Kódolástechnika

Buttyán Levente Györfi László Győri Sándor Vajda István

2006. december 18.

Tartalomjegyzék

Előszó

A digitális kommunikáció, adattárolás, -védelem és -feldolgozás tudományos alaposságú vizsgálata Claude Shannon 1948-as, A kommunikáció matematikai elmélete című cikkével vette kezdetét, így az információ- és kódelmélet tudományága alig fél évszázados múltra tekint vissza. Napjainkban azonban szinte észrevétlenül, ám mégis kikerülhetetlenül jelen van az élet és az infokommunikációs ipar tömegszolgáltatásainak számos területén. Lehetővé teszi a filmek és zeneművek korábban elképzelhetetlenül jó minőségű és gazdaságos tárolását DVD-n illetve CD-n, és az alkotások tetszőleges számú lejátszását minőségromlás nélkül. Ki álmodott volna a múlt század derekán arról, hogy egy egész estés mozifilm elférhet egy korongon, akár többféle szinkronnal együtt? Gondolhatunk az utóbbi években rohamosan elterjedt második és harmadik generációs mobiltelefonokra, vagy a közeljövőben hasonló karriert befutó intelligens chipkártyákra. Említhetnénk ezeken kívül számos más számítástechnikai és távközlési alkalmazást.

Ahhoz, hogy ezekben az információtovábbító illetve -tároló berendezésekben az információkezelés egyszerre legyen gazdaságos és biztonságos, szükség van olyan informatikusokra, akik a különböző kódolási technikákat és algoritmusokat készségszinten tudják használni.

Az informatikusok és villamosmérnökök oktatásában éppen ezért régóta lényeges szerepet kap ez a terület. A Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Karán, a mérnök-informatikus egyetemi szakon a kezdetektől a tanterv részei az Információelmélet és a Kódelmélet tárgyak, öt éve pedig az Adatbiztonság tárgy is. Ezek a tárgyak a kétciklusú képzésben az eredeti célkitűzéssel az MSc szintre kerülnek.

A kétszintű mérnökképzés bevezetésével fel kell készülnünk ezen ismeretanyag alapjainak az eddigiektől célkitűzésében és módszertanában eltérő átadására. A BSc, vagyis az alapképzés jól használható gyakorlati ismeretek elsajátítására épít, míg az MSc, vagyis a mesterképzés feladata az elméleti háttér részletes bemutatása, amely alapján a mérnök képes az információtechnológia területén fejlesztési, tervezési illetve kutatási problémák megoldására. A BME mérnök-informatikus BSc képzésében önálló alaptárgyként jelenik meg a Kódolástechnika, melyet a hallgatók egy félévben, heti négy órában tanulnak. Jegyzetünk ehhez a tárgyhoz készült, de törekedtünk arra, hogy az elkészült mű más felsőoktatási intézmények informatikus alapképzéseibe is könnyűszerrel beilleszthető legyen. A jegyzetnek a mérnök-informatikus alapképzésen kívül jóval szélesebb a potenciá-

lis olvasóközönsége. Az egyetemi és főiskolai oktatásban haszonnal forgathatják más mérnökszakos vagy matematikus, alkalmazott matematikus szakos hallgatók is, de nem szabad megfeledkeznünk a végzett mérnökök szakmai önképzéséről vagy szervezett tanfolyamok keretében folyatott továbbképzéséről sem.

A korlátozott terjedelem és az oktatásra fordítható korlátos idő miatt nem lehet célunk a bevezető matematikai ismeretek ismételt elmondása, hiszen ezzel a hallgatók más alapozó tárgyak keretében ismerkednek meg (a BME mérnökinformatikus szakán ilyen tárgy például a Bevezetés a számításelméletbe vagy a Valószínűségszámítás). Ugyanakkor csak az elemi lineáris algebrai és a bevezető valószínűségszámítási tudásra építünk.

Célkitűzésünk, hogy ismertessük az információ átvitelének illetve tárolásának alapvető kódolási algoritmusait és ezek tulajdonságait. A szóba jövő információtechnológiai feladatokat négy csoportba szokás sorolni:

- hibajavító kódolás,
- adatbiztonság,
- adattömörítés és
- veszteséges forráskódolás.

Ehhez igazodnak a következő fejezetek, amelyekben a legfontosabbb technikák után alkalmazási példákat adunk (pl. CD, GSM, Internet, videotömörítés), majd gyakorló feladatok és azok megoldásai következnek. Végül a legfontosabb ismeretek összefoglalása és irodalomjegyzék zárja a fejezeteket.

Ez a jegyzet a Korszerű Mérnökért Alapítvány „Tankönyv, szakkönyv, jegyzet” projektjének keretében készült. Ezúton is szeretnénk köszönetet mondani az Alapítvány támogatásáért.

Budapest, 2006. december 18.

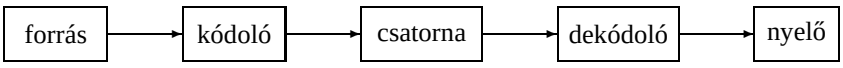
Buttyán Levente
Györfi László
Győri Sándor
Vajda István

1. fejezet

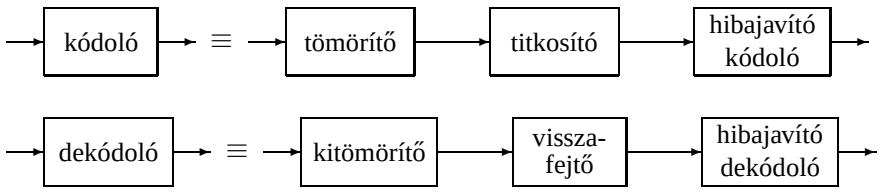
Bevezetés

A következő fejezetekben a hírközlési rendszerek működését egzakt matematikai eszközökkel fogjuk vizsgálni. Persze sokféle matematikai modellt állíthatunk hírközlési feladatokra, de kezdetben az egyik legegyszerűbbet választva jól kifejezhetjük a probléma lényegét. A hírközlés alapfeladata az, hogy valamely jelsorozatot („információt”) el kell juttatni egyik helyről a másikra (vagy tárolni kell). A távolságot (vagy időt) áthidaló hírközlési eszköz — a csatorna — azonban csak meghatározott típusú jeleket képes átvinni. Az információforrás által előállított jelfolyamot kódolással meg kell feleltetni egy, a csatorna által használt jelekből álló jelfolyamnak. A felhasználó (vevő, nyelő) a csatorna kimenetén pontosan, vagy megközelítőleg visszaállítja, dekódolja az üzenetet. Az 1.1. ábrán egy ilyen rendszer blokkdiagrammja látható.

A kódoló általános esetben három részből áll (1.2. ábra). Az első a forráskódoló vagy tömörítő, amelynek célja a forrás kimenetén jelenlévő felesleges ismétlődések, függőségek, vagyis az ún. redundancia eltávolítása. Ez a tömörítés akkor lehetséges, ha vagy az egyes betűk nem egyformán valószínűek, vagy az egymás után következő betűk nem függetlenek. Megkövetelhetjük, hogy a forrásdekódoló (kitömörítő) pontosan helyreállíthassa az adatokat, de megelégedhetünk részleges helyreállíthatósággal is. A forráskódoló tehát a forrás üzeneteit gazdaságosan, tömören reprezentálja. Ezzel foglalkozik a 4. és 5. fejezet. A kódoló második része a titkosító (3. fejezet), amely egyrészt az adatvédelmet garantálja, vagyis azt, hogy illetéktelenek ne férhessenek hozzá az üzenet tartalmához, másrészt pedig a hitelesítést oldja meg, vagyis annak bizonyítását, hogy az üzenet valóban a feladótól származik. A harmadik rész a csatornakódoló vagy hibajavító kódoló (2. fejezet), amelynek feladata a tömörítőével éppen ellentétes. Irányított módon visz be re-



1.1. ábra. Hírközlési rendszer blokkdiagramja



1.2. ábra. A kódoló és a dekódoló felépítése

dundanciát az adatokba úgy, hogy a csatorna átviteli hibái javíthatók legyenek. Ezeknek megfelelően az 1.2. ábrán látható dekódoló is három részből áll: csatornadekódoló (hibajavító dekódoló), visszafejtő és forrásdekódoló (kitömörítő).

A második és a harmadik rész feladatának érzékeltetésére tegyük fel, hogy egy ügyfél az interneten akar egy banki tranzakciót lebonyolítani. Ekkor nyilván elvárja, hogy a megadott adatok pontosan legyenek továbbítva (hibajavító kódolás), más személy ne tudja meg ezeket az adatokat még akkor sem, ha az információtovábbítás nyilvános hálózaton, például mobil eszközön történik (titkosítás), a bank számára pedig bizonyított legyen, hogy valóban ő kezdeményezte a tranzakciót (hitelesítés, digitális aláírás).

A forráskódoló és -dekódoló együttese foglalja magába a forráskódot, a csatornakódoló és -dekódoló együttese pedig a csatornakódot. A csatorna a kommunikációs rendszer tervezője számára adott, meg nem változtatható tulajdonságokkal rendelkező modell, amely leírja, hogyan zajlik az adatok átvitele vagy tárolása. A tervező ennek figyelembe vételével azonban szabadon megválaszthatja a forráskódot és a csatornakódot úgy, hogy ez minél jobb adattovábbítást eredményezzen.

2. fejezet

Hibajavító kódolás

A hibakorlátozó kódolás célja információ hibázó kommunikációs csatornákon történő megbízható átvitele illetve hibázó adattárolókon történő megbízható tárolása. E célból az információt kisebb egységekre bontjuk, majd redundáns információval bővítve kódszavakba képezzük, kódoljuk. A redundancia hozzáadása teszi lehetővé az információ védelmét hibázás esetén. A hibakorlátozó kódolás két fő technikája a hibajelzés illetve a hibajavítás. Hibajelzés esetén a cél annak eldöntése, hogy történt-e meghibásodás az információ továbbítása illetve tárolása során. Hibajavítás esetén ezen túlmenően a hibák kijavítása is feladat. Több kapcsolatos kérdés merül fel: hogy történjen a kódszavakba képezés, azaz a kódolás, ha célunk a minél erősebb hibakorlátozó képesség, azon feltétel mellett, hogy a rekonstrukció (dekódolás) a számításigényét tekintve hatékony maradjon.

Az alábbiakban a hibakorlátozó kódok konstrukciós és dekódolási alapelveit tekintjük át, a hangsúlyt a kapcsolatos fogalmakra és alapvető algoritmusokra helyezve.

2.1. Kódolási alapfogalmak

A hibajavító kódolás alapvető módszereit a 2.1. ábrán látható egyszerű hírközlési struktúra kapcsán vizsgáljuk.

Az $\mathbf{u}$ és $\mathbf{u}'$ vektorok koordinátái egy F halmazból veszik értékeiket, mely halmazt **forrásábécé**nek nevezzük. A kódoló a k hosszú $\mathbf{u}$ vektort (az **üzenet**et) egy n hosszú $\mathbf{c}$ vektorba (a **kódszó**ba) képezi le. A $\mathbf{c}$ koordinátái egy Q halmazból veszik értékeiket. A Q-t **kódábécé**nek vagy csatorna bemeneti ábécének fogjuk hívni.

2.1. ábra. Hírközlési rendszer blokkdiagramja

A csatorna kimenete $\mathbf{v}$, szintén egy n hosszú vektor, melynek koordinátái szintén Q-beliek.

Egy

$$\mathbf{c} = (c_1, \ldots, c_n)$$

bemeneti és

$$\mathbf{v} = (v_1, \ldots, v_n)$$

kimeneti sorozat esetén azt mondjuk, hogy az m-edik időpontban a csatorna hibázott, ha $c_m \neq v_m$. Jelölje $d(\mathbf{c},\mathbf{v})$ azon i pozíciók számát, ahol $c_i \neq v_i$.

$d(\mathbf{c},\mathbf{v})$ neve a $\mathbf{c},\mathbf{v}$ sorozatok **Hamming-távolság**a, és azt mondjuk, hogy a $\mathbf{c}$ sorozat küldésekor és a $\mathbf{v}$ sorozat vételekor a hibák száma $t = d(\mathbf{c},\mathbf{v})$. Ezt az esetet nevezzük **egyszerű hibázás**nak, amikor a hiba helye és értéke egyaránt ismeretlen. $d(\mathbf{c},\mathbf{v})$ valóban távolság, hiszen

$$d(\mathbf{c},\mathbf{v}) \geq 0,$$

$$d(\mathbf{c},\mathbf{v}) = d(\mathbf{v},\mathbf{c}),$$

és igaz a háromszög-egyenlőtlenség:

$$d(\mathbf{c},\mathbf{v}) \leq d(\mathbf{c},\mathbf{w}) + d(\mathbf{w},\mathbf{v}).$$

Kód (blokk-kód) alatt a Q^n halmaz egy C részhalmazát értjük, azaz C minden eleme egy n hosszú vektor, melynek koordinátái Q-beliek. C elemeit kódszavaknak nevezzük. A továbbiakban a $C(n,k,d)$, illetve rövidítettebb formában $C(n,k)$ jelölést alkalmazzuk, kiemelve a kód paramétereit. A **kódolás** egy invertálható függvény, mely k hosszú F-beli sorozatot — üzenetet — képez le egy kódszóba, formalizálva:

$$f : F^k \to C,$$

és minden különböző $\mathbf{u}$, $\mathbf{u}'$-re $f(\mathbf{u})$, $f(\mathbf{u}')$ is különböző.

Dekódolás alatt két függvény egymásutánját értjük. Az egyik a csatorna kimenetének n hosszú szegmensét képezi le C-be, azaz igyekszik eltalálni a küldött kódszót, a másik pedig az f függvény inverze, tehát

$$g : Q^n \to C, \qquad f^{-1} : C \to F^k.$$

Mivel f egyértelműen meghatározza f^{-1}-et, ezért dekódolás alatt a későbbiekben csak a g függvényt értjük. A g dekódoló függvényként az algebrai hibajavító kódok elméletében speciális függvényt választunk, nevezetesen a $\mathbf{v}$ vektorhoz megkeressük azt a $\mathbf{c}' \in C$ kódszót, mely Hamming-távolság szerint hozzá a legközelebb van, vagy ha több ilyen van, akkor az egyiket, tehát teljesül, hogy ha $\mathbf{c}' = g(\mathbf{v})$, akkor

$$d(\mathbf{c}',\mathbf{v}) = \min_{\mathbf{c}\in C} d(\mathbf{c},\mathbf{v}).$$

A dekódolás feladata ezek után arra a messze nem triviális feladatra szűkül, hogy egy **v** vett szóhoz hogyan keressük meg a hozzá legközelebbi $\mathbf{c}'$ kódszót anélkül, hogy minden $d(\mathbf{c},\mathbf{v})$-t kiszámítanánk. Ha mégis kiszámítjuk ezeket a távolságokat, és minden **v**-hez megkeressük a hozzá legközelebbi **c** kódszót, majd a neki megfelelő üzenetet, akkor elvben azt eltárolhatjuk, és így egy táblázathoz jutunk, melynek címét **v** adja, tartalma pedig a **v**-nek megfelelő dekódolt üzenet. Ez a **táblázatos dekódolás**nak a legegyszerűbb, de legpazarlóbb esete, hiszen a táblázat q^n darab üzenetből áll, ahol q a Q kódábécé elemszáma.

2.1. példa (ismétléses kód). Tekintsük azt a nagyon egyszerű kódolást, amikor bináris forrásunk egyes bitjeit tekintjük üzenetnek, s háromszor megismételve küldjük a kommunikációs csatornába a következő leképezés szerint:

$$\begin{array}{lll} u & c_1\,c_2\,c_3 & \\ 0 \to & 0\;0\;0 & \mathbf{c}_1 \\ 1 \to & 1\;1\;1 & \mathbf{c}_2 \end{array}$$

A kód egy hibát képes javítani, mivel 1 hiba esetén a vett szó az átküldött kódszótól egy, míg a másik kódszótól kettő Hamming-távolságra van. A kód egy illetve kettő hibát képes jelezni, mivel ezen esetekben a vett szó nem lehet kódszó. A konstrukció általánosítható: legyen az ismétlések száma n, $n \geq 3$ páratlan szám. Könnyen látható, hogy a kód $(n-1)/2$ hibát képes javítani, valamint $n-1$ hibát jelezni.

2.2. példa (egyszerű paritáskód). Tekintsük azt a feladatot, amikor a forrás a következő négy lehetséges üzenetet bocsátja ki, a 00, 01, 10, 11 üzeneteket, amelyekhez a kódoló a következő, 3 hosszú kódszavakat rendeli hozzá:

$$\begin{array}{lll} u_1u_2 & c_1\,c_2\,c_3 & \\ 0\;0 \to & 0\;0\;0 & \mathbf{c}_1 \\ 0\;1 \to & 0\;1\;1 & \mathbf{c}_2 \\ 1\;0 \to & 1\;0\;1 & \mathbf{c}_3 \\ 1\;1 \to & 1\;1\;0 & \mathbf{c}_4 \end{array}$$

azaz az u_1u_2 bitet kiegészítjük egy paritásbittel. A kód egy hibát képes jelezni. A kód nem képes hibát javítani: pl. ha $\mathbf{c}_1$ kód továbbításakor a második bit meghibásodik, akkor a $\mathbf{v} = (0,1,0)$ vett szó azonos, egy távolságra lesz a $\mathbf{c}_1$ és $\mathbf{c}_2$ kódszavakhoz.

2.3. példa. A 2.2 példában szereplő üzenetekhez rendeljünk 5 hosszú kódszavakat:

$$\begin{array}{lll} u_1u_2 & c_1\,c_2\,c_3\,c_4\,c_5 & \\ 0\;0 \to & 0\;0\;0\;0\;0 & \mathbf{c}_1 \\ 0\;1 \to & 0\;1\;1\;0\;1 & \mathbf{c}_2 \\ 1\;0 \to & 1\;0\;1\;1\;0 & \mathbf{c}_3 \\ 1\;1 \to & 1\;1\;0\;1\;1 & \mathbf{c}_4 \end{array}$$

A g és az f^{-1} függvényt a következő táblázat foglalja össze:

$v_1\,v_2\,v_3\,v_4\,v_5$		$c'_1\,c'_2\,c'_3\,c'_4\,c'_5$		$u'_1\,u'_2$
0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1	$\rightarrow$	0 0 0 0 0	$\rightarrow$	0 0
0 1 1 0 1 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 1 1 1 0 1 1 0 0	$\rightarrow$	0 1 1 0 1	$\rightarrow$	0 1
1 0 1 1 0 0 0 1 1 0 1 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 1 1	$\rightarrow$	1 0 1 1 0	$\rightarrow$	1 0
1 1 0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 0	$\rightarrow$	1 1 0 1 1	$\rightarrow$	1 1
0 0 0 1 1 0 1 0 1 0 1 0 0 0 1 1 1 0 0 0	$\rightarrow$	0 0 0 0 0	$\rightarrow$	0 0
0 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 1 0 0	$\rightarrow$	0 1 1 0 1	$\rightarrow$	0 1

Az első 24 kimeneti szó dekódolásakor nincs probléma, hiszen minden 6 szóból álló csoport minden szava olyan, hogy vagy kódszó (az első helyen álló), vagy a kódszó egy bitjének megváltoztatásával (egy hibával) képződött. A 25–28. szavak mindegyike $\mathbf{c}_1$-től és $\mathbf{c}_4$-től 2, míg $\mathbf{c}_2$-től és $\mathbf{c}_3$-tól 3 távolságra van. Itt önkényesen

a dekódolás eredménye $\mathbf{c}_1$ (jobb érvünk nincs, mint az, hogy jobban szeretjük az almát, mint a cseresznyét). A 29–32. szavak mindegyike $\mathbf{c}_1$-től és $\mathbf{c}_4$-től 3, míg $\mathbf{c}_2$-től és $\mathbf{c}_3$-tól 2 távolságra van. Itt (szintén önkényesen) a dekódolás eredménye $\mathbf{c}_2$.

A későbbiekben kiderül, hogy a kódoló f függvény leglényegesebb tulajdonsága a C kód egy paramétere, amit **kódtávolság**nak nevezünk, és $d_{\min}$-nel jelölünk:

$$d_{\min} = \min_{\substack{\mathbf{c}\neq\mathbf{c}' \\ \mathbf{c},\mathbf{c}'\in C}} d(\mathbf{c},\mathbf{c}').$$

A 2.1. és a 2.3. példákban $d_{\min} = 3$, míg a 2.2. példában $d_{\min} = 2$.

A **hibajelzés** a hibakorlátozó kódolás azon feladata, amikor a vevőben csupán detektálni akarjuk a hibázás tényét, azaz azt kérdezzük, hogy van-e hiba. Nyilván egy $\mathbf{v}$ vett szó esetén akkor tudjuk a hibázást észrevenni, ha $\mathbf{v}$ nem kódszó, amire garancia, hogy ha $\mathbf{c}$ küldött kódszó esetén

$$d_{\min} > d(\mathbf{v},\mathbf{c}),$$

azaz a hibák számára

$$d_{\min} > t,$$

tehát egy $d_{\min}$ kódtávolságú kód minden, legfeljebb $d_{\min} - 1$ számú hibát jelezni tud.

Mivel a 2.1. és a 2.3. példákban $d_{\min} = 3$, ezért ez a kód 2 hibát tud jelezni, míg a 2.2. példa kódja $d_{\min} = 2$ miatt 1-et.

Hibajavítás esetén azt kérdezzük, hogy ha t a hibák száma, akkor mi biztosítja, hogy a $\mathbf{v}$ vett szóból a $\mathbf{c}$ küldött kódszó egyértelműen visszaállítható legyen, azaz minden más $\mathbf{c}'$ kódszóra

$$d(\mathbf{v},\mathbf{c}') > d(\mathbf{v},\mathbf{c}) \tag{2.1}$$

legyen. Mivel a Hamming-távolság valóban távolság, ezért teljesíti a háromszög-egyenlőtlenséget, azaz

$$d(\mathbf{v},\mathbf{c}') \geq d(\mathbf{c},\mathbf{c}') - d(\mathbf{v},\mathbf{c}), \tag{2.2}$$

tehát (2.1) úgy biztosítható, hogy

$$d(\mathbf{c},\mathbf{c}') - d(\mathbf{v},\mathbf{c}) > d(\mathbf{v},\mathbf{c}),$$

ugyanis, ha ez utóbbi teljesül, akkor (2.1) is teljesül, azaz minden $\mathbf{c}' \neq \mathbf{c}$-re

$$d(\mathbf{c},\mathbf{c}') > 2d(\mathbf{v},\mathbf{c}),$$

vagyis

$$\frac{d_{\min}}{2} > d(\mathbf{v},\mathbf{c}).$$

Összefoglalva: **egyszerű hibázás** esetén $\left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$ hiba javítható.

A 2.1. és a 2.3. példa kódja 1 hibát tud javítani, míg a 2.2. példa kódjának hibajavító képessége 0.

Gyakran fordul elő olyan hibázás, amikor tudjuk, hogy egy pozícióban hiba lehet, vagyis tudjuk, hogy más pozíciókban nincs hiba, tehát a hiba helyét ismerjük, csak a hiba értékét nem. Az ilyen hibát **törléses hibá**nak nevezzük. Egyszerűen belátható, hogy minden $d_{\min}-1$ törléses hiba javítható, ugyanis a legrosszabb esetben sem fordulhat elő, hogy két $\mathbf{c},\mathbf{c}'$ kódszó ugyanazon, de legfeljebb $d_{\min}-1$ pozíciójának törlésével ugyanazt a szót kapnánk.

A 2.1. és a 2.3. példa kódja 2 törléses hibát tud javítani, míg a 2.2. példa kódja 1-et.

Nyilván adott n kódszóhosszúság és $d_{\min}$ kódtávolság esetén nem lehet akármilyen nagy méretű kódot konstruálni:

2.1. tétel (Singleton-korlát). *Egy M kódszóból álló, n hosszú és d_{min} kódtávolságú kódra*

$$M \leq q^{n-d_{min}+1}.$$

BIZONYÍTÁS: Legyen k egy természetes szám, melyre

$$q^{k-1} < M \leq q^k.$$

Mivel a $k-1$ hosszú különböző sorozatok száma q^{k-1}, ezért $q^{k-1} < M$ miatt létezik két kódszó $\mathbf{c}$ és $\mathbf{c}'$, melyek az első $k-1$ koordinátában megegyeznek. Ezekre

$$d(\mathbf{c},\mathbf{c}') \leq n-k+1,$$

következésképpen

$$d_{\min} \leq n-k+1,$$

azaz

$$M \leq q^k \leq q^{n-d_{\min}+1}.$$ ■

Jellegzetes esetben $M = q^k$, vagyis a kódoló k hosszú forrásszegmensekhez rendel n hosszú vektorokat. Azt mondjuk ilyenkor, hogy a kódunk (n,k) paraméterű. Ebben az esetben a Singleton-korlát alakja

$$d_{\min} \leq n-k+1.$$

2.1. definíció. *Azon kódot, melyre a Singleton-korlátban egyenlőség áll,* ***maximális távolságú*** *vagy* ***MDS*** *(maximum distance separable) kódnak nevezzük.*

A 2.1. példában $k=1$, $n=3$, $d_{\min}=3$, tehát ez a kód MDS kód. Ugyanakkor a 2.3. példában $k=2$, $n=5$, $d_{\min}=3$, így ez a kód nem MDS.

A 2.1. és a 2.3. példák kódjai egy hibát képesek javítani. Ehhez a következő szemléletes geometriai képet rendelhetjük. A kódszavak mint középpontok körül képzeljünk el 1 Hamming-sugarú gömböket, azaz a gömb felületén olyan —

kódszóhossz hosszúságú — bináris szavak találhatók, amelyeknek a középpontban levő szótól való Hamming-távolsága 1. Ha egy hiba keletkezik, akkor a vett szó a leadott kódszó körüli gömbön helyezkedik el. Más szavakkal ezen gömbök a dekódolási tartományok. Tekintsük először a 2.1. példa kódját. A két gömb a következő szavakat tartalmazza:

$$(0,0,0) : (1,0,0), (0,1,0), (0,0,1)$$

$$(1,1,1) : (0,1,1), (1,0,1), (1,1,0)$$

A két gömb tartalmazza az összes 3 bit hosszúságú bináris szót, teljesen kitölti 3 bit hosszúságú bináris szavak terét. A 2.3. példák kódja esetén a négy 1 Hamming-sugarú gömb nem tölti ki az 5 bit hosszúságú bináris szavak terét. Ugyanakkor nem is tudjuk növelni 2 Hamming-sugárra a gömbök méretét anélkül, hogy azok átlapolódnának. Abban az esetben, ha gömbi dekódolási tartományokkal hézagmentesen képesek vagyunk lefedni a teret perfekt kódokról beszélünk. Ennek megfelelően a 2.1. példa kódja perfekt, míg a 2.3. példa kódja nem az. Nagyon egyszerű összefüggés adódik az 1 hibát javító bináris perfekt kódok paraméterei közötti összefüggésre. A tér elemeinek száma 2^n, a kódszavak száma 2^k, így egy gömb elemeinek száma ezek hányadosa, azaz 2^{n-k}. Másfelől, egy 1 Hamming sugarú gömbben $1+n$ elem van, így adódik, hogy 1 hibát javító bináris perfekt kódok esetén

$$1+n = 2^{n-k}. \tag{2.3}$$

A következő szakaszban mutatunk egy az ismétléses kódnál hatékonyabb perfekt kódot, a bináris Hamming-kódot. A (2.3) összefüggés könnyen általánosítható tetszőleges $t \geq 1$ hiba javítás esetére: a képlet bal oldalán a gömb elemeinek száma, a középpontól $0,1,2,\ldots,t$ Hamming-távolságra levő szavak számának összege, azaz

$$V(n,t) = 1+n+\binom{n}{2}+\cdots+\binom{n}{t}$$

áll. A fenti gondolatmenet alapján adódó Hamming-korlát bináris esetben az alábbi

$$V(n,t) \leq 2^{n-k}.$$

A Hamming-korlát nembináris esetben a következő alakot ölti:

$$\sum_{i=0}^{t}\binom{n}{i}(q-1)^i \leq q^{n-k}$$

2.2. Lineáris kódok

Bináris lineáris kódok

Ebben a szakaszban először bináris kódok egy fontos csoportjával ismerkedünk meg. A továbbiakban a kódjainkban szereplő kódszavakat alkotó szimbólumok legyenek 0 vagy 1 értékűek, az összeadás és a szorzás pedig a bináris összeadás és a bináris szorzás, azaz a modulo 2 összeadás és a modulo 2 szorzás.

Vezessük be a lineáris kód fogalmát:

2.2. definíció. *Egy bináris C kód **lineáris**, ha a C halmaza lineáris tér, azaz ha minden $\mathbf{c}, \mathbf{c}' \in C$-re $\mathbf{c} + \mathbf{c}' \in C$.*

A lineáris kód definíciójából következik, hogy a $\mathbf{0}$ vektor eleme minden lineáris kódnak, vagyis minden lineáris kód esetén a $\mathbf{0}$ kódszó. Egyszerűen belátható, hogy a 2.3. és 2.2. példa kódja lineáris.

A lineáris kódok jelentőségét az adja, hogy az egyes üzenetekhez tartozó kódszavak viszonylag egyszerűen generálhatók, és ugyancsak egyszerű módszer található a vett kódszavak hibamentességének vizsgálatára, vagyis a hibadetektálásra, és a hibák javítása sem bonyolult. A következőkben e módszereket fogjuk bemutatni.

Jelentsen C továbbra is egy lineáris kódot, a kódszóhossz legyen n. Ekkor C az n hosszúságú bináris koordinátájú vektorok terének egy altere; „kódszó” helyett gyakran „vektor”-t fogunk mondani.

A valós vektortérben megszokott lineáris függetlenség és bázis fogalmak itt is teljesen hasonlóan értelmezhetők, vagyis

2.3. definíció. *A $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_k \in C$ vektorok lineárisan függetlenek, ha $\alpha_i \in \{0,1\}$ mellett*

$$\sum_{i=1}^{k} \alpha_i \mathbf{g}_i = 0$$

csak úgy állhat elő, ha $\alpha_i = 0$ minden $i = 1, 2, \ldots, k$-ra.

2.4. definíció. *A $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_k \in C$ vektorok a C lineáris tér egy bázisát alkotják, ha lineárisan függetlenek, továbbá igaz az, hogy minden $\mathbf{c} \in C$ vektor előállítható*

$$\mathbf{c} = \sum_{i=1}^{k} u_i \mathbf{g}_i \tag{2.4}$$

alakban, ahol $u_i \in \{0,1\}$ minden $i = 1, 2, \ldots, k$-ra.

Az utóbbi definícióban a bázist alkotó vektorok lineáris függetlenségéből következik, hogy a kódszavak fenti típusú előállítása egyértelmű is, ha ugyanis létezne két különböző előállítás valamely $\mathbf{c} \in C$-re, tehát

$$\mathbf{c} = \sum_{i=1}^{k} u_i \mathbf{g}_i$$

és

$$\mathbf{c} = \sum_{i=1}^{k} y_i \mathbf{g}_i,$$

ahol nem áll fenn $u_i = y_i$ minden i-re, akkor a két egyenletet kivonva egymásból a nullvektornak egy nem triviális előállítását kapnánk a bázisvektorokkal, ami ellentmondana azok lineáris függetlenségének.

A (2.4) egyenlőség felírható mátrixalakban:

$$\mathbf{c} = \mathbf{u}\mathbf{G}, \tag{2.5}$$

ahol $\mathbf{u} = (u_1, u_2, \ldots, u_k)$, $\mathbf{G}$ pedig a bázisvektorokból mint sorvektorokból álló mátrix. A (2.5) egyenlettel tehát egy k-dimenziós és egy n-dimenziós vektort rendelünk össze lineáris transzformációval, mégpedig kölcsönösen egyértelmű módon. Azt fogjuk mondani, hogy az $\mathbf{u}$ üzenethez a $\mathbf{c}$ kódszó tartozik.

A k-dimenziós $\mathbf{u}$ vektorokkal 2^k-féle üzenetet fejezhetünk ki, s ezeket kódoljuk a C kóddal. C elemei azonban n-dimenziós vektorok, és n nem kisebb k-nál, hiszen k az n-dimenziós vektorok C alterének dimenziószáma. A $k = n$ esetnek nincs most jelentősége, ha k kisebb, mint n, akkor viszont világos, hogy nem minden vektort kell felhasználni kódszónak, vagyis kódunk redundáns lesz, s ezt a redundanciát tudjuk hibajavításra felhasználni.

Az üzenetekhez a kódszavakat a $\mathbf{G}$ mátrix segítségével rendeljük hozzá, vagyis a $\mathbf{G}$ mátrix jelöli ki az n-dimenziós vektortérnek a kódot jelentő C alterét, a kódot $\mathbf{G}$ „generálja”.

2.5. definíció. *A fenti tulajdonságú* $\mathbf{G}$ *mátrixot a* C *kód* ***generátormátrix****ának nevezzük.*

Vegyük észre, hogy ha nem törődünk azzal, hogy melyik kódszó melyik üzenethez tartozik, csak a kódszavak halmazát tekintjük, akkor $\mathbf{G}$ nem egyértelmű, vagyis több mátrix is generálhatja ugyanazt a C kódszóhalmazt. A következő definíció egy megfeleltetést definiál az üzenetek és a kódszavak között.

2.6. definíció. *Egy* (n,k) *paraméterű lineáris kód* ***szisztematikus****, ha minden kódszavára igaz, hogy annak utolsó* $n-k$ *szimbólumát elhagyva éppen a neki megfelelő* k *hosszúságú üzenetet kapjuk, más szavakkal a* k *hosszú üzenetet egészítjük ki* $n-k$ *karakterrel.*

A 2.1. szakaszban már leszögeztük, hogy dekódolás alatt csak az esetleges hibák kijavítását értjük, aminek eredményeképp egy kódszót kapunk. Az üzenetvektor visszanyeréséhez még el kell ugyan végezni a kódolás inverz műveletét, ez azonban rendszerint triviális lépés, szisztematikus kód esetén például csak el kell hagyni a kódszó egy részét (a végét).

Szisztematikus kód esetén a generátormátrix is egyértelmű, mégpedig

$$\mathbf{G} = (\mathbf{I}_k, \mathbf{B}) \tag{2.6}$$

alakú, ahol $\mathbf{I}_k$ a $k \times k$ méretű egységmátrix, $\mathbf{B}$ pedig $k \times (n-k)$ méretű mátrix. Az $\mathbf{u}$ üzenethez tartozó $\mathbf{c}$ kódszó szerkezete tehát:

$$\mathbf{c} = (u_1, u_2, \ldots, u_k, c_{k+1}, c_{k+2}, \ldots, c_n).$$

A $\mathbf{c}$ első k koordinátájából álló szegmensét **üzenetszegmens**nek, az utolsó $n-k$ koordinátájából állót **paritásszegmens**nek nevezzük.

A lineáris kódok további tulajdonságai elvezetnek az ígért egyszerű hibadetektáláshoz illetve hibajavításhoz.

2.7. definíció. *Ha egy $n-k$ sorból és n oszlopból álló $\mathbf{H}$ mátrixra*

$$\mathbf{H}\mathbf{c}^T = \mathbf{0}$$

*akkor és csak akkor, ha $\mathbf{c} \in C$ ($\mathbf{c}^T$ a $\mathbf{c}$ transzponáltja), akkor $\mathbf{H}$-t a C kód **paritásellenőrző mátrix**ának nevezzük. (Röviden paritásmátrixot fogunk mondani.)*

$\mathbf{H}$ segítségével tehát meg tudjuk állapítani, hogy egy vett szó valóban kódszó-e.

2.2. tétel. *Ha $\mathbf{G}$ és $\mathbf{H}$ ugyanazon C lineáris kód generátormátrixa illetve paritásmátrixa, akkor*

$$\mathbf{H}\mathbf{G}^T = \mathbf{0}.$$

BIZONYÍTÁS: Jelölje Q^k a k hosszú bináris sorozatok halmazát. Ekkor minden $\mathbf{u} \in Q^k$-hoz létezik $\mathbf{c} \in C$, amire $\mathbf{c} = \mathbf{u}\mathbf{G}$. Ugyanakkor $\mathbf{c} \in C$ miatt $\mathbf{H}\mathbf{c}^T = \mathbf{0}$, azaz

$$\mathbf{H}\mathbf{c}^T = \mathbf{H}(\mathbf{u}\mathbf{G})^T = \mathbf{H}\mathbf{G}^T\mathbf{u}^T = \mathbf{0}.$$

Az utolsó egyenlőség pedig csak úgy állhat fönn minden $\mathbf{u} \in Q^k$-ra, ha $\mathbf{H}\mathbf{G}^T = \mathbf{0}$, amint állítottuk. ■

A 2.2. tétel alapján szisztematikus generátormátrix felhasználásával könnyen előállíthatjuk a kód egy paritásellenőrző mátrixát. Keressük $\mathbf{H}$-t

$$\mathbf{H} = (\mathbf{A}, \mathbf{I}_{n-k})$$

alakban. A 2.2. tétel alapján

$$\mathbf{H}\mathbf{G}^T = (\mathbf{A}, \mathbf{I}_{n-k})(\mathbf{I}_k, \mathbf{B})^T = \mathbf{A} + \mathbf{B}^T = \mathbf{0}.$$

Azaz

$$\mathbf{A} = -\mathbf{B}^T$$

kell teljesüljön. (Bináris esetben $-\mathbf{B}^T = \mathbf{B}^T$.)

2.4. példa. Adjuk meg a 2.3. példa kódjának szisztematikus generátormátrixát, ha van, és a paritásmátrixot! Ezt úgy kapjuk meg, ha **G** első sora $\mathbf{c}_3$, míg a második $\mathbf{c}_2$, mivel ekkor az első 2×2-es részmátrix egységmátrix:

$$\mathbf{G} = \begin{pmatrix} 1\,0\,1\,1\,0 \\ 0\,1\,1\,0\,1 \end{pmatrix}.$$

A fentiek alapján a paritásmátrix:

$$\mathbf{H} = \begin{pmatrix} 1\,1\,1\,0\,0 \\ 1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1 \end{pmatrix}.$$

Ugyanígy kapjuk a 2.2. példa szisztematikus generátormátrixát és paritásmátrixát:

$$\mathbf{G} = \begin{pmatrix} 1\,0\,1 \\ 0\,1\,1 \end{pmatrix},$$

$$\mathbf{H} = \begin{pmatrix} 1\,1\,1 \end{pmatrix}.$$

A következőkben a súly fogalmát definiáljuk, majd megmutatjuk, hogy lineáris kódoknál a minimális súly a kódtávolsággal egyenlő. (Emlékeztetünk, hogy két kódszó távolsága azon koordinátáik száma, ahol a két kódszó különbözik.)

2.8. definíció. *Egy* **c** *vektor* ***súly****a a koordinátái között levő nem nulla elemek száma, jelölése* $w(\mathbf{c})$.

2.9. definíció. *Egy* C *kód* ***minimális súly****án a*

$$w_{min} = \min_{\substack{\mathbf{c} \in C \\ \mathbf{c} \neq \mathbf{0}}} w(\mathbf{c})$$

számot értjük.

2.3. tétel. *Ha* C *lineáris kód, akkor a kódtávolsága megegyezik a minimális súlyával, azaz*

$$d_{min} = w_{min}.$$

BIZONYÍTÁS:

$$d_{\min} = \min_{\mathbf{c} \neq \mathbf{c}'} d(\mathbf{c}, \mathbf{c}') = \min_{\mathbf{c} \neq \mathbf{c}'} w(\mathbf{c} - \mathbf{c}') = \min_{\mathbf{c}'' \neq \mathbf{0}} w(\mathbf{c}'') = w_{\min},$$

ahol az utolsó előtti egyenlőség felírásakor a C kód linearitását használtuk ki, ebből következik ugyanis, hogy $\mathbf{c}'' = \mathbf{c} - \mathbf{c}'$ is kódszó, továbbá, az is, hogy minden kódszó előáll ilyen különbség alakjában. (Utóbbi ahhoz szükséges, hogy a minimum képzésekor valóban minden $\mathbf{c}'' \in C$-t figyelembe vehessünk.) ■

A 2.3. tétel jelentősége abban áll, hogy segítségével a $d_{\min}$ definíció alapján történő kiszámításához szükséges $\frac{|C|(|C|-1)}{2}$ műveletet a $w_{\min}$ kiszámításához szükséges $|C|-1$ műveletre redukálhatjuk. ($|C|$-vel a C elemszámát jelöltük.)

A 2.1. és a 2.3. példák nemzérus kódszavaira tekintve látható, hogy a minimális súly 3, míg a 2.3. példa esetén a minimális súly 2.

Szindróma dekódolás

A **H** mátrix hasznosnak bizonyul dekódolás során.

2.10. definíció. *Az* $\mathbf{s} = \mathbf{e}\mathbf{H}^T$ *mennyiséget* ***szindróma****nak nevezzük.*

Legyen az adott kódszó **c**, a vett szó **v**. Az $\mathbf{e} = \mathbf{v} - \mathbf{c}$ vektort **hibavektor**nak nevezzük. Vegyük észre, hogy

$$\mathbf{H}\mathbf{v}^T = \mathbf{H}(\mathbf{c}+\mathbf{e})^T = \mathbf{H}\mathbf{c}^T + \mathbf{H}\mathbf{e}^T = \mathbf{H}\mathbf{e}^T,$$

vagyis $\mathbf{H}\mathbf{v}^T$ értéke csak a hibavektortól függ, az adott kódszótól nem. A szindróma tehát a hibavektor egy lineáris leképezése.

A dekódolás leggyakoribb módja a **szindróma dekódolás**. A fentiek alapján a dekódolás a következőképpen mehet végbe: a vett **v** szóból kiszámítjuk az $\mathbf{s}^T = \mathbf{H}\mathbf{v}^T = \mathbf{H}\mathbf{e}^T$ szindrómát, ennek alapján megbecsüljük a hibavektort, s ezt **v**-ből levonva megkapjuk a kódszóra vonatkozó becslésünket.

A szindrómának hibamintára történő leképezési módját táblázatba szokás foglalni, az ún. **standard elrendezési táblázat**ba.

Valamely **e** hibaminta által generált halmaz (szokásos nevén mellékosztály) az $\mathbf{e}+\mathbf{c}$, $\mathbf{c} \in C(n,k)$ vektorok halmaza. Adott mellékosztály elemeihez azonos szindróma tartozik. Az $\mathbf{e} = \mathbf{0}$ zérus hibavektorhoz tartozó mellékosztály a $C(n,k)$ kóddal azonos. Ha egy **e** hibaminta $\mathbf{e} = \mathbf{e}' + \mathbf{c}$ alakban írható fel, akkor a két hibaminta (**e** és $\mathbf{e}'$) azonos mellékosztályt generál. Azonos mellékosztályba tartozó hibaminták közül válasszuk ki a legkisebb súlyút, s azt mellékosztály-vezetőnek nevezzük. Ennek megfelelően a standard elrendezési táblázat az alábbi struktúrájú:

szindróma	mellékosztály-vezető			
$\mathbf{s}^{(0)}$	$\mathbf{e}^{(0)} = \mathbf{0}$	$\mathbf{c}^{(1)}$		$\mathbf{c}^{(2^k-1)}$
$\mathbf{s}^{(1)}$	$\mathbf{e}^{(1)}$	$\mathbf{c}^{(1)} + \mathbf{e}^{(1)}$		$\mathbf{c}^{(2^k-1)} + \mathbf{e}^{(1)}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\mathbf{s}^{(2^{n-k}-1)}$	$\mathbf{e}^{(2^{n-k}-1)}$	$\mathbf{c}^{(1)} + \mathbf{e}^{(2^{n-k}-1)}$		$\mathbf{c}^{(2^k-1)} + \mathbf{e}^{(2^{n-k}-1)}$

mellékosztály elemek

A $w(\mathbf{e}^{(i+1)}) \geq w(\mathbf{e}^{(i)})$, $\mathbf{e}^{(0)} = \mathbf{0}$, $i = 0,1,\ldots,2^{n-k}-2$ a szokásos sorrend. Könnyen látható, hogy a táblázat elemei különbözőek. Egy soron belül ez nyilvánvaló. Különböző sorokat tekintve tegyük fel, hogy $\mathbf{e}^{(i)} + \mathbf{c}^{(j)} = \mathbf{e}^{(k)} + \mathbf{c}^{(m)}$, ahol $i > k$. Mivel ebből $\mathbf{e}^{(i)} = \mathbf{e}^{(k)} + \mathbf{c}^{(m)} - \mathbf{c}^{(j)} = \mathbf{e}^{(k)} + \mathbf{c}^{(n)}$ következik, ahol $\mathbf{c}^{(j)}, \mathbf{c}^{(m)}, \mathbf{c}^{(n)} \in C(n,k)$, ezért $\mathbf{e}^{(i)}$-nek is az $\mathbf{e}^{(k)}$ mellékosztály-vezetőjű sorban kell lennie, ami ellentétes kiindulási feltételünkkel.

Az $\mathbf{e}^{(i)}$, $i = 1,2,\ldots,2^{n-k}-1$ mellékosztály-vezetőket **javítható hibamintá**knak nevezzük, ugyanis ha a $\mathbf{v}$ vett szó szindrómája $\mathbf{s}^{(i)}$, akkor a $\widehat{\mathbf{c}} = \mathbf{v} - \mathbf{e}^{(i)}$ kódszóra döntünk. A szindróma dekódolásnak ezt az — elsősorban elvi — módját **táblázatos dekódolás**nak nevezzük. (Megjegyezzük, hogy a fenti dekódolási módszer nembináris ábécé esetére történő kiterjesztésekor 2^{n-k} helyett q^{n-k} áll.)

A szindrómát használó táblázatos dekódoló tárja a vizsgált bináris esetben 2^{n-k} darab hibavektort tartalmaz, és a táblázat elemeit a szindróma segítségével címezzük. Következésképp a táblázatos módszer gyakorlatban addig használható, amíg gyorselérésű tárunk mérete lehetővé teszi a javítható hibaminták tárolását.

2.5. példa. Adjuk meg a javítható hibamintákat a

$$\mathbf{H} = \begin{pmatrix} 0\,0\,1\,1\,1 \\ 0\,1\,0\,1\,0 \\ 1\,0\,0\,1\,1 \end{pmatrix}$$

mátrixszal adott kód esetére.

A dekódolási táblázat a következő:

szindróma	javítható hibaminták
000	00000
001	10000
010	01000
011	00110
100	00100
101	00001
110	01100
111	00010

Tehát a standard elrendezés fenti táblázata alapján történő szindróma dekódolással az egyszeres hibák és a 00110, 01100 két hibát tartalmazó hibaminták javíthatók.

Illusztrációként egy klasszikusnak számító kódot mutatunk be, mely **bináris Hamming-kód** néven ismeretes. Konstrukciónkat az alábbi tételre alapozzuk:

2.4. tétel. *C lineáris kód kódtávolsága legalább d^* akkor és csak akkor, ha a paritásellenőrző mátrixa tetszőlegesen választott d^*-1 oszlopa lineárisan független.*

A 2.4. tétel alapján 1 hibát javító bináris kódot kapunk, ha **H** tetszőleges két oszlopa lineárisan független, azaz oszlopai különbözők. Mivel a különböző, nemzérus, $n-k$ hosszú bináris vektorok száma $2^{n-k}-1$, ezért ezen vektorokat használva a **H** mátrix különböző oszlopaiként, az

$$n = 2^{n-k} - 1$$

összefüggésre jutunk, ami azt is jelenti, hogy a kapott kód perfekt tulajdonságú. Ennek alapján bináris Hamming-kód paraméterei az alábbi számpárok ($d_{min} = 3$):

$n =$ 3	$k =$ 1
7	4
15	11
31	26
63	57
127	120

2.6. példa. A $(7,4)$ paraméterű Hamming-kód paritásmátrixa

$$\mathbf{H} = \begin{pmatrix} 1\,1\,0\,1\,1\,0\,0 \\ 1\,0\,1\,1\,0\,1\,0 \\ 0\,1\,1\,1\,0\,0\,1 \end{pmatrix}.$$

A generátormátrixa ebből könnyen kiszámítható a már szerepelt $\mathbf{A} = -\mathbf{B}^T$ összefüggés alapján:

$$\mathbf{G} = \begin{pmatrix} 1\,0\,0\,0\,1\,1\,0 \\ 0\,1\,0\,0\,1\,0\,1 \\ 0\,0\,1\,0\,0\,1\,1 \\ 0\,0\,0\,1\,1\,1\,1 \end{pmatrix}.$$

2.3. Véges test

Hatékony hibajavító kódok konstrukciójához szükséges, hogy a nembináris Q kódábécé struktúrált legyen, mely például úgy lehetséges, hogy műveleteket vezetünk be Q-n.

2.11. definíció. *Egy Q halmazt **test**nek nevezünk, ha értelmezve van tetszőleges két eleme között két művelet, amelyeket összeadásnak illetve szorzásnak nevezünk, $+$ illetve $*$ szimbólumokkal jelöljük, és Q rendelkezik a következő tulajdonságokkal:*

1. *Q az összeadásra nézve kommutatív csoport, azaz*
   a) *Minden $\alpha, \beta \in Q$ esetén $\alpha + \beta \in Q$, tehát Q az összeadásra nézve zárt.*
   b) *Minden $\alpha, \beta, \gamma \in Q$ esetén $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$ (asszociativitás).*

c) *Létezik egy 0-val jelölt eleme Q-nek úgy, hogy minden $\alpha \in Q$-re* $0+\alpha=\alpha+0=\alpha$. *0-t nullelemnek nevezzük.*

d) *Minden $\alpha \in Q$-hez létezik $\beta \in Q$ úgy, hogy* $\alpha+\beta=0$. *β-t az α additív inverzének nevezzük és $-\alpha$-val jelöljük.*

e) *Minden $\alpha,\beta \in Q$-re* $\alpha+\beta=\beta+\alpha$ *(kommutativitás).*

2. $Q\setminus\{0\}$ *a szorzásra nézve kommutatív csoport, azaz*

a) *Minden $\alpha,\beta \in Q\setminus\{0\}$ esetén* $\alpha\cdot\beta \in Q\setminus\{0\}$ *(zártság).*

b) *Minden $\alpha,\beta,\gamma \in Q\setminus\{0\}$ esetén* $(\alpha\cdot\beta)\cdot\gamma=\alpha\cdot(\beta\cdot\gamma)$ *(asszociativitás).*

c) *Létezik egy 1-gyel jelölt eleme $Q\setminus\{0\}$-nak úgy, hogy* $1\cdot\alpha=\alpha\cdot 1=\alpha$. *1-et egységelemnek nevezzük.*

d) *Minden $\alpha \in Q\setminus\{0\}$ esetén létezik $\beta \in Q\setminus\{0\}$ úgy, hogy* $\alpha\cdot\beta=\beta\cdot\alpha=1$. *$\beta$-t az α multiplikatív inverzének nevezzük, és α^{-1}-gyel jelöljük.*

e) *Minden $\alpha,\beta \in Q\setminus\{0\}$-ra* $\alpha\cdot\beta=\beta\cdot\alpha$ *(kommutativitás).*

3. *Minden $\alpha,\beta,\gamma \in Q$-re* $\alpha\cdot 0=0\cdot\alpha=0$ *és* $\alpha\cdot(\beta+\gamma)=(\alpha\cdot\beta)+(\alpha\cdot\gamma)$ *(disztributivitás).*

Egyszerű konvenciókkal egy Q testben definiálható a kivonás és az osztás a következő módon: $\alpha-\beta$ alatt az α-nak és a β additív inverzének összegét értjük, azaz $\alpha+(-\beta)$-t. α/β alatt az α-nak és a β multiplikatív inverzének a szorzatát értjük, azaz $\alpha\cdot\beta^{-1}$-et, amennyiben β nem 0.

Példák testre:

- Valós számok halmaza a valós összeadással és szorzással.
- Racionális számok halmaza a valós összeadással és szorzással.
- Komplex számok halmaza a komplex összeadással és szorzással.
- $\{0,1\}$ a bináris összeadással és szorzással.

Egy q elemszámú Q testet **véges test**nek nevezünk és GF(q)-val jelöljük.

Mielőtt a véges testek aritmetikáját tárgyalnánk, néhány a továbbiakban felhasznált fontos tulajdonságukat ismertetjük.

Egy GF(q) esetén q nem lehet bármilyen:

2.5. tétel. *Egy GF(q) esetén $q=p^m$ alakú, ahol p prímszám, tehát q vagy prímszám, vagy prímhatvány.*

2.1. lemma. *Minden* $0 \neq a \in GF(q)$*-ra*

$$a^{q-1}=1.$$

2.2. lemma. *Minden $0 \neq a \in GF(q)$-ra létezik egy legkisebb m természetes szám, amit az a **elem rendjé**nek nevezünk, melyre*

$$a^m = 1,$$

és az $a, a^2, \ldots, a^m$ elemek mind különbözők. m osztója $q-1$-nek.

2.12. definíció. *Egy $\alpha \in GF(q)$-t a $GF(q)$ **primitív elem**ének nevezünk, ha α rendje $q-1$.*

2.6. tétel. *Minden GF(q)-ban létezik primitív elem.*

Aritmetika GF(p)-ben

2.7. tétel. *A $G = \{0, 1, \ldots, p-1\}$ halmaz a **modulo** p **aritmetiká**val egy p prímszám esetén véges test, azaz a testműveletek*

$$a + b = a + b \mod p,$$

$$a \cdot b = a \cdot b \mod p,$$

ahol $+$ illetve $\cdot$ jelöli a valós összeadást illetve szorzást.

2.7. példa. GF(3)
A GF(3) testben a műveletek modulo 3 összeadás és szorzás. A kapcsolatos műveleti táblákat láthatjuk alább:

$+$	**0**	**1**	**2**
0	0	1	2
1	1	2	0
2	2	0	1

$*$	**0**	**1**	**2**
0	0	0	0
1	0	1	2
2	0	2	1

2.8. példa. GF(7)

elem ($\neq 0$)	hatványai	rendje
1	1	1
2	2, 4, 1	3
3	3, 2, 6, 4, 5, 1	6 (primitív elem)
4	4, 2, 1	3
5	5, 4, 6, 2, 3, 1	6 (primitív elem)
6	6, 1	2

A prímitív elem egyrészt igen fontos hatékony kódok konstrukciójakor, másrészt GF(q)-beli szorzások és osztások elvégzésekor. Ha α a GF(q) egy primitív eleme, akkor bevezethetjük egy $a \in \mathrm{GF}(q)$ testelem α alapú logaritmusát az

$$\alpha^{\log a} = a$$

egyenlet (egyértelmű) megoldásával, ahol $a \neq 0$. Ha a, b a GF(q) nem 0 elemei, akkor

$$a \cdot b = \alpha^{\log a} \cdot \alpha^{\log b} = \alpha^{\log a + \log b},$$

tehát egy α alapú logaritmustábla és egy inverzlogaritmus-tábla segítségével a szorzás (illetve az osztás) visszavezethető valós összeadásra (illetve kivonásra).

A következőkben nagyon hasznosnak bizonyulnak a GF(q) feletti polinomok, így többek között egy fontos kódcsalád (a ciklikus kódok) leírásában, illetve a prímhatvány méretű véges testek aritmetikája generálásakor fogjuk használni őket.

Véges test feletti polinomok

GF(q) feletti vektorok reprezentálására, és vektorok közötti szorzás kényelmes bevezetésére egy célszerű eszköz a polinomreprezentáció:

2.13. definíció. $a(x) = a_0 + a_1 x + \ldots + a_m x^m$ ***GF(q) feletti*** *m-edfokú* ***polinom****, ha*

$$a_i \in \mathrm{GF}(q), \qquad i = 0, \ldots, m, \quad a_m \neq 0,$$

$$x \in \mathrm{GF}(q).$$

A polinom m fokszámát $\deg a(x)$ *jelöli. (Az* $a(x) \equiv 0$ *polinom fokszáma definíció szerint legyen* $-\infty$*.)*

2.14. definíció. $a(x) = b(x)$*, ha* $a_i = b_i$ *minden i-re.*

Műveletek polinomok között:

1. Polinomok összeadása: $c(x) = a(x) + b(x)$ tagonként történik GF(q) feletti műveletekkel: $c_i = a_i + b_i$. Nyilvánvalóan

$$\deg c(x) \leq \max\{\deg a(x), \deg b(x)\}.$$

2. Polinomok szorzása: $c(x) = a(x)b(x)$ minden tagot minden taggal szorzunk, majd az azonos fokú tagokat csoportosítjuk (az összeadások és szorzások GF(q) felettiek):

$$c_i = \sum_{j=0}^{\min\{i, \deg a(x)\}} a_j \cdot b_{i-j}.$$

Nyilván

$$\deg c(x) = \deg a(x) + \deg b(x)$$

2.9. példa. Ha GF(2) felett $a(x) = 1+x$ és $b(x) = 1+x+x^3$, akkor $a(x)+b(x) = x^3$ és $a(x)b(x) = 1+x^2+x^3+x^4$.

2.8. tétel (Euklidészi osztás polinomokra). *Adott $a(x)$ és $d(x) \neq 0$ esetén egyértelműen létezik olyan $q(x)$, $r(x)$, hogy*

$$a(x) = q(x)d(x) + r(x),$$

és $\deg r(x) < \deg d(x)$.

2.15. definíció. *$r(x)$-et az $a(x)$-nek $d(x)$-re vonatkozó maradékának nevezzük. Jelölés: $r(x) = a(x) \mod d(x)$.*

2.16. definíció. *$d(x)$ osztja $a(x)$-et, ha $a(x) \mod d(x) = 0$. Ezt a továbbiakban $d(x) \mid a(x)$ formában fogjuk jelölni.*

2.17. definíció. *$b \in GF(q)$ gyöke az $a(x)$ polinomnak, ha $a(b) = 0$.*

2.9. tétel. *Ha c az $a(x)$ polinom gyöke, akkor az előáll*

$$a(x) = b(x)(x-c)$$

alakban.

2.10. tétel. *Egy k-adfokú polinomnak legfeljebb k gyöke lehet.*

Aritmetika GF(p^m)-ben

Lényeges különbség van a prím illetve prímhatvány méretű testek aritmetikája között. Prím méretű testben a modulo aritmetika megfelelt. Prímhatvány méret esetén sajnos a modulo aritmetika nem teljesíti a testaxiómákat, például egy 4 elemű halmazban $2 \cdot 2 \mod 4 = 0$, tehát két nem 0 elem szorzata 0 lenne, ami sérti a 2. a) axiómát. A GF(p^m) feletti aritmetika konstrukciója azért alapvető fontosságú, mert manapság a hibajavító kódokat tömegesen alkalmazzuk számítástechnikai környezetben, ahol a természetes ábécé a GF(2^8), vagyis a bájt.

A GF(p^m)-beli elemek legyenek a $0, 1, \ldots, p^m - 1$ számok, melyeknek m hosszú vektorokat feleltetünk meg, ahol a koordináták GF(p)-beliek. Ezt megfogalmazhatjuk például úgy is, hogy a $0, 1, \ldots, p^m - 1$ számokat p-s számrendszerben írjuk fel. Ezek után a GF(p^m)-beli aritmetikát m hosszú vektorok közötti műveletekkel definiáljuk. A két művelet közül az összeadás az egyszerűbb: két vektor összegén a koordinátánkénti GF(p)-beli összeget értjük, vagyis a koordinátánkénti mod p összeget. A szorzás egy kicsit bonyolultabb. A két m hosszú vektort legfeljebb $(m-1)$-edfokú polinom formájában reprezentáljuk, és összeszorozzuk. Az eredmény fokszáma meghaladhatja $(m-1)$-et, ezért itt egy speciális polinom szerinti maradékot képezünk. Ezt a speciális polinomot irreducíbilis polinomnak nevezzük, és ez a polinom ugyanolyan szerepet játszik, mint a prímszám a GF(p)-beli aritmetikában.

2.18. definíció. *A GF(p) feletti, nem nulladfokú $P(x)$ polinomot **irreducíbilis polinom**nak nevezzük, ha nem bontható fel két, nála alacsonyabb fokú GF(p) feletti polinom szorzatára, azaz nincs GF(p) feletti $a_1(x), a_2(x)$ polinom, melyekre*

$$P(x) = a_1(x) \cdot a_2(x)$$

és

$$0 < \deg(a_i(x)) < \deg(P(x)), \qquad i = 1,2.$$

Bizonyítás nélkül megjegyezzük, minden véges testben található tetszőleges fokszámú irreducíbilis polinom. Példát mutatunk viszont arra, hogy hogyan lehet GF(2) feletti irreducíbilis polinomokat generálni. A definícióból következik, hogy minden elsőfokú polinom (x és $x+1$) irreducíbilis. Ha találunk olyan másodfokú polinomot, mely különbözik az x^2, az $x(x+1)$ és az $(x+1)^2$ mindegyikétől, akkor találtunk irreducíbilis másodfokú polinomot. Egy ilyen van: x^2+x+1. Más testben és nagyobb fokszám esetén ennél hatékonyabb konstrukciókat érdemes használni, de bináris esetben így is találhatók irreducíbilis polinomok, amelyeket táblázatban foglalunk össze:

fokszám	irreducíbilis polinom
2	x^2+x+1
3	x^3+x+1
4	x^4+x+1
5	x^5+x^2+1
6	x^6+x+1
7	x^7+x^3+1
8	$x^8+x^4+x^3+x^2+1$
9	x^9+x^4+1

2.11. tétel. *Legyen p egy prím, m egy természetes szám, $P(x)$ egy GF(p) feletti m-edfokú irreducíbilis polinom és $Q = \{0,1,\ldots,p^m-1\}$. Egy $a \in Q$-nak és $b \in Q$-nak kölcsönösen egyértelműen feleltessünk meg GF(p) feletti, legfeljebb $(m-1)$-edfokú $a(x)$ és $b(x)$ polinomot. $a+b$ definíció szerint az a $c \in Q$, melynek megfelelő $c(x)$ polinomra*

$$c(x) = a(x) + b(x).$$

$a \cdot b$ az a $d \in Q$, melynek megfelelő $d(x)$ polinomra

$$d(x) = \{a(x) \cdot b(x)\} \mod P(x).$$

Ezzel az aritmetikával Q egy GF(p^m).

2.10. példa. Készítsük el a GF(2^2)-beli aritmetikát! Tudjuk, hogy a $P(x) = x^2 + x + 1$ egy másodfokú irreducíbilis polinom. A kölcsönösen egyértelmű megfelelte-

téseket egy táblázatban foglaljuk össze:

testelemek	$m=2$ hosszú vektorok	polinomok
0	00	0
1	01	1
2	10	x
3	11	$x+1$

Az összeadást egyszerűen a 2 hosszú vektorok koordinátánkénti bináris összegével kapjuk. Nézzünk a szorzásra példát! $2\cdot 3$-at úgy számoljuk ki, hogy a 2-nek és a 3-nak megfelelő polinomot összeszorozzuk, és vesszük a $P(x)$ szerinti maradékot:

$$x(x+1)=1 \pmod{x^2+x+1},$$

amely megfelel az 1 testelemnek. Az összeadó és a szorzó tábla ennek megfelelően a bináris vektorokra:

$+$	**00**	**01**	**10**	**11**
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00

$\cdot$	**00**	**01**	**10**	**11**
00	00	00	00	00
01	00	01	10	11
10	00	10	11	01
11	00	11	01	10

majd testelemekre

$+$	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

$\cdot$	**0**	**1**	**2**	**3**
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

2.4. Nembináris lineáris kód

Ebben a szakaszban kódok egy fontos csoportjával ismerkedünk meg, melyek a 2.2. szakaszban megismert bináris lineáris kódok kiterjesztései nembináris esetre.

A továbbiakban a kódjainkban szereplő kódszavakat alkotó szimbólumokat vegyük GF(q)-ból, a lehetséges szimbólumok tehát a $0,1,2,\ldots,q-1$ számoknak feleltethetők meg.

2.19. definíció. *Egy C kód **lineáris**, ha a C halmaz lineáris tér GF(q) fölött, azaz ha minden $\mathbf{c},\mathbf{c}'\in C$-re*

$$\mathbf{c}+\mathbf{c}'\in C$$

illetve $\beta\in$ GF(q) esetén

$$\beta\mathbf{c}\in C.$$

A 2.2. szakaszhoz hasonló módon belátható, hogy tetszőleges C lineáris kódhoz létezik egy k lineárisan független sorból és n oszlopból álló $\mathbf{G}$ mátrix, melyre

$$\mathbf{c} = \mathbf{uG}, \tag{2.7}$$

ahol a k hosszú $\mathbf{u}$ üzenethez a $\mathbf{c}$ kódszó tartozik, és a $\mathbf{G}$ mátrixot a $\mathbf{C}$ kód **generátormátrix**ának nevezzük.

A bináris esethez hasonlóan a C lineáris kódhoz egy $n-k$ sorból és n oszlopból álló $\mathbf{H}$ mátrixot **paritásmátrix**nak nevezünk, amennyiben

$$\mathbf{Hc}^T = 0$$

akkor és csak akkor teljesül, ha $\mathbf{c} \in C$.

A bináris eset másolataként kaphatjuk, hogy

2.12. tétel. *Minden C lineáris kódnak van paritásellenőrző mátrixa.*

Példaként bemutatjuk a **nembináris Hamming-kód**ot. Ismét 1 hibát javító kódot akarunk konstruálni. A bináris esetben a hiba javításához elég volt ismerni a hiba helyét, amihez elégséges volt, ha a $\mathbf{H}$ paritásmátrix minden oszlopa különböző. Nembináris esetben nemcsak a hiba helyét, hanem a hiba értékét is meg kell állapítani, ezért a $\mathbf{H}$ mátrix oszlopait úgy választjuk, hogy azok nem 0-k, mind különbözők legyenek, és az első nem 0 elem minden oszlopban 1 értékű legyen. Ekkor, ha egy hiba esetén az az i-edik helyen fordul elő és értéke e_i, akkor a szindróma $\mathbf{s} = e_i\mathbf{a}_i$, ahol $\mathbf{a}_i^T$ a $\mathbf{H}$ i-edik oszlopa. Tehát a hiba értéke, e_i éppen a szindróma első nem 0 értéke, míg $\mathbf{a}_i = \frac{\mathbf{s}}{e_i}$, amiből az i visszakereshető.

Ha $\mathbf{H}$ tartalmazza az összes lehetséges, a fenti módon megengedett oszlopvektort, akkor

$$n = \frac{q^{n-k}-1}{q-1},$$

azaz

$$1 + n(q-1) = q^{n-k},$$

másrészt a Hamming-korlát miatt

$$1 + n(q-1) \leq q^{n-k},$$

tehát

2.13. tétel. *A maximális hosszúságú nembináris Hamming-kód perfekt kód.*

A nembináris Hamming-kódok közül különösen érdekes az az eset, amikor a kód szisztematikus és a paritásszegmens hossza 2, azaz $n-k=2$. Legyen α a GF(q) egy nem 0 eleme, melynek rendje $m \geq 2$. Válasszunk $n \leq (m+2)$-t és $k = (n-2)$-t. Ekkor a paritásmátrix:

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & 0 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-3} & 0 & 1 \end{pmatrix}.$$

Ez egy $(n, n-2)$ paraméterű nembináris Hamming-kód paritásmátrixa.

A 2.2. tétel alkalmazásával nyerjük a kód generátormátrixát:

$$\mathbf{G} = \begin{pmatrix} 1\,0\,0\,0\cdots 0\,-1 & -1 \\ 0\,1\,0\,0\cdots 0\,-1 & -\alpha \\ 0\,0\,1\,0\cdots 0\,-1 & -\alpha^2 \\ \vdots \qquad\quad \ddots \qquad\quad & \vdots \\ 0\,0\,0\,0\cdots 1\,-1 & -\alpha^{n-3} \end{pmatrix}.$$

Mivel ez a kód 1 hibát tud javítani, ezért $d_{\min} \geq 3$, de a Singleton-korlát miatt $d_{\min} \leq n-k+1 = 3$, ezért

2.14. tétel. *Az $(n, n-2)$ paraméterű nembináris Hamming-kód MDS kód.*

2.11. példa. Írjuk fel a GF(7) feletti, $(8,6)$ paraméterű Hamming-kód generátormátrixát és paritásmátrixát! GF(7)-ben a 3 primitív elem (lásd a 2.8. példát), tehát

$$\mathbf{H} = \begin{pmatrix} 1\,1\,1\,1\,1\,1\,1\,0 \\ 1\,3\,2\,6\,4\,5\,0\,1 \end{pmatrix}$$

$$\mathbf{G} = \begin{pmatrix} 1\,0\,0\,0\,0\,0\,-1\,-1 \\ 0\,1\,0\,0\,0\,0\,-1\,-3 \\ 0\,0\,1\,0\,0\,0\,-1\,-2 \\ 0\,0\,0\,1\,0\,0\,-1\,-6 \\ 0\,0\,0\,0\,1\,0\,-1\,-4 \\ 0\,0\,0\,0\,0\,1\,-1\,-5 \end{pmatrix} = \begin{pmatrix} 1\,0\,0\,0\,0\,0\,6\,6 \\ 0\,1\,0\,0\,0\,0\,6\,4 \\ 0\,0\,1\,0\,0\,0\,6\,5 \\ 0\,0\,0\,1\,0\,0\,6\,1 \\ 0\,0\,0\,0\,1\,0\,6\,3 \\ 0\,0\,0\,0\,0\,1\,6\,2 \end{pmatrix}.$$

Reed–Solomon-kód

Ebben a szakaszban a lineáris kódok egyik leggyakrabban használt osztályával, a **Reed–Solomon-kód**okkal, azok különböző konstrukcióival ismerkedünk meg.

2.1. konstrukció. *Legyenek $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ a GF(q) különböző elemei ($n \leq q$), és $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ ($u_i \in$ GF(q)) a k hosszúságú üzenetszegmens, amelyhez az*

$$u(x) = u_0 + u_1 x + \ldots + u_{k-1} x^{k-1}$$

üzenetpolinomot rendeljük. Ekkor a Reed–Solomon-kódnak az **u** *üzenethez tartozó n hosszú* **c** *kódszavát a következő módon állítjuk elő:*

$$\begin{aligned} c_0 &= u(\alpha_0) \\ c_1 &= u(\alpha_1) \\ c_2 &= u(\alpha_2) \\ &\vdots \\ c_{n-1} &= u(\alpha_{n-1}). \end{aligned}$$

Egyszerűen belátható, hogy a Reed–Solomon-kód lineáris, és a generátormátrixa

$$\mathbf{G} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{n-1} \\ \vdots & & & \ddots & \vdots \\ \alpha_0^{k-1} & \alpha_1^{k-1} & \alpha_2^{k-1} & \cdots & \alpha_{n-1}^{k-1} \end{pmatrix}.$$

2.15. tétel. *Az (n,k) paraméterű Reed–Solomon-kód kódtávolsága*

$$d_{min} = n-k+1,$$

vagyis a Reed–Solomon-kód maximális távolságú.

BIZONYÍTÁS:

$$\begin{aligned} w(\mathbf{c}) &= |\{\mathbf{c} \text{ nem 0 koordinátái}\}| = \\ &= n - |\{\mathbf{c} \text{ 0 koordinátái}\}| \geq \\ &\geq n - |\{u(x) \text{ gyökei}\}| \geq \\ &\geq n-(k-1), \end{aligned}$$

tehát

$$w_{\min} \geq n-k+1.$$

Ugyanakkor a 2.1. tétel és a 2.3. tétel miatt

$$n-k+1 \geq d_{\min} = w_{\min},$$

következésképp az állítást bebizonyítottuk. ■

Az (n,k) paraméterű Reed–Solomon-kód tehát $n-k$ hibát tud jelezni, $\left\lfloor \frac{n-k}{2} \right\rfloor$ egyszerű hibát javítani és $n-k$ törléses hibát javítani. Ez utóbbi azt is jelenti, hogy az $\mathbf{u}$ ismeretlenre vonatkozó

$$\mathbf{uG} = \mathbf{c}$$

n darab egyenletből bármelyik $n-k$ egyenlet elhagyásával egy egyértelműen megoldható egyenletrendszer marad, tehát a $\mathbf{G}$ mátrix minden $k \times k$-s négyzetes részmátrixa invertálható.

2.2. konstrukció. *Legyen α a GF(q) egy nem 0 eleme, melynek rendje m, $m \geq n$ és a 2.1. konstrukcióban legyen $\alpha_0 = 1, \alpha_1 = \alpha, \ldots, \alpha_{n-1} = \alpha^{n-1}$. Ekkor a generátormátrix:*

$$\mathbf{G} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-1)} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2(k-1)} & \cdots & \alpha^{(k-1)(n-1)} \end{pmatrix}$$

2.5. Ciklikus kódok

2.20. definíció. *Egy*

$$\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$$

vektor ciklikus eltoltja az

$$S\mathbf{c} = (c_{n-1}, c_0, \ldots, c_{n-2}).$$

*S-et a **ciklikus eltolás** operátorának nevezzük.*

2.21. definíció. *A C kódot **ciklikus**nak nevezzük, ha bármely kódszó ciklikus eltoltja is kódszó.*

2.12. példa. Legyen C a

000
101
110
011
111

vektorok halmaza. Egyszerűen belátható, hogy C ciklikus. Megjegyezzük, hogy a ciklikusságból nem következik a linearitás, például a 2. és az 5. kódszó összege 010, amely nem eleme a kódnak.

A ciklikus eltolás operátorát kényelmesebben tudjuk kezelni, ha a kódszavakat mint vektorokat a már megszokott polinomos formában reprezentáljuk.

2.22. definíció. *Rendeljünk polinomot az egyes kódszavakhoz a következő módon:*

$$\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \;\mapsto\; c(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1},$$

ekkor a $\mathbf{c}$ *kódszónak megfeleltetett* $c(x)$ *polinomot **kódszópolinom**nak vagy röviden kódpolinomnak nevezzük. A kódszópolinomok halmazát* $C(x)$*-szel jelöljük.*

2.3. lemma. *Legyen* $c'(x)$ *a* $\mathbf{c}$ *kódszó* $S\mathbf{c}$ *eltoltjához rendelt kódszópolinom, ekkor*

$$c'(x) = [xc(x)] \mod (x^n - 1).$$

2.16. tétel. *Minden* (n,k) *paraméterű, ciklikus, lineáris* C *kódban a nem azonosan nulla kódszópolinomok között egyértelműen létezik egy minimális fokszámú* $g(x)$ *polinom, amelynek legmagasabb fokú tagja együtthatója* 1. $g(x)$ *fokszáma* $n-k$*, és egy* $\mathbf{c} \in C$ *akkor és csak akkor, ha* $g(x) \mid c(x)$*, azaz létezik egy* $u(x)$ *polinom úgy, hogy* $c(x) = g(x)u(x)$*.*

2.23. definíció. $g(x)$*-et a kód **generátorpolinom**jának nevezzük.*

2.17. tétel. *Minden ciklikus, lineáris kód $g(x)$ generátorpolinomjára*

$$g(x) \mid x^n - 1.$$

Másrészről, ha egy $g(x)$ főpolinomra $g(x) \mid x^n - 1$, akkor létezik egy lineáris ciklikus kód, melynek $g(x)$ a generátorpolinomja.

A paritásmátrixnak is van polinomos megfelelője:

2.24. definíció. *Egy $g(x)$ generátorpolinomú lineáris, ciklikus kód esetén a*

$$h(x) = \frac{x^n - 1}{g(x)}$$

*polinomot **paritásellenőrző polinom**nak nevezzük.*

2.18. tétel. *Egy lineáris, ciklikus kódra $c(x)$ akkor és csak akkor kódszópolinom, ha*

$$c(x)h(x) = 0 \mod (x^n - 1)$$

és

$$\deg(c(x)) \leq n - 1.$$

Ciklikus kódok szisztematikus generálása

A ciklikus kódok előnyös tulajdonságai egyrészt a generálási lehetőségek sokféleségében, másrészt egyszerű dekódolási eljárásokban jelentkeznek. Egy lineáris ciklikus kódot lehet például a generátorpolinom és az üzenetpolinom szorzásával generálni. Ez a módszer megfogalmazható egy olyan **G** generátormátrix segítségével is, amelyhez legegyszerűbben úgy juthatunk el, ha **G** sorai a $g(x)$-nek megfelelő vektor eltoltjai:

$$\mathbf{G} = \begin{pmatrix} g_0 & g_1 & g_2 & \cdots & g_{n-k-1} & 1 & 0 & \cdots & & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k-2} & g_{n-k-1} & 1 & \cdots & & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & g_0 & g_1 & g_2 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & g_0 & g_1 & \cdots & g_{n-k-1} & 1 \end{pmatrix},$$

kihasználva, hogy $g(x)$ főpolinom, így $g_{n-k} = 1$.

Egy másik generálási módszer kapcsán azt is megmutatjuk, hogy

2.19. tétel. *Minden lineáris ciklikus kód generálható szisztematikusan.*

BIZONYÍTÁS: Vegyük észre, hogy $g_0 \neq 0$, mert ha 0 lenne, akkor a **g**-t balra eltolva **g**-nél 1-gyel kisebb fokszámú kódszópolinomot kapnánk, ami lehetetlen. $g_0 \neq 0$ miatt viszont a Gauss-eliminációt balról jobbra végrehajtva szisztematikus generátormátrixot kapunk. ■

A **szisztematikus generálás** egy praktikus módszere a következő:
Legyen $u(x)$ egy legfeljebb $(k-1)$-edfokú üzenetpolinom és

$$c(x) = u(x)x^{n-k} - [u(x)x^{n-k}] \mod g(x),$$

akkor $c(x)$ kódszópolinom, mivel $c(x) = 0 \mod g(x)$. Ezen generálás szisztematikus: a $c(x)$-et definiáló egyenlőség jobb oldalának első tagja adja az üzenetszegmenst, míg a második tagja a paritásszegmenst.

2.13. példa. Tekintsük a GF(2) feletti $g(x) = 1+x+x^3$ polinomot! Mivel

$$x^7 - 1 = (1+x)(1+x+x^3)(1+x^2+x^3),$$

ezért $g(x)$ osztja (x^7-1)-et, tehát $g(x)$ egy $(7,4)$ paraméterű bináris, lineáris, ciklikus kód generátorpolinomja. Egy generátormátrixához jutunk a $g(x)$ eltoltjaival:

$$\mathbf{G} = \begin{pmatrix} 0\,0\,0\,1\,0\,1\,1 \\ 0\,0\,1\,0\,1\,1\,0 \\ 0\,1\,0\,1\,1\,0\,0 \\ 1\,0\,1\,1\,0\,0\,0 \end{pmatrix}.$$

A második módszer szerinti szisztematikus generátormátrixhoz úgy jutunk el, ha kiszámítjuk az $[x^{3+i}] \mod (1+x+x^3)$ maradékokat $i = 0,1,2,3$-ra, melyek

$$\begin{aligned} x^3 &= 1+x \mod (1+x+x^3) \\ x^4 &= x(1+x+x^3) - x(1+x) = \\ &= x(1+x) = \\ &= x+x^2 \mod (1+x+x^3) \\ x^5 &= x^2(1+x+x^3) - x^2(1+x) = \\ &= x^2(1+x) = \\ &= 1+x+x^2 \mod (1+x+x^3) \\ x^6 &= x^3(1+x+x^3) - x^3(1+x) = \\ &= x^3 + x^4 = \\ &= 1+x+x+x^2 = \\ &= 1+x^2 \mod (1+x+x^3) \end{aligned}$$

Ezek alapján

$$\mathbf{G} = \begin{pmatrix} 1\,0\,0\,0\,1\,0\,1 \\ 0\,1\,0\,0\,1\,1\,1 \\ 0\,0\,1\,0\,1\,1\,0 \\ 0\,0\,0\,1\,0\,1\,1 \end{pmatrix},$$

amely a már jól ismert $(7,4)$ paraméterű Hamming-kód szisztematikus generátormátrixa.

Ciklikus Reed–Solomon-kód

A Reed–Solomon-kódok legfontosabb gyakorlati előállítási módja az alábbi tételen alapszik:

2.20. tétel. *A Reed–Solomon-kódok esetén legyen az n kódszóhossz egyenlő az ott szereplő α elem m rendjével. Ekkor a kód ciklikus, és generátorpolinomja*

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha^i),$$

továbbá paritásellenőrző polinomja

$$h(x) = \prod_{i=n-k+1}^{n} (x - \alpha^i),$$

tehát a nem rövidített Reed–Solomon-kód ciklikus.

2.21. tétel. *A 2.20. tétel Reed–Solomon-kódjának paritásellenőrző mátrixa*

$$\mathbf{H} = \begin{pmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-1)} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{2(n-k)} & \cdots & \alpha^{(n-1)(n-k)} \end{pmatrix}$$

2.6. Dekódolási algoritmus

A 2.2. szakaszban lineáris kódok táblázatos dekódolását mutattuk be. A táblázat sorainak száma q^{n-k}, azaz a módszer gyakorlati alkalmazhatóságát a paritásszegmens hossza és a kódábécé mérete határozza meg. Ha a táblázat mérete meghaladja a tárkapacitásunkat, nem tudjuk előre tárolni az egyes szindróma értékekhez tartozó javítható hibamintákat, ehelyett a hibamintát a vett szó szindrómájának meghatározása után mindig újra kiszámítjuk. Az általános hibajavító algoritmusok bemutatása meghaladja ezen jegyzet kereteit, ugyanakkor módunk van arra, hogy egyszerűbb esetekben bemutassuk az általános algoritmusok alapvető lépéseit. Az alábbiakban Reed–Solomon-kód esetén egy hiba javításának, illetve a $t \geq 1$ törlés javításának algoritmusát mutatjuk be.

Egy hibát javító ciklikus Reed–Solomon-kód generátorpolinomja

$$g(x) = (x - \alpha)(x - \alpha^2),$$

következésképp tetszőleges $c(x)$ kódszó esetén $c(\alpha) = c(\alpha^2) = 0$. A vett szó $\mathbf{v} = \mathbf{c} + \mathbf{e}$ felbontása polinomos alakban $v(x) = c(x) + e(x)$. Az egy hiba javítás esetét tekintve $e(x) = ex^i$, $e \in GF(q)$, $i \in \{0, 1, \ldots, n-1\}$, ahol e a hiba értéke, és i a hiba helye. A 2.21. tételbeli paritásellenőrző-mátrix alapján láthatjuk, hogy $s = (s_1, s_2)$, $s_i \in GF(q)$, $i = 1, 2$ szindróma vektor komponenseit ekvivalensen kiszámíthatjuk a következő módon: $s_1 = v(\alpha)$, $s_2 = v(\alpha^2)$. Innen

$$s_1 = v(\alpha) = c(\alpha) + e(\alpha) = 0 + e(\alpha) = e(\alpha) = e \cdot \alpha^i$$
$$s_2 = v(\alpha^2) = c(\alpha^2) + e(\alpha^2) = 0 + e(\alpha^2) = e(\alpha^2) = e \cdot \alpha^{2i},$$

ahonnan

$$e \cdot \alpha^i = s_1$$
$$e \cdot \alpha^{2i} = s_2,$$

egyenletrendszer adódik e, i ismeretlenekben. Így $s_2/s_1 = \alpha^i$, amiből az i hibahelyet kapjuk, majd ezután meghatározzuk az $e = s_1\alpha^{-i}$ hibaértéket.

Törléses hiba esetén ismerjük a hibahelyeket, de továbbra sem ismerjük a hiba értékeit. A dekódolási algoritmus alapja ismét a szindrómákra vonatkozó lineáris egyenletrendszer

$$\begin{aligned} s_1 &= v(\alpha) = e_1 \cdot \alpha^{i_1} + \cdots + e_t \cdot \alpha^{i_t} \\ s_2 &= v(\alpha^2) = e_1 \cdot \alpha^{2i_1} + \cdots + e_t \cdot \alpha^{2i_t} \\ &\vdots \\ s_t &= v(\alpha^t) = e_1 \cdot \alpha^{ti_1} + \cdots + e_t \cdot \alpha^{ti_t} \end{aligned}$$

ahol $e(x) = e_1 \cdot x^{i_1} + \cdots + e_t \cdot x^{i_t}$, továbbá $t \leq n - k$.

2.14. példa. Egy GF(11) feletti $g(x) = (x-2)(x-4)$ generátorpolinomú Reed–Solomon-kód dekóderéhez érkezett vett szóból 2 karakter törlődött:

0.	1.	2.	3.	4.	5.	6.	7.	8.	9.
(8	2	0	0	2	0	0	?	0	?)

Határozzuk meg a törlődött karaktereket!

$u, v \in GF(11)$ ismeretleneket bevezetve az ismeretlen értékekre, a kódszó polinom alakban a következő:

$$c(x) = ux^9 + vx^7 + 2x^4 + 2x + 8.$$

A $c(2) = 0$, $c(4) = 0$ egyenletek alapján:

$$u \cdot 2^9 + v \cdot 2^7 + 2 \cdot 2^4 + 2 \cdot 2 + 8 = 0,$$
$$u \cdot 4^9 + v \cdot 4^7 + 2 \cdot 4^4 + 2 \cdot 4 + 8 = 0,$$

$GF(11)$ feletti egyenletrendszer adódik, amelynek megoldása $u = 0$, $v = 0$.

2.7. Kódkombinációk

A standard kódkonstrukciók során kapott kódok paraméterei nem mindig illeszkednek közvetlenül az adott alkalmazásban megkövetelt értékekhez. Hatékony, ugyanakkor egyszerű módszerek léteznek arra, hogy változtassuk a kódszóhossz, üzenethossz, kódtávolság paraméterek értékét az eredeti konstrukcióhoz képest. Az alábbiakban ezen módszereket tekintjük át röviden.

Kódátfűzés és a csomós hibák javítása

Adott $C(n,k)$ kód m-szeres **átfűzés**ével egy $C^m = C(mn,mk)$ kódot kapunk, olyan módon, hogy a C kód $\mathbf{c}^{(i)}$, $i = 1,\ldots,m$ m darab kódszavát egy $m \times n$ dimenziós mátrixba rendezzük soronként, s a C^m átfűzéses kód $\mathbf{c}$ kódszavát ezen mátrix oszlapainak sorrendben való kiolvasásával képezzük. Azaz a kódszavakat (komponens szavakat) fésű módon egymásba toljuk:

$$\mathbf{c} = \left(c_0^{(1)}, c_0^{(2)}, \ldots, c_0^{(m)}, c_1^{(1)}, c_1^{(2)}, \ldots, c_1^{(m)}, \ldots, c_{n-1}^{(1)}, c_{n-1}^{(2)}, \ldots, c_{n-1}^{(m)}\right) \tag{2.8}$$

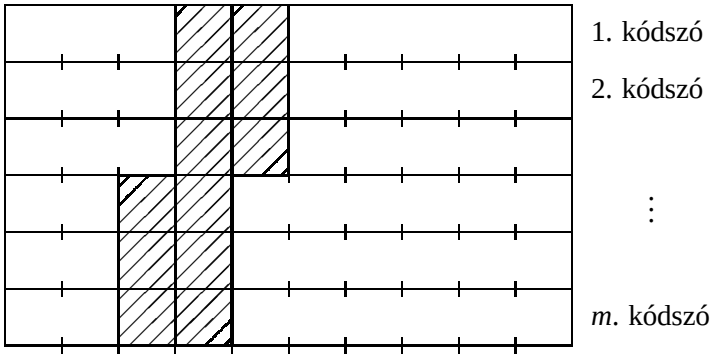
Lineáris kódot átfűzve nyilván lineáris kódot kapunk. Az is könnyen látható, hogy ha d a C kód kódtávolsága, akkor a C^m átfűzéses kód távolsága is d marad. Lineáris C kódot tekintve legyen $\mathbf{c}^{(i)}$, $i = 1,\ldots,m$ sorozat egyik kódszavának súlya d, míg a többi kódszó legyen a zérus kódszó. Általános esetben tekintsük a C kódbeli kódszavak $\mathbf{c}^{(1,i)}$, $i = 1,\ldots,m$, $\mathbf{c}^{(2,i)}$, $i = 1,\ldots,m$ két sorozatát, ahol $\mathbf{c}^{(1,1)}$ és $\mathbf{c}^{(2,1)}$ távolsága d, míg $\mathbf{c}^{(1,i)} = \mathbf{c}^{(2,i)}$, $i = 2,\ldots,m$. (A később bemutatott CD példájában $m = 2, n = 28, k = 24$.)

Ciklikus kódot átfűzve ciklikus kódot kapunk. Legyen S az egyszeri ciklikus jobbra léptetés operátora. Könnyen ellenőrizhető, hogy a (2.8) szerinti $\mathbf{c}$ kódszó $S\mathbf{c}$ ciklikus eltoltja az $S\mathbf{c}^{(m)}, \mathbf{c}^{(1)}, \mathbf{c}^{(2)}, \ldots, \mathbf{c}^{(m-1)}$ sorozat átfűzésének felel meg, s mivel $S\mathbf{c}^{(m)} \in C$, ezért $S\mathbf{c} \in C^m$ is fennáll.

Mint láttuk, a kódtávolság nem változik átfűzés során, ami azt jelenti, hogy a C^m átfűzéses kód szokásos képességei (véletlen hibák javítása, törlésjavítás, detekciós képesség) romlanak az átfűzéssel, hiszen ezen képességek m-szeres kódszóhosszon érvényesek. Ha valaki itt arra gondolna, hogy például az egyes komponensszavak javítóképessége nem változott, s így a teljes javító képesség a komponensek m-szeresének tűnik, az ott hibázik, hogy t javítóképesség azt jelenti, hogy tetszőleges t pozícióban eshet hiba, nem pedig azt, hogy az a komponens szavaknak megfelelően kerül „szétosztásra”. Ezen a ponton felmerül a természetes kérdés: egyáltalán mire jó akkor az átfűzés? A válasz: hibacsomók javítására.

A hibavektor egy l hosszúságú szegmense **hibacsomó** l hosszal, ha a szegmens első és utolsó karaktere nem zérus. Egy kód l hosszúságú hibacsomót javító, ha minden legfeljebb l hosszúságú hibacsomó javítható.

2.22. tétel. *A C^m átfűzéses kód $m \cdot t$ hosszúságú hibacsomót javító, ahol t a C kód javítóképessége.*



2.2. ábra. Kódátfűzés $t = 2$ esetén

BIZONYÍTÁS: A (2.8) szerinti **c** kódszóban egy legfeljebb $m \cdot t$ hosszúságú hibacsomónak megfelelő hibázás a komponens szavakban legfeljebb t számú hibát okozhat, amit azok javítani képesek. (A $t = 2$ esetet szemlélteti a 2.2. ábra.) ■

Egy tetszőleges lineáris $C(n,k)$ kód n,k paramétere alapján a kód l **hibacsomójavító képesség**ére az alábbi egyszerű korlát adható:

2.23. tétel. *Egy $C(n,k)$ lineáris kód l hibacsomójavító képességére fennáll, hogy $l \leq \lfloor \frac{n-k}{2} \rfloor$.*

A tételbeli korlát Reiger-korlát néven ismert. Azokat a hibacsomó javító kódokat, amelyre $l = \lfloor \frac{n-k}{2} \rfloor$ fennáll, **Reiger-optimális**nak hívjuk.

MEGJEGYZÉS: Egy MDS tulajdonságú lineáris kód Reiger-optimális.

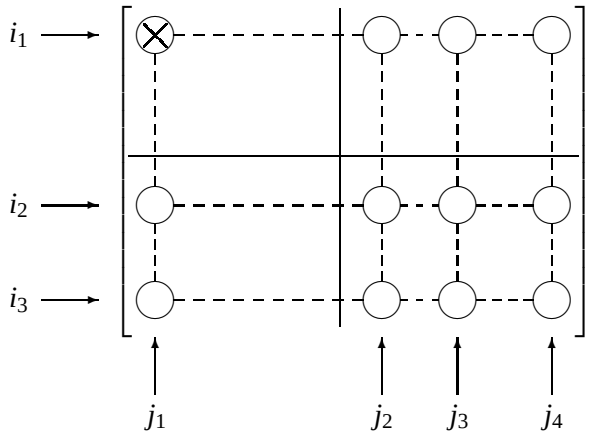
Szorzatkód

Egy $C_1(n_1,k_1,d_1)$ és egy $C_2(n_2,k_2,d_2)$ lineáris kód (komponenskódok) felhasználásával $C_1 \times C_2(n_1 \cdot n_2, k_1 \cdot k_2, d_1 \cdot d_2)$ **szorzatkód**ot készíthetünk, amelynek kódszavai $n_1 \times n_2$ dimenziós mátrixok, ahol a mátrix sorai C_1 kódbeli, oszlopai C_2 kódbeli kódszavak. Szisztematikus komponenskódok esetén a szorzatkódbeli mátrix-kódszó bal felső $k_1 \times k_2$ dimenziós minorja tartalmazza az üzenetet. A mátrix-kódszavakat soronként kiolvasva kapjuk a szorzatkód — soros — kódszavát. A kapott $C_1 \times C_2(n_1 \cdot n_2, k_1 \cdot k_2)$ kód lineáris.

A mátrix-kódszó képzése a következőképp történik. Az első k_1 oszlopot a $C_2(n_2,k_2)$ kód alapján szisztematikus kódolással kapjuk, kiegészítve a k_2 hosszú üzenetszegmenst $n_2 - k_2$ hosszú paritásszegmenssel (2.3. ábra). Az első k_2 sort a $C_1(n_1,k_1)$ kód alapján szisztematikus kódolással kapjuk, kiegészítve a k_1 hosszú üzenetszegmenst $n_1 - k_1$ hosszú paritásszegmenssel. A mátrix jobb alsó sarkába kerül a paritások paritása, amit — mint azt hamarosan belátjuk — képezhetjük akár az első k_1 oszlop, akár az első k_2 sor paritásai alapján szisztematikus kódolással a C_2 illetve C_1 kódbeli szavakkal. A fenti módon képezett szorzatkódot —

$$\left[\begin{array}{cccc|ccc}
c_0^{(0)} & c_1^{(0)} & \cdots & c_{k_1-1}^{(0)} & c_{k_1}^{(0)} & \cdots & c_{n_1-1}^{(0)} \\
c_0^{(1)} & c_1^{(1)} & \cdots & c_{k_1-1}^{(1)} & c_{k_1}^{(1)} & \cdots & c_{n_1-1}^{(1)} \\
\vdots & & & & & & \vdots \\
c_0^{(k_2-1)} & c_1^{(k_2-1)} & \cdots & c_{k_1-1}^{(k_2-1)} & c_{k_1}^{(k_2-1)} & \cdots & c_{n_1-1}^{(k_2-1)} \\
\hline
\vdots & & & & & & \vdots \\
c_0^{(n_2-1)} & c_1^{(n_2-1)} & \cdots & c_{k_1-1}^{(n_2-1)} & c_{k_1}^{(n_2-1)} & \cdots & c_{n_1-1}^{(n_2-1)}
\end{array}\right]$$

2.3. ábra. A szorzat-kódszó képzése

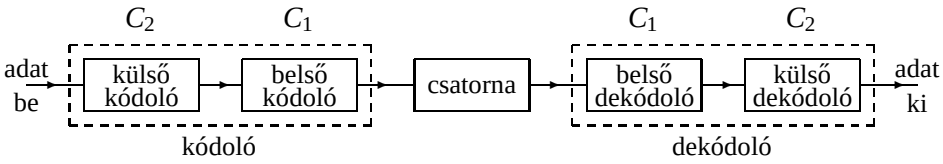


2.4. ábra. A paritások paritásainak képzése

amelynek sorai illetve oszlopai az alapkódok kódszavai — kanonikus elrendezésűnek nevezzük.

A paritások paritásai képzésével kapcsolatos alábbi gondolatmenetünket illusztrálja a 2.4. ábra.

Képezzük azt a $C_1 \times C_2$ kódbeli kódszót, amelynek üzenetmátrixa csak az (i_1, j_1) koordinátájú helyen tartalmaz nullától különböző elemet. Ehhez az üzenethez képezzük a C_2 illetve C_1 kódolás szerint a $C_1 \times C_2$ kódbeli kódszó i_1-edik sorát és j_1-edik oszlopát. A kétdimenziós paritásszegmens jobb felső illetve bal alsó részmátrixa az i_1-edik sor illetve a j_1-edik oszlop kivételével csak 0 elemeket tartalmaz. Innen már egyszerűen látszik, hogy a jobb alsó részmátrixot megkaphatjuk, akár az (i_2, j_1) illetve (i_3, j_1) elemekből C_2 kód szerinti, akár $(i_1, j_2), (i_1, j_3)$ illetve (i_1, j_4) elemekből C_1 kód szerinti kódolással. S miután a $C_1 \times C_2$ kód lineáris, ezért tetszőleges üzenetmátrixú szorzatkód kódszót a 2.4. ábrán is illusztrált elem-kódszavakból koordinátánként vett összeadással képezhetjük.



2.5. ábra. Kaszkád kódoló

Annak igazolása, hogy a szorzatkód kódtávolsága a komponenskódok távolságainak szorzata, a minimális nemzérus súlyú, azaz $d_1 \cdot d_2$ súlyú kódszó előállításával történhet. Válasszunk ehhez egy-egy minimális súlyú kódszót a C_1 illetve C_2 kódból, amelyeket jelöljön $\mathbf{c}'$ illetve $\mathbf{c}''$, ekkor egy minimális súlyú mátrix-kódszó az i-edik sorában a $\mathbf{c}''$ kódszót tartalmazza, ha $\mathbf{c}'$ i-edik komponense 1, egyébként a csupa zérus kódszó kerül a sorba. Az, hogy a kapott kódszó minimális súlyú, onnan látható, hogy ha nem minimális súlyú $\mathbf{c}''$ kódszót helyeznénk el valamelyik sorba, akkor több nemzérus oszlopot kellene elhelyezni a mátrixban a C_1 kódszavai közül és viszont.

2.15. példa. Az egyik legismertebb és egyben legegyszerűbb konstrukciójú hibajavító kód a **kétdimenziós paritáskód**. Ez egy $C \times C$ szorzatkód, ahol a C komponenskód $(n, n-1, 2)$ paraméterű egy paritásbittel rendelkező, egy hibát jelző bináris kód. A kapott szorzatkód kódtávolsága 4, azaz egyszerű paritásbites konstrukcióval 1 hiba javítására vagy 3 hiba jelzésére alkalmas kódot kaptunk.

Kaszkád kódok

Vegyünk egy $C_1(n_1, k_1, d_1)$ GF(q) feletti és egy $C_2(N_2, K_2, D_2)$ GF(q^{k_1}) feletti lineáris kódot, amelyből az alábbi módon generálhatjuk a szisztematikus, $C(n_1N_2, k_1K_2, d)$ paraméterű GF(q) feletti **kaszkád kód** kódszavait. A k_1K_2 hosszú üzenetet osszuk fel K_2, egyenként k_1 hosszú szegmensre. A C_2 kód egy k_1 hosszú üzenetszegmenst egy üzenetkarakternek vesz, és K_2 ilyen karakter alkot számára egy üzenetszegmenst, amelyből N_2 karakter hosszúságú kódszót képez $N_2 - K_2$ paritáskarakternek az üzenethez való illesztésével. A C_2-beli kódszó elkészülte után a kódszó mindegyik koordinátáját a C_1 kód kódolója újra k_1 hosszúságú üzenetként értelmezi, és $n_1 - k_1$ paritáskarakterrel kiegészíti. Így kapjuk az n_1N_2 hosszú kódszót, ami a kaszkád kód adott k_1K_2 hosszú üzenethez tartozó kódszava. A kaszkád kód kódtávolsága $d \geq d_1D_2$.

A C_1 kódot **belső**, a C_2 kódot **külső kód**nak is nevezik. A kód az elnevezését onnan kapta, hogy a külső kód kódolójának és a belső kód kódolójának a kaszkádba kötése képezi a generált kód kódolóját (2.5. ábra).

A dekódolás során először a C_1 kódszavakat dekódoljuk, majd értelemszerűen, a C_1 kódszavai paritásszegmensének törlése után a C_2 kódszó dekódolását végezzük el.

A kaszkád kódok igen alkalmasak az együttes csomós és véletlen hibák javítására, ahol a csomós hibákat a C_2 kód, a véletlen hibákat a C_1 kód javítja elsősorban. A C_2 kód egy karakterének tetszőleges meghibásodása legfeljebb k_1 méretű q-áris hibaszámnak felel meg. Ugyanakkor ritka egyedi hibák javítása C_1-beli kódszavakban könnyen elvégezhető, míg ezen egyedi hibák C_2-beli karakterszintű javítása „pazarlás" lenne.

Rövidített kód

Egy $C(n,k)$ kód **rövidítés**ével egy $C(n-i,k-i)$, $1 \leq i < k$ kódot kapunk olyan módon, hogy a $C(n,k)$ szisztematikus kód kódszavai közül csak azokat hagyjuk meg, amelyek az első i karakterén zérust tartalmazó üzenetekhez rendeltek. Ekkor a $C(n,k)$ kód i karakterrel történő rövidítéséről beszélünk. Mivel a rövidített kód kódszavai a $C(n,k)$ kód kódszavai is egyben, ezért a rövidített kód minimális távolsága legalább akkora, mint az eredeti kódé volt. Praktikusan természetesen a kódoló és a dekódoló úgy van kiképezve, hogy az első i zérus karaktert nem is továbbítjuk, s a dekóder zérusnak tekinti azokat. A kódrövidítés elsődleges célja a kódhossznak az alkalmazásbeli paraméterekhez való igazítása.

2.16. példa. Konstruáljunk kódszórövidítés módszerével 5 bit hosszon 1 bit hiba javítására bináris kódot. Ehhez rövidítsük a $g(x) = x^3 + x + 1$ generátorpolinomú Hamming-kódot. A rövidített kód az alapkód altere, amelynek szavai az eredeti kód rövidítésnek megfelelő számú 0 bittel kezdődő kódszavai. Az alapkód szisztematikus generátormátrixa

$$\mathbf{G} = \begin{pmatrix} 1000101 \\ 0100111 \\ 0010110 \\ 0001011 \end{pmatrix}$$

amelynek alapján a keresett alteret a

$$\mathbf{G} = \begin{pmatrix} 10110 \\ 01011 \end{pmatrix}$$

mátrix generálja, ahonnan a keresett kód szavai:

$$(00000), (10110), (01011), (11101).$$

Paritásbittel bővítés

A paritáskarakterrel történő kiegészítés után a $C(n,k)$ bináris lineáris alapkódból egy $\widehat{C}(n+1,k)$ lineáris kódot kapunk, amelynek a minimális távolsága az alapkód d minimális távolságával azonos, ha d páros, illetve $d+1$ lesz, ha d páratlan.

A $\mathbf{H}_{\widehat{C}}$ paritásmátrix $\mathbf{H}_C$ ismeretében az alábbi alakú:

$$\mathbf{H}_{\widehat{C}} = \begin{pmatrix} 1\,1 & 1 & \cdots & 1 \\ & & & 0 \\ & \mathbf{H}_C & & \vdots \\ & & & 0 \\ & & & 0 \end{pmatrix} \tag{2.9}$$

2.17. példa. Adjuk meg a $C(7,4)$ Hamming-kód $\widehat{C}(8,4)$ paritásbittel bővített kódjának paritásmátrixát. Az x^3+x+1 generátorpolinomú Hamming-kód bővítésével a (2.9) képlet alapján

$$\mathbf{H}_{\widehat{C}} = \begin{pmatrix} 1\,1\,1\,1\,1\,1\,1\,1 \\ 0\,0\,1\,0\,1\,1\,1\,0 \\ 0\,1\,0\,1\,1\,1\,0\,0 \\ 1\,0\,0\,1\,0\,1\,1\,0 \end{pmatrix}.$$

A kapott $\widehat{C}(8,4)$ kód nyilván nem ciklikus, például a (10001101) a $\widehat{C}$ kódszava, de (11000110) már nem az. A bővített kód minimális távolsága 4, ezért 3 véletlen hiba detektálására alkalmas.

2.8. Hibajavítás és hibajelzés hibavalószínűsége

A bináris kommunikációs csatorna egyik modellje az emlékezetnélküli bináris szimmetrikus csatorna p hibázási valószínűséggel (röviden emlékezetnélküli BSC(p)). Ekkor az egyes bitek átvitele során függetlenül adódnak a meghibásodások, továbbá annak az eseménynek valószínűsége, hogy 0 (1) bit átvitele esetén hibásan 1 (0) bit jelenik meg a csatorna kimenetén, p valószínűségű.

A kódszavak körüli döntési tartományok t sugarú gömbök, ezért a következő felső becslést kapjuk a hibajavító dekódolás kódszó-hibavalószínűségére:

$$P_e \leq 1 - \sum_{i=0}^{t} \binom{n}{i} p^i (1-p)^{n-i}. \tag{2.10}$$

Ha a kód perfekt, akkor a t sugarú gömbök hézagmentesen kitöltik az n bit hosszú szavak terét, ezért egy t-nél nagyobb Hamming-súlyú hiba mindig téves dekódolásra vezet. Így (2.10) jobb oldali formulája a hibavalószínűség pontos értékét adja. Például Hamming-kód esetén a

$$P_{e,\mathrm{corr}} = 1 - ((1-p)^n + np(1-p)^{n-1}) \tag{2.11}$$

formulát kapjuk. Táblázatos szindróma dekódolás esetén tetszőleges kódra ugyancsak pontosan meg tudjuk adni a hibajavítás hibavalószínűségét a tekintett emlékezetnélküli BSC(p) csatorna esetén. A javítható hibavektorok halmazát jelölje

B, amely perfekt kód esetén a zéró vektor körüli t sugarú gömb, egyéb esetekben ennél bővebb (lásd az alábbi példát). A hibavalószínűség formulája az alábbi:

$$P_{e,\text{det}} = \sum_{c \in B \setminus \{0\}} p^{w(c)} (1-p)^{n-w(c)}. \tag{2.12}$$

Tekintsük most a hibajelzés esetét. Ezesetben pontosan akkor keletkezik hiba, ha a vett szó egy nemzérus kódszóval egyezik meg, amely különbözik a továbbított kódszótól. Lineáris kód esetén ez az esemény ekvivalens azzal az eseménnyel, hogy a hibavektor valamely nemzérus kódszóval egyenlő. Így kapjuk egy C lineáris hibajelző blokk kód és emlékezetnélküli BSC(p) csatorna esetén az alábbi általános formulát a hibajelzés hibavalószínűségére:

$$P_{e,\text{det}} = \sum_{c \in C \setminus \{0\}} p^{w(c)} (1-p)^{n-w(c)}. \tag{2.13}$$

2.18. példa. Tekintsük a következő generátormátrixszal definiált kódot:

$$\mathbf{G} = \begin{pmatrix} 11001 \\ 01110 \end{pmatrix}.$$

A kódot emlékezetnélküli BSC(p) csatornán hibajelzésre illetve hibajavításra használjuk. Adjuk meg mindkét alkalmazás esetén a hibázás valószínűségét!

1. Hibajelzés esete:
   Hibajelzés hibája akkor következik be, ha a hibavektor pontosan egy nemzérus kódszónak felel meg. A (2.13) képletet alkalmazva, tekintettel hogy a kódnak 3 nemzérus kódszava lözül kettő kódszó 3 súlyú, egy 4 súlyú, a következő eredmény adódik: $P_{e,\text{det}} = 2p^3(1-p)^2 + p^4(1-p)$.

2. Hibajavítás esete:
   A standard elrendezési táblázat az alábbi:

00000	11001	01110	10111
00001	11000	01111	10110
00010	11011	01100	10101
00100	11101	01010	10011
01000	10001	00110	11111
10000	01001	11110	00111
00011	11010	01101	10100
10001	01000	11111	00110

   Ezen táblázatban minden lehetséges vett szó fel van sorolva, s oszloponként látható az egyes kódszavak döntési tartománya. Az utolsó két sorban vannak olyan szavak, amelyek azonos távolságban vannak több kódszótól is, s ezek döntési tartományba helyezése önkényes (szerencsére az adott statisztikai csatornamodell erre érzéketlen). Ennek megfelelően dekódolási hiba akkor

keletkezik, ha egy kódszó úgy hibásodik meg, hogy a vett szó már kívül esik a kódszó körüli döntési tartományon. Az egyes kódszavak átküldését azonos valószínűségűnek véve

$$P_e = 1 - ((1-p)^5 + 5p(1-p)^4 + 2p^2(1-p)^3).$$

2.9. Alkalmazások

Teletext kód. A $(7,4)$-es Hamming-kódot egy páratlan paritásúra kiegészítő paritásbittel kapunk egy $(8,4)$ paraméterű, továbbra is egy hibát javító kódot, amelyet a Teletextben használnak.

CRC. A ciklikus kódok gyakorlata a leghosszabb múlttal a hibajelzés területén rendelkezik, amikor a kód bináris, a kódokat a szabványok generátorpolinomjuk segítségével adják meg és generálásuk a 2.19. tétel bizonyításában leírt módon, a generátorpolinom szerinti maradékos osztással, szisztematikusan történik. Ezeket a kódokat **CRC** kódoknak hívják (Cyclic Redundancy Check). A hibajelzést legtöbbször zajos, visszacsatolásos csatornáknál használják, amikor a vevő hiba detektálása esetén értesíti az adót, amely ezután az adást ugyanazzal a kóddal vagy egy jobbal megismétli. Ezt az eljárást **ARQ**-nak nevezzük (Automatic Repeat reQuest).

A CCITT 16 paritásbitet tartalmazó szabványában a CRC generátorpolinomja

$$g_1(x) = x^{16} + x^{12} + x^5 + 1.$$

Ezt a generátorpolinomot alkalmazzák pl. az SNC 2653 (Polinomial Generator Checker), Intel 82586 (Local Communication Controller), Intel 8274 (Multi-Protocol Serial Controller), Signetics 2652 (Multi-Protocol Communications Circuit) integrált áramkörökben. A két utóbbiban még választhatjuk a

$$g_2(x) = x^{16} + x^{15} + x^2 + 1$$

polinomot is. Az Intel 82586-os Ethernet chip tartalmaz egy 32 bites generátorpolinomot is:

$$g_3(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1.$$

A 2.17. tétel értelmében ezek a polinomok akkor generátorpolinomjai ciklikus kódoknak, ha osztják az $x^n - 1$ polinomot, tehát csak bizonyos kódszóhosszakra ciklikus kódok. Ugyanakkor erre nem figyelmeztetik a felhasználót. Ez azért nem okoz problémát, mert tetszőleges üzenethossz esetén azért jó kódot kapunk, ugyanis az vagy eleve ciklikus, vagy egy ciklikus kód rövidítése. Legyen C egy (n,k) paraméterű szisztematikus lineáris kód és $k' < k$. Egy

$$\mathbf{u}' = (u'_0, u'_1, \ldots, u'_{k'})$$

üzenethez rendeljük a k hosszú

$$\mathbf{u} = (0,0,\ldots,0,u'_0,u'_1,\ldots,u'_{k'})$$

üzenetet, ahhoz a

$$\mathbf{c} = (0,0,\ldots,0,u'_0,u'_1,\ldots,u'_{k'},c_{k+1},c_{k+2},\ldots,c_n)$$

kódszót és ahhoz a rövidített kódszót:

$$\mathbf{c}' = (u'_0,u'_1,\ldots,u'_{k'},c_{k+1},c_{k+2},\ldots,c_n).$$

Az ilyen $\mathbf{c}'$ kódszavak C' halmazát nevezzük a C kód rövidített kódjának. Nyilván

$$n - n' = k - k'$$

és C' kódtávolsága legalább akkora, mint a C kódé.

Ez utóbbi miatt elég összefoglalni a CRC kódok alapvető tulajdonságait akkor, amikor az ciklikus kód, azaz az n kódszóhosszra a generátorpolinom osztja $x^n - 1$-et.

Ezek után legyen n az a legkisebb természetes szám, melyre $g_1(x) \mid x^n - 1$, és jelölje C az $(n, n-16)$ paraméterű, ciklikus, lineáris kódot, melynek a generátorpolinomja $g_1(x)$, ekkor

1. tulajdonság: $n = 2^{15} - 1 = 32767$.
2. tulajdonság: C jelez minden legfeljebb 3 súlyú hibát.
3. tulajdonság: C jelez minden páratlan súlyú hibát.
4. tulajdonság: C jelez minden olyan hibát, ahol a hibahelyek maximumának és minimumának a távolsága kisebb, mint 16. (Ez utóbbit úgy szokás mondani, hogy a kód jelez minden legfeljebb 16 hosszú hibacsomót.)

Közvetlen műholdas műsorszórás. A közvetlen műholdas műsorszórás (Direct Broadcasting Satellite, DBS) digitalizált hangját is hibajavító kóddal védik. Így a D2-MAC/PACKET szabványa szerint az egyik változatban a 14 bites hangminta felső 11 bitjét egy $(16,11)$ paraméterű kóddal kódolják, ami a $(15,11)$ paraméterű Hamming-kód kiegészítése egy páratlan paritásbittel. A másik változatban a 10 bites hangminta felső 6 bitjét kódolják egy $(11,6)$-os kóddal. Megjegyezzük még, hogy a csomagolt, kódolt beszédmintákat egy olyan csomagfejjel látják el, melyet 2 hibát javító $(71,57)$ ill. $(94,80)$ paraméterű úgynevezett BCH-kóddal védenek, míg a legfontosabb adatokat, az úgynevezett szolgáltatásazonosítást egy három hibát javító $(23,12)$ paraméterű Golay-kóddal kódolják. Ennek a kódnak a kódtávolsága 7, ezért 3 hibát tud javítani. Könnyen ellenőrizhető, hogy a kód paramétereire a Hamming-korlátban az egyenlőség teljesül:

$$\sum_{i=0}^{3} \binom{23}{i} = 2^{11},$$

tehát ez a kód perfekt. Eredetileg Golay a kódot szisztematikus generátormátrixával adta meg:

$$\mathbf{G} = \begin{pmatrix}
1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,0\,1\,0\\
0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,0\,1\\
0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,1\,0\,1\,1\,1\,0\,0\,0\\
0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,1\,0\,1\,1\,1\,0\,0\\
0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,1\,0\,1\,1\,1\,0\\
0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,1\,0\,1\,1\,1\\
0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,0\,1\,1\,0\,1\,1\\
0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,1\,0\,0\,0\,1\,0\,1\,1\,0\,1\\
0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,1\,1\,0\,0\,0\,1\,0\,1\,1\,0\\
0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,1\,1\,0\,0\,0\,1\,0\,1\,1\\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,1\,0\,1\,1\,1\,0\,0\,0\,1\,0\,1\\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1
\end{pmatrix}$$

Kiderült, hogy ez a kód ciklikus is, és a generátorpolinomja

$$g(x) = x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1.$$

Ugyanilyen paraméterű kódot kapunk a

$$g'(x) = x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1$$

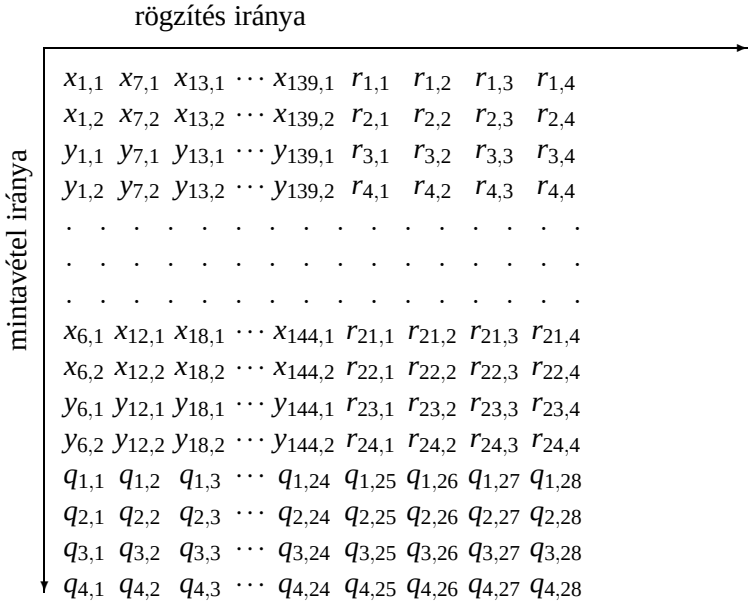
generátorpolinommal. Ezek valóban 23 hosszú ciklikus kódok generátorpolinomjai, ugyanis

$$(x-1)g(x)g'(x) = x^{23} - 1.$$

Mivel a $g(x)$ együtthatói közül 7 darab 1, ezért a minimális súly nem lehet 7-nél nagyobb. Megmutatható, hogy pontosan 7:

2.24. tétel. *A* $(23,12)$ *paraméterű Golay-kód egy 3 hibát javító perfekt, lineáris, ciklikus kód.*

Compact Disc hibavédelme. A digitális hangrögzítésben (CD és DAT) alkalmazott hibavédelem Reed–Solomon-kódra épül. A kódolási eljárás lényegét közelítőleg a következő módon lehet összefoglalni: a 44.1 kHz-cel mintavételezett és 16 bitre kvantált mintákat két bájtban ábrázoljuk, és egy mátrixba írjuk be oszlopfolytonosan.



$$
\begin{array}{cccccccccc}
x_{1,1} & x_{7,1} & x_{13,1} & \cdots & x_{139,1} & r_{1,1} & r_{1,2} & r_{1,3} & r_{1,4} \\
x_{1,2} & x_{7,2} & x_{13,2} & \cdots & x_{139,2} & r_{2,1} & r_{2,2} & r_{2,3} & r_{2,4} \\
y_{1,1} & y_{7,1} & y_{13,1} & \cdots & y_{139,1} & r_{3,1} & r_{3,2} & r_{3,3} & r_{3,4} \\
y_{1,2} & y_{7,2} & y_{13,2} & \cdots & y_{139,2} & r_{4,1} & r_{4,2} & r_{4,3} & r_{4,4} \\
\vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_{6,1} & x_{12,1} & x_{18,1} & \cdots & x_{144,1} & r_{21,1} & r_{21,2} & r_{21,3} & r_{21,4} \\
x_{6,2} & x_{12,2} & x_{18,2} & \cdots & x_{144,2} & r_{22,1} & r_{22,2} & r_{22,3} & r_{22,4} \\
y_{6,1} & y_{12,1} & y_{18,1} & \cdots & y_{144,1} & r_{23,1} & r_{23,2} & r_{23,3} & r_{23,4} \\
y_{6,2} & y_{12,2} & y_{18,2} & \cdots & y_{144,2} & r_{24,1} & r_{24,2} & r_{24,3} & r_{24,4} \\
q_{1,1} & q_{1,2} & q_{1,3} & \cdots & q_{1,24} & q_{1,25} & q_{1,26} & q_{1,27} & q_{1,28} \\
q_{2,1} & q_{2,2} & q_{2,3} & \cdots & q_{2,24} & q_{2,25} & q_{2,26} & q_{2,27} & q_{2,28} \\
q_{3,1} & q_{3,2} & q_{3,3} & \cdots & q_{3,24} & q_{3,25} & q_{3,26} & q_{3,27} & q_{3,28} \\
q_{4,1} & q_{4,2} & q_{4,3} & \cdots & q_{4,24} & q_{4,25} & q_{4,26} & q_{4,27} & q_{4,28}
\end{array}
$$

Nevezetesen egy 24×24-es mátrix oszlopai egymás után következő 6 mintavételi időpontban vett két minta (bal és jobb hangcsatorna) $2 \times 2 = 4$ bájtját tartalmazzák. Ha $x_{i,1}, x_{i,2}$ jelöli a jobb csatorna mintáját az i-edik időpillanatban, és $y_{i,1}, y_{i,2}$ a bal csatornáét, akkor a fenti ábra mutatja a minták beírását a táblázatba. A kapott 24×24-es mátrix minden oszlopát kódoljuk egy $(28, 24)$ paraméterű, GF(2^8) feletti szisztematikus Reed–Solomon-kóddal. A j-edik oszlop paritásbájtjait jelöltük $q_{1,j}, q_{2,j}, q_{3,j}, q_{4,j}$-vel. Ennek a kódnak a kódtávolsága 5, tehát 4 hibát tud jelezni, 2 egyszerű hibát tud javítani és 4 törléses hibát tud javítani. A digitális lemezen előforduló hibák jól modellezhetők egy kétállapotú csatornával. Az egyik állapotot nevezzük JÓ állapotnak, melyben átlagosan 10000–20000 bitideig tartózkodik, és ekkor a hibák előfordulása független egymástól és valószínűsége kb. 10^{-4}. A másik állapotot nevezzük ROSSZ állapotnak, amiben 30–40 bitideig tartózkodik, és ekkor gyakorlatilag használhatatlan a vétel. Ekkor azt mondjuk, hogy a hibázás csomós (burst-ös). Az ilyen csatornák kódolására találták ki a kódátfűzés (interleaving) technikát, amikor az előbbi mátrixot sorfolytonosan olvassák ki, de előtte minden sort kódolnak ugyanazzal a $(28, 24)$ paraméterű Reed–Solomon-kóddal. A j-edik sor paritásbájtjait jelöli $r_{j,1}, r_{j,2}, r_{j,3}, r_{j,4}$. Ennek előnye az, hogy a fizikailag összefüggő, csomós hiba hatását több kódszóra osztja szét.

A Sony és a Philips megegyezett a fentihez hasonló (kicsit bonyolultabb) kódolásban azért, hogy a tömeges digitális hanglemezgyártás elindulhasson. A verseny nyitott viszont a lejátszó készülékben, vagyis a dekódolás terén. A különböző dekódolások igaziból a következő egyszerű eljárás finomításai: számítsuk ki soronként a szindrómát! Ha a szindróma 0, akkor azzal a sorral készen vagyunk. Ha 1 hiba volt, akkor azt kijavítjuk. Ha 2 hiba volt, akkor azt kijavítjuk, és az oszloponkénti javításhoz ezeket a hibahelyeket megjegyezzük, azaz mesterségesen törléses hibákat generálunk. Minden egyéb esetben az egész sort törléses hiba-

ként regisztráljuk. Ezek után oszloponként javítunk, ha ott legfeljebb két törléses hiba volt (emlékeztetünk, hogy 4 törléses hibát képes a rendszer javítani). Ha a hibák száma nagyobb, mint 2, akkor a környező hibátlan mintákból interpolálunk. Látható, hogy a hibajavítás nem használja ki a Reed–Solomon-kód hibajavítási lehetőségeit, aminek elsősorban technológiai okai vannak, mivel a dekódolás bonyolultsága a javítandó hibák számának négyzetével arányos, és itt igen gyorsan kell dekódolni (a forrás sebessége $2 \cdot 44100 \cdot 16 = 1.4112$ Mbit/sec)

2.10. Feladatok

Lineáris blokk-kódok

2.1. feladat. Egy $\{0,1,2\}$ kódábécéjű GF(3) feletti lineáris kód generátormátrixa:

$$\mathbf{G} = \begin{pmatrix} 1021 \\ 0122 \end{pmatrix}$$

Adja meg a kódszavakat, valamint a d minimális távolságot!

2.2. feladat. Egy lineáris bináris kód paritásellenőrző mátrixa
$\mathbf{H} = \begin{pmatrix} 1\,1\,1\,1\,1\,1 \end{pmatrix}$. Adja meg a kód következő paramétereit: n,k,d, kódszavak száma!

2.3. feladat. Egy lineáris bináris blokk-kód generátormátrixa:

$$\mathbf{G} = \begin{pmatrix} 10110 \\ 01101 \end{pmatrix}$$

Adja meg

a) a kód paramétereit: n,k,d,

b) standard elrendezési táblázatát,

c) szindróma dekódolási táblázatát,

d) a kódszó dekódolási hibavalószínűséget emlékezetnélküli BSC(p) esetére!

2.4. feladat. Egy lineáris bináris kód paritásellenőrző mátrixa:

$$\mathbf{H} = \begin{pmatrix} 11100 \\ 10010 \\ 11001 \end{pmatrix}$$

Adja meg a szindróma dekódolási táblázatot!

2.5. feladat. Egy lineáris bináris kód generátormátrixa az alábbi:

$$\mathbf{G} = \begin{pmatrix} 101011 \\ 011101 \\ 011010 \end{pmatrix}$$

Adja meg:

a) egy ekvivalens szisztematikus kód generátor- és paritásellenőrző mátrixát,

b) a duális kód kódszavait (duális kód = a paritásellenőrző mátrix mint generátormátrix által generált kód).

2.6. feladat. Adja meg a GF(4) feletti $C(4,2)$ paraméterű,

$$\mathbf{G} = \begin{pmatrix} 1022 \\ 0112 \end{pmatrix}$$

generátormátrixú kód szindróma dekódolási táblázatát! ($0 \leftrightarrow 0,\ 1 \leftrightarrow 1,\ 2 \leftrightarrow x,\ 3 \leftrightarrow x+1$)

2.7. feladat. Definiáljon egy $(5,3)$ paraméterű GF(4) feletti kódot a generátormátrixa, amely

$$\mathbf{G} = \begin{pmatrix} 10011 \\ 01012 \\ 00113 \end{pmatrix}.$$

a) Mennyi a kód minimális távolsága?

b) Perfekt-e a kód?

c) Mi lehetett az átküldött kódszó, ha a vett szó $\left(1\ ?\ 1\ 3\ ?\right)$?
A kód tisztán $0,1$ elemeket tartalmazó kódszavakat is tartalmaz.

d) Adja meg a bináris kódszavakat!

e) Igazolja, hogy ezen bináris kódszavak lineáris részkódot alkotnak az eredeti kódban!

f) Adja meg ezen részkód (n,k,d) paraméterhármasát!

g) Adja meg a részkód generátormátrixát!

2.8. feladat. Egy GF(5) feletti $(5,3)$ paraméterű lineáris kód kódszavai között vannak a $(0,1,0,1,2)$, $(1,0,0,1,4)$, $(0,0,1,1,3)$ szavak is. Adja meg a kód hibajavító képességét!

2.9. feladat. Adja meg egy $(21,18)$ paraméterű GF(4) feletti, egy hibát javító kód paritásmátrixát és generátormátrixát.

2.10. feladat. Adja meg a legkisebb kódszóhosszú GF(3) feletti, 1 hibát javító, $k = 2$ üzenethosszú szisztematikus kódot paritásellenőrző mátrixával!

2.11. feladat. Létezik-e $C(n,k)$, $n-k=2$, GF(3) feletti 1 hibát javító kód? Ha igen, adja meg szisztematikus mátrixaival!

2.12. feladat. Adja meg a $C(n,1,d=n)$ paraméterű bináris kód C' duális kódját (duális kód = paritásellenőrző mátrix mint generátormátrix által generált kód), s annak paraméterhármasát! Adja meg C' szavait $n=4$ esetén!

2.13. feladat. Létezhet-e olyan GF(q) feletti lineáris blokk-kód, amelynek generátormátrixa egyben a kód paritásellenőrző mátrixa is? Ha válasza igen, mutasson példát rá, mind $q=2$, mind pedig $q>2$ esetben.

2.14. feladat. Valaki azt állítja, hogy ha egy $C(n=2m-1,k)$ lineáris bináris kódnak a csupa 1 kódszó eleme, akkor pontosan eggyel kevesebb páros paritású nemzérus kódszava van, mint páratlan paritású. Igaza van-e?

2.15. feladat. Legyen $C(n,k)$ egy lineáris bináris blokk-kód, amelynek generátormátrixában nincsen csupa zérus oszlop. Igaz-e, hogy az összes kódszó egyeseinek összesített darabszáma $n \cdot 2^{k-1}$?

2.16. feladat. Egy GF(q) feletti lineáris blokk-kód paritásellenőrző mátrixa

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{q-2} \end{pmatrix},$$

ahol α a test primitív eleme. Adja meg a kódtávolságot!

2.17. feladat. A legegyszerűbb konstrukciójú hibajavításra már alkalmas nemtriviális kód a kétdimenziós bináris paritáskód. (Az üzenetet mátrixba rendezzük, soronként és oszloponként paritásbittel egészítjük ki, majd a jobb alsó sarokba írjuk a paritások paritását.) Mennyi a minimális távolság?

2.18. feladat. Igazolja, hogy a kétdimenziós bináris paritáskód jobb alsó paritáselemét (azaz a paritások paritását) képezhetjük akár a sorparitások paritásaként, akár az oszlopparitások paritásaként, azaz mindkét esetben azonos eredményre jutunk.

2.19. feladat. Konstruáljon egy GF(q), $q=2^m$ feletti $(q+1,q-1)$ paraméterű 1 hibát javító lineáris blokk-kódot.

a) Adja meg a szisztematikus paritásellenőrző mátrixot!

b) Perfekt-e a kód?

c) Adja meg $q=4$ esetre a generátormátrixot! ($0 \leftrightarrow 0$, $1 \leftrightarrow 1$, $2 \leftrightarrow x$, $3 \leftrightarrow x+1$)

2.20. feladat. Igaz-e, hogy tetszőleges $C(n,k,d=3)$ paraméterű lineáris kódot egy paritásszimbólummal kiegészítve $C'(n+1,k,d=4)$ paraméterű kódot kapunk?

Kalkulus

2.21. feladat. Tekintse az $S=\{0,1,2,3\}$ halmazt az alábbi műveleti táblák szerinti „+” és „$*$” műveletekkel:

$+$	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

$*$	**0**	**1**	**2**	**3**
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Testet kapunk-e?

2.22. feladat. Konstruálja meg GF(4) műveleti tábláit!

2.23. feladat. Konstruálja meg GF(8) műveleti tábláit:

a) Az x^3+x+1 bináris irreducíbilis polinom felhasználásával!

b) Ismételje meg a konstrukciót az x^3+x^2+1 bináris irreducíbilis polinom felhasználásával, s mutassa meg hogy a két test izomorf (az elemek átnevezésével azonos műveleti táblákhoz jutunk)!

2.24. feladat. Legyen adva GF(4) a következő műveleti táblákkal:

$+$	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

$*$	**0**	**1**	**2**	**3**
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

a) Oldja meg az alábbi GF(4) feletti egyenletrendszert:

$$\begin{aligned} 2x+y &= 3 \\ x+2y &= 3 \end{aligned}$$

b) Számítsa ki az alábbi GF(4) feletti mátrix determinánsát!

$$\det\begin{pmatrix} 2 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 0 & 1 \end{pmatrix} = ?$$

2.25. feladat. Adja meg az $x+1 \in$ GF(8) polinom alakban megadott testelem inverzét, ha x^3+x^2+1 az aritmetika generáló polinom!

2.26. feladat.

a) Mutassa meg, hogy a $p(x) = x^3+x^2+2$ GF(3) feletti polinom irreducíbilis!

b) Adja meg GF(27) elemeinek rendjét!

c) Mi az x polinom által reprezentált elem rendje GF(27)-ben, ha $p(x)$ az aritmetika generáló polinom?

2.27. feladat. Konstruálja meg GF(9) műveleti tábláit!
Segítség: $f(x) = x^2+x+1$ GF(3) feletti irreducíbilis (primitív) polinomot használhatja aritmetika generálásra.

2.28. feladat. A GF(16) test összeadó- és szorzótábláját többféleképpen is megkonstruálhatjuk:

a) GF(2) feletti 4-edfokú irreducíbilis polinommal,

b) GF(4) feletti másodfokú irreducíbilis polinommal.

Kövessük a b) utat!
Segítség: A polinomegyüttható aritmetikája GF(4), amelyhez a műveleti táblák például a 2.24. feladatnál találhatók. Az aritmetika generáló polinomot szita módszerrel kaphatjuk. Ezzel az alábbi másodfokú GF(4) feletti irreducíbilis polinomokat kapjuk: x^2+x+2, x^2+x+2, x^2+2x+1, x^2+2x+2, x^2+3x+1, x^2+3x+3.

Ciklikus kódok

2.29. feladat. Valaki azt állítja, hogy egy 1 hibát javító bináris ciklikus kód egyik szava 0001111. Lehetséges ez?

2.30. feladat. Tekintsük a $g(x) = x^3+x^2+1$ generátorpolinomú, $n = 7$ kódszóhosszú bináris Hamming-kódot.

a) Adja meg a kód $h(x)$ paritásellenőrző polinomját és szisztematikus alakú generátormátrixát!

b) Tekintsük a kód nem páros súlyú szavainak halmazát. Adja meg ezen részkód méretét, valamint minimális távolságát!

c) Tekintsük a nem páratlan súlyú szavainak halmazát. Lineáris, illetve ciklikus-e ez a halmaz, s mik a paraméterei?

2.31. feladat. A $g(x) = x^3+x+1$ bináris polinom egy 7 kódszóhosszú Hamming-kód generátorpolinomja. Adja meg

a) a kódszavak halmazát,

b) a minimális távolságot,

c) a paritásellenőrző polinomot!

2.32. feladat. Egy $n = 7$ kódszóhosszú bináris ciklikus blokk-kód generátorpolinomja $g(x) = x - 1$. Adja meg a

a) lehetséges kódszósúlyokat, és a k, d paramétereket,

b) paritásellenőrző polinomot,

c) szisztematikus paritásmátrixot!

2.33. feladat. Egy $n = 7$ hosszú bináris ciklikus kód generátorpolinomja $g(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$. Adja meg a kódszavak halmazát!

2.34. feladat.

a) Hány különböző 7 kódszóhosszú bináris ciklikus kód van?

b) Adja meg (n,k,d) paramétereivel és $g(x)$ generátorpolinomjával az összes lehetséges bináris $n = 7$ kódszóhosszú ciklikus kódot!

2.35. feladat. Egy C' kódot úgy származtatunk, hogy egy $g(x)$ generátorpolinomú $C(n,k)$, GF(q) feletti Reed–Solomon-kód kódszavait tükrözzük, azaz elemeit fordított sorrendben tekintjük ($c'_i = c_{n-1-i},\ i = 0,1,\ldots,n-1$).

a) Ciklikus-e C'?

b) Ha az a) kérdésre a válasz igen, akkor adja meg a C' kód $g'(x)$ generátorpolinomját $g(x)$ alapján, továbbá annak gyökeit, ha $g(x)$ gyökei $\alpha_1, \alpha_2, \ldots, \alpha_{n-k}$.

2.36. feladat. Egy $C(n,k),\ n = 2^m - 1$ bináris ciklikus kód $g(x)$ generátorpolinomját osztja az $x + 1$ polinom. Eleme-e a kódnak a csupa 1 szó?

2.37. feladat. Egy $C_1(n,k_1,d_1)$ illetve egy $C_2(n,k_2,d_2)$ ciklikus kód $h_1(x)$ illetve $h_2(x)$ paritásellenőrző polinomja közötti kapcsolat $h_1(x) \mid h_2(x)$. Mi a kapcsolat:

a) C_1 és C_2 között,

b) d_1 és d_2 között?

2.38. feladat. Egy $C_1(n_1,k_1)$ ciklikus kód egy $C_2(n_2,k_2)$ ciklikus kód részkódja. Mi az algebrai kapcsolat a megfelelő

a) $h_1(x), h_2(x)$ paritásellenőrző polinomok között,

b) $g_1(x), g_2(x)$ generátorpolinomok között?

2.39. feladat. Igazolja, hogy egy $h(x)$ paritásellenőrző polinomú ciklikus kód paritásellenőrző mátrixának sorait a $(0,0,\ldots,h_k,h_{k-1},\ldots,h_0)$ vektor ciklikus eltolásaival nyerhetjük.
Segítség:
Ellenőrizze, hogy az így kapott **H** sorai lineárisan függetlenek és ortogonálisak a kódszavakra.

2.40. feladat. Képezzük a CRC-t a $g(x) = x^5 + x^3 + x^2 + 1$ generátorpolinommal. Jelez-e hibát a detektor, ha a vett szó $v = (0000000100111011)$, ahol a jobb oldali bit a zéró helyiértékű?

2.41. feladat. A következőket állítja valaki:

a) Egy $C(n,k)$ ciklikus lineáris kód $h(x)$ paritásellenőrző polinomját használhatom egy n kódszóhosszú C' kód generátorpolinomjaként.

b) A C' kód minimális kódtávolsága elérheti a $k+2$ értéket is.
Igazak-e az állítások?

Kódkorlátok

2.42. feladat. Konstruálható-e $n = 11, k = 5$ paraméterű $t = 2$ hibát javító bináris kód?

2.43. feladat. Valaki azt állítja, hogy olyan kódot tervezett, amely 7 redundanciakarakterrel meghosszabbítja az üzenetblokkot, és 4 véletlen hibát képes javítani a kódszóban. Lehetséges ez?

2.44. feladat. Létezik-e $C(n,k)$, $n-k = 2$, GF(3) feletti egy hibát javító kód? Ha igen adjon példát, megadva a szisztematikus paritásellenőrző mátrixát és a kódszavait!

2.45. feladat. Perfekt-e a $\{(1\ 1\ 1\ 1\ 1\ 1\ 1), (0\ 0\ 0\ 0\ 0\ 0\ 0)\}$ kódszavakat tartalmazó kód?

2.46. feladat. Perfekt-e egy $C(11,6)$ paraméterű GF(3) feletti 2 hibát javító kód?

RS-kódok

2.47. feladat. Tekintsünk egy GF(11) feletti Reed–Solomon-kódot $g(x) = (x-2)(x-4)(x-8)(x-5)$ generátorpolinommal. Adja meg a kód következő jellemzőit: minimális távolság (d), hibajavító képesség (t_c), hibadetektáló képesség (t_d), törlésjavító képesség (t_e)!

2.48. feladat. Adja meg egy GF(11) feletti három hibát javító, $n = 10$ szóhosszú Reed–Solomon-kód paramétereit, generátorpolinomját, paritásellenőrző polinomját!

2.49. feladat. Eleme-e az $(1,1,\ldots,1)$ csupa 1 vektor egy GF(q) feletti $C(n,k)$, $n = q-1$ Reed–Solomon-kódnak, ahol a generátorpolinom gyökei az α primitív elem $0,1,2,\ldots,n-k-1$ hatványai?

2.50. feladat. Egy t hibát javító GF(q) feletti Reed–Solomon-kód generátorpolinomjának gyöke az α primitív elem $2,\ldots,2t$-edik hatványa. Lehetséges-e, hogy a kódszavak elemeinek (koordinátáinak) összege a test zéró eleme legyen?

2.51. feladat. Legyen $\boldsymbol{\alpha}^{(r)} = \left(1,\alpha^r,\alpha^{2r},\ldots,\alpha^{(n-1)r}\right),\ r = 0,1,\ldots,n-1$, ahol $\alpha \in$ GF(q) egy n-edrendű elem. Igaz-e, hogy ha egy $C(n,k)$ kód $\mathbf{G}$ generátormátrixának sorai rendre $\boldsymbol{\alpha}^{(r)},\ r = 0,1,\ldots,k-1$, akkor $\mathbf{H}$ mátrixának sorai lehetnek rendre az $\alpha^{(r)},\ r = 1,\ldots,n-k$ vektorok?

2.52. feladat. Legyen $\boldsymbol{\alpha}^{(r)} = \left(1,\alpha^r,\alpha^{2r},\ldots,\alpha^{(n-1)r}\right),\ r = 0,1,\ldots,n-1$, ahol GF($q$) egy n-edrendű elem. Van-e olyan $C(n,k)$ kód, amelyre a $\mathbf{G}$ és $\mathbf{H}$ mátrixainak sorai rendre ugyanazok az $\boldsymbol{\alpha}^{(r)}$ alakú vektorok?

2.53. feladat. Legyen $g(x) = x^3 + x + 1$ egy $C(7,4)$ bináris Hamming-kód generátorpolinomja. Mutassuk meg, hogy C lineáris részkódja egy $C'(7,5)$ GF(8) feletti Reed–Solomon-kódnak!

2.54. feladat. Igaz-e a következő állítás? Egy $C(n,k,d)$ GF(q) feletti Reed–Solomon-kód kódszavaiból kiemelve bármely, rögzített k méretű koordinátahalmaz által meghatározott részvektorokat, azok különbözők a különböző kódszavakra.

Kódkombinációk, kódmódosítások

2.55. feladat. Adja meg a $g(x) = x^3 + x + 1$ generátorpolinomú $C(7,4)$ kód nem páratlan súlyú szavai C' részkódjának halmazát, s ezen részkód paramétereit!

2.56. feladat. Perfekt marad-e a $C(n,k)$ bináris Hamming-kód, ha kódrövidítést hajtunk végre, amelynek mértéke

a) 1 bit,

b) 2 bit?

2.57. feladat. A $C(7,4)$ bináris Hamming-kód kódszavait paritáskarakterrel bővítjük páros paritásúra egészítve ki a kódszavakat. Adja meg a kapott kód paramétereit!

2.58. feladat. A $(7,4)$ bináris Hamming-kódból kiindulva konstruáljon bináris kódot, amelynek 8 kódszava van, 7 a szóhossza és alkalmas 3 hiba detektálására.

2.59. feladat. A $g(x) = x^3 + x^2 + 1$ generátorpolinomú szisztematikus Hamming-kódon 3 bites kódrövidítést hajtunk végre.

a) Adja meg a rövidített kód (n,k) paramétereit!

b) Adja meg a kódszavakat és a kód d paraméterét!

2.60. feladat. Adja meg a 8 bitnyi kódrövidítéssel kapható kód paramétereit és kódszavait, ha a $g(x) = x^4 + x + 1$ generátorpolinomú bináris Hamming-kódot rövidítettük.

2.61. feladat. Egy bináris lineáris $C(n,k,d)$ kód nem tartalmazza a csupa egyesekből $(\mathbf{1})$ álló kódszót. Mit mondhatunk a $C' = C \oplus \mathbf{1}$ kódról, ahol $\oplus$ a koordinátánkénti mod 2 összeadás?

a) Lineáris-e?

b) Mik a paraméterei: n', k', d' ?
Mit mondhatunk a $C'' = C \cup C'$ kódról, ahol $\cup$ a halmazegyesítés:

a) Lineáris-e?

b) Mik a paraméterei: n'', k'', d'' ?

2.62. feladat. Egy GF(q) feletti $n = q - 1$ szóhosszú C Reed–Solomon-kód generátorpolinomjának gyökei $\alpha, \alpha^2, \ldots, \alpha^{d-1}$, $\alpha \in$ GF(q). A kódot egy „paritás” karakterrel bővítjük, olyan módon, hogy a kódszó karaktereinek testbeli aritmetika szerinti összege lesz az $n+1$-edik karakter. MDS tulajdonságú marad-e a kapott q szóhosszú kód?

2.63. feladat. Egy kommunikációs csatornán nagyon ritkán maximum 8 bit hosszú hibacsomók keletkeznek. A következő beállítható paraméterű kódolási elemekben gondolkozunk:

a) bináris Hamming-kódoló,

b) bájt karakter alapú Reed–Solomon-kódoló,
valamint alkalmazhatjuk a kódátfűzés technikát is. A cél minimális redundancia mellett elvégezni a javítást. Milyen konstrukciót alkalmazzunk?

2.64. feladat. Mutassuk meg, hogy egy C és a m-szeres átfűzése, C^m kód generátorpolinomja között a következő egyszerű kapcsolat áll fenn: Ha $g(x)$ a C kód generátorpolinomja, akkor $g(x^m)$ a C^m kód generátorpolinomja.

Hibajavító dekódolás

2.65. feladat. Egy GF(11) feletti lineáris blokk-kód paritásellenőrző mátrixa

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^9 \end{pmatrix},$$

ahol $\alpha = 2$ a test primitív eleme.

a) Adja meg a kód paramétereit!

b) Adja meg a dekódolás menetét az $e(x) = 5x^3$ hibapolinom esetére!

2.66. feladat. Valaki azt állítja, hogy nem feltétlenül kell egy GF(2^m) feletti $C(n = 2^m - 1, k)$ bináris kód generátorpolinomjában 4 ciklikusan egymás utáni gyöknek lennie ahhoz, hogy a kód $t = 2$ hibát javíthasson. Szerinte az is megfelelő, ha a $g(x)$ generátorpolinom olyan, hogy $g(\alpha) = g(\alpha^{-1}) = 0$, ahol α a GF(2^m) primitív eleme. Igaza van-e?
Segítség:
A szindrómaegyenletek megoldhatóságnak közvetlen vizsgálatával ellenőrizze az állítást.

2.67. feladat. Valaki a következő gondolatmenetet mondja a társának:
Én úgy tudom, hogy egy $C(n,k)$ lineáris bináris blokk-kód szindróma dekódolási táblázata mindig $2^{(n-k)}$ javítható hibamintát tartalmaz, tehát végülis a minimális távolságnak nincs jelentősége, s mindegyik (n,k) paraméterű kód egyformán hasznos (hasznos = emlékezetnélküli BSC csatornán használva azonos kódszó-dekódolási hibavalószínűséget kapunk). Tömör érveléssel tegyen igazságot! Mutasson egy egyszerű példát is!

2.68. feladat. Egy $C(n,k)$ blokk-kód dekódolását szindróma dekódolási táblázat alapján végezzük. Az alábbi két állítás közül melyik az igaz és miért?

a) A dekóder kimenetén az aktuális hibázástól függetlenül mindig valamilyen — esetleg hibás — kódszó jelenik meg.

b) A dekóder a táblázat alapján egyszerűen levon a vett szóból egy hibamintát, így súlyosabb hibázás esetén előfordulhat, hogy nem egy kódszó jelenik meg a kimeneten.

2.69. feladat. Van GF(256) aritmetikában gyorsan számoló egységünk. A csatorna hibázása olyan, hogy ritkán, legfeljebb 16 bit hosszú hibacsomók keletkezhetnek. Milyen kódot javasol a javításra, ha maximalizálni szeretnénk a kódolási sebességet?

2.70. feladat. Egy üzenetforrás kimenetén 8 különböző karakter jelenhet meg. Ha ezeket a karaktereket az átviteli csatornán továbbítjuk, akkor alkalmanként egy

karakter meghibásodik, de két hibás karakter közti távolság legalább 10 karakter. Blokk-kódos hibajavítást szeretnénk alkalmazni azzal a megkötéssel, hogy a relatív redundancia nem haladhatja meg a 30 %-ot. Javasoljon megoldást a hibajavításra, adja meg az alkalmazandó kódot!

Hibadetekció

2.71. feladat. Valaki azt állítja, hogy tetszőleges m fokszámú, bináris $f(x)$ polinom, amelynek konstans tagja 1, alkalmas arra, hogy CRC generátorpolinomként detektáljunk vele tetszőleges, legfeljebb m bit hosszúságú hibacsomagot. (Pl. 1xxx1 egy 5 hosszú hibacsomag.) Igaza van-e?

2.72. feladat. Egy hibadetekciós protokollban a $000\ldots0011$ (két utolsó bitjén 1-et tartalmazó) üzenetcsomaghoz a $g(x) = x^{16} + x^{12} + x^5 + 1$ szabványos generátorpolinommal ciklikus redundancia ellenőrző összeget (CRC) alkalmazunk. Adja meg az ellenőrzőösszeggel ellátott blokkot!

Hibavalószínűség

2.73. feladat. Emlékezetnélküli q-áris ($0,1,\ldots,q$ bemeneti ábécé, $0,1,\ldots,q$ kimeneti ábécé) csatornán kommunikálunk, ahol a hibázás valószínűsége p ($\mathbf{P}(i \mid j) = p,\ i \neq j$). Egy (n,k) paraméterű GF(q) feletti t hibát javító perfekt kódot használunk csatornakódként. Adja meg egy kódszó téves dekódolásának valószínűségét!

2.74. feladat. Tekintsük a $g(x) = x^4 + x + 1$ generátorpolinomú bináris Hamming-kódot. A kódot hibajavításra használjuk. Adja meg egy kódszó téves dekódolásának valószínűségét!

2.75. feladat. A $(8,4)$ paraméterű paritásbittel bővített bináris Hamming-kódot hibadetekcióra használjuk p hibázási valószínűségű emlékezetnélküli bináris szimmetrikus csatornán. Adja meg a detekció mulasztás valószínűségét!

2.76. feladat. A $C(n,k=1)$ paraméterű bináris ismétléses kódot

a) tisztán törléses csatornán

b) véletlen bithibázásos csatornán

használjuk, ahol a törlés illetve hibázás valószínűsége p. Adja meg mindkét esetben egy kódszó téves dekódolásának valószínűségét!

2.77. feladat. 1 bitnyi üzenetet úgy viszünk át a csatornán, hogy ismételjük azt, azaz a $(00\ldots0)$ illetve az $(11\ldots1)$ szavak valamelyikét küldjük át. Legyen $p = 0.01$ a bithibázás valószínűsége az emlékeznélküli bináris csatornában. Mennyivel javul a téves dekódolás valószínűsége, ha $n = 3$ hosszú szavak helyett $n = 5$ hosszúakat használunk?

2.78. feladat. Az alábbi méretű kétdimenziós paritáskódot paritásellenőrzésre használjuk (u: üzenetbit, p: paritásbit)

$$\begin{pmatrix} u & u & p \\ u & u & p \\ p & p & p \end{pmatrix}.$$

Adja meg a hibadetekció elmulasztásának valószínűségét emlékezetnélküli BSC(p) csatorna esetén!

2.79. feladat. Egy bináris, tisztán törléses emlékezetnélküli csatornán $p = 0.05$ a törlés és $1 - p = 0.95$ a hibátlan továbbítás valószínűsége. Félbájtos (4 bit) egységekben továbbítjuk a forrás információját, amelyet 4 bit redundanciával kiegészítünk kódszóvá. Hasonlítsuk össze az alábbi két kódolási eljárást a kódolási hatékonyság szempontjából:

a) a redundancia nem más mint az üzenet félbájt megismétlése,

b) a $(8,4)$ paraméterű, paritásbittel kiegészített Hamming-kódot használjuk.

2.80. feladat. Tisztán törléses hibát okozó emlékezetnélküli bináris csatornán $p = 0.05$ a törlés valószínűsége. 4 bites üzeneteinket paritásbittel bővített $(7,4)$ Hamming-kóddal továbbítjuk. Elfogadhatóan választottuk-e a kódot, ha üzeneteinket legalább 0.999 valószínűséggel szeretnénk a vevőben helyesen rekonstruálni?

2.81. feladat. 18 bájt méretű üzenetcsomagjainkat 2 bájt méretű CRC-vel védjük egy $p = 0.001$ bithibázás valószínűségű emlékezetnélküli BSC csatornán.

a) Tegyük fel, hogy zajmentes nyugtázócsatorna áll rendelkezésre, s a hibadetekció tökéletes! Mennyi a csomagismétlések átlagos száma?

b) Mekkora ugyanez a szám, ha a nyugtázó csatorna is ugyanilyen mértékben hibázhat, ahol az 1 bájt méretű nyugta szintén 2 bájt méretű CRC-vel védett. Az adó csak akkor nem ismétel, ha hibátlan nyugta érkezik. Mennyi a csomagismétlések átlagos száma?

2.82. feladat. p hibázási valószínűségű emlékezetnélküli csatornán N bájt méretű, T időtartamú csomagokat továbbítunk. A hibakontroll CRC alapú hibadetekció. Az adó addig nem küldi el a következő csomagot, amíg az utoljára elküldött csomag sikeresen át nem jutott a csatornán, s erről a visszairányú csatornán nyugtát nem kapott. Tegyük fel, hogy a nyugtázás szintén T időt vesz igénybe. Mekkora a csomagkésleltetés várható értéke, ha a visszairányú csatorna hibamentes?

2.11. Megoldások

2.1. megoldás. A kódszavak halmaza $\{0000, 1021, 0122, 2012, 0211, 1110, 1202, 2101, 2220\}$, $d = 3$

2.2. megoldás. A **H** mátrix dimenziója $(n-k) \times n$, ezért $n = 6, k = 5$. Mivel azok és csak azok a kódszavak, amelyek ortogonálisak **H** soraira, azaz $\mathbf{Hc}^T = 0$, ezért adott esetben a nemzérus kódszavak nyilván a páros súlyú 6 bites szavak. Így $d = 2$. A kódszavak száma $2^k = 32$.

2.3. megoldás.

a) A **G** mátrix dimenziója $k \times n$, ezért $n = 5, k = 2$.
A négy kódszó: $00000, 10110, 01101, 11011$, ahonnan $d = 3$.

b)

00000 10110 01101 11011
00001 10111 01100 11010
00010 10100 01111 11001
00100 10010 01001 11111
01000 11110 00101 10011
10000 00110 11101 01011
00011 10101 01110 11000
10001 00111 11100 01010

amelynek alapján a kód javítani képes minden 1 súlyú hibát, továbbá két rögzített 2 súlyú hibát.

c)

szindróma	hibavektor
000	00000
001	00001
010	00010
011	00011
100	00100
101	01000
110	10000
111	10001

d) A szindróma dekódolási táblázatot kiegészítettük a 2 súlyú hibákkal. Ennek alapján láthatjuk, hogy a 10 darab 2 súlyú hibaminta esetén 6 esetben — a fenti táblázatban az azonos sorban levő — 1 súlyú hibamintával próbál javítani a dekóder. A fennmaradó 4 esetből kettőt helyesen javít, kettőt hibásan 2 súlyúval. Tehát a dekódoló kétsúlyú hibák esetén 0.2 valószínűséggel javít helyesen, s

0.8 valószínűséggel hibázik. A kérdezett kódszó-dekódolási hibavalószínűség: $P_e = 1-((1-p)^5+5p(1-p)^4+2p^2(1-p)^3))$

2.5. megoldás.

a) A **G** mátrix 2. és 4. valamint a 3. és 5. oszlopát felcseréljük,

$$\mathbf{G}' = \begin{pmatrix} 101011 \\ 010111 \\ 001110 \end{pmatrix}$$

majd a 3. sort az 1. sorból kivonjuk:

$$\mathbf{G}'' = \begin{pmatrix} 100101 \\ 010111 \\ 001110 \end{pmatrix}$$

szisztematikus alakú ekvivalens generátor mátrixot kapunk. Innen a keresett ekvivalens szisztematikus paritásellenőrző mátrix:

$$\mathbf{H} = \begin{pmatrix} 111100 \\ 011010 \\ 110001 \end{pmatrix}$$

b) A **H** mátrix sortere:
000000, 111100, 011010, 110001, 100110, 001101, 101011, 010111

2.6. megoldás.

$$\mathbf{H} = \begin{pmatrix} 2110 \\ 2201 \end{pmatrix}$$

GF(4) műveleti táblák:

$+$	**0**	**1**	**2**	**3**
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

$*$	**0**	**1**	**2**	**3**
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

szindróma	hibavektor	szindróma	hibavektor
00	0000	20	0020
01	0001	21	0021
02	0002	22	1000
03	0003	23	0200
10	0010	30	0030
11	3000	31	0300
12	0100	32	0032
13	0013	33	2000

A kód tehát minden egy súlyú hibát javít, továbbá 3 kettő súlyút.

2.7. megoldás.

a)
$$\mathbf{H} = \begin{pmatrix} 12310 \\ 11101 \end{pmatrix}$$
$d = 3$, mivel van 3 súlyú kódszó, s a **H** mátrix oszlopai lineárisan függetlenek.

b) Igen. $4^3(1+3\cdot 5) = 4^5$

c) Két törlést tud javítani a kód: a kódszó $(1,3,1,3,3)$. Ilyen kisméretű iskolapéldánál egyszerű kombinálással is látható ez az eredmény a generátormátrix alapján. Egy megoldási technika x,y ismeretlenek bevezetése a törlések helyén, majd $\mathbf{H}\mathbf{c}^T = 0$ egyenlet alapján egy kétismeretlenes $\mathrm{GF}(4)$ feletti egyenletrendszer felírása:
$$1\cdot 1+2\cdot x+3\cdot 1+1\cdot 3+0\cdot y = 0$$
$$1\cdot 1+1\cdot x+1\cdot 1+0\cdot 3+1\cdot y = 0$$
ahonnan
$$2\cdot x = 1$$
$$y = x$$
s innen — az első egyenlet mindkét oldalát $2^{-1} = 3$-mal szorozva — az $x = 3, y = 3$ megoldást kapjuk.

d) A **G** mátrix alapján — egyszerű kombinálással — az alábbi bináris kódszavakat kapjuk:
$\{(00000),(10011),(01101),(11110)\}$

e) A bináris szavak halmaza nyilván zárt a bináris kombinációkra, így egy részkódot képez.

f) A kódszavak halmaza alapján: $n = 5, k = 2, d = 3$.

g) A $\mathbf{G}'$ generátormátrix a következő
$$\mathbf{G}' = \begin{pmatrix} 10011 \\ 01101 \end{pmatrix}.$$

2.8. megoldás. A megadott kódszavak lineárisan függetlenek, továbbá a kód dimenziója $k = 3$, ezért a három megadott kódszó kifeszíti a kódszavak terét:
$$\mathbf{G} = \begin{pmatrix} 10014 \\ 01012 \\ 00113 \end{pmatrix},$$
ahonnan
$$\mathbf{H} = \begin{pmatrix} 42310 \\ 11101 \end{pmatrix}.$$

H oszlopai lineárisan függetlenek, továbbá a Singleton-korlát alapján $d = 3$, tehát $t = 1$.

2.9. megoldás.

$$\mathbf{H} = \left(\begin{array}{c|c} 111111111111111000 & 100 \\ 000111122223333111 & 010 \\ 123012301230123123 & 001 \end{array}\right)$$

A **H** mátrix elemekkel való kitöltésének technikája jól követhető a fenti mátrixon (az 100 oszlop logikusan az első lenne, ezt azonban a szisztematikus generálás kedvéért a mátrix végén levő egységmátrixhoz használtuk fel, s hasonlóan a 010 oszlop a fenti mátrix 15. és 16. sora közé illeszkedik). Ez a technika egyrészt biztosítja, hogy lineráisan függtelenek legyenek az oszlopok, másrészt ezzel a módszerrel maximális számú páronként lineárisan független oszlopokat tartalmazó **H** mátrixot kapunk: a kitöltés logikája szerint ez a kódszóhossz $q^3+q+1 = (q^3-1)/(q-1) = 63/3 = 21$. GF$(q)$ feletti m soros **H** mátrix általános esetében a kódszóhossz: $q^m + q^{m-1} + \cdots + q + 1 = (q^m-1)/(q-1)$.

2.10. megoldás. A Singleton-korlát szerint $n-2+1 \geq 3$, azaz $n \geq 4$, tehát a legrövidebb hossz legalább 4. A

$$\mathbf{H} = \begin{pmatrix} 1110 \\ 2301 \end{pmatrix}$$

mátrix könnyen láthatóan megfelel a kívánságnak.

2.11. megoldás. A Singleton-korlát szerint $n-k+1 \geq 3$, azaz ez a korlát nem zárja ki a létezést. Az egyszerű ismétléses kód megfelelő, azaz a kódszavak halmaza: $\{(000),(111),(222)\}$. A mátrixok:

$$\mathbf{G} = \begin{pmatrix} 111 \end{pmatrix},$$

$$\mathbf{H} = \begin{pmatrix} 210 \\ 201 \end{pmatrix}.$$

2.12. megoldás. A $C(n,1,n)$ paraméterű bináris kód az ún. bináris ismétléses kód, amelynek két kódszava van: $\{(00...000),(11...111)\}$. A kód generátormátrixa $\mathbf{G} = (11...111)$
így

$$\mathbf{H} = \begin{pmatrix} 1000 \ldots 0001 \\ 0100 \ldots 0001 \\ 0010 \ldots 0001 \\ \vdots \quad \ddots \quad \vdots \\ 1000 \ldots 0011 \end{pmatrix}.$$

$n' = n$, $k' = n-k$, $d' = 2$. C' szavai **H** sorainak lineáris kombinációi (páros súlyú 4 hosszú bináris vektorok):
$\{(0000),(1100),(1010),(1001),(0110),(0101),(0011),(1111)\}$.

2.13. megoldás. $q = 2$-re a $\mathbf{H} = (\mathbf{P}\,|\,\mathbf{I})$ szisztematikus alakot tekintve, öndualitás esetén $\mathbf{G}\mathbf{H}^T = 0$ miatt $\mathbf{P}\mathbf{P}^T = -\mathbf{I}$ fenn kell álljon, azaz $\mathbf{P}$ csak négyzetes mátrix lehet. Ekkor $k = n-k$, azaz $k = n/2$. Bináris esetre példa

$$\mathbf{H} = \left(\begin{array}{1111|1000} 1101|0100 \\ 1011|0010 \\ 0111|0001 \end{array}\right).$$

$q > 2$, $2|n$: Nembináris esetben az $(n, n/2)$ paraméterű Reed–Solomon-kód standard $\mathbf{H}$ mátrixa alapján kaphatunk önduális kódot, ahol

$$\mathbf{H} = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \dots & \alpha^{2(n-1)} \\ \vdots & & & \ddots & \vdots \\ 1 & \alpha^{n/2} & \alpha^n & \dots & \alpha^{n/2(n-1)} \end{pmatrix}$$

2.14. megoldás. Igen. Bináris lináris kódok esetén igaz, hogy egy kódnak vagy minden szava páros súlyú (a 0 súlyt is ideértve), vagy a szavaknak pontosan fele páros, fele páratlan súlyú. Ez abból következik, hogy páros súlyú szavak bináris összege páros súlyú, páratlan súlyúak összege páratlan súlyú, míg páros és páratlan súlyúak összege páratlan súlyú. Ha tehát van egy páratlan súlyú szó, akkor ezt a szót a páros súlyúakhoz adva a páratlan súlyú szavak száma legalább akkora lesz mint a párosaké. Ugyanakkor ha egy páratlan súlyú szót adunk a páratlan súlyúakhoz — önmagához is — akkor láthatóan legalább annyi páros súlyú szónak kell lennie a kódban, mint páratlannak. Tehát a számuk egyforma.

2.15. megoldás. Igen. Írjuk gondolatban a kódszavakat egymás alá, táblázatszerűen. A $\mathbf{G}$ mátrix sorainak kombinációi a kódszavak. Tekintsük $\mathbf{G}$ első oszlopát. Ezen oszlop bitjei között van legalább egy egyes. Ha tekintjük az összes bináris kombinációt a $\mathbf{G}$ első oszlopában (azaz skalárszorzás a különböző bináris, oszlophosszú vektorokkal), akkor könnyen láthatóan pontosan az esetek felében 1, másik felében pedig 0 bit adódik. Ennek megfelelően a kódszavak táblázatának első oszlopában pontosan 2^{k-1} darab 1 bit és ugyanennyi 0 bit áll. Ugyanez a gondolatmenet igaz a többi $n-1$ oszlop vonatkozásában is.

2.16. megoldás. Bármely két oszlop linárisan független, ezért $d \geq 3$. A Singleton-korlát szerint $d \leq 3$, így $d = 3$.

2.17. megoldás. $d = 4$.

2.18. megoldás. A kérdéses sarok-bit, az üzenet bitjeinek paritása, amit számolhatunk soronkénti, vagy oszloponkénti paritások paritásaként is (más szavakkal, az üzenet bitjeinek modulo 2 összegét számítjuk ki kétféle sorrendben).

2.19. megoldás.

a) Legyen α a GF(q) primitív eleme:

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 & 1 & 0 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{q-2} & 0 & 1 \end{pmatrix}$$

b) Igen. $q^{q-1}(1+(q+1)(q-1)) = q^{q+1}$

c)

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & \alpha \\ 0 & 0 & 1 & 1 & \alpha^2 \end{pmatrix}$$

2.20. megoldás. Nem igaz. Azt kell észrevenni, hogy egy nembináris páratlan súlyú kódszó elemeinek összege lehet 0. Például $C(4,2)$ GF(3) feletti kódot tekintve az alábbi mátrixokkal:

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{pmatrix}$$

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 2 & 2 \\ 0 & 1 & 2 & 1 \end{pmatrix}$$

Például az $(1,1)$ üzenethez tartozó $(1,1,1,0)$ kódszóhoz 0 paritáskarakter tartozik, azaz a minimális kódszósúly 3 marad.

2.21. megoldás. Nem. A disztributivitás nem teljesül: $(1+1)\cdot 2 = 3 \neq 1\cdot 2 + 1\cdot 2 = 0$.

2.22. megoldás. Mivel $4 = 2^2$, ezért bináris másodfokú irreducíbilis polinomot választunk a modulo polinom aritmetika generálásához. Ez a polinom $f(x) = x^2 + x + 1$. A testelemek elnevezése 0,1,2,3, ahol a következő megfeleltetést választjuk: $0 \leftrightarrow 0$, $1 \leftrightarrow 1$, $2 \leftrightarrow x$, $3 \leftrightarrow x+1$.
Például: $2 \cdot 3 = x(x+1) = x^2 + x = 1 \mod f(x)$.

2.24. megoldás.

a) $x = 1$, $y = 1$.

b) A harmadik sor szerint kifejtve a determinánst: $D = 0 + 3 = 3$.

2.25. megoldás. $(x+1)x^2 = 1 \mod x^3 + x^2 + 1$, ezért $(x+1)^{-1} = x^2$.

2.26. megoldás.

a) Ha $p(x)$ GF(3) feletti kisebb fokszámú faktorokra lenne bontható, akkor lenne azok között elsőfokú, tehát 0,1,2 $GF(3)$-beli elemek közül kellene, hogy legyen gyöke. De
$p(0) = 0+0+2 \neq 0 \pmod 3$
$p(1) = 1+1+2 \neq 0 \pmod 3$
$p(2) = 8+4+2 \neq 0 \pmod 3$

b) $d|26 \Longrightarrow 1,2,13,26$

c) $(2x+1)(x^2+2) = 2x^3+4x+x^2+2 = 2x^2+x+1$

2.29. megoldás. Nem, mivel ezen kódszó és egylépéses balra forgatottja közötti Hamming-távolság csak 2 lenne.

2.30. megoldás.

a) A részkód mérete 8, mivel ha egy lineáris kódban van páratlan súlyú kódszó, akkor a kódszavak fele páratlan súlyú

b) $d = 4$, mivel a csupa 1 szó eleme a kódnak, ugyanis $(x^7-1)/(x-1) = (x^3+x^2+1)(x^3+x+1)$, így a részkódnak is , s az alapkódban emiatt nem lehet 6 súlyú szó, következésképp a részkódnak csak 7 darab 4 súlyú és 1 darab 0 súlyú eleme van, s két 4 súlyú szó különbsége esetünkben csak 4 lehet.

2.31. megoldás.

a) a szisztematikus kódszógenerálás technikája alapján

$$\mathbf{G} = \left(\begin{array}{c|c} 1000 & 101 \\ 0100 & 111 \\ 0010 & 110 \\ 0001 & 011 \end{array}\right)$$

a szisztematikus generátormátrix, s ennek alapján a 16 kódszó

0000000	1000101
0001011	1001110
0010110	1010011
0011101	1011000
0100111	1100010
0101100	1101001
0110001	1110100
0111010	1111111

b) a kódszavak halmaza alapján a minimális nemzérus kódszósúly 3

c) $x^7-1=(x-1)(x^3+x^2+1)(x^3+x+1)$, ezért $h(x)=(x^7-1)/g(x)=(x-1)(x^3+x^2+1)=x^4+x^2+x+1$

2.32. megoldás.

a) súlyok $=0,2,4,6$, $k=6$, $d=2$

b) $h(x)=(x^7-1)/(x-1)=x^6+x^5+x^4+x^3+x^2+x+1$

c) $\mathbf{H}=(111111)$

2.33. megoldás. A kódszavak száma: $|C|=2^k=2$.
A két kódszó $\{(0000000),(1111111)\}$.

2.34. megoldás.

a) A választ az x^7-1 polinom valódi, nemzérus fokszámú bináris osztóinak száma adja, s mivel $x^7-1=(x-1)(x^3+x^2+1)(x^3+x+1)$, ezért a válasz 6.

b) $C_1(7,1,6)$: $g_1(x)=(x^7-1)/(x-1)=x^6+x^5+x^4+x^3+x^2+x+1$
$C_2(7,4,3)$: $g_2(x)=x^3+x+1$ (Hamming-kód)
$C_3(7,4,3)$: $g_3(x)=x^3+x^2+1$ (Hamming-kód)
$C_4(7,3,4)$: $g_4(x)=(x-1)(x^3+x^2+1)$ (páros súlyú részkód)
$C_5(7,3,4)$: $g_5(x)=(x-1)(x^3+x+1)$ (páros súlyú részkód)
$C_6(7,6,2)$: $g_6(x)=(x-1)$

2.37. megoldás.

a) $C_1\subseteq C_2$

b) $d_1\geq d_2$

2.38. megoldás.

a) $h_1(x)\mid h_2(x)$

b) $g_2(x)\mid g_1(x)$

2.40. megoldás. Igen, $v(x)=x^3+1 \mod g(x)$.

2.41. megoldás.

a) Igen, mivel $h(x)\mid x^n-1$.

b) Nem, mivel a Singleton-korlát miatt C' minimális távolsága legfeljebb $k+1$.

2.42. megoldás. Nem, ezt láthatjuk, ha ellenőrizzük a gömbi kitöltést:
$2^5(1+11+11\cdot 10/2)>2^{11}$.

2.43. megoldás. Nem, az állítás ellentmond a Singleton-korlátnak.

2.44. megoldás. Igen. Tekintsük a (000), (111), (222) szavakból álló ismétléses kódot. A minimális távolság nyilván 3. A szisztematikus paritásellenőrző mátrix: $\mathbf{H} = (111)$.

2.45. megoldás. Igen, mivel teljesül a Hamming-korlát szerinti egyenlőség: $2(1+7+21+35) = 2^7$.

2.46. megoldás. Igen, mivel teljesül a Hamming-korlát szerinti egyenlőség: $3^6(1+11\cdot 2+(11\cdot 10/2)\cdot 2^2) = 3^{11}$

2.47. megoldás. A generátorpolinom gyökhalmaza alapján: $d = 5$, $t_c = 2$, $t_d = 4$, $t_e = 4$.

2.49. megoldás. Nem, mivel $g(1) = 0$, de a $(x^{q-1}-1)/(x-1) = x^{n-1}+x^{n-2}+\cdots+x+1$ polinomnak az 1 testelem nem gyöke.

2.50. megoldás. Nem lehetséges. A kódszavak elemeinek összege zérus, ekvivalens azzal, hogy $c(1) = 0$, tetszőleges $\mathbf{c}$ kódszóra. Mivel a kód t hibát javít, ezért az α primitív elemnek vagy $1,2,\ldots,2t$ vagy a $2,3,\ldots,2t,2t+1$ hatványai — azaz $2t$ egymást követő kitevőjű hatvány — a generátorpolinom gyökei. Mindkét esetben igaz, hogy $g(1) \neq 0$, azaz α primitív elem 0 kitevőjű hatványa nem gyöke a generátorpolinomnak, következésképp nem lehetséges, hogy $c(1) = 0$ fenállhasson, hiszen egy kódszó a generátorpolinom többszöröse.

2.51. megoldás. Igaz. Az $\mathbf{a} = \left(1,\alpha^i,\alpha^{2i},\ldots,\alpha^{(n-1)i}\right)$ és $\mathbf{b} = \left(1,\alpha^j,\alpha^{2j},\ldots,\alpha^{(n-1)j}\right)$ vektorok ortogonálisak, azaz

$$\mathbf{ab} = \prod_{k=0}^{n-1} \alpha^{(i+j)k} = 0$$

ha $i+j \neq 0 \mod n$, ami esetünkben teljesül.

2.52. megoldás. Van. Ha n páratlan szám, akkor $\boldsymbol{\alpha}^{(r)} = \left(1,\alpha^r,\alpha^{2r},\ldots,\alpha^{(n-1)r}\right)$, $r = 0,1,\ldots,n-2$ vektorok lehetnek mind $\mathbf{G}$ és $\mathbf{H}$ mátrix sorai az előző feladat megoldása értelmében. Ha például $q = 2^m$ alakú, akkor az elemek rendje mindig páratlan szám.

2.53. megoldás. A C' Reed–Solomon-kód $g'(x)$ generátorpolinomjának gyökei α, α^2 ahol $\alpha \in GF(8)$ primitív eleme. A C Hamming-kód $g(x)$ generátorpolinomjának gyökei pedig α,α^2,α^4. Így $c(x) = (x-\alpha^4)g'(x) = g(x)$ egyben a C' kód egy kódszava, azaz C minden kódszava eleme a C' kódnak.

2.54. megoldás. Igaz. Ugyanis a Reed–Solomon-kód MDS tulajdonságú, s így ha valamely k elemű koordinátahalmazon két kódszó azonos elemeket tartalmazna, akkor a különbségük súlya legfeljebb $n-k$ lehetne, ami ellentmodana annak, hogy a minimális távolság $n-k+1$.

2.55. megoldás. Mivel egy bináris kódban vagy minden szó páros súlyú vagy pontosan a szavak fele az, s a Hamming-kódban van páratlan súlyú kódszó, ezért C' dimenziója $k' = 3$. A szisztematikus kódszógenerálás technikája alapján

$$\mathbf{G} = \left(\begin{array}{c|c} 1000 & 101 \\ 0100 & 111 \\ 0010 & 110 \\ 0001 & 011 \end{array}\right)$$

a szisztematikus generátormátrix, amelynek 2. sora, a 3. és 4. sor összege, valamint az 1. és 3. sor összege páros súlyú. Ezen vektorok kifeszítik a keresett részkódot, így a részkód egy generátormátrixa

$$\mathbf{G}' = \left(\begin{array}{c|c} 0100 & 111 \\ 0010 & 110 \\ 0001 & 011 \end{array}\right),$$

C' kódszavai: $0000000, 1001110, 1010011, 0011101, 1011111, 0100111, 1110100,$ 0111010. C' paraméterei: $n' = 7$, $k' = 3$, $d' = 4$.

2.56. megoldás.

a) Nem: $2^3(1+6) < 2^6$.

b) Nem: $2^2(1+5) < 2^5$.

2.57. megoldás. $n = 8$, $k = 4$, $d = 4$

2.59. megoldás.

a) $n = 4$, $k = 1$

b) A rövidített kódot a generátorpolinomnak megfelelő (0001101) szó generálja, azaz a kódszavak $(0000000), (0001101)$, így $d = 3$.

2.61. megoldás.

a) Nem, mivel a C' kódnak nem eleme a csupa zérus kódszó.

b) $n' = n$, $k' = k$, $d' = d$, mivel C' két különböző nemzérus szavának különbsége C szava, s C minden nemzérus szava így előállítható.

c) Igen. Bármely két kódszó összege is kódszó, azaz vagy C vagy C' eleme.

d) A C és C' diszjunkt halmazok, hiszen ha C' egy $\mathbf{c} \oplus \mathbf{1}$ eleme C eleme lenne, ahol $\mathbf{c} \in C$, akkor nyilván $\mathbf{1} \in C$ is fennnállna, ugyanis $(\mathbf{c} \oplus \mathbf{1}) \oplus \mathbf{c} = \mathbf{1}$, ami ellentmondás. Így $n' = n$, $k' = k+1$, $d' = \min\{d, n-d\}$.

2.62. megoldás. Azon szavak részhalmazát tekintve, ahol az $n+1$-edik karakter 0, a szavak polinom alakjának az $1 (= \alpha^0)$ testelem is gyöke, tehát ezen részhalmazbeli szavak minimális súlya legalább $d+1$. Azon szavak súlya pedig, ahol ahol az $n+1$-edik karakter nem 0 nyilván szintén legalább $d+1$.

2.64. megoldás. Mivel $g(x)$ — mint generátorpolinom — a C kód egyben legkisebb fokszámú nemzérus főpolinom kódszava, ezért a $c^{(1)}(x) = g(x), c^{(2)}(x) = 0, \ldots, c^{(m)}(x) = 0$ sorozat átfűzésével kapható kódszó polinom alakban $g(x^m)$, a C^m kód legkisebb fokszámú nemzérus főpolinom kódszava, azaz generátorpolinomja.

2.65. megoldás.

a) $n = 10$, $k = 9$, $d = 3$

b) Alkalmazzuk a 2.6. pontban megadott módszert.

2.66. megoldás. Igaza van. Közvetlenül megadhatjuk a dekódolás menetét két hiba javítása esetére. A hibapolinom $e(x) = x^{i_1} + x^{i_2}$, ahol $0 \leq i_1 < i_2 \leq n-1$ a hibapozíciók. Felírva a szindrómákra vonatkozó két egyenletből álló egyenletrendszert:

$$S_1 = e(\alpha) = X_1 + X_2,$$
$$S_{-1} = e(\alpha^{-1}) = X_1^{-1} + X_2^{-1},$$

ahol $X_1 = \alpha^{i_1}$, $X_2 = \alpha^{i_2}$ az úgynevezett hibahelycímkék. A második egyenletet X_1X_2-vel beszorozva:

$$X_1 + X_2 = S_1,$$
$$X_1X_2 = S_1/S_{-1},$$

egyenletrendszert kapjuk X_1 és X_2 ismeretlenekben. Az

$$(x - X_1)(x - X_2) = x^2 + (X_1 + X_2)x + X_1X_2 = x^2 + S_1x + S_1/S_{-1}$$

másodfokú egyenletre jutunk, amelynek a gyökei a keresett hibahelycímkék.

2.67. megoldás. Nincs igaza. Az ugyan igaz, hogy 2^{n-k} javítható hibamintát tudunk felmutatni, de a dekódolási hibavalószínűség úgy minimálizálható, ha a javítható hibaminták súlyai az 1 súlyoktól kezdve súly szerint folyamatosan lefedik a hibamintákat, azaz az egy súlyúakat, majd a két súlyúakat, s.í.t.

2.68. megoldás. Az a) válasz az igaz, mivel a vett szóhoz tartozó szindróma alapján választjuk a javító hibamintát, azaz mindig zérusra korrigáljuk a szindrómát, tehát mindig kódszó az eredmény.

2.69. megoldás. Bájt karakteres Reed–Solomon-kódot választva három, egyenként egy bájt hibát javító kódot átfűzünk, mivel 16 bites hibacsomó legrosszabb esetben 3 egymást követő bájt hibáját okozza. A komponenskódok paramétere $C(255, 253)$, s ezzel a kérdéses optimális sebesség $R = 253/255 = 0.992$.

2.71. megoldás. Igen. Ugyanis egy m bites hibacsomó, amely az $r, r+1, \ldots, r+m-1$ bitpozíciókat foglalja el, nyilván felírható $b(x)x^r$ alakban, ahol $b(x)$ egy $m-1$ fokszámú bináris polinom. De $b(x)x^r$ polinom modulo $f(x)$ maradéka nem lehet zérus, mivel $f(x)$ tetszőleges polinomszorosa legalább $m+1$ bites „hibacsomagnak” felel meg.

2.72. megoldás. $x^{17}+x^{16}$ mod $g(x)$ maradékot kell meghatározni. Így a keresett 16 bites CRC: 00110000|01100011.

2.73. megoldás. Használjuk a (2.10) formulát: $P_e(n,t,(q-1)p)$.

2.74. megoldás. A kód perfekt, ezért a (2.10) formula felhasználásával egyszerűen számítható a dekódolási hibavalószínűség: $P_e = 1-((1-p)^{15}+15p(1-p)^{14})$.

2.75. megoldás. A $g(x) = x^3+x+1$ bináris generátorpolinomú Hamming-kód kódszavai

0000000	1000101
0001011	1001110
0010110	1010011
0011101	1011000
0100111	1100010
0101100	1101001
0110001	1110100
0111010	1111111

Ennek alapján a paritásbittel kibővített kód súlyeloszlása:

súly	darab
4	14
8	1

Így $P_e = 14p^4(1-p)^4+p^8$.

2.76. megoldás.

a) Nyilván, ha egy pozíció is marad, törlés nélkül felismerhető a kódszó, lévén hogy a két kódszó a csupa 0 illetve a csupa 1. Ez megfelel a $d-1=n-1$ törlésjavító képességnek. Így $P_e = 1/2\cdot q^n$.

b) A javítóképesség $t=\lfloor(n-1)/2\rfloor$. Ha n páratlan, akkor a (2.10) képlet alapján számolhatjuk a hibavalószínűséget. Ha n páros, akkor $t'=\lfloor n/2\rfloor$ behelyettesésével a (2.10) képletbe felső becslést kapunk, mivel ekkor $n/2$ súlyú hiba esetén egyenlő távolságra vagyunk a két kódszótól.

2.77. megoldás. A (2.10) képletet használva a javulás:

$$P_e(5,2,0.01)-P_e(3,1,0.01).$$

2.78. megoldás.

a)

súly	darab
0	1
4	9
6	6

b) $P_e = 9p^4(1-p)^5 + 6p^6(1-p)^3$

2.80. megoldás. Mivel $d = 4$, ezért ha a törléses hibák száma legfeljebb 3, azt biztosan javítani tudjuk. Ezért a (2.10) képlet $t = 3$ helyettesítéssel felső becslést ad a dekódolási hibavalószínűségre. Innen $P_e \leq 0.000372$ adódik, azaz elfogadhatóan választottuk a kódot.

2.81. megoldás.

a) $q = \mathbf{P}\{\text{20 bájt hibátlan vétele}\} = (1-p)^{160}$,
$P_{\text{ism}}(q) = 0 \cdot q + 1 \cdot (1-q)q + 2 \cdot (1-q)^2 q + \ldots = (1-q)/q \simeq 0.174$.

b) $q' = q \cdot q_{ACK}$, $q_{ACK} = \mathbf{P}\{\text{3 bájt hibátlan vétele}\} = (1-p)^{24} = 0.976$,
$P_{\text{ism}}(q') = 0.202$.

2.82. megoldás. $q = \mathbf{P}\{N \text{ bájt hibátlan vétele}\} = (1-p)^{8N}$, a késleltetés várható értéke:
$2Tq + 4T(1-q)q + 6T(1-q)^2 q + \ldots = 2T/q$.

2.12. Összefoglalás

Ebben a fejezetben a hibakontroll kódolás alapelveit, alapvető konstrukcióit és alkalmazási területeit tekintettük át. [2, 5, 6, 7, 9, 12, 13, 15, 17] Ezen kódolás célja az, hogy hibázó csatornán keresztül is megbízhatóan tudjunk üzeneteket küldeni.

A célok és alapfogalmak tisztázása után először a hibakontroll blokk-kódok legfontosabb osztályát, a lineáris kódokat vezettük be, amelynél a kódszavak halmaza egy egyszerű algebrai struktúrába, egy lineáris térbe rendeződik. Ezzel mind a kód generálása, mind a dekódolás feladata lényegesen egyszerűsödik az általános nemlineáris kódokhoz képest. A bináris kód általánosítása a tetszőleges véges test feletti kód, amelynek felhasználását először egy egyszerű lináris kódkonstukción, a továbbiakban a Reed–Solomon-kódokon szemléltettük. A ciklikus kódok bevezetése polinomalgebrai eszköztár felhasználásával még hatékonyabb kódok konstrukcióját teszi lehetővé. Egy kód hibakontroll képességei vonatkozásában fontos a kódtávolság minél nagyobb értéke, de egy kód csak akkor lesz felhasználható a gyakorlatban, ha hatékony dekódolási algoritmust is tudunk adni hozzá. Az alapvető dekódolási módszerekbe betekintést nyújtottunk, ahol a legfontosabb módszerünk a szindróma-dekódolás táblázatos illetve algoritmikus módja volt. Ismert kódokból mint építőelemekből újabb, adott feladatra jobb kódot készíthetünk: ilyen technikákat mutattunk a kódkombinációk alfejezetben. Számos fontos gyakorlati alkalmazás [6, 8, 12] és nagy számú kidolgozott feladat zárta a fejezetet.

Irodalomjegyzék

[1] Bahl, L.R., Cocke, J., Jelinek, F., Raviv, J. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, IT-20:248–287, 1974.

[2] Berlekamp, E.R. *Algebraic Coding Theory*. McGraw Hill, 1968.

[3] Berlekamp, E.R. The technology of error-correcting codes. *Proceedings of the IEEE*, 68, May, 1980.

[4] Berlekamp, E.R., Peile, R.E., Pope, S.P. The application of error control to communications. *IEEE Communications Magazine*, 25, Apr, 1987.

[5] Blahut, R.E. *Theory and Practice of Error Control Codes*. Addison-Wesley, 1983.

[6] Csibi S. (szerk.) *Információ közlése és feldolgozása*. Tankönyvkiadó, Budapest, 1986.

[7] Csiszár I., Körner J. *Information Theory: Coding Theorems for Discrete Memoryless Systems*. Akadémiai Kiadó, Budapest, 1981.

[8] Doi, T. *Error Correction for Digital Audio Recordings, No. 1991 AES 73. Convention*. Eindhoven, 1983.

[9] Gallager, R.G. *Information Theory and Reliable Communication*. Wiley, 1968.

[10] Géher K. (szerk.) *Híradástechnika*. Műszaki Könyvkiadó, Budapest, 1993.

[11] Györfi L., Vajda I. *A hibajavító kódolás és a nyilvános kulcsú titkosítás elemei*. Műegyetemi Kiadó, Budapest, 1990.

[12] Györfi L., Győri S., Vajda I. *Információ- és kódelmélet*. TypoTEX Kiadó, Budapest, 2002.

[13] MacWilliams, F.J., Sloane, N.J.A. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

[14] Massey, J.L. *Applied Digital Information Theory*. Course Notes, ETH Zürich, 1984.

[15] McEliece, R.J. *The Theory of Information and Coding*. Addison-Wesley, 1977.

[16] Niven, I., Zuckerman, H.S. *Bevezetés a számelméletbe*. Műszaki Könyvkiadó, Budapest, 1978.

[17] Peterson, W.W. *Error Correcting Codes*. MIT Press – Wiley, Cambridge, 1961.

[18] Vajda I. *Hibajavító kódolás és műszaki alkalmazásai*. BME Mérnöki Továbbképző Intézet, Budapest, 1982.

3. fejezet

Kriptográfia

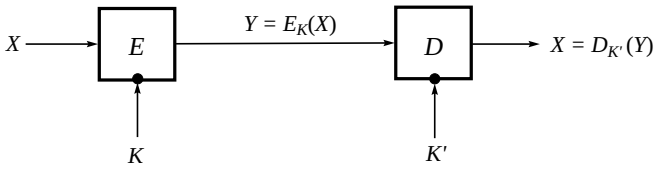
A kriptográfia azon algoritmikus módszerekkel foglalkozik, amelyek biztosítják a kommunikáció biztonságát. Itt biztonság alatt a szándékos támadásokkal szembeni ellenállóképességet értjük. A támadások lehetnek passzívak vagy aktívak. Passzív támadás esetén a támadó nem avatkozik be észrevehető módon a rendszer működésébe. Ennek tipikus példája a kommunikáció lehallgatása, és ily módon az átvitt üzenetek tartalmához történő illetéktelen hozzáférés. Ezzel szemben, aktív támadás esetén a támadó észrevehető módosításokat végez a rendszerben, például az átvitt üzeneteket módosítja, esetleg törli, vagy csalárd módon üzeneteket fabrikál más nevében. Jellegüknél fogva, a passzív támadásokat nehéz detektálni, míg az aktív támadásokat nehéz megakadályozni. Ezért a kriptográfiai módszerek a passzív támadások megelőzésére és az aktív támadások detektálására törekszenek.

A kriptográfia segítségével megvalósítható főbb biztonsági szolgáltatások a következők:

- **Titkosítás:** A titkosítás során az üzeneteket oly módon kódoljuk (rejtjelezzük), hogy az üzenet megértéséhez szükséges dekódolást csak az arra illetékes felek tudják elvégezni. Így az üzenet tartalmához egy támadó nem tud hozzáférni.

- **Integritásvédelem:** Az integritásvédelem feladata az, hogy detektálhatóvá tegye az üzenetek szándékos módosítását. Ezt tipikusan úgy érjük el, hogy az üzenetet egy kriptográfiai ellenőrzőösszeggel egészítjük ki. Ez hasonlít a hibadetektáló kódolás során alkalmazott ellenőrzőösszeghez (pl. CRC), ám azzal ellentétben a kriptográfiai ellenőrzőösszeget csak az arra illetékes felek tudják kiszámolni. Így az üzenet tartalmát egy támadó nem tudja észrevétlenül módosítani.

- **Hitelesítés:** A hitelesítés lehetővé teszi egy üzenet küldőjének megbízható azonosítását. Ezen szolgáltatás segítségével tehát detektálható ha egy támadó más nevében próbál üzenetet küldeni.



3.1. ábra. A rejtjelezés klasszikus modellje

- **Letagadhatatlanság:** A letagadhatatlanság szolgáltatás segítségével elérhető, hogy egy üzenet küldője később ne tudja letagadni, hogy ő küldte az üzenetet.

A továbbiakban a fenti biztonsági szolgáltatások megvalósításához szükséges kriptográfiai technikákat mutatjuk be, majd ezek gyakorlati alkalmazásait szemléltetjük a valós életből vett példákon.

3.1. Rejtjelezési technikák

Alapfogalmak

A rejtjelezés egy olyan kódolási transzformáció, melynek segítségével az üzenet küldője az üzenet tartalmát érthetetlenné teszi egy illetéktelen lehallgató számára. A rejtjelezés klasszikus modelljét a 3.1. ábra szemlélteti. Ennek segítségével vezetjük be a rejtjelezéssel kapcsolatos alapfogalmakat.

Az elküldendő X üzenetet **nyílt szöveg**nek nevezzük. A küldő a nyílt szövegen alkalmazza a rejtjelezés $E_K(.)$ **kódoló transzformáció**ját, melynek eredményeként előáll az $Y = E_K(X)$ **rejtjeles szöveg**. A küldő továbbítja a rejtjeles szövegetet a nem biztonságos csatornán. A vevő a $D_{K'}(.)$ **dekodóló transzformáció**t használja az eredeti nyílt szöveg visszaállítására: $X = D_{K'}(Y)$. Mind a kódoló, mind a dekódoló transzformációnak két bemente van: az X nyílt illetve az Y rejtjeles szöveg, valamint a K kódoló illetve a K' dekódoló **kulcs**, melyeket a transzformációk indexében szokás feltüntetni. Fontos, hogy a dekódoló kulcs értéke titkos, azt illetéktelenek nem ismerik. Mivel a csatorna nem biztonságos, ezért egy támadó le tudja hallgatni az Y rejtjeles szöveget, ám a K' dekódoló kulcs ismeretének hiányában nem tudja abból előállítani a nyílt szöveget.

A rejtjelezésnek több fajtája létezik. Ha a kódoló és a dekódoló kulcsok megegyeznek (azaz $K' = K$), vagy az egyik a másikból könnyen számítható, akkor **szimmetrikus kulcsú rejtjelezés**ről beszélünk. Ha a kódoló és a dekódoló kulcs nem azonos (azaz $K' \neq K$), és egymásból nehezen számolhatók, akkor **aszimmetrikus kulcsú rejtjelezés**ről beszélünk.

Szimmetrikus kulcsú rejtjelezés esetén a küldőnek és a vevőnek meg kell egyeznie a K kódoló/dekódoló kulcsban. Ehhez valamilyen biztonságos csatornát kell használniuk, hiszen ellenkező esetben egy támadó is hozzájuthat a kulcshoz, s azzal dekódolni tudja a rejtjeles üzeneteket. A gyakorlatban, a kulcsot vagy egy fizikai találkozó során hozzák létre és telepítik a felek, vagy egy már korábban telepített kulcs segítségével, rejtjelezett formában közlik egymással a kulcs értékét. Ezek a módszerek azonban nem minden alkalmazásban használhatók. Ezzel szemben, az aszimmetrikus kulcsú rejtjelezés előnye, hogy a küldőnek és a vevőnek nem kell egy közös titokban megegyeznie a kommunikáció előtt; elegendő, ha a vevő nyilvánosságra hozza a kódoló kulcsát. A K kódoló kulcs segítségével bárki képes rejtjeles üzeneteket előállítani a vevő számára, ám azokat csak a vevő képes dekódolni a K' titkos dekódoló kulcsával. Ez a működési elv megkönnyíti a kulcscsere probléma megoldását. Mivel az aszimmetrikus kulcsú rejtjelezés esetében a kódoló kulcs nyilvános, ezt a fajta rejtjelezést szokás **nyilvános kulcsú rejtjelezés**nek is nevezni.

A szimmetrikus kulcsú rejtjelezőket szokás tovább osztályozni **kulcsfolyam rejtjelező**kre és **blokkrejtjelező**kre. A kulcsfolyam rejtjelezők a nyílt szöveget karakterenként dolgozzák fel. A karakterhossz változó, tipikus értéke 8 bit vagy 1 bit. A kulcsfolyam rejtjelező lelke a kulcsfolyam generátor, mely egy valódi véletlen vagy egy véletlennek tűnő, ún. álvéletlen karaktersorozatot állít elő. Ezt a sorozatot kulcsfolyamnak nevezzük. A kódolás úgy történik, hogy a küldő a kulcsfolyam karaktereit bitenként XOR-olja a nyílt szöveg karaktereivel, és így nyeri a rejtjeles szöveg karaktereit. A dekódolás hasonlóképpen történik: a vevő előállítja ugyanazt a kulcsfolyamot amit a küldő használt a kódolás során, majd ezt bitenként a rejtjeles szöveghez XOR-olja. Az XOR művelet tulajdonságai miatt így pont a nyílt szöveget kapja vissza. Valódi véletlen kulcsfolyam használata esetén, a kulcs maga a kódoláshoz és a dekódoláshoz használt kulcsfolyam. Álvéletlen kulcsfolyam használata esetén, a kulcs egy kis méretű (tipikusan 128 bites) titok, melyből egy álvéletlen generátor mind a küldő, mind a vevő oldalán ugyanazt az álvéletlen kulcsfolyamot állítja elő.

A blokkrejtjelezők ezzel szemben hosszabb blokkokban dolgozzák fel a nyílt szöveget. Tipikus blokkméret a 64 bit vagy a 128 bit. A küldő kódolás előtt a nyílt szöveget blokkokra osztja, majd a blokkokat kódolja a blokkrejtjelező segítségével, így kapja a rejtjeles szöveg blokkjait. A rejtjeles blokkok hossza megegyezik a nyílt blokkok hosszával. A dekódolás során a vevő a blokkrejtjelező inverzét használja a nyílt blokkok rejtjeles blokkokból történő visszaállításához. A blokkrejtjelező és annak inverze bemenetként kapja a kódoló illetve a dekódoló kulcsot, melynek tipikus mérete 128 bit.

A rejtjelezők biztonságát különböző **támadómodellek**ben szokták vizsgálni. Ezek a következők:

- **Kizárólag rejtjeles szövegekre épülő támadás (ciphertext only attack):** Ebben a modellben a támadónak csak rejtjeles szövegek állnak a rendelkezésére, melyek mind ugyanazzal a kulccsal lettek előállítva. Tipikusan ez

a helyzet, mikor a támadó csak lehallgatni képes a küldő és a vevő közötti csatornát.

- **Ismert nyílt szöveg – rejtjeles szöveg párokra épülő támadás (known plaintext attack):** Ebben a modellben azt feltételezzük, hogy a támadó hozzá tud jutni egymáshoz tartozó nyílt szöveg – rejtett szöveg párokhoz, ahol minden pár esetében ugyanazt a kulcsot használta a küldő a rejtett szöveg előállításához. Bizonyos esetekben, ilyen párokhoz egyszerű lehallgatással is hozzájuthat a támadó (pl. egy kihívás–válasz típusú partnerhitelesítő protokoll használata esetén).

- **Választott nyílt szövegekre épülő támadás (chosen plaintext attack):** Ez már egy erősebb támadó modell abban az értelemben, hogy ebben a modellben feltételezzük, hogy a támadó nemcsak lehallgatni képes a csatornát, hanem valamilyen módon rá tudja venni a küldőt, hogy bármely a támadó által választott nyílt szöveget rejtjelezze. Azt szokták mondani, hogy a támadó rejtjelező **orákulum**ként használja a küldőt. Ezen modell egyszerű változata az, mikor a támadó csak egyszer hívhatja meg az orákulumot, és ezzel a hívással az összes kódoltatni kívánt nyílt szöveget egyszerre átadja, majd az összes rejtett szöveget egyszerre megkapja. Ennek a modellnek létezik egy adaptív változata is, amikor a támadó többször hívhatja meg az orákulomot, s minden hívásnál adaptívan választhatja a kódoltatni kívánt nyílt szöveget az előző hívások eredményeit is figyelembe véve. A gyakorlatban, rejtjelezést végző reaktív eszközök használhatók rejtjelező orákulumként. Tipikus példa egy partnerhitelesítést végző szerver vagy intelligens chip-kártya.

- **Választott rejtjeles szövegekre épülő támadás (chosen ciphertext attack):** Ez a modell nagyban hasonlít az előzőhöz azzal a különbséggel, hogy itt a támadó egy dekódoló orákulumot használ, s ennek ad át dekódoltatni kívánt rejtjeles szövegeket. Ennek a modellnek is létezik egyszerű és adaptív változata is. A gyakorlatban, dekódolást végző reaktív eszközök használhatók dekódoló orákulumként.

- **Összefüggő kulcsokra épülő támadás (related keys attack):** Ez a modell abban különbözik az előzőektől, hogy itt a támadó olyan rejtett szövegeket, vagy nyílt szöveg – rejtett szöveg párokat használ a támadásban, melyek mind különböző kulccsal lettek előállítva, ám feltételezzük, hogy ezen kulcsok között létezik valamilyen kapcsolat, és ez a támadó számára is ismert. Tipikus példa mikor a támadó ismeri a kulcsok XOR különbségét.

Minden fent említett modellben a támadó célja a kulcs (összefüggő kulcsokra épülő támadás esetén a kulcsok) megfejtése. Minden modell feltételezi továbbá, hogy a kódoló és a dekódoló transzformáció működése (maga az algoritmus) ismert a támadó számára. Ez utóbbi feltételezést **Kerckhoff-elv**nek[1] nevezik.

[1] Auguste Kerckhoff francia kriptográfus után, aki ezen feltevés gyakorlati fontosságát már 1883-ban hangsúlyozta *La cryptographie militaire* című tanulmányában.

Egy rejtjelező biztonságos egy adott támadómodellben, ha az adott modell által megengedett képességekkel rendelkező támadó nem képes a rejtjelezőt feltörni, azaz hatékony módszert találni a kulcs szisztematikus megfejtésére. Attól függően, hogy hogyan értelmezzük a „nem képes" kifejezést, kétféle biztonság-fogalmat különböztethetünk meg: a **feltétel nélküli** és a **feltételes**, vagy **algoritmikus biztonság**ot. A feltétel nélküli biztonság azt jelenti, hogy a támadó tetszőleges nagyságú erőforrás mellett sem képes feltörni a rejtjelezőt. Ezzel szemben, algoritmikus biztonság esetén a rejtjelező csak addig nem törhető fel, amíg a támadó erőforrása egy megadott korlát alá esik. Erőforrás alatt itt a tárkapacitást, a futási időt, vagy a támadáshoz szükséges adatok mennyiségét (pl. nyílt szöveg – rejtett szöveg párok számát), illetve ezek együttesét értjük.

Talán meglepő, de léteznek feltétel nélkül biztonságos rejtjelezők. Ilyen például a **one-time pad (OTP)**, melyet később részletesen tárgyalunk. A legtöbb gyakorlatban használt rejtjelező azonban nem feltétel nélkül biztonságos. Ezen rejtjelezők biztonságát úgy szokták jellemezni, hogy a jelenleg ismert támadási módszerekkel és a ma tipikusan feltételezhető erőforásokkal nem törhetők fel. Előfordulhat azonban, hogy felfedeznek egy új támadási módszert, vagy drasztikus változás következik be a számítógép-technológiában (pl. megépítik a gyakorlatban is használható kvantumszámítógépet), s egy korábban feltételesen biztonságosnak tartott rejtjelező az új körülmények között feltörhetővé válik.

Végül megjegyezzük, hogy minden algoritmikusan biztonságos rejtjelező ki van téve a **kimerítő kulcskeresés** támadásnak. Ez a támadás abból áll, hogy a támadó az összes lehetséges kulcsot kipróbálja mindaddig amíg az éppen használt kulcsot meg nem találja. A csak rejtjeles szövegek ismeretét feltételező támadómodellben ez azt jelenti, hogy a támadó addig próbálkozik, míg meg nem találja azt a kulcsot, mely minden rendelkezésre álló rejtett szöveget értelmes nyílt szövegbe dekódol. Hasonlóképpen, az ismert nyílt szöveg – rejtett szöveg párokat feltételező modellben, a támadó addig próbálkozik, míg meg nem találja azt a kulcsot, amelyik minden pár nyílt szövegét az adott pár rejtjeles szövegébe kódolja. Ahhoz, hogy a kimertő kulcskeresés támadás hatástalan legyen (azaz a rejtjelező algoritmikusan biztonságos maradjon) a lehetséges kulcsok számának, más szavakkal, a kulcstér méretének elegendően nagynak kell lennie. Ezért van az, hogy a mai tipikus kulcsméret (szimmetrikus kulcsú rejtjelezők esetén) 128 bit, azaz a kulcstér mérete 2^{128}. Összehasonlításul, az univerzum becsült életkora 2^{59} másodperc, a Földön található atomok becsült száma pedig 2^{170}. Fontos azonban megérteni, hogy a kulcstér nagy mérete csak szükséges feltétele az algoritmikus biztonságnak. Elképzelhető ugyanis, hogy a rejtjelező valamely algebrai tulajdonságát kihasználva a rejtjelező sikeresen feltörhető annak ellenére, hogy kulcstere megfelelően nagy. Erre később mutatunk majd példát.

Történelmi példák

Ebben a szakaszban néhány egyszerű rejtjelezőt mutatunk be melyek közül párat a történelem valamely korábbi szakaszában a gyakorlatban is használtak a diplomáciai vagy a taktikai kommunikáció védelmére.

A shift rejtjelező. A shift rejtjelező esetén a nyílt és a rejtett üzenetek terét az ábécé betűi alkotják. Az egyszerűség kedvéért tegyük fel, hogy ez a 26 elemű angol ábécé. Ezen ábécé betűivel műveleteket szeretnénk végezni, s ennek érdekében minden betűhöz egy számértéket rendelünk, méghozzá az ábécében elfoglalt helyének sorszámát, ahol a számozást a 0-tól kezdjük. Így például az `A` betű értéke 0, a `B` betű értéke 1, stb. Az ábécét alkotó betűk halmazát ennek megfelelően a Z_{26} halmazzal, azaz a modulo 26 egész számok halmazával reprezentáljuk.

A shift rejtjelező k kódoló kulcsa a Z_{26} halmaz valamely eleme. A rejtjelezés abból áll, hogy k-t modulo 26 hozzáadjuk az x nyílt üzenethez. Vagyis a rejtjeles üzenet $y = (x+k) \bmod 26$ formában áll elő. A dekódolás ennek megfelelően a következő: $x = (y-k) \bmod 26$. Mivel a $k = 0$ kulcs az identitás transzformációra vezet, ezért azt kizárjuk a lehetséges kulcsok halmazából. Így a shift rejtjelező kulcsterének mérete $|Z_{26}| - 1 = 25$.

A shift rejtejelezővel hosszabb szövegeket is rejtjelezhetünk úgy, hogy a szöveg betűit külön-külön kódoljuk a k kulcs hozzáadásával. Állítólag ezt a módszert használta Julius Caesar hadvezéreinek szánt üzenetei titkosítására ($k = 3$ fix kulccsal).

3.1. példa. Rejtjelezzük a `VENIVIDIVICI` üzenetet a $k = 3$ kukccsal! Ellenőrizzük, hogy a rejtjeles szöveg a követező lesz: `YHQLYLGLYLFL`.

Látható, hogy a shift rejtjelezőt ily módon hosszabb szövegek rejtjelezésére használni nem biztonságos, mert a kulcstér kis mérete miatt, az összes lehetséges dekódoló kulcs gyorsan kipróbálható. A helyes dekódoló kulcsot akkor találtuk meg, mikor az adott kulccsal végrehajtott dekódolás eredménye értelmes nyílt szöveg.

Monoalfabetikus helyettesítés. A monoalfabetikus rejtjelező nyílt és rejtett üzeneteinek tere szintén a Z_{26} halmaz. Most azonban a kódoló kulcs a Z_{26} elemeinek valamely P permutációja. A kódolás úgy történik, hogy az x nyílt betűt a $P(x)$ betűvel helyettesítjük, azaz $y = P(x)$, ahol $P(x)$ az a betű amit a permutáció x-hez rendel. A dekódoláshoz szükségünk van a P permutáció P^{-1} inverzére. A dekódolást az $x = P^{-1}(y)$ kifejezés definiálja.

A shift rejtjelezőhöz hasonlóan, a monoalfabetikus rejtjelezés is használható hosszabb szövegek rejtjelezésére a szöveg betűinek kölön-külön történő helyettesítésével.

3.2. példa. Tekintsük a Z_{26} alábbi P permutációját:

$$\{\mathtt{T},\mathtt{Q},\mathtt{L},\mathtt{R},\mathtt{B},\mathtt{W},\mathtt{J},\mathtt{G},\mathtt{F},\mathtt{Y},\mathtt{N},\mathtt{C},\mathtt{P},\mathtt{E},\mathtt{K},\mathtt{S},\mathtt{D},\mathtt{U},\mathtt{V},\mathtt{I},\mathtt{X},\mathtt{M},\mathtt{Z},\mathtt{A},\mathtt{H},\mathtt{O}\}$$

Ekkor például $P(\mathtt{A}) = \mathtt{T}$, $P(\mathtt{B}) = \mathtt{Q}$, $P(\mathtt{C}) = \mathtt{L}$, stb. A VENIVIDIVICI üzenethez tartozó rejtjeles szöveg pedig MBEFMFRFMFLF lesz.

A kulcstér mérete most 26!, ami óriási szám, ezért a kimerítő kulcskeresés nem használható. Vegyük észre azonban, hogy a monoalfabetikus helyettesítés megtartja az eredeti szöveg betűstatisztikáját, s ezt sikerrel használhatjuk ki a rejtjeles üzenetek megfejtésénél.

Minden nyelvnek van egy jellegzetes betűstatisztikája, azaz egy átlagos szövegben minden betű az adott nyelvre jellemző valószínűséggel fordul elő. Például az angol nyelvben a leggyakoribb betű az E (12.7%), majd a T (9.1%). A legritkábban előforduló betűk a Z (0.1%) és a J (0.2%). Ezért ha egy átlagos angol szöveget rejtjelezünk egy monoalfabetikus rejtjelezővel, akkor a kódolt szövegben leggyakrabban előforduló betű nagy valószínűséggel az E betű kódoltja, a következő leggyakoribb betű valószínűleg a T kódoltja, stb. Ez a betűstatisztikára épülő fejtési módszer az esetek többségében sikerre vezet a monoalfabetikus rejtjelező esetében.

A monoalfabetikus rejtjelező fontos tanulsága, hogy a kulcstér méretének nagysága még nem garancia a rejtjelező erősségére vonatkozóan. Azaz, ahogy ezt korábban már említettük, a kulcstér nagy mérete csak a biztonság szükséges feltétele.

Polialfabetikus helyettesítés. A polialfabetikus rejtjelező hasonlít a monoalfabetikus rejtjelezőhöz, a különbség az, hogy a polialfabetikus rejtjelező a nyílt szöveg minden betűjét más permutációt használva helyettesíti. Ennek egyszerű példája a Vigenère-rejtjelező[2]. A Vigenère-rejtjelező működésének megértéséhez tekintsük a 3.2. ábra táblázatát.

A táblázat minden sora az ábécé különböző mértékű eltolásából áll, továbbá a táblázat sorai és oszlopai az ábécé betűivel vannak indexelve. Minden nyílt betű helyettesítésénél más sort használunk, melyet a kulcs határoz meg. A kulcs által kijelölt sorból a nyílt betűnek megfelelő oszlopban található betűt használjuk a nyílt betű helyettesítésére.

3.3. példa. Legyen a kulcs a CAESAR szó. Ekkor a VENIVIDIVICI szöveget a következő módon rejtjelezzük. Először a kulcsszót megismételjük annyiszor, hogy az üzenet hosszával megegyező betűsorozatot kapjunk. Esetünkben ez a sorozat a CAESARCAESAR lesz. Tekintsük a fenti táblázatban az első kulcsbetű által megcímzett sort, és az első nyílt betű által megcímzett oszlopot. Jelen esteben ez a táblázat (C, V) eleme, melynek értéke X. Tehát az első rejtjeles betű az X lesz. Ezután tekintsük a táblázatban a második kulcsbetű által megcímzett sort, és a második nyílt

[2]Blaise de Vigenère után, aki a módszert 1586-ban publikálta *Traicté des Chiffres* című értekezésében.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

3.2. ábra. A Vigenère-rejtjelező

betű által megcímzett oszlopot. Jelen esteben ez a táblázat (A, E) eleme, melynek értéke E. Tehát a második rejtjeles betű az E lesz, stb. A VENIVIDIVICI szöveghez tartozó rejtjeles szöveg a XERAVZFIZACZ.

Figyeljük meg, hogy a fenti példában a nyílt szöveg azonos betűit különböző rejtjeles betűkkel helyettesítettük. Például a V betűket rendre X, V, és Z betűkkel helyettesítettük. Ez annak köszönhető, hogy polialfabetikus rejtjelezés esetén minden nyílt betűt más permutáció segítségével helyettesítünk, amit a kulcs határoz meg. Emiatt az egyszerű betűstatisztikára épülő feltörési módszerünk most nem használható közvetlenül.

A Vigenère-rejtjelezőt a középkorban használták és sokáig feltörhetetlennek gondolták, míg végül Friedrich Kasiski porosz katonatiszt 1863-ban publikált egy törési módszert[3]. A módszer lényege, hogy először azonosítja a kulcsszó hosszát

[3] Állítólag Charles Babbage ennél korábban feltörte a Vigenère-rejtjelezőt, de nem publikálta módszerét.

a rejtjeles szöveg jellemzői periodicitásának megállapításával. Ha a kulcshossz k, akkor minden k. nyílt betű ugyanazon permutáció segítségével van helyettesítve. Így ha minden k. betűt tekintünk, akkor lényegében monoalfabetikus helyettesítést kapunk, s erre már a korábban bemutatott fejtési módszer alkalmazható.

Az affin rejtjelező. Az affin rejtjelező nyílt üzeneteinek és rejtett üzeneteinek tere a Z_{26} halmaz. A kódolási transzformációt az $y = (a \cdot x + b) \bmod 26$ kifejezés definiálja, ahol az $(a,b) \in Z_{26} \times Z_{26}$ pár a kódoló kulcs.

A dekódoláshoz szükségünk van a inverzére, azaz arra az $a^{-1} \in Z_{26}$ elemre, melyre $(a \cdot a^{-1}) \bmod 26 = 1$. Ismert, hogy Z_{26} felett csak azoknak az a elemknek létezik inverze, melyek relatív prímek a 26-hoz, azaz melyekre $(a,26) = 1$. Ezen elemek száma $\varphi(26) = 12$. A kulcstér tehát az $\{(a,b) \in Z_{26} \times Z_{26} : (a,26) = 1\}$ párok halmaza, és a kulcstér mérete $12 \cdot 26 = 312$.

Tegyük fel tehát, hogy a-nak létezik a^{-1} inverze. Ekkor a dekódoló transzformáció a következő: $x = (a^{-1} \cdot (y - b)) \bmod 26$.

Az affin rejtjelező könnyen feltörhető. Ha a támadónak lehetősége van választott nyílt szöveges támadásra, akkor kódoltatja az $x_1 = 0$ (= `A`) és az $x_2 = 1$ (= `B`) betűket. Ezzel hozzájut az $y_1 = (a \cdot 0 + b) \bmod 26 = b$ és az $y_2 = (a \cdot 1 + b) \bmod 26 = (a + b) \bmod 26$ értékekhez. Ezekből $b = y_1$ és $a = (y_2 - y_1) \bmod 26$.

Tegyük most fel, hogy a támadó hozzájutott két ismert nyílt szöveg – rejtett szöveg párhoz, (x_1,y_1)-hez és (x_2,y_2)-höz. Ekkor $y_1 = (a \cdot x_1 + b) \bmod 26$ és $y_2 = (a \cdot x_2 + b) \bmod 26$. Ebből

$$y_1 - y_2 \equiv a(x_1 - x_2) \pmod{26}$$

azaz

$$a = (y_1 - y_2) \cdot (x_1 - x_2)^{-1} \bmod 26$$

és

$$b = (y_1 - a \cdot x_1) \bmod 26 = (y_1 - (y_1 - y_2) \cdot (x_1 - x_2)^{-1} \cdot x_1) \bmod 26$$

feltéve, hogy $(x_1 - x_2)^{-1}$ létezik.

3.4. példa. Tegyük fel, hogy a támadó megszerzi a következő két párt: $(x_1,y_1) = (\mathtt{F},\mathtt{E}) = (5,4)$ és $(x_2,y_2) = (\mathtt{C},\mathtt{U}) = (2,21)$. Ekkor

$$a = (4 - 21) \cdot (5 - 2)^{-1} \bmod 26 = 9 \cdot 3^{-1} \bmod 26 = 9 \cdot 9 \bmod 26 = 3$$

és

$$b = (4 - 3 \cdot 5) \bmod 26 = -11 \bmod 26 = 15$$

A kódoló kulcs tehát $(a,b) = (3,15)$.

A Hill-rejtjelező. A Hill-rejtjelező nyílt és rejtjeles üzeneteinek tere a Z_{26}^n tér, ahol n egy rögzített pozitív egész. A kulcstér a Z_{26} feletti $n \times n$-es invertálható mátrixok halmaza. A kódolást az $\mathbf{y} = \mathbf{x} \cdot \mathbf{A}$ kifejezés definiálja, ahol $\mathbf{A}$ a kulcsmátrix, és a műveleteket modulo 26 végezzük. A dekódolást az $\mathbf{x} = \mathbf{y} \cdot \mathbf{A}^{-1}$ kifejezés adja meg, ahol $\mathbf{A}^{-1}$ az $\mathbf{A}$ mátrix inverze modulo 26.

3.5. példa. Legyen $n = 2$. Rejtjelezzük a `JULY` szöveget a következő kulccsal:

$$\mathbf{A} = \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix}$$

Mivel $n = 2$, ezért a nyílt szöveget 2 hosszú vektorokra kell bontanunk:

$$\texttt{JULY} \;\rightarrow\; (9,20);(11,24)$$

Ezután elvégezzük a szorzásokat (modulo 26):

$$(9,20) \cdot \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix} = (99+60 \bmod 26, 72+140 \bmod 26) = (3,4)$$

és

$$(11,24) \cdot \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix} = (121+72 \bmod 26, 88+168 \bmod 26) = (11,22)$$

Az eredényül kapott rejtjeles szöveg tehát:

$$(3,4);(11,22) \;\rightarrow\; \texttt{DELW}$$

Bizonyítás nélkül közöljük a következő két tételt, melyek a Hill rejtjelező dekódoló kulcsának kiszámításához nyújtanak segítséget:

3.1. tétel. *Egy* $\mathbf{A}$ *mátrix akkor és csak akkor invertálható* Z_k *felett, ha* $(k, \det \mathbf{A}) = 1$.

3.2. tétel. *Az* $\mathbf{A}$ *mátrix* Z_k *feletti inverze a következőképpen számolható:*

$$\mathbf{A}^{-1} = (\det \mathbf{A})^{-1} \mathbf{A}^* \tag{3.1}$$

ahol az $\mathbf{A}^*$ *az* $\mathbf{A}$ *adjungált mátrixa, azaz*

$$\left(a_{ij}^*\right) = \left((-1)^{i+j} \det \mathbf{A}_{ji}\right) \tag{3.2}$$

ahol $\mathbf{A}_{ji}$ *az a mátrix amelyet* $\mathbf{A}$*-ból nyerünk a* j*. sor és az* i*. oszlop törlésével.*

Alkalmazzuk a fenti tételt az $n = 2$ esetre, azaz amikor $\mathbf{A}$ a következő alakú:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

Ekkor

$$\mathbf{A}^* = \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix} \tag{3.3}$$

és

$$\mathbf{A}^{-1} = (\det\ \mathbf{A})^{-1} \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix} \tag{3.4}$$

3.6. példa. Számoljuk ki az előző példában használt $\mathbf{A}$ mátrix inverzét!

$$\mathbf{A} = \begin{pmatrix} 11 & 8 \\ 3 & 7 \end{pmatrix}$$

$$\det\ \mathbf{A} = (11 \cdot 7 - 3 \cdot 8) \bmod 26 = 1$$

$$(\det\ \mathbf{A})^{-1} = 1$$

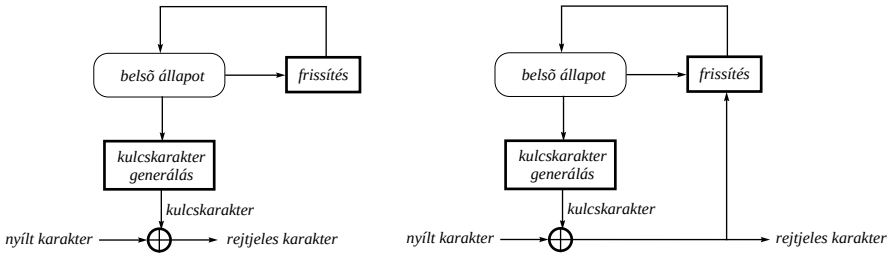
$$\mathbf{A}^{-1} = \begin{pmatrix} 7 & -8 \\ -3 & 11 \end{pmatrix} = \begin{pmatrix} 7 & 18 \\ 23 & 11 \end{pmatrix}$$

Kulcsfolyam rejtjelezők

A kulcsfolyam rejtjelezők a nyílt szöveget karakterenként kódolják, mégpedig úgy, hogy a nyílt szöveg karaktereit kombinálják a kulcsfolyam karaktereivel. Ez a kombináció tipikusan a (bitenkénti) XOR műveletet jelenti. A kulcsfolyamot a kulcsfolyam generátor állítja elő egy belső állapotból. A generátor minden kulcskarakter előállítása után frissíti a belső állapotát, így a következő kulcskaraktert már ebből a friss állapotból számolja. A kezdő állapotot egy közös titokból állítja elő a küldő és a vevő.

A belső állapot frissítésének módjától függően megkülönböztetünk **szinkron** és **önszinkronizáló** kulcsfolyam rejtjelezőket. Szinkron rejtjelezők esetében, a következő állapot csak az előző állapottól vagy állapotoktól függ, míg önszinkronizáló rejtjelező esetén a következő állapot függ a rejtjeles karakterektől (s azon keresztül a nyílt szövegtől). A szinkron és az önszinkronizáló kulcsfolyam rejtjelezők működési vázlatát a 3.3. ábra szemlélteti.

Szinkron kulcsfolyam rejtjelezők esetében ügyelni kell arra, hogy a küldő és a vevő állapota szinkronban legyen. Ellenkező esetben a vevő nem a megfelelő állapotból generálja a vett rejtjeles karakter dekódolásához szükséges kulcskaraktert,



3.3. ábra. A szinkron (bal oldali ábra) és az önszinkronizáló (jobb oldali ábra) kulcsfolyam rejtjelezők működési vázlata.

és a dekódolás eredménytelen lesz. Önszinkronizáló kulcsfolyam rejtjelezők esetében megfelelő számú helyesen vett rejtjeles karakter automatikusan szinkronizálja a küldő és a vevő állapotát. Ezekből a tulajdonságokból származik a szinkron és az önszinkronizáló elnevezés.

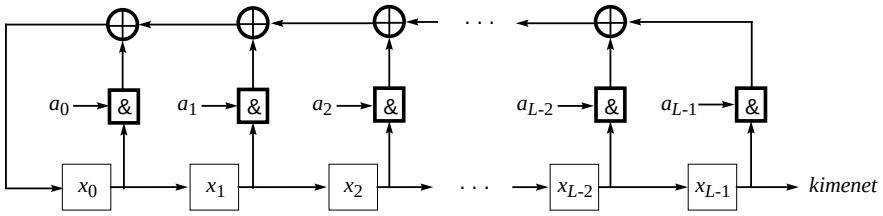
Szoftver megvalósítás esetén a rejtjelező állapotát egy belső változóban tároljuk. Például a szoftver megvalósításra tervezett RC4 rejtjelező esetében a belső állapot az összes lehetséges bájt egy permutációja, melyet egy 256 elemű bájttömbben tárolunk, és két egybájtos index változó. Így a belső állapot tárolására 258 bájtra van szükség, és a rejtjelező állapotterének mérete $2^{16} \cdot 256!$.

Hardver megvalósítás esetén a belső állapotot regiszterekben tároljuk. Ennek tipikus példája a lineárisan visszacsatolt shift-regiszter (Linear Feedback Shift Register – LFSR) használata. Mivel az LFSR-re épülő rejtjelezők a gyakorlatban igen elterjedtek, ezért ezeket a következő szakaszban kicsit részletesebben tárgyaljuk.

LFSR alapú kulcsfolyam rejtjelezők

A lineárisan visszacsatolt shift-regiszter működését a 3.4. ábrán vázoltuk. A shift-regiszter rekeszeiben tárolt bitek reprezentálják az aktuális belső állapotot. A shift-regisztert induláskor egy kezdőértékkel töltjük fel. Ezután, minden ütemben, kiszámítjuk a rekeszek tartalmának egy lineárkombinációját, és az eredményt balról beléptetjük a shift-regiszterbe. Ennek hatására minden rekesz tartalma a tőle jobbra található rekeszbe másolódik, a jobb szélső rekesz tartalma pedig az LFSR kimenetére kerül. Ily módon az LFSR minden ütemben egy bit kimenetet generál, és a visszacsatoláson keresztül frissíti belső állapotát.

Ha az LFSR-t kulcsfolyam generátorként szeretnénk használni, akkor a visszacsatolás módját titokban tartjuk, azaz azt csak a küldő és a vevő ismeri. Azonos kezdőállapotból indulva így a küldő és a vevő LFSR-e ugyanazt a bitsorozatot generálja, mely kulcsfolyamként használható. Azt reméljük továbbá, hogy a visszacsatolás ismeretének hiányában egy támadó nem tudja előállítani az LFSR kimenetén megjelenő kulcsfolyamot.



3.4. ábra. A lineárisan visszacsatolt shift-regiszter működési vázlata.

Sajnos a fent vázolt, egyetlen LFSR-re épülő kulcsfolyam rejtjelező nem biztonságos. Ennek elmagyarázásához be kell vezetnünk egy bitsorozat lineáris komplexitásának fogalmát, és egy kapcsolódó tételt:

3.1. definíció. *Egy bitsorozat lineáris komplexitása alatt a sorozatot előállító legrövidebb LFSR hosszát értjük.*

3.3. tétel. *Tekintsünk egy s bináris sorozatot, melynek lineáris komplexitása L. Ekkor s egy legalább 2L hosszúságú részsorozatából megkonstruálható az az LFSR, mely előállítja az s sorozatot.*

Legyen a rejtjelezéshez használt LFSR hossza L'. Ezen LFSR által előállított bináris sorozat L lineáris komplexitására nyilván teljesül, hogy $L \leq L'$. A fenti tétel értelmében tehát, ha a támadó megfigyel az LFSR kimenetén $2L'$ egymásutáni bitet, akkor meg tudja konstruálni az LFSR-t, azaz meg tudja fejteni a visszacsatolás módját, ami jelen esetben maga a titkos kulcs. Vegyük észre továbbá, hogy az LFSR kimenete egy nyílt szöveg – rejtett szöveg párokat használó támadás esetén egyszerűen a nyílt és a rejtett szövegek XOR összegeként számolható.

A fenti gyengeség miatt, a gyakorlatban használt LFSR alapú rejtjelezők nem egyetlen LFSR-t használnak, hanem több LFSR kimenetét kombinálják egy nemlineáris függvény segítségével.

A one-time pad

A one-time pad (OTP) egy olyan kulcsfolyam rejtjelező, ahol a K kulcsfolyam egy igazi véletlen bitsorozat. Rejtjelezésnél ezt XOR-oljuk az X nyílt szöveg bitjeihez, és így kapjuk az Y rejtjeles szöveg bitjeit. Azaz $Y = X \oplus K$. Dekódolásnál hasonlóképpen járunk el, s az XOR művelet tulajdonságai miatt visszakapjuk a nyílt szöveget: $Y \oplus K = X \oplus K \oplus K = X$.

A one-time pad segítségével **tökéletes titkosítás**t valósíthatunk meg, ahol a tökéletes titkosítást a következőképpen definiáljuk:

3.2. definíció. *Tökéletes titkosításról akkor beszélünk, ha a támadó a rejtett szöveg megfigyelésével nem jut több információhoz a nyílt szöveget illetően, mint*

amennyi információ birtokában a rejtett szöveg megfigyelése előtt volt. Formálisan, tökéletes titkosítás esetén, a nyílt szöveget és a rejtett szöveget reprezentáló valószínűségi változók függetlenek.

Például, ha minden nyílt szöveg egyforma valószínűséggel fodul elő, akkor tökéletes rejtjelezés esetén, a támadó megfigyeli a rejtett szöveget, s továbbra is minden nyílt szöveg egyforma valószínűségű számára. Azaz a megfigyeléssel nem jut több információhoz, és továbbra sem tudja eldönteni, hogy melyik nyílt szöveget rejtjelezték.

Sajnos a one-time pad esetén minden egyes üzenet rejtjelezéséhez véletlenül sorsolt új kulcs szükséges, melynek mérete lényegében megegyezik az üzenet méretével. Ezt az állítást pontosítja a következő tétel:

3.4. tétel. *A tökéletes titkosítás szükséges feltétele, hogy a kulcs hossza legalább akkora legyen, mint a tömörített nyílt szöveg hossza.*

A fenti tétel alapján, a tökéletes titkosításhoz egy az üzenet méretével megegyező hosszúságú titkos kulcsot kell biztonságosan eljuttatni a küldőtől a vevőhöz. Ha erre a célra létezne titkosított csatorna a küldő és a vevő között, akkor azon maga a rejtjelezni kívánt üzenet is továbbítható lenne. Csak arról lehet tehát szó, hogy a küldő és a vevő a kommunikáció előtt előre megállapodik a későbbi kommunikációhoz szükséges mennyiségű véletlen kulcsbitben. Például, személyesen találkoznak, és a küldő valamilyen adathordozón átadja a később használni kívánt kulcsbiteket. Ez nyilvánvalóan nagy mértékben szűkíti a tökéletes titkosítás gyakorlati alkalmazhatóságának körét.

Szimmetrikus kulcsú blokkrejtjelezők

A blokkrejtjelező egy $E : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$ transzformáció, amely az n bites nyílt blokkot és a k bites kulcsot az n bites rejtett blokkba transzformálja. Rögzített kulcs mellett az E transzformációnak természetesen invertálhatónak kell lennie, különben a rejtjeles blokkból nem tudnánk egyértelműen visszaállítani a nyílt blokkot. A fent definiált blokkrejtjelezőt tehát úgy is tekinthetjük, mint az $E_K : \{0,1\}^n \to \{0,1\}^n$ invertálható leképezések egy családját, ahol a család egyes elemeit a $K \in \{0,1\}^k$ kulcs választja ki. Az E_K transzformáció invertálhatóságából, valamint a bemenet és a kimenet méretének azonosságából következik, hogy E_K egy permutációt határoz meg az n bites bináris vektorok halmazán.

A blokkrejtjelezővel szemben támasztott minimális tervezési követelmények a következők:

- **Statisztikai függetlenség:** Ne legyen egyértelmű statisztikai kapcsolat a nyílt blokkok és a rejtjeles blokkok között.
- **Teljesség:** A kimenet minden bitje minden bementi bittől és minden kulcsbittől függjön.

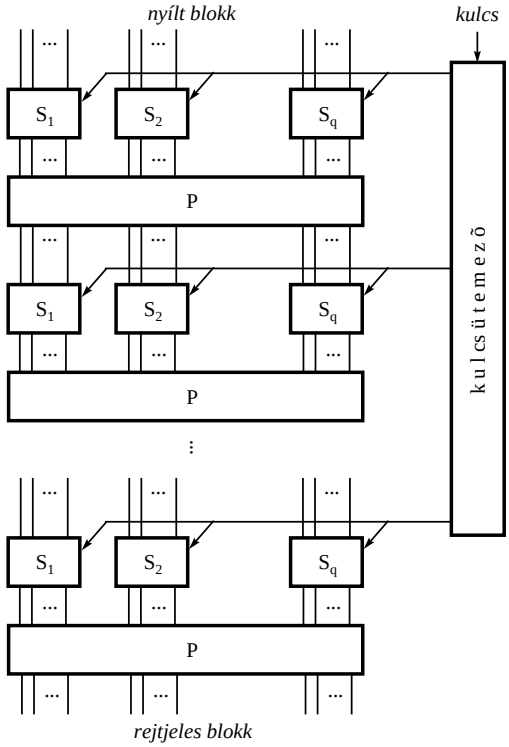
- **Lavinahatás:** Ha a bemeneten egy bitet megváltoztatunk, akkor a kimeneten minden bit közel $\frac{1}{2}$ valószínűséggel változzon meg. Ez a követelmény azt biztosítja, hogy a blokkrejtjelező majdnem azonos nyílt szövegeket is teljesen eltérő rejtett szövegekbe kódol. Így például eltérő sorszámmal rendelkező, de azonos tartalmú üzenetek a rejtett szövegek megfigyelése alapján nem csoportosíthatók. Hasonlóképpen, szeretnénk, ha egy kimeneti bit megváltoztatása nem predikálható változásokat eredményezne a bemeneten.

A legtöbb gyakorlatban használt blokkrejtjelező **szorzat rejtjelező**. A szorzat rejtjelezők önmagukban egyszerű transzformációk egymásután többszöri alkalmazásával próbálják meg elérni, hogy az eredményül kapott transzformáció (az egyszerű transzformációk szorzata) kielégítse a fenti kritériumokat. A szorzat rejtjelezőben használt egyszerű transzformációk lehetnek egyszerű logikai és aritmetikai műveletek, kis méretű helyettesítések, vagy bitpermutációk. Nyilvánvaló azonban, hogy ezek tetszőleges kombinációja nem fog biztonságos blokkrejtjelezőt eredményezni. A blokkrejtjelező tervezése nagy szakértelmet igényel, és nem tanácsos azt hozzá nem értőre bízni.

A szorzat rejtjelezők egyik gyakori típusa a **helyettesítéses-permutációs rejtjelező** (substitution-permutation, vagy röviden SP rejtjelező), mely kis méretű helyettesítések és nagy méretű bitpermutációk sorozatából áll. Ennek jellegzetes példája a DES, amelyet később részletesen ismertetünk. A klasszikus SP rejtjelező felépítése a 3.5. ábrán látható. A bemenetre kerülő nyílt blokk permutációs és kulcsvezérelt helyettesítő rétegeken keresztül jut el a kimenetre. A permutációs réteg egyetlen P bitpermutációból (ún. P-dobozból) áll. A helyettesítő réteg több kis méretű S_i helyettesítő transzformációból (ún. S-dobozból) épül fel, melyek mindegyike több helyettesítő táblát tartalmaz. Az S-dobozba bevezetett kulcsbitek határozzák meg, hogy melyik táblát kell használni a kódolás során.

A permutációk és helyettesítések egymásutáni használatának célja a lavinahatás elérése. Ha például az S-dobozok olyanok, hogy tetszőleges bemenet esetén a bemeneten egy bit megváltoztatása a kimeneten legalább két bit megváltozását okozza, akkor az első réteg bemenetén fellépő egy bit változás az első helyettesítő réteg kimenetén legalább 2 bit változását eredményezi, amit a bit permutáció szétszór a teljes blokkon, így azok tipikusan a következő réteg két különböző S-dobozának bemenetére kerülnek. A második réteg kimenetén így már legalább 4 bit változik meg, és így tovább. Az eredmény az lesz, hogy egymáshoz Hamming-távolságban közeli nyílt szöveg blokkok álvéletlen módon tipikusan távolra kerülnek egymástól.

Fontos továbbá, hogy az S-dobozok nemlineáris leképezést valósítsanak meg, azaz a rejtett szöveg vektor a nyílt szöveg vektorból egy bináris mátrixszal történő szorzással ne legyen előállítható. Ellenkező esetben a teljes rejtjelező lineáris transzformációt valósítana meg, hiszen a bitpermutációk lineáris leképezések. A klasszikus történelmi példák tárgyalása során azonban láttuk, hogy a lineáris rejtjelezők könnyen feltörhetőek.

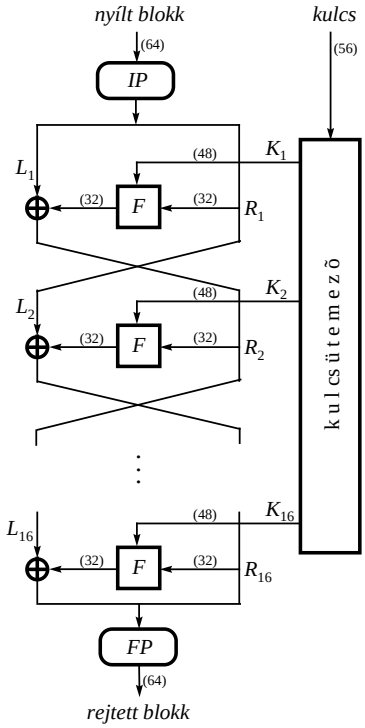


3.5. ábra. A klasszikus helyettesítéses-permutációs rejtjelező felépítése.

A DES blokkrejtjelező

A DES (Data Encrytpion Standard) a legismertebb helyettesítéses-permutációs blokkrejtjelező, melyet az IBM szakemberei fejlesztettek ki a hetvenes években. Később a világ több országában szabványként fogadták el, és azóta is használják. A DES egy jól megtervezett blokkrejtjelező, legalábbis nem ismert ellene olyan igazán hatékony támadás mely valamilyen tervezési gyengeséget használna ki. Ugyanakkor, a DES 56 bites kulcsmérete miatt ma már nem zárható ki a kimerítő kulcskeresés mint hatékony támadás a DES ellen. Ezen probléma megoldására 2001-ben új blokkrejtjelező szabványt fogadtak el, melynek kulcsmérete legalább 128 bit. Ez az új blokkrejtjelező, melyet AES-nek (Advanced Encryption Standard) hívnak, fokozatosan felváltja a DES-t a különböző alkalmazásokban. Az AES-t a DES után tárgyaljuk majd.

A DES struktúrája a 3.6. ábrán látható. A bemenet és a kimenet mérete 64 bit, a kulcs 56 bites. A rejtjelező 16 rétegből áll, ahol a rétegek felépítése egy kicsit eltér a 3.5. ábra rétegeinek klasszikus felépítésétől. Itt az egyes rétegek bemenete két részből áll, jelöljük ezeket L_i-vel és R_i-vel, ahol L_i és R_i mérete 32 bit. A jobb oldali blokk, R_i áthalad egy kulcsvezérelt, S-dobozokból és egy P-dobozból felépülő F



3.6. ábra. A DES szerkezete.

transzformáción. Az i. réteg kulcsát jelöljük K_i-vel. K_i 48 bites, és a kulcsütemező állítja elő az eredeti 56 bites kulcsból. Az F transzformáció kimenetét XOR-oljuk a bal oldali blokkal, L_i-vel, majd a két oldal blokkjait felcseréljük (kivéve az utolsó rétegben). Így kapjuk a következő réteg bemenetének két blokkját. Formálisan:

$$\begin{aligned} L_{i+1} &= R_i \\ R_{i+1} &= L_i \oplus F(R_i, K_i) \end{aligned} \tag{3.5}$$

Ezt a struktúrát Feistel-struktúrának[4] nevezik. Az az előnye, hogy az így felépülő rejtjelező mindenképpen invertálható, függetlenül attól, hogy F invertálható, vagy sem. Vegyük észre ugyanis, hogy L_i és R_i előállítható L_{i+1}-ből és R_{i+1}-ből anélkül, hogy F inverzét használnánk, hiszen (3.5)-ből:

$$\begin{aligned} L_i &= R_{i+1} \oplus F(L_{i+1}, K_i) \\ R_i &= L_{i+1} \end{aligned}$$

A Feistel-struktúra egy másik előnye, hogy a dekódoló eljárás megegyezik a kódoló eljárással, csupán a kulcsütemezőt kell fordítva működtetni (pl. a kódolásnál az első rétegben alkalmazott 48 bites kulcsot a dekódolásnál az utolsó rétegben

[4]Horst Feistel, az IBM kriptugráfusa után, aki ezt a struktúrát kitalálta.

kell használni). Ez különösen hasznos tulajdonság, ha a DES-t hardverben szeretnénk implementálni, mert ugyanaz az áramkör használható a kódoláshoz és a dekódoláshoz, s így kevesebb kapura van szükség.

A továbbiakban a DES építőelemeit vizsgáljuk meg részletesebben:

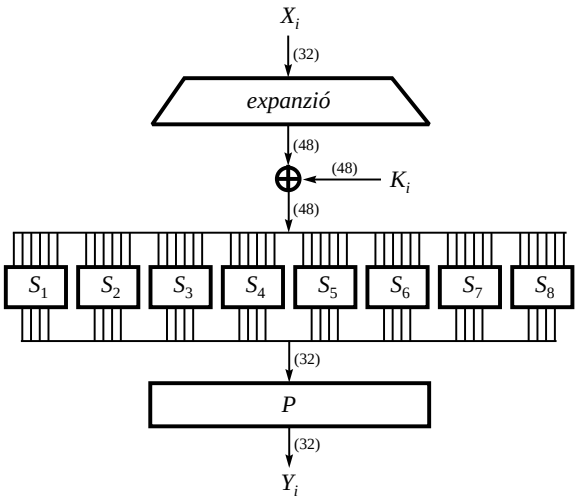
A kezdeti és a végső permutáció (*IP* és *FP*). E két 64 bit bemenetű és 64 bit kimenetű bitpermutációnak nincs különösebb szerepe, a DES biztonságára nincsenek hatással. Mivel szoftverben viszonylag nehezen implementálhatók, ezért gyakran el is hagyják őket, eltérve az eredeti szabványtól.

A kulcsütemező. A kulcsütemező feladata, hogy az egyes rétegek számára előállítsa a megfelelő méretű kulcsot. Ezt a következőképpen teszi. Először is az 56 bites kulcsot két 28 bites részre vágja. Mindkét felet balra rotálja egyszer vagy kétszer, attól függően, hogy melyik réteg kulcsáról van éppen szó. A rotálások száma természetsen a DES szabványban rögzített. A rotálás után egy *PC* permutáció szerint kiválaszt az 56 bitből 48-at. A rotálások következtében minden rétegben más 48 bit kerül kiválasztásra, annak ellenére, hogy a választó *PC* permutáció minden rétegben azonos. Minden bit kb. 14 rétegbe lép be. Az összes rotálások száma pontosan 28, így egy blokk rejtjelezése után az 56 bites kulcs az eredeti állapotába áll vissza, és kezdődhet a következő blokk kódolása. Ennek elsősorban hardver megvalósításoknál van szerepe.

Az *F* függvény. Az *F* függvény felépítését a 3.7. ábra szemlélteti. A bemenet először egy expanziós transzformáción halad keresztül, mely a 32 bites vektorból egy 48 bites vektort állít elő. Ezt XOR-oljuk a 48 bites rétegkulccsal. A következő elem egy S-doboz réteg, melyben 8 darab 6 bitet 4 bitbe képző S-doboz található. Az S-doboz réteg kimenete így 32 bites. Az F függvény kiemenetét végül egy 32 bites P-doboz állítja elő.

Az S-dobozok (S_i). Minden S-doboz 4 darab 4 bitet 4 bitbe helyettesítő táblázatból áll. Ezen táblázatok közül az S-doboz bemenetének két szélső bitje választja ki az aktuálisan alkalmazandót. Úgy is elképzelhetjük, hogy az S-dobozok 4 sorból és 16 oszlopból álló táblázatok, és a bemenet két szélső (tehát az 1. és a 6.) bitje címzi meg a sort, a középső 4 (tehát a 2–5.) bit pedig az oszlopot.

A P-doboz (*P*). A P-doboz egy egyszerű bitpermutáció. A P-doboz feladata a bitek összekeverése oly módon, hogy egy S-doboz kimeneti bitjei a következő rétegben más S-dobozok bemeneti bitjei legyenek.



3.7. ábra. A DES F rétegfüggvényének felépítése.

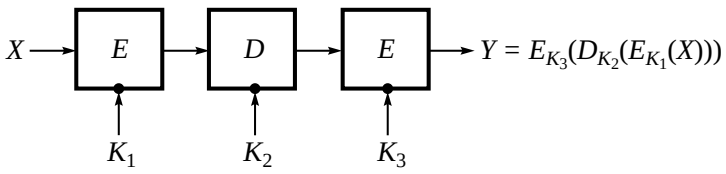
Többszörös rejtjelezés és a 3DES

Több különböző kódolási eljárás kombinálásától azt reméljük, hogy egy erősebb rejtjelezőre jutunk. Az egyik legközvetlenebb ezen irányba tett lépés a kétszeres kódolás: $Y = E_{K_2}(E_{K_1}(X))$

A kulcshossz a kétszeres kódolással formailag kétszeresére nő, hiszen két kulcsot használunk. Formailag, mert ténylegesen csak akkor van így, ha a kétszeres kódolással a kétszeres kulcshossznak megfelelő számú különböző transzformációt állítunk elő. Ha a kódolónk csoport tulajdonságú, akkor ez biztosan nincs így, hiszen ezesetben $E_{K_2}(E_{K_1}(X)) = E_{K_3}(X)$, valamely $E_{K_3} \in \mathcal{E}$ esetén, ahol $\mathcal{E} = \{E_{K_i}\}$ a kódoló transzformációk halmaza. Ekkor tehát egyáltalán nem változott a különböző transzformációk száma, azaz kétszeres kódolással csak ugyanazon transzformációkat tudjuk előállítani, mint egyszeres kódolással.

Mivel a DES esetén a csoport tulajdonság nem áll fenn, ezért van értelme a DES-sel többszörösen rejtjelezni különböző kulcsokkal. Ugyanakkor a kétszeres kódolásnak van egy nagy hátránya. Nevezetesen az, hogy a kimerítő kulcskeresés támadásnál lényegesen egyszerűbb támadás kínálkozik ellene, melyet **középen találkozás támadás**nak neveznek.

A középen találkozás támadás lényege a következő: Tegyük fel, hogy a támadó birtokában van egy összetartozó nyílt blokk – rejtjeles blokk pár, amit jelöljünk (X,Y)-nal. Tudjuk, hogy a keresett kódoló transzformáció két egylépéses kódoló transzformáció szorzata. A támadó az X nyílt blokkot kódolja egylépéseses kódolással az összes lehetséges kulccsal, s hasonlóan, az Y rejtett blokkot dekódolja egylépéseses dekódolással az összes lehetséges kulccsal. Ezután a támadó a kódolás és dekódolás eredményei halmazának metszetét vizsgálja. Ha egy



3.8. ábra. A 3DES EDE konfigurációban.

$E_{K_i}(X)$ kódolási lépés eredménye egybeesik egy $D_{K_j}(Y)$ dekódolási lépés eredményével, akkor a keresett ismeretlen kétlépéses transzformációra a támadó sejtése $E_{K_j}(E_{K_i}(.))$, amit további birtokában levő nyílt blokk – rejtett blokk párokon ellenőrizhet. Jelöljük az egylépéses kódolás kulcshosszát k-val. Ekkor, a támadás komplexitása $2 \cdot 2^k = 2^{k+1}$, ami sokkal kisebb, mint a kétszeres kulcshosszhoz tartozó kimerítő kulcskeresés 2^{2k} komplexitása.

Kétszeres DES rejtjelezéssel tehát nem érünk el jelentős javulást. A következő lehetőség a háromszoros kódolás. Az így nyert rejtjelezőt 3DES-nek nevezik, és a gyakorlatban igen elterjedten használják. A 3DES-t EDE és DED konfigurációban lehet használni, ahol az E betű a kódoló, a D betű pedig a dekódoló transzformációra utal. A 3DES-EDE vázlatát mutatja a 3.8. ábra.

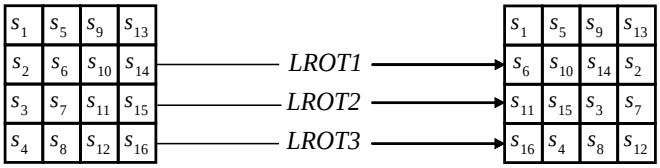
Ha $K_1 = K_3$, ekkor két kulcsos 3DES-ről beszélünk, melynek effektív kulcshossza 112 bit. Ha a K_1, K_2, és K_3 kulcsok mind különbözők, akkor három kulcsos 3DES-ről beszélünk, melynek effektív kulcshossza 168 bit.

Az AES blokkrejtjelező

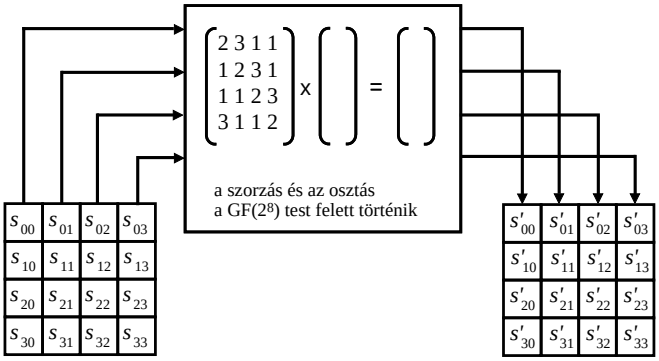
Az AES-t (Advanced Encryption Standard) egy több évig tartó eljárás során választották a DES utódjának több tucat jelölt közül. A DES lecserélése azért vált időszerűvé, mert 56 bites kulcsmérete a mai technológia mellett túlságosan rövidnek számít. Az AES kulcsmérete változó: 128, 192, vagy 256 bit (a rétegek számától függően). Az AES blokkmérete 128 bit (azaz 16 bájt).

Az AES is réteges szerkezetű, azaz szorzat-rejtjelező. A rétegek száma változó: 10, 12 vagy 14. Továbbá a DES-hez hasonlóan az AES is használ S-dobozt, azaz helyettesítéses-permutációs rejtjelező. A DES-sel ellentétben azonban az AES nem Feistel-struktúrájú. Így az AES kódoló és dekódoló algoritmusa nem azonos, a dekódoló algoritmus a kódolás során alkalmazott transzformációk inverzét használja. Az AES úgy van tervezve, hogy a kódoló algoritmus hatékonyabb, mint a dekódoló. Ez azért van így, mert több blokkrejtelezési mód (lásd később a 3.2. szakaszt) a kódoló transzformációt használja a dekódoláshoz is.

Az AES kódoló minden rétegében a következő négy művelet kerül végrehajtásra:



3.9. ábra. Az AES sor-eltolás művelete.



3.10. ábra. Az AES oszlop-keverés művelete.

- **Bájt-helyettesítés:** Az AES-nek egyetlen 8 bitet 8 bitbe képző S-doboza van. A bájt-helyettesítés során ezt az S-dobozt használjuk az adott réteg 16 bájtos bemenete minden egyes bájtjának helyettesítésére.

- **Sor-eltolás:** A sor-eltoláshoz a bájt-helyettesítés eredményeként kapott 16 bájtos vektort egy 4×4-es mátrixba rendezzük, majd a mátrix i. sorát $(i-1)$-gyel balra rotáljuk. Ezt a 3.9. ábra szemlélteti.

- **Oszlop-keverés:** Az oszlop-keverés abból áll, hogy a sor-eltolás eredményeként kapott 4×4-es mátrixot megszorozzuk egy, az AES specifikációban megadott konstans mátrixszal. A műveleteket GF(2^8) felett végezzük. Ezt a 3.10. ábra szemlélteti. Végül az eredményül kapott mátrix elemeit újra sorba rendezzük.

- **Kulcs-injekció:** A kulcs-injekció során a 16 bájtos rétegkulcsot (melyet egy kulcsütemező állít elő minden réteg számára) bitenként XOR-oljuk az oszlop-keverés eredményeként kapott 16 bájtos vektorral.

Az AES blokkrejtjelező, szabványként való elfogadása óta, a kriptanalízis szakértők egyik kedvelt célpontja, ugyanúgy, ahogy közel harminc évvel ezelőtt a DES blokkrejtjelező. Az AES eddig sikeresen ellenállt minden feltörési kísérletnek.

S-doboz tervezési kritériumok

A helyettesítéses-permutációs rejtjelezők legfontosabb építőelemei az S-dobozok. A rejtjelező ereje nagy mértékben függ S-dobozainak tulajdonságaitól, ezért ezek tervezése igen nagy körültekintést és tapasztalatot igényel. Ebben a szakaszban összefoglaljuk az S-doboz tervezés főbb kritériumait.

Elsőként formálisan is definiáljuk, hogy mit értünk S-doboz alatt:

3.3. definíció. *Egy $f : \{0,1\}^m \to \{0,1\}^n$ leképzést S-doboznak nevezünk, ha $1 < n \leq m$. f-et gyakran tekintjük úgy, mint n darab Boole-függvény együttesét:*

$$f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \tag{3.6}$$

ahol az $f_i : \{0,1\}^m \to \{0,1\}^1$ függvényeket komponens Boole-függvényeknek nevezzük.

Balansz tulajdonság. A balansz tulajdonság azt jelenti, hogy az S-doboz nem torzítja el egy egyenletes eloszlású bemenet gyakoriság-statisztikáját.

3.4. definíció. *Egy $f : \{0,1\}^m \to \{0,1\}^n$ S-doboz akkor balansz, ha tetszőleges $y \in \{0,1\}^n$ esetén*

$$|\{x \in \{0,1\}^m : f(x) = y\}| = 2^{m-n} \tag{3.7}$$

Teljesség. Egy $f : \{0,1\}^m \to \{0,1\}^n$ S-doboz teljes, ha minden kimeneti bitje minden bemeneti bittől függ. Más szavakkal ez azt jelenti, hogy az $f(x) \oplus f(x \oplus e_m^{(i)})$ függvény j. komponens Boole-függvényének igazságtáblájában legalább egy darab 1 található, és ez minden $1 \leq i \leq m$ és $1 \leq j \leq n$ esetén igaz. Formálisan:

3.5. definíció. *Egy $f : \{0,1\}^m \to \{0,1\}^n$ S-doboz teljes, ha minden $1 \leq i \leq m$ és $1 \leq j \leq n$ esetén teljesül a következő:*

$$|\{x \in \{0,1\}^m : e_n^{(j)} \odot f(x) \neq e_n^{(j)} \odot f(x \oplus e_m^{(i)})\}| > 0 \tag{3.8}$$

ahol $e_m^{(i)}$ az m dimenziós egységvektor, mely egyetlen egyest tartalmaz az i. pozícióban.

Linearitás és affinitás. A lineritás és az affinitás definíciója a következő:

3.6. definíció. *Egy $f : \{0,1\}^m \to \{0,1\}^n$ S-doboz lineáris, ha felírható $f(x) = A \odot x$ alakban, és affin, ha felírható $f(x) = A \odot x \oplus b$ alakban, ahol A egy $n \times m$-es bináris mátrix, $b \in \{0,1\}^n$, és $\odot$ a bináris skalárszorzatot jelöli, azaz $A \odot x$ egy olyan bináris vektor, melynek komponensei az A mátrix sorainak és az x vektornak a bináris skalárszorzataként állnak elő.*

Mivel a lineáris (affin) S-dobozokból felépülő rejtjelező könnyen analizálható, ezért a lineáris (affin) S-dobozok kerülendők.

Lineáris struktúrák, lineáris dimenzió. Előfordulhat, hogy egy S-doboz nem lineáris (affin), mégis rendelkezik bizonyos parciális linearitással, azaz kimenete néhány bemeneti bittel lineáris (affin) kapcsolatban áll. Ezen parciális linearitás mérőszáma a lineáris dimenzió.

A lineáris dimenzió formális definíciójához először a lineáris struktúra definícióját vezetjük be:

3.7. definíció. *Az $a \in \{0,1\}^m$ bináris vektor lineáris struktúrája az $f : \{0,1\}^m \to \{0,1\}^n$ S-doboznak, ha $f(x) \oplus f(x \oplus a)$ minden $x \in \{0,1\}^m$ esetén konstans.*

Könnyen belátható, hogy egy $f : \{0,1\}^m \to \{0,1\}^n$ függvény lineáris struktúrái alteret alkotnak a $\{0,1\}^m$ térben. Ezen altér dimenzióját nevezzük f lináris dimenziójának:

3.8. definíció. *Egy $f : \{0,1\}^m \to \{0,1\}^n$ S-doboz lineáris dimenziója az f lineáris struktúrái által alkotott altér dimenziója.*

Nemlinearitás. Mivel egy helyettesítéses-permutációs rejtjelező egyetlen nemlineáris eleme a helyettesítő réteg, ezért ez a kritérium központi szerepet játszik az S-dobozok tervezése során.

3.9. definíció. *Egy $f : \{0,1\}^m \to \{0,1\}^1$ Boole-függvény $N(f)$ nemlinearitásán az affin Boole-függvények halmazától vett távolságát értjük:*

$$N(f) = \min_{u \in \{0,1\}^m, v \in \{0,1\}^1} d(f, L_{u,v}) \tag{3.9}$$

ahol az $L_{u,v}(x)$ affin függvény $u \odot x \oplus v$ alakú, továbbá két Boole-függvény távolsága alatt azon bemenetek számát értjük, melyekre a két függvény kimenete különbözik egymástól.

Egy S-doboz nemlinearitását komponens Boole-függvényeinek segítségével definiáljuk a következőképpen:

3.10. definíció. *Képezzük egy S-doboz komponens Boole-függvényeinek lehetséges lineárkombinációit. Az így kapott Boole-függvények közül válasszuk ki azt, amelyiknek nemlinearitása minimális. Ez a minimális nemlinearitás az S-doboz nemlinearitása:*

$$N(f) = \min_{\beta \in \{0,1\}^n, \beta \neq 0} N(\beta \odot f) \tag{3.10}$$

Differenciális egyenletesség. A differenciális egyenletesség a nemlinearitás egy másik mérőszáma, mely a lineáris struktúráktól való távolsággal van összefüggésben. Ez a kritérium a differenciális kriptanalízissel kapcsolatban jelent meg. A differenciális kriptanalízis egy olyan támadási módszer, mely az S-dobozok azon tulajdonságát használja ki, hogy adott bemeneti differencia esetén a kimeneten nem minden differencia jelenik meg azonos valószínűséggel. A differenciális egyenletesség ennek az egyenetlenségnek a mérőszáma.

3.11. definíció. *Egy $f:\{0,1\}^m \to \{0,1\}^n$ S-doboz $D(f)$ differenciális egyenletességét a következőképpen definiáljuk:*

$$D(f) = 2^m - \max_{a\in\{0,1\}^m, b\in\{0,1\}^n, a\neq 0} |\{x \in \{0,1\}^m : f(x)\oplus f(x\oplus a) = b\}|. \quad (3.11)$$

Terjedési kritériumok, szigorú lavinahatás kritérium. Szintén a nemlinearitással függenek össze a terjedési kritériumok:

3.12. definíció. *Egy $f:\{0,1\}^m \to \{0,1\}^1$ Boole-függvény kielégíti a k-adfokú terjedési kritériumot, ha autokorrelációs függvényére igaz, hogy $r_f(a) = 0$ minden k-nál nem nagyobb súlyú $a \in \{0,1\}^m$ esetén, ahol $a \neq 0$.*

3.13. definíció. *Egy $f:\{0,1\}^m \to \{0,1\}^n$ S-doboz kielégíti a k-adfokú terjedési kritériumot, ha minden komponens Boole-függvénye kielégíti azt.*

Az első fokú terjedési kritériumot szokás szigorú lavinahatás kritériumnak (Strict Avalanche Criterion – SAC)is nevezni. Egy SAC tulajdonságot kielégítő S-doboz tetszőleges bemeneti bitjét megváltoztatva a kimenet minden egyes bitje $\frac{1}{2}$ valószínűséggel változik meg.

S-dobozok esetén, a SAC tulajdonság teljesülésének ellenőrzésére a diffúziós mátrix szolgál:

3.14. definíció. *Egy $f:\{0,1\}^m \to \{0,1\}^n$ S-doboz diffúziós mátrixa alatt azt az $m \times n$ méretű W_f mátrixot értjük, melyre*

$$W_f^{(i,j)} = \frac{1}{2^m}|\{x \in \{0,1\}^m : e_n^{(j)} \odot f(x) \neq e_n^{(j)} \odot f(x\oplus e_m^{(i)})\}| \quad (3.12)$$

A diffúziós mátrix $W_f^{(i,j)}$ eleme tehát azt mondja meg, hogy megváltoztatva az f függvény i. bemeneti bitjét a kimenet j. bitje milyen valószínűséggel változik meg. Ha a diffúziós mátrix minden eleme $\frac{1}{2}$, akkor az S-doboz kielégíti a szigorú lavinahatás kritériumot.

A kritériumok közötti kapcsolatok. Bizonyítás nélkül közöljük a következő tételt, mely néhány kritérium közötti fontos kapcsolatra világít rá:

3.5. tétel. *Az $f:\{0,1\}^m \to \{0,1\}^n$ S-dobozra az alábbi állítások ekvivalensek:*

- *f nemlinearitása maximális, $N(f) = 2^{m-1} - 2^{\frac{m}{2}-1}$;*
- *f differenciálisan egyenletes, $D(f) = 2^{m-1} - 2^{m-n}$;*

Továbbá a fenti állítások bármelyikéből következik, hogy f kielégíti az m-edfokú terjedési kritériumot.

A 3.5. tétel szerint léteznek olyan S-dobozok, melyek a nemlinearitás, a differenciális egyenletesség és a terjedési kritériumok szempontjából is optimálisak. Ezeket az S-dobozokat **tökéletes S-doboz**oknak nevezzük. Ugyanakkor belátható, hogy a tökéletes S-dobozok nem teljesítik a szintén fontos balansz tulajdonságot.

S-doboz tervezés. A fenti kritétiumokat kielégítő S-dobozok tervezése nem könynyű feladat. Alapvetően két módszer létezik: a véletlen generálás és a konstrukció. A véletlen generálás esetén véletlen módon választunk S-dobozokat, majd ezek közül kiválasztjuk azokat, melyek kielégítik a kívánt kritériumokat. Ez csak akkor lehet hatákony módszer, ha a kívánt kritériumokat kielégítő S-dobozok száma nagy, ellenkező esetben ugyanis nincsen arra remény, hogy véletlen választással pont megfelelő S-dobozokat találunk. Az ily módon kielégíthető kritériumok közé tartozik a balansz tulajdonság és a teljesség. A tökéletes S-dobozok száma azonban nagyon kicsi, így ilyen S-dobozokat véletlen választassál keresni reménytelen. Ekkor folyamodhatunk a konstrukcióra épülő tervezési hozzáálláshoz, melynek segítségével közvetlenül a kívánt kritériumokat kielégítő S-dobozokat generálhatunk. Létezik például olyan konstrukciós eljárás, mellyel először generálunk egy tökéletes (maximális nemlinearitással rendelkező) S-dobozt, majd ezt módosítjuk oly módon, hogy az eredémyül kapott S-doboz kielégítse a balansz tulajdonságot és nemlinearitása ne csökkenjen jelentős mértékben.

Aszimmetrikus kulcsú rejtjelezés

Az aszimmetrikus kulcsú kriptográfia alapja, hogy a kódoló és a dekódoló transzformációt paraméterező kódoló és dekódoló kulcsok nem azonosak, sőt egyiket a másikból kiszámítani nagy komplexitású feladat, azaz praktikusan lehetetlen. Ezért a két kulcs közül az egyiket, általában a kódoló kulcsot, nyilvánosságra lehet hozni. Ez azért hasznos, mert a titkos kommunikáció megvalósításához nincsen szükség egy közös titkos kulcs létrehozására a küldő és a vevő között; elegendő a rejteni kívánt üzenetet a vevő nyilvános kódoló kulcsával rejtjelezni. Megjegyezzük azonban, hogy gondoskodni kell a nyilvános kulcsok hitelességéről. Más szavakkal, a küldőnek meg kell tudnia győződni arról, hogy a használni kívánt nyilvános kulcs valóban a vevő nyilvános kulcsa, és nem egy harmadik, feltehetően rossz-szándékú félé. A nyilvános kulcsú kriptográfia tehát oly módon egyszerűsíti a kulcscsere problémát, hogy titkos csatorna helyett, hiteles csatorna létezését követeli meg a vevő és a küldő között, melyen a vevő eljuttathatja nyilvános kulcsát a küldőnek.

Az alábbiakban ismertetjük az egyik leggyakrabban használt nyilvános kulcsú rejtjelező módszert, az RSA algoritmust, és áttekintjük annak ismert gyengeségeit. Mint látni fogjuk, az RSA kulcsok generálásához nagy prímszámokra van szükség. Ezért röviden összefoglaljuk a prímtesztelés módszereit is.

Az RSA algoritmus

Az RSA rejtjelező rendszer három algoritmusból áll: kulcsgenerálás, kódolás (rejtjelezés) és dekódolás.

A kulcsgenerálás a következőképpen történik: Véletlenszerűen választunk két nagy prímszámot, p-t és q-t, ahol $p \neq q$. Jelenleg nagy prímszámnak a legalább 500 bites prímszámokat tekintjük. Kiszámítjuk az $n = pq$ modulust és a $\varphi(n) =$

$(p-1)(q-1)$ szorzatot, melyek tehát legalább 1000 bites számok. Választunk továbbá egy $1 < e < \varphi(n)$ számot, amelyik $\varphi(n)$-hez relatív prím (azaz mind $(p-1)$-hez, mind $(q-1)$-hez relatív prím). Kiszámítjuk az e inverzét modulo $\varphi(n)$, azaz megkeressük azt a $1 < d < \varphi(n)$ számot, melyre $ed \bmod \varphi(n) = 1$. A nyilvános kulcs az (e,n) pár, a titkos privát kulcs pedig d.

A fenti számítások hatékonyan végrehajthatók. Egyedül d kiszámítása tűnik nehéznek, de ez is könnyen megoldható a **kiterjesztett euklidészi algoritmus**sal. A kiterjesztett euklidészi algoritmus meghatározza két szám, mondjuk a és b, legnagyobb közös osztóját, r-et, valamint azt a legkisebb t számot, melyre $t \cdot b \equiv r \pmod{a}$ fennáll. A kiterjesztett euklidészi algoritmus pszeudokódja a következő:

```
INPUT:  a (egyik szám)
        b (másik szám)

OUTPUT: r (a és b legnagyobb közös osztója)
        t (az a legkisebb szám melyre t*b mod a = r)

// feltesszük, hogy a > b
// (ellenkező esetben felcseréljük a bementeket)

x = a
y = b
t0 = 0
t1 = 1
r = b
t = 1
WHILE (x mod y) != 0
    q = x div y  // maradékos osztás eredménye
    r = x mod y  // és maradéka
    t = (t0 - q*t1) mod a
    x = y
    y = r
    t0 = t1
    t1 = t
ENDWHILE
RETURN r, t
```

Az RSA esetén $\varphi(n)$ és e legnagyobb közös osztója 1, ezért ha a kiterjesztett euklidészi algoritmust $a = \varphi(n)$-nel és $b = e$-vel hívjuk meg, akkor megkapjuk azt a t számot, melyre $t \cdot e \equiv 1 \pmod{\varphi(n)}$. Azaz a keresett d érték pontosan a t szám. Az euklidészi algoritmus legfeljebb kétszer annyi lépést igényel, mint az n modulus logaritmusa.

3.7. példa. Legyen $p = 73$, $q = 151$, így $n = 73 \cdot 151 = 11023$, $\varphi(n) = (73-1)(151-1) = 10800$. Az e paramétert $e = 11$-re választhatjuk, mivel $(10800, 11) = 1$, hiszen $10800 = 2^4 \cdot 3 \cdot 5^2 \cdot 9$. Az e inverzét modulo $\varphi(n)$ az kiterjesztett euklidészi algoritmussal határozzuk meg. Az algoritmusban szereplő változók értékét

az egyes iterációs lépésekben (a WHILE ciklus végrehajtása során) a következő táblázat tartalmazza:

iteráció	q	r	t	x	y	t_0	t_1	$x \bmod y$
WHILE előtt	–	–	–	10800	11	0	1	9
1.	981	9	9819	11	9	1	9819	2
2.	1	2	982	9	2	9819	982	1
3.	4	1	5891	2	1	982	5891	0

A táblázatból látható, hogy mikor a WHILE ciklus befejeződik, t értéke 5891, azaz $d = 5891$. A nyilvános kulcs tehát az $(e,n) = (11,11023)$ pár, és a titkos privát kulcs $d = 5891$.

A kódolás menete a következő: A küldő megszerzi a vevő (e,n) nyilvános kulcsát. A küldő előkódolja az elküldendő üzenetet, ami azt jelenti, hogy egy n-nél kisebb x egész számmá konvertálja azt. Ha az üzenet túl nagy akkor a küldő az üzenetet blokkokra osztja, és a blokkokat egyenként konvertálja n-nél kisebb egésszé. Végül a küldő kiszámolja az $y = x^e \bmod n$ rejtjeles üzenetet.

A dekódolás abból áll, hogy a vevő kiszámolja az $x' = y^d \bmod n$ számot, majd azt visszakonvertálja üzenetté az előkódolás inverz transzformációját használva.

A következő tétel azt mondja ki, hogy a dekódolás helyesen működik, azaz $x' = x$:

3.6. tétel. *Bármely x egész számra fennáll, hogy*

$$(x^e)^d \equiv x \pmod{n}$$

BIZONYÍTÁS: Mivel d az e inverze modulo $\varphi(n)$, ezért valamely v egész számra $ed = v\varphi(n)+1$, s így a tétel bizonyításához az

$$x \equiv x^{v\varphi(n)+1} \pmod{n}$$

modulo egyenlőséget kell belátni.

Először tetszőleges r prímre belátjuk, hogy tetszőleges x és s egész esetén fennáll a következő:

$$x \equiv x^{s(r-1)+1} \pmod{r} \tag{3.13}$$

Ha r prím nem osztója x-nek, akkor a Fermat-tétel (lásd 2.1 Lemma) szerint

$$\left(x^{r-1}\right)^s \equiv 1^s \equiv 1 \pmod{r}$$

Ebből azonnal következik, hogy ebben az esetben (3.13) fennáll. Ha r prím osztója x-nek, azaz $x \equiv 0 \pmod{r}$, akkor nyilván

$$x^{s(r-1)+1} \equiv 0 \pmod{r}$$

azaz (3.13) ekkor is fennáll.

Visszatérve az általános esetre, mivel $p-1\,|\,\varphi(n)$ és $q-1\,|\,\varphi(n)$, ezért (3.13) felhasználásával

$$x \equiv x^{v\varphi(n)+1} \pmod{p}$$
$$x \equiv x^{v\varphi(n)+1} \pmod{q}$$

fennállnak. Innen $p\,|\,(x^{v\varphi(n)+1}-x)$ és $q\,|\,(x^{v\varphi(n)+1}-x)$ teljesülnek, amelyből $pq\,|\,(x^{v\varphi(n)+1}-x)$ is következik, s ezzel az állítást beláttuk. ■

A kódolás és — a dekódoló kulcs ismeretében — a dekódolás is hatékonyan végrehajtható műveletek, ugyanis a moduláris hatványozásra létezik hatékony algoritmus, amit **ismételt négyzetreemelés és szorzás algorimus**nak neveznek. Ezt az algoritmust használva az e-edik hatvány kiszámítása legfeljebb $2\log_2 e$ számú modulo n szorzással megoldható. Az ismételt négyzetreemelés és szorzás algoritmus pszeudokódja a következő:

```
INPUT:  x (hatványozandó szám)
        e (k bites kitevő)
        n (modulus)

OUTPUT: x^e mod n

// az e kitevő i. legkisebb helyiértékű
// bitje e[i]

c = 1
FOR i = k-1 TO 0
    c = c^2 mod n
    IF e[i] = 1 THEN c = c*x mod n
ENDFOR
RETURN c
```

3.8. példa. Rejtjelezzük az $x = 17$ üzenetet az előző példában generált nyilvános kulcs segítségével. Az $e = 11$ kitevő bináris felbontása `1011`. Az ismételt négyzetreemelés és szorzás algoritmusát használjuk:

$$\begin{aligned}
&c = 1\\
&e[3] = 1 \;\rightarrow\; c = c^2 \cdot x \bmod n = 17\\
&e[2] = 0 \;\rightarrow\; c = c^2 \bmod n = 289\\
&e[1] = 1 \;\rightarrow\; c = c^2 \cdot x \bmod n = 1419857 \bmod 11023 = 8913\\
&e[0] = 1 \;\rightarrow\; c = c^2 \cdot x \bmod n = 1350506673 \bmod 11023 = 1782
\end{aligned}$$

azaz a rejtjelezett üzenet az $y = 1782$.

Az RSA biztonsága

Az RSA algoritmus biztonsága szorosan összefügg az egész számok faktorizációjának nehézségével. Pontosabban a d dekódoló kulcs kiszámítása a nyilvános (e,n) kódoló kulcsból ekvivalens feladat n faktorizációjával, amiről azt sejtik, hogy nem oldható meg hatékonyan. Az egyik irányt könnyű bizonyítani: Ha egy támadó hatékonyan tudná faktorizálni n-t, akkor p és q ismeretében ő is hatékonyan végre tudná hajtani a kulcsgenerálás algoritmusát, azaz ki tudná számolni e inverzét mod $(p-1)(q-1)$. Az is igaz, hogy ha létezne hatékony algoritmus d kiszámolására e-ből és n-ből, akkor ezen algoritmus segítségével hatékonyan tudnánk faktorizálni n-et. Ezt azonban nehezebb bizonyítani, ezért a bizonyítást itt nem ismertetjük.

Vegyük észre, hogy az RSA feltörése nem jelenti feltétlenül a d dekódoló kulcs megfejtését. A támadó célja valójában az, hogy az $y = x^e \bmod n$ rejtjeles üzenet és az (e,n) pár ismeretében kiszámolja az x nyílt üzenetet, azaz e-edik gyököt vonjon y-ból modulo n. Ezt nevezik RSA problémának. Nem tudjuk biztosan, hogy az RSA probléma és a faktorizáció problémája ekvivalensek-e. Mindenesetre a sejtés az, hogy az RSA probléma is nehéz feladat.

Az RSA implementációja és konkrét felhasználása során sok részleten múlhat a biztonság. Számos hatékony támadás ismert az üzenethalmaz, a prímpár, a kódoló és dekódoló kulcs speciális választása, valamint nem biztonságos protokollok kapcsán. Mindezt az alábbiakban három problémával illusztráljuk.

Fermat-faktorizáció: p és q közeli prímek. Sikeres támadásra van lehetőségünk, ha a p és q prímek távolsága nem eléggé nagy, ekkor ugyanis egyszerű faktorizációs lehetőség adódik az n modulusra: Tegyük fel, hogy $p > q$. Legyen $t = (p+q)/2$ és $s = (p-q)/2$, azaz $p = t+s$, $q = t-s$. Innen $n = pq = (t+s)(t-s) = t^2 - s^2$ adódik. Ha $p-q$ különbség kicsi, akkor s is kicsi, azaz $t \approx \sqrt{n}$. Találgassuk t értékét a következőképpen: Számítsuk ki a $t_1 = \sqrt{n}+1$, $t_2 = \sqrt{n}+2, \ldots$ értékeket, s ellenőrizzük hogy $t_i^2 - n$ különbség négyzetszámot ade. Ha igen, akkor a különbség egyenlő s^2-tel. A megkapott t és s alapján már könnyen kiszámítható p és q.

3.9. példa. Faktorizáljuk $n = 200819$-et. n lefelé kerekített négyzetgyöke 448. $t_1 = 449$, ám $449^2 - 200819 = 782$ nem négyzetszám. $t_2 = 450$, s $450^2 - 200819 = 1681 = 41^2$ már négyzetszám, ahonnan $200819 = 450^2 - 41^2 = (450+41)(450-41) = 491 \cdot 409$.

Kicsi kódoló kulcsok problémája. Tegyük fel, hogy egy központ r távoli ügynöknek azonos x üzenetet küld RSA rejtjelezéssel, ahol az ügynökök e nyilvános RSA kitevőjét azonosra választották, azon az alapon, hogy az RSA rejtjelezés biztonsága nem múlik ezen kitevő választásán. Az ügynökök modulusai rendre $n_1, n_2, \ldots, n_r$. A csatornákat lehallgató támadó az alábbi rejtjeles üzeneteket szerzi meg.

$$y_1 = x^e \bmod n_1$$

$$y_2 = x^e \bmod n_2$$
$$\vdots$$
$$y_r = x^e \bmod n_r$$

Tegyük fel, hogy e kicsi, s ezért fennáll az $e \leq r$ egyenlőtlenség. Ekkor a támadó a **kínai maradéktétel**t használhatja az x üzenet megfejtésére, mely a következőképpen szól:

3.7. tétel. *Tekintsük az alábbi kongruencia-rendszert, ahol az $n_1, n_2, \ldots n_r$ pozitív egész számok páronként relatív prímek:*

$$X \equiv y_1 \pmod{n_1}$$
$$X \equiv y_2 \pmod{n_2}$$
$$\vdots$$
$$X \equiv y_r \pmod{n_r}$$

Ennek a kongruencia-rendszernek létezik egyértlemű megoldása modulo $N = n_1 \cdot n_2 \cdot \ldots \cdot n_r$, s ez a következő alakú $X = (y_1 \cdot N_1 \cdot M_1 + y_2 \cdot N_2 \cdot M_2 + \ldots + y_r \cdot N_r \cdot M_r) \mod N$, ahol $N_i = N/n_i$ és $M_i = N_i^{-1} \pmod{n_i}$ $(i = 1, 2, \ldots, r)$.

Jelen esetben feltehetjük, hogy az $n_1, n_2, \ldots, n_r$ modulusok páronként relatív prímek, ugyanis ügynökönként véletlenszerűen választottuk a prímpárt, így ezen feltevés teljesülése gyakorlatilag biztos. A kínai maradéktételt alkalmazva, a támadó hatékonyan ki tudja számítani az $X = x^e \bmod n_1 \cdot n_2 \cdot \ldots \cdot n_r$ maradékot. Vegyük észre, hogy $x < n_i$ $(i = 1, 2, \ldots, r)$ valamint $e \leq r$ miatt $x^e < n_1 \cdot n_2 \cdot \ldots \cdot n_r$, azaz az X megoldás e-edik gyökét kiszámítva a támadó magát a keresett x nyílt szöveget kapja.

Közös modulus problémája. Tudjuk, hogy az RSA kódolás igen számításigényes, ezért arra a gondolatra juthatunk, hogy célhardverben legyártatjuk egy választott n modulus (egy adott (p, q) prímpár) esetére a modulo n hatványozó aritmetikát. Mivel azonban több felhasználós a rendszer, a központ úgy különbözteti meg az egyes felhasználókat, hogy az adott n modulus mellett különböző nyilvános kitevőket választ számukra, amelyekből különböző dekódoló kulcsok adódnak. A központ ezek után egy x titkos üzenetet küld két felhasználónak, A-nak és B-nek. A támadó lehallgatja a csatornákat, s ezzel megismeri az $y_A = x^{e_A} \bmod n$ és az $y_B = x^{e_B} \bmod n$ rejtett szövegeket. Tegyük fel, hogy e_A és e_B relatív prímek. Így létezik t és s egész, hogy $t \cdot e_A + s \cdot e_B = 1$, ahol $t \cdot s < 0$, mivel $e_A > 0$ és $e_B > 0$. Az általánosság korlátozása nélkül tegyük fel, hogy $t < 0$, azaz $t = -1 \cdot |t|$. Továbbá feltehető, hogy y_A és n is relatív prímek, ezért létezik az y_A^{-1} inverz modulo n. Innen

$$\left(y_A^{-1}\right)^{|t|} \cdot (y_B)^s \equiv (x^{e_A})^t \cdot (x^{e_B})^s \equiv x^{t \cdot e_A + s \cdot e_B} \equiv x \pmod{n}$$

alapján a támadó, kizárólag nyilvános információk és a lehallgatott rejtett szövegpár felhasználásával, dekódolni képes a titkos üzenetet.

Prímszámok keresése

Az RSA kapcsán szeretnénk egy hatékony (azaz polinomiális idejű) algoritmust annak eldöntésére, hogy egy véletlenül kisorsolt egész szám prím-e vagy sem. Megnyugtató, hogy a prímek elég sűrűn találhatók az egész számok között, s így, ha véletlenszerűen választunk a páratlan egész számok közül, az alkalmazások szempontjából elegendően nagy valószínűséggel prímszámot választunk.

A prímek sűrűségével kapcsolatban bizonyítás nélkül utalunk a számelmélet ismert tételére, amely kimondja, hogy az n pozitív egésznél kisebb prímek $\pi(n)$ számára:

$$\pi(n) \cong \frac{n}{\ln n}.$$

3.10. példa. Számpéldaként tekintsünk egy 10^{200} nagyságrendű modulust, azaz $10^{100} = 2^{332}$ nagyságrendű prímeket választunk. Annak a valószínűsége, hogy egy véletlenszerűen választott 332 bites s szám prím, a fenti kifejezés alapján becsülhető:

$$\frac{\pi(2^{332}) - \pi(2^{331})}{2^{332} - 2^{331}} \cong \frac{2^{331}\frac{1}{331 \cdot \ln 2}}{2^{331}} \cong \frac{1}{230}$$

Továbbá, mivel az adott számtartomány fele páros szám, elég csak a páratlanok közül válogatnunk, s ezzel $1/115$-re duplázzuk meg a találati valószínűséget. Így tehát átlagosan 115 próbálkozásonként jutunk prímszámhoz a 10^{100} nagyságrendű számok között.

Ezek után már csak az a kérdés, hogy hogyan vesszük észre, hogy prímet sorsoltunk ki. Ez utóbbi feladatra szolgálnak a prímtesztelő algoritmusok. Több ilyen algortimus is létezik. Az alapelveket a Fermat prímtesztelő algoritmus kapcsán mutatjuk be.

A Fermat-prímteszt. Elsőként bevezetjük a **Fermat-álprím** fogalmát:

3.15. definíció. *Egy n összetett szám Fermat-álprím egy b bázisra nézve, ha*

$$b^{n-1} \equiv 1 \pmod{n} \tag{3.14}$$

ahol $b \in W_n$, $W_n = \{z : 1 < z < n \text{ és } (z,n) = 1\}$.

A 3.15. definíció a Fermat-tételen alapul, amely szerint, ha n prímszám, akkor (3.14) teljesül (lásd 2.1 Lemma). Ha n összetett szám, de mégis teljesíti (3.14) egyenletet, akkor álprím.

A Fermat-prímteszt tehát a következő: Ha egy véletlenszerűen sorsolt n egész szám egy b bázissal nem teljesíti a (3.14) egyenletet, akkor biztosan nem prím, míg ha teljesíti, akkor prímgyanús és tovább vizsgálandó egy következő bázissal. Ha a tesztelt szám k egymás utáni vizsgálat során prímgyanúsnak bizonyult, akkor nagy valószínűséggel prímszám.

A kérdés az, hogy van-e olyan összetett szám, amely tetszőleges szóbajövő bázisra átmegy a Fermat-próbán? Sajnos van, s ezen számokat **Carmichael-számok**nak nevezzük. Bizonyságul a legkisebb ilyen szám az 561.

Felmerül továbbá az a kérdés is, hogy mekkora a valószínűsége annak az eseménynek, hogy egy n összetett szám, amely nem Carmichael-szám, egy véletlenszerűen választott $b \in W_n$ bázissal kielégíti (3.14) egyenletet. Erre vonatkozik a következő tétel:

3.8. tétel. *Ha egy n összetett szám nem Carmichael-szám, akkor legfeljebb $\frac{n}{2}$ különböző $b \in W_n$ bázissal elégíti ki (3.14) egyenletet.*

A 3.8. tétel következménye, hogy ha egy n összetett szám nem Carmichael-szám, akkor annak valószínűsége, hogy k alkalommal átmegy a Fermat-próbán, kisebb, mint 2^{-k}.

3.2. Blokkrejtjelezési módok

Egy blokkrejtjelező n bit hosszú nyílt szöveg blokkokat kódol n bit hosszú rejtjeles blokkokba. n tipikus értéke 64 vagy 128. Sok alkalmazásban ennél jóval hosszabb (vagy rövidebb) üzeneteket szeretnénk rejtjelezni. Ezért szükségünk van olyan eljárásokra, melyek lehetővé teszik tetszőleges hosszúságú üzenetek kódolását egy n bites blokkrejtjelező segítségével. Ezeket az eljárásokat blokkrejtjelezési módoknak nevezzük.

Az öt leggyakrabban használt blokkrejtjelezési mód a következő: ECB (Electronic Code Book), CBC (Cipher Block Chaining), CFB (Cipher Feedback), OFB (Output Feedback) és CTR (Counter). Ezek működését mutatjuk be az alábbiakban.

Az ECB mód

ECB módban az X nyílt üzentet n bites blokkokra osztjuk. Jelölje $X_1, X_2, \ldots, X_N$ az így nyert nyílt blokkokat. Az i. rejtjeles blokkot, Y_i-t, a következőképpen kapjuk:

$$Y_i = E_K(X_i) \quad i = 1, 2, \ldots, N, \tag{3.15}$$

ahol E_K jelöli a blokkrejtjelező kódoló függvényét a K kulccsal paraméterezve. A dekódolás hasonlóan egyszerű:

$$X_i = D_K(Y_i) \quad i = 1, 2, \ldots, N, \tag{3.16}$$

ahol D_K jelöli a blokkrejtjelező dekódoló függvényét a K kulccsal paraméterezve (azaz az E_K függvény inverzét).

Előfordulhat, hogy a rejtjelezni kívánt nyílt üzenet bitekben mért hossza nem többszöröse a rejtjelező n blokkhosszának. Ekkor az üzenetet ki kell tölteni néhány további bit hozzáadásával oly módon, hogy a kitöltéssel együtt az üzenet hossza n

többszörese legyen. Ez sok esetben csak az utolsó, csonka blokk kitöltését jelenti, de néha egy teljes blokk hozzáadásával is járhat. Sőt, vannak olyan kitöltési sémák is, melyek több, véletlenszámú blokkot is hozzáadnak az üzenethez kitöltésként, ezzel próbálva az üzenet valódi hosszát elrejteni egy forgalmat analizáló támadó elől.

Akármennyi bitet is alkalmazunk kitöltésként, egy dolgot minden esetben biztosítani kell: a vevőnek képesnek kell lennie a kitöltésként alkalmazott bitek azonosítására, majd azok eltávolításával az eredeti nyílt üzenet visszaállítására. Ennek elérése érdekében sok kitöltő sémában a kitöltés utolsó bájtja a kitöltés hosszára vonatkozó információt tartalmaz. Ezen sémák használata esetén minden üzenetnek tartalmaznia kell kitöltést, még azoknak is, amelyek hossza n többszöröse. Ha ugyanis megengednénk olyan üzeneteket, melyek nem tartalmaznak kitöltést, akkor egy adott üzenet vétele esetén a vevő nem tudná egyértelműen megállapítani, hogy az üzenet tartalmaz-e kitöltést vagy sem. Ha viszont megköveteljük, hogy minden üzenet tartalmazzon kitöltést, akkor ilyen probléma nem lép fel.

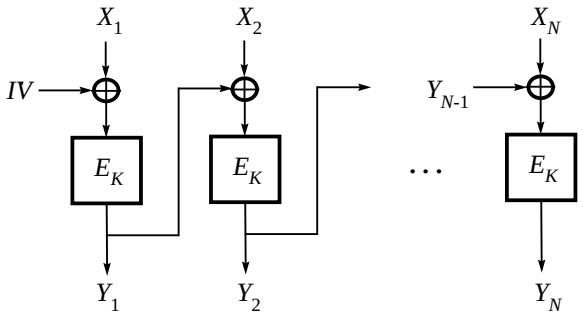
Biztonság. ECB mód használata esetén, azonos nyílt blokkokhoz mindig azonos rejtjeles blokkok tartoznak (feltéve persze, hogy minden blokk ugyanazzal a K kulccsal lett kódolva). Ez azt jelenti, hogy ha adott két különböző X és X' nyílt szöveghez tartózó Y és Y' rejtjeles szöveg, akkor egy támadó a K kulcs ismerete nélkül is azonosítani tudja X és X' egyforma blokkjait Y és Y' egyforma blokkjainak azonosításával.

Ha a támadó valamilyen módon nagy számú nyílt blokk – rejtjeles blokk párhoz jut, akkor ezekből egy szótárat (vagy kódkönyvet) építhet. Ezt a szótárat aztán később megfigyelt rejtjeles szövegek részleges dekódolásához használhatja a K kulcs ismerete nélkül.

További probléma, hogy egy rejtjeles üzenet blokkjai egymással tetszőleges módon felcserélhetők, törölhetők, vagy más rejtjeles üzenetekből kimásolt blokkokkal helyettesíthetők. Az ECB mód nem teszi lehetővé az ilyen jellegű blokkcserék, törlések és helyettesítések megbízható detektálását, mivel a rejtjeles blokkok nem függnek a szomszédos blokkoktól.

Bár a fent említett problémák a nyílt üzenet rejtjelezés előtti tömörítésével és integritásvédelmi mechanizmusok (pl. kriptográfiai ellenőrzőösszeg) alkalmazásával enyhíthetők, az ECB mód mégsem javasolt egynél több blokkból álló nyílt üzenet rejtjelezésére. Helyette a később ismertetett CBC mód használata ajánlott.

Hatékonyság. Az ECB mód előnye, hogy a kódolás és dekódolás sebessége megegyezik a blokkrejtjelező sebességével, valamint az, hogy a kódolás és a dekódolás is párhuzamosítható. Továbbá, az ECB mód lehetővé teszi a nyílt blokkokhoz történő véletlen hozzáférést a rejtjeles szövegben. Ez azt jelenti, hogy a rejtjeles szöveg tetszőleges sorszámú blokkját dekódolni tudjuk a többi blokk dekódolása nélkül, sőt a dekódolt blokk módosítása majd újrakódolása sem érinti a



3.11. ábra. Kódolás CBC módban.

többi blokkot. Ez igen előnyös tulajdonság például adatbázisfájlok rejtjelezésénél, ami a rekordok hatékony lekérdezését és módosítását teszi lehetővé.

Hibaterjedési tulajdonságok. A biztonsággal és hatékonysággal kapcsolatos tulajdonságok mellett meg szokták még említeni egy adott blokkrejtjelezési mód ún. hibaterjedési tulajdonságait. Egészen pontosan arra vagyunk kíváncsiak, hogy a rejtjeles üzenet átvitele során keletkezett egy bites hibának milyen hatása van a dekódolt nyílt üzenetre. Az egy bites hiba lehet egy bit értékének változása, valamint egy bit törlése vagy beszúrása a rejtjeles üzenetbe. Az utóbbit kerethibának (framing error) nevezzük.

ECB mód esetén a rejtjeles szöveg egy bitjének megváltozása csak a bitet tartalmazó rejtjeles blokkhoz tartozó nyílt blokkra van hatással, mégpedig oly módon, hogy az adott nyílt blokk teljes egészében használhatatlan pénzfeldobás sorozattá válik (a blokk minden bitje 1/2 valószínűséggel lesz helyes dekódolás után). Ezzel szemben egy bit beszúrása vagy törlése a hiba helye után található blokkhatárok elcsúszását eredményezi (innen a kerethiba elnevezés), és ily módon hatással van a beszúrást illetve törlést tartalmazó rejtjeles blokkhoz tartozó nyílt blokkra és az összes azt követő nyílt blokkra is. Ezek a blokkok használhatatlan pénzfeldobás sorozattá válnak.

A CBC mód

A CBC mód működését a 3.11. ábra szemlélteti. Az ECB módhoz hasonlóan a nyílt üzenetet először kitöltjük, majd a kitöltött nyílt üzenetet n bit hosszúságú blokkokra osztjuk. Ezután az i. rejtjeles blokkot úgy állítjuk elő, hogy az i. nyílt blokkot XOR-oljuk az $(i-1)$. rejtjeles blokkal, majd az eredményt kódoljuk a blokkrejtjelezővel. A dekódolásnál az i. rejtjeles blokkot dekódoljuk, majd az eredményt XOR-oljuk az $(i-1)$. rejtjeles blokkal és így kapjuk az i. nyílt blokkot.

Jelölje ismét $X_1, X_2, \ldots, X_N$ a nyílt blokkokat és $Y_1, Y_2, \ldots, Y_N$ a rejtjeles blokkokat. A kódolást tehát a következőképpen írhatjuk le formálisan:

$$Y_1 = E_K(X_1 \oplus IV) \tag{3.17}$$

$$Y_i = E_K(X_i \oplus Y_{i-1}) \quad i = 2, 3, \ldots, N \tag{3.18}$$

ahol E_K jelöli a blokkrejtjelező K kulccsal paraméterezett kódoló függvényét, $\oplus$ a bitenkénti XOR művelet, és IV egy kezdeti változó (Initial Vector – IV). Az IV az előző rejtjeles blokk szerepét tölti be az első nyílt blokk kódolásánál, ahol még nem áll rendelkezésre egy valódi előző rejtjeles blokk. A dekódolás képlete a következő:

$$X_1 = D_K(Y_1) \oplus IV \tag{3.19}$$

$$X_i = D_K(Y_i) \oplus Y_{i-1} \quad i = 2, 3, \ldots, N \tag{3.20}$$

ahol D_K jelöli a blokkrejtjelező K kulccsal paraméterezett dekódoló függvényét.

Mint látható, az első blokk dekódolásánál szükség van az IV-re, így gondoskodni kell ennek a vevőhöz történő eljuttatásáról. Ez megoldható úgy, hogy az IV-t ECB módban rejtjelezzük, majd a kódolt IV-t az átküldött rejtjeles üzenet elé csatoljuk. Az IV rejtjelezéséhez ugyanazt a kulcsot használjuk, mint az üzenet rejtjelezéséhez. A vevő először az IV-t dekódolja, majd ennek ismeretében a rejtjeles üzenetből (3.19) és (3.20) szerint visszaállítja a nyílt üzenetet.

Valójában az IV titkossága nem követelmény, ügyelni kell azonban az IV integritásának védelmére. Erre azért van szükség, mert (3.19) szerint az IV értéke közvetlen hatással van az első visszaállított nyílt blokkra. Az IV bitjeinek módosításával tehát egy támadó az első nyílt blokk adott bitjeit manipulálni (invertálni) tudja. A rejtjelezés alkalmazása ezt megakadályozza, mert a rejtjelezett IV módosítása a támadó által nem predikálható változásokat eredményez a dekódolt IV-ben és így a visszaállított első nyílt blokkban is.

Biztonság. A CBC mód egyik legfontosabb tulajdonsága, hogy az Y_i rejtjeles blokk értéke nemcsak az X_i nyílt blokktól függ, hanem az azt megelőző blokkoktól és az IV-től is[5]. Ennek a tuljadonságának több kedvező hatása is van. Egyrészt azonos nyílt blokkokhoz nagy valószínűséggel különböző rejtjeles blokkok tartoznak. Ez akkor is igaz, ha az azonos nyílt blokkok egy nyílt üzenet részei és akkor is, ha két különböző nyílt üzenethez tartoznak. Pontosabban, ha két különböző X és X' nyílt üzenetben valamely X_i és X'_j blokkok azonosak, akkor az ezen blokkokhoz tartozó Y_i és Y'_j rejtjeles blokkok nagy valószínűséggel csak akkor lesznek azonosak, ha $i = j$ és minden $k = 1, 2, \ldots, i-1$ esetén $X_k = X'_k$, valamint $IV = IV'$ (azaz X és X' minden i-nél kisebb sorszámú blokkja, valamint a két kezdeti változó értéke is megegyezik). Ebből az is látszik, hogy ha teljesen azonos nyílt üzeneteket

[5]Meg kell azonban jegyezni, hogy ez a függés csak az előző rejtjeles blokkon, Y_{i-1}-en keresztül valósul meg.

különböző IV-t alkalmazva rejtjelezünk, akkor különböző rejtjeles üzenetekhez jutunk. Az IV üzenetenkénti változtatása megoldható az IV véletlen módon történő generálásával, vagy egy sorozatszám IV-ként történő használatával.

Az ECB móddal ellentétben, a CBC mód korlátozott mértékben lehetőséget nyújt a rejtjeles blokkok felcserélésének, törlésének és beszúrásának detektálására. Ez azért van, mert egy Y_i rejtjeles blokkot csak akkor lehet helyesen dekódolni, ha azt a helyes Y_{i-1} blokk előzi meg. Két blokk, Y_i és Y_j átvitel során történő felcserélésekor azonban Y_i dekódolásánál Y_{j-1}-et használja a vevő. Az így eredményül kapott $D_K(Y_i) \oplus Y_{j-1}$ egy pénzfeldobás sorozat lesz, amit a vevő könnyen kiszűrhet mint egy várt struktúrától való durva eltérést. Meg kell azonban jegyezni, hogy a nyílt üzenet maga is tartalmazhat véletlen blokkokat (pl. egy véletlen módon generált tranzakció azonosítót), ezért a véletlen blokkok detektálására alapozott integritásvédelem nem mindig megbízható. Általában is igaz, hogy ha az alkalmazás megkívánja az üzenetek integritásának védelmét, akkor erre a célra speciális integritásvédő mechanizmusokat (lásd később) ajánlott használni és nem tanácsos kizárólag a CBC mód által nyújtott lehetőségekre hagyatkozni.

A CBC mód egy ismert gyengesége a következő: Ha két rejtjeles blokk, Y_i és Y'_j megegyezik, akkor (3.18) használatával kapjuk, hogy

$$Y_{i-1} \oplus Y'_{j-1} = X_i \oplus X'_j. \tag{3.21}$$

Mivel a támadó Y_{i-1}-et és Y'_{j-1}-et ismeri, ezért (3.21) alapján két nyílt blokk XOR összegére vonatkozóan jut információhoz. Innen a nyílt üzenetek redundanciáját felhasználva, a támadó sikerrel kísérelheti meg X_i és X'_j megfejtését. A gyakorlatban ennek a támadásnak az szab korlátot, hogy a rejtjeles blokkok egyenletes eloszlásúak, ezért annak a valószínűsége, hogy kettő megegyezik a születésnapi paradoxon alapján becsülhető. A támadás akkor hatékony, ha a megfigyelt rejtjeles blokkok száma $2^{\frac{n}{2}}$ nagyságrendjébe esik. A tipikus $n = 64$ esetben ez több gigabájt adat megfigyelését igényli.

Hatékonyság. A CBC módban történő kódolás és dekódolás sebessége lényegében megegyezik a blokkrejtjelező sebességével, mert a blokkonként végrehajtott XOR művelet által okozott késleltetés elhanyagolható a blokkrejtjelezéséhez szükséges időhöz képest. A CBC mód hátránya, hogy a kódolás nem párhuzamosítható. Egy másik hátrány, hogy egy nyílt blokk módosítása az összes őt követő blokk újrakódolását igényli. Ezzel szemben a dekódolás párhuzamosítható és egy nyílt blokkhoz történő módosítás nélküli hozzáférés (olvasás) nem érinti a többi blokkot. Az ECB móddal ellentétben tehát a nyílt blokkokhoz történő véletlen hozzáférés a rejtjeles szövegben csak olvasás esetén hatékony.

Hibaterjedési tulajdonságok. Ha a rejtjeles üzenet átvitele során az i. rejtjeles blokk j. bitje megváltozik, akkor ez hatással lesz az i. és az $(i+1)$. nyílt blokk visszaállítására. Egészen pontosan az i. visszaállított nyílt blokk pénzfeldobás sorozat lesz, míg az $(i+1)$. visszaállított nyílt blokkban csak a j. bit lesz hibás. Az

$(i+2)$. nyílt blokkra a hiba hatása már nem terjed ki. Az egy bithibából való felépülés képességét **önszinkronizáció**nak is nevezzük. A CBC mód tehát ebben az értelemben önszinkronizáló.

A CBC mód azon tulajdonságát, hogy az i. rejtjeles blokk j. bitjének megváltoztatása az $(i+1)$. visszaállított nyílt blokkban csak a j. bit megváltozását eredményezi, egy támadó bizonyos esetekben sikerrel használhatja ki egy adott nyílt blokk bitjeinek manipulálására. Ezért mégegyszer hangsúlyozzuk a kriptográfiai integritásvédő mechanizmusok alkalmazásának szükségességét.

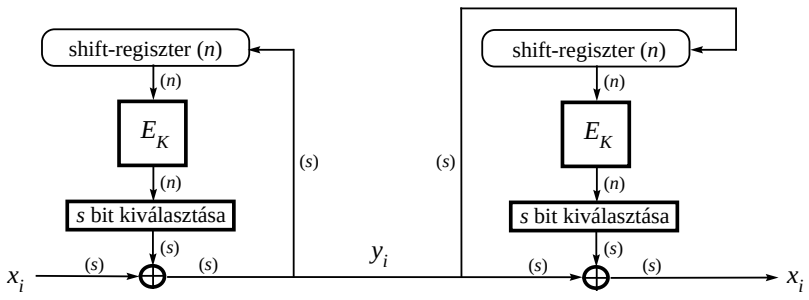
Az ECB módhoz hasonlóan egy bit elvesztése vagy beszúrása a rejtjeles üzenetből illetve üzenetbe a blokkhatárok elcsúszását eredményezi, és a hiba helyétől kezdve minden visszaállított nyílt blokk használhatatlanná válik. Bitvesztésből és -beszúrásból tehát nem áll helyre a rendszer.

A CFB mód

Vannak olyan alkalmazások, melyekben a rejtjelező blokkhosszánál rövidebb üzeneteket (pl. karaktereket vagy akár biteket) kell rejtjelezni. Ráadásul az alkalmazás késleltetésre vontakozó követelményei olyanok lehetnek, hogy a küldőnek nincs ideje megvárni, míg e rövid üzenetekből összegyűlik egy blokkra való. Rövid üzenetenként egy blokkot elküldeni viszont pazarló lenne, így a CBC mód nem jöhet szóba. A megoldás a CFB mód alkalmazása lehet, mely a blokkrejtjelezőt egy önszinkronizáló kulcsfolyam rejtjelezővé alakítja.

A CFB mód működését a 3.12. ábra szemlélteti. Tegyük fel, hogy s bites karaktereket szeretnénk rejtjelezni. Ekkor az i. karakter, x_i rejtjelezéshez egy n bites belső shift-regiszter tartalmát rejtjelezzük a blokkrejtjelezővel, majd az eredmény első s bitjét XOR-oljuk x_i bitjeivel és így kapjuk az i. rejtjeles karaktert, y_i-t. Ezt egyfelől elküldjük a vevőnek, másfelől jobbról beléptetjük a shift-regiszterbe. A következő karakter rejtjelezésekor tehát már a shift-regiszter új tartalmát fogjuk rejtjelezni. A dekódolás hasonlóan történik. A shift-regiszter aktuális tartalmát rejtjelezzük, majd az eredmény első s bitjét XOR-oljuk az y_i rejtjeles karakter bitjeivel és így kapjuk vissza az x_i nyílt karaktert. Ezután y_i-t jobbról beléptetjük a shift-regiszterbe. Figyeljük meg, hogy a kódolásnál és a dekódolásnál is a blokkrejtjelező E_K kódoló függvényét használjuk.

Mielőtt az első karaktert kódolnánk, a shift-regisztert fel kell tölteni egy kezdeti értékkel, amit a CBC módhoz hasonlóan itt is IV-nek nevezünk. Az első néhány rejtjeles karakter dekódolásához a vevőnek is szüksége van a küldő által használt IV-re, így azt valamilyen módon el kell juttatni a vevőhöz. Mivel CFB módban az IV a rejtjelező függvényen keresztül lép be a folyamatba, ezért az IV nyíltan is elküldhető a vevőnek. Egyrészt, az IV ismerete nem segíti a támadót, mert a blokkrejtjelező által használt K kulcs nélkül úgysem tudja megfejteni az üzenetet. Vegyük észre, hogy a shift-regiszter tartalma úgyis ismertté válik a támadó számára az első n/s karakter feldolgozása után, hiszen a támadó által is megfigyelhető rejtjeles karaktereket léptetjük be a shift-regiszterbe. A rendszer biztonsága tehát



3.12. ábra. Kódolás és dekódolás CFB módban.

nem a shift-regiszter állapotának titokban tartására épül, és ez a kezdeti állapotra éppen úgy igaz, mint a későbbi állapotokra. Másrészt, a blokkrejtjelező tulajdonságai miatt, az IV bitjeinek módosítása a támadó által nem predikálható változásokat eredményez az első (és esetleg néhány további) visszaállított nyílt karakterben. A támadó tehát az IV manipulálásával nem tudja a nyílt szöveg kiválasztott bitjeit manipulálni.

Az IV vevőhöz történő eljuttatásának legegyszerűbb módja az, amikor a küldő az üzenet karakterei előtt, nyíltan küldi el az IV-t a vevőnek. A vevő nem alkalmaz külön eljárást az IV feldolgozására, hanem ugyanazt a dekódoló algoritmust futtatja, mint a rejtjeles karakterek vételénél. Így a vevő karakterenként lépteti be az IV-t a shift-regiszterébe. A dekódoló kimenetén eközben generálódó karaktereket a vevő eldobja. Mire az első rejtjeles karakter megérkezik, a vevő shift-regisztere már helyes állapotban van, és így a rejtjeles karakter dekódolása sikeres lesz. Ez az eljárás a CFB mód önszinkronizáló tulajdonságát használja ki, azaz azt, hogy n/s rejtjeles karakter helyes vétele után a dekódoló helyes állapotba kerül, függetlenül attól, hogy milyen állapotban volt a karakterek vétele előtt.

Megjegyezzük, hogy bár a CFB mód bevezetését a rövid üzenetek rejtjelezésének képessége motiválta, semmi nem szól az ellen, hogy teljes blokkok rejtjelezésére is használjuk. Más szóval, lehetséges az $s = n$ eset. Ekkor az ECB és CBC módok mintájára, a CFB kódoló működését a következő módon írhatjuk le:

$$Y_1 = X_1 \oplus E_K(IV) \tag{3.22}$$

$$Y_i = X_i \oplus E_K(Y_{i-1}) \quad i = 2,3,\ldots \tag{3.23}$$

A dekódoló működését pedig a következő egyenletek definiálják:

$$X_1 = Y_1 \oplus E_K(IV) \tag{3.24}$$

$$X_i = Y_i \oplus E_K(Y_{i-1}) \quad i = 2,3,\ldots \tag{3.25}$$

Biztonság. A CFB mód hasonló láncolási tulajdonságokkal rendelkezik, mint a CBC mód: az i. rejtjeles karakter értéke nemcsak az i. nyílt karaktertől függ, hanem

az összes azt megelőző nyílt karaktertől és az IV-től is. Ez a függés azonban csak az előző n/s rejtjeles karakteren keresztül valósul meg. Ez tehát azt jelenti, hogy egy adott rejtjeles karakter helyes dekódolásához elegendő a megelőző n/s rejtjeles karakter helyes értékének ismerete. Mindenesetre, ha két azonos nyílt szöveget különböző IV-ket alkalmazva rejtjelezünk, akkor a rejtjeles szövegek különbözők lesznek. Továbbá, rejtjeles karakterek módosítása, törlése, beszúrása, és felcserélése a hiba helye után (mindaddig amíg a hibás karakterek a shift-regiszterben jelen vannak) pénzfeldobás sorozatokól álló nyílt karakterek visszaállítását eredményezi, tehát sok esetben detektálható.

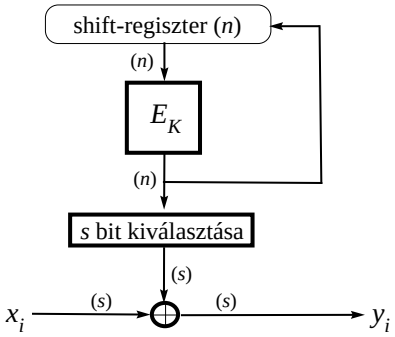
Hatékonyság. CFB módban a kódolás és a dekódolás sebessége a blokkrejtjelező sebességének s/n-szerese. Ez $s = n$ esetén egyenlőséget jelent, $s < n$ esetén viszont a CFB módú kódolás sebessége kisebb, mint a blokkrejtjelező sebessége, ami egyszerűen abból adódik, hogy a blokkrejtjelező kimenetének csak töredék részét használjuk. A CBC módhoz hasonlóan a dekódolás CFB módban is párhuzamosítható, a kódolás viszont nem. Tetszőleges rejtjeles karaktert dekódolni lehet a többi karakter dekódolása nélkül, ha viszont a dekódolt karaktert módosítani is akarjuk, akkor az összes őt követő karaktert újra kell rejtjelezni. A nyílt karakterekhez való véletlen hozzáférés a rejtjeles szövegben tehát csak olvasás esetén hatékony.

Hibaterjedési tulajdonságok. Tegyük fel, hogy egy rejtjeles karakter egy bitje megváltozik. Mivel a hibás rejtjeles karakter bekerül a shift-regiszterbe, a hiba hatása kiterjed és mindaddig érezhető lesz, amíg a hibás karakter a shift-regiszterben van. Egészen pontosan, a j. bit hibája egy rejtjeles karakterben a j. bit hibáját okozza az adott rejtjeles karakterből visszaállított nyílt karakterben, az azt követő n/s visszaállított nyílt karakter pedig használhatatlan pénzfeldobás sorozat lesz. Egy támadó tehát egy adott nyílt karakter bitjeit manipulálni tudja, bár az adott karaktert követően megjelenő véletlen karakterek azonosítása a támadás detektálásához vezethet. A véletlen karakterek azonban lehet, hogy túl későn érkeznek, és addigra a támadó már elérte a célját. Továbbá, az utolsó átküldött karakter mindig támadható, mert azt nem követi semmi, amiből a támadásra következtetni lehetne.

CFB mód használata esetén bitvesztésből és -beszúrásból származó hibákból is felépül a rendszer, mert a hiba után érkező helyes karakterek „kitolják” a hibát a shift-regiszterből és n/s lépés után visszaáll a helyes állapot.

Az OFB mód

A CFB mód hátránya, hogy egy egy bites hiba a rejtjeles szövegben több dekódolt karaktert tesz használhatatlanná. Előfordulhat, hogy az alkalmazás jellege nem teszi lehetővé ezen hibák dekódolás előtti javítását (pl. a vett karaktereket azonnal fel kell dolgozni, de nem hatékony minden átküldött rejtjeles karakterhez hibajavítást lehetővé tevő redundáns biteket csatolni). Ezért ebben az esetben, ha



3.13. ábra. Kódolás OFB módban. A dekódolás ugyanilyen séma szerint történik, csak a rejtjeles karakterek lépnek be, és a nyílt karakterek lépnek ki.

a bithiba valószínűsége nagy (azaz a csatorna zajos), akkor a CFB mód nem használható. Ilyenkor OFB vagy CTR módot használhatunk. Ezen módok egyik közös jellemezője, hogy a blokkrejtjelezőt egy szinkron kulcsfolyam rejtjelzővé alakítják, melyben a rejtjelező belső állapota (a shift-regiszter tartalma) nem függ sem a nyílt sem a rejtjeles karakterektől, és így egy rejtjeles karakterben bekövetkező bithiba hatása nem terjed ki más karakterekre.

Az OFB mód működését a 3.13. ábra szemlélteti. Az ábrán a kódolás vázlata látható. Mivel szinkron kulcsfolyam rejtjelezők esetében a dekódolás sémája megegyezik a kódolással (csupán annyi a különbség, hogy a bemeneten rejtjeles karakterek lépnek be és a kimeneten nyílt karakterek jönnek ki), ezért a dekódoló működését sem OFB sem CTR mód esetén nem illusztráljuk külön ábrán.

Az OFB kódoló vázlata feltűnő hasonlóságot mutat a CFB kódoló vázlatával. A két rendszer között annyi a különbség, hogy OFB esetén nem a rejtjeles karakter van visszacsatolva, hanem a blokkrejtjelező kimenete. Az i. nyílt karakter, x_i rejtjelezése tehát a CFB módhoz hasonlóan történik: a shift-regiszter aktuális tartalmát rejtjelezzük, majd az eredmény első s bitjét XOR-oljuk x_i-vel (melyről feltesszük, hogy szintén s bites) és így kapjuk az i. rejtjeles karaktert, y_i-t. Ezután a blokkrejtjelező kimenetét beléptetjük a shift-regiszterbe. Dekódolásnál ugyanez történik: a shift-regiszter aktuális tartalmát rejtjelezzük, majd az eredmény első s bitjét XOR-oljuk y_i-vel és így kapjuk vissza x_i-t. Ezután a blokkrejtjelező kimenetét beléptetjük a shift-regiszterbe.

A CFB módhoz hasonlóan, az első karakter kódolása és dekódolása előtt a shift-regisztereket egy IV kezdeti értékkel kell feltölteni.

Biztonság. Az IV titkosságát nem kell biztosítani. Ügyelni kell azonban arra, hogy minden nyílt üzenet (ahol üzenet alatt most egy karaktersorozatot értünk) kódolásához különböző IV-t használjunk. Ellenkező esetben mindkét nyílt üzenetet pontosan ugyanazzal a kulcsfolyammal fogjuk kódolni, és egy támadó a két rejtje-

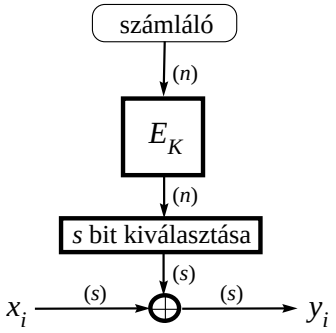
les üzenetet egymással XOR-olva a nyílt üzenetek XOR összegét kapja, melyből az üzenetek redundanciáját ismerve sikerrel kísérleheti meg az üzenetek megfejtését. Az IV-k különbözőségét például úgy lehet biztosítani, hogy az üzenetek sorszámát használjuk IV-nek.

A CFB móddal ellentétben, az OFB mód esetén az i. rejtjeles karakter értéke csak az i. nyílt karaktertől (és a shift-regiszter tartalmától) függ. Ennek ellenére, a CFB módhoz hasonlóan, a rejtjeles karakterek törlését, beszúrását, és sorrendjének megváltoztatását korlátozott mértékben mégis detektálni lehet. Ez azért van, mert egy karakter helyes visszaállításához a shift-regiszter tartalmának meg kell egyeznie a karakter kódolásánál használt shift-regiszter-tartalommal. Ha azonban y_i-t felcseréljük y_j-vel, akkor y_i dekódolásánál a shift-regiszter tartalma az y_j előállításánál használt tartalom lesz. Ebből kifolyólag az y_i-ből visszaállított nyílt karakter bitjei pénzfeldobás sorozatot alkotnak.

Hatékonyság. A CFB módhoz hasonlóan, a kódolás és dekódolás sebessége a blokkrejtjelező sebességének s/n-szerese. Míg CFB módban az y_i rejtjeles karakter dekódolásához elegendő az $y_{i-n/s}$, $y_{i-n/s+1}$, $\ldots$, y_{i-1} rejtjeles karaktereket beléptetni a shift-regiszterbe, addig OFB mód esetén az IV-ből indulva elő kell állítani az y_i kódolásánál használt shift-regiszter-tartalmat. Ezt nem lehet párhuzamosítnai, ezért a CFB móddal ellentétben, sem a kódolás sem a dekódolás nem párhuzamosítható. Ugyanakkor, mivel a generált kulcsfolyam nem függ a nyílt szövegtől, ezért az off-line módon előre kiszámítható még mielőtt a nyílt szöveg rendelkezésre állna. Ez nagy mértékben gyorsíthatja a kódolás és dekódolás műveletét. A megfelelő shift-regiszter-állapot elérése után tetszőleges rejtjeles karakter dekódolható, módosítható és újrakódolható a többi karakter változtatása nélkül. Az OFB mód tehát támogatja a nyílt karakterekhez való véletlen hozzáférést a rejtjeles szövegben, mind olvasás, mind írás esetén.

Hibaterjedési tulajdonságok. Mint azt már említettük, a CFB móddal ellentétben, az OFB mód nem terjeszti ki a rejtjeles szövegben keletkezett bithibát több visszaállított nyílt karakterre. Egy rejtjeles karaker j. bitjének változása csak a visszaállított nyílt karakter j. bitjére van hatással. Ez egyfelől előny, mert az OFB módot zajos átviteli csatorna esetén is használhatóvá teszi. Másrészt viszont hátrány, mert egy támadó a visszaállított nyílt szöveg bitjeit manipulaláni tudja a megfelelő rejtjeles bitek módosításával.

Bitvesztésből és -beszúrásból származó hibából nem épül fel a rendszer, mert a hiba helyét követően a karakterhatárok elcsúsznak és minden további karakter hibásan áll vissza. Ezért OFB mód használata esetén speciális újraszinkronizáló pontokat (markereket) kell beiktatni a karakterfolyamba, melyek lehetővé teszik a szinkronból történt esetleges kiesésekből való felépülést.



3.14. ábra. Kódolás CTR módban. A dekódolás ugyanilyen séma szerint történik, csak a rejtjeles karakterek lépnek be, és a nyílt karakterek lépnek ki.

A CTR mód

A CTR mód működése nagyon hasonlít az OFB módhoz. A különbség annyi, hogy nincs visszacsatolás, helyette a shift-regiszter által tárolt értéket minden lépésben eggyel növeljük. Ezt úgy is felfoghatjuk, mintha egy számláló aktuális értékét rejtjeleznénk a blokkrejtjelezővel. A CTR mód működését a 3.14. ábrán szemléltetjük.

A működés hasonlóságából fakadóan a CTR mód tulajdonságai nagyban hasonlítanak az OFB mód tulajdonságaihoz. Ezért itt most csak a lényeges különbségekre térünk ki.

Biztonság. CTR mód esetén a generátor periódusát könnyű megállapítani, hiszen azt a számláló mérete határozza meg. n bites számláló esetében a periódushossz 2^n. A biztonsággal kapcsolatos többi tulajdonság megegyezik az OFB mód tulajdonságaival.

Hatékonyság. Az OFB móddal ellentétben, CTR módban mind a kódolás, mind a dekódolás párhuzamosítható, hiszen az i. karakter feldolgozásához szükséges shift-regiszter-tartalmat a számláló értékének megfelelő beállításával azonnal elő tudjuk állítani. A hatékonysággal kapcsolatos többi tulajdonság megegyezik az OFB mód tulajdonságaival.

3.3. Hitelesítési feladatok

A titkosításon kívül fontos biztonsági szolgáltatás még az integritásvédelem, a hitelesítés, és a letagadhatatlanság. Ezekkel a szolgáltatásokkal foglalkozunk ebben a szakaszban. Először bevezetünk egy újabb kriptográfiai primitívet, a hash függ-

vényt, mely gyakran alkalmazott építőelem az integritásvédelmi és a hitelesítlsi feladatok megoldásában. Ezután bemutatjuk a szimmetrikus kulcsú üzenethitelesítési technikákat, majd az aszimmetrikus kulcsú digitális aláírást. Utóbbi már letagadhatatlanság szolgáltatást is biztosít. Végül röviden összefoglaljuk a nyilvános kulcsok hitelesítésének alapjait, valamint a kihívás-válasz alapú partnerhitelesítés módszereit.

Hash függvények

A kriptográfiai hash függvényeket leggyakrabban arra használjuk, hogy segítségükkel a hosszú üzeneteket kompakt módon reprezentáljuk. Ez azért hasznos, mert így számos alkalmazásban nagy méretű üzenetek helyett azok kompakt reprezentációján kell csak műveleteket végeznünk. Az egyik legtipikusabb ilyen alkalmazás az, amikor nem magára az üzenetre készítünk digitális aláírást, hanem annak kompakt reprezentációját írjuk csak alá (lásd később a 3.3. szakaszt).

Formálisan, egy $h : \{0,1\}^* \to \{0,1\}^n$ hash függvény tetszőleges hosszúságú bináris sorozatot rögzített n hosszúságú bináris sorozatba képez le. A hash függvény kimenetét szokás **hash érték**nek vagy **lenyomat**nak is nevezni.

Mivel a hash függvény ősképtere definíció szerint nagyobb, mint a képtere, ezért az ütközések elkerülhetetlenek. Itt ütközés alatt két különböző $x \neq x'$ bemenetet értünk, melyeknek hash értéke megegyezik, azaz $h(x) = h(x')$. Kriptográfiai hash függvények esetében azonban azt követeljük meg, hogy ilyen ütközéseket nehéz legyen találni. Pontosabban, a kriptográfiai hash függvényekkel kapcsolatban a következő három tulajdonságot szokás megkövetelni:

- **(Erős) ütközés-ellenállóság:** Egy h hash függvény (erősen) ütközés-ellenálló, ha nehéz feladat két olyan különböző ősképtérbeli $x \neq x'$ elemet találni, melyeknek megegyezik a hash értéke, azaz $h(x) = h(x')$.

- **Gyenge ütközés-ellenállóság:** Egy h hash függvény gyengén ütközés-ellenálló, ha bármely adott ősképtérbeli x elemhez nehéz olyan másik ősképtérbeli $x' \neq x$ elemet találni, melynek hash értéke megegyezik x hash értékével, azaz $h(x') = h(x)$.

- **Egyirányúság:**Egy h hash függvény egyirányú, ha bármely y képtérbeli elemhez, melynek ősképe a priori nem ismert, nehéz feladat olyan ősképtérbeli x elemet találni, melynek hash értéke pontosan y, azaz $h(x) = y$.

Figyeljük meg a különbséget a gyenge és az erős ütközés-ellenállóság tulajdonság között. Gyenge ütközés-ellenállóság esetén azt követeljük meg, hogy egy adott x-hez nehéz legyen találni egy x'-t, melynek hash értéke megegyezik x hash értékével. Ezzel szemben, ütközés-ellenállóság esetén nem rögzítjük egyik ősképtérbeli elemet sem, hanem azt követeljük meg, hogy nehéz legyen tetszőleges ütközést találni.

A fenti tulajdonságok bizonyos mértékben összefüggnek egymással. Könnyen igazolható például, hogy az ütközés-ellenállóságból következik a gyenge ütközés-ellenállóság. Tegyük fel ugyanis, hogy létezik olyan h hash függvény amely ütközés-ellenálló, de nem gyengén ütközés-ellenálló. Ekkor egy adott x bemenethez könnyen tudunk találni egy olyan $x' \neq x$ bemenetet, melyre $h(x') = h(x)$. Ez azonban azt jelenti, hogy könnyen találtunk egy ütközést, mégpedig az (x, x') párt, ami ellentmond azon feltevésünknek, hogy h ütközés-ellenálló.

Kicsit bonyolultabban bizonyítható, hogy az ütközés-ellenállóságból következik az egyirányúság is. A bizonyítást itt nem részletezzük. Látjuk tehát, hogy az ütközés-ellenállóság a legerősebb tulajdonság, ezért érdemes az ütközés-ellenálló hash függvények tervezésére koncentrálni.

A születésnapi paradoxon

Az ütközés-ellenállósággal szoros kapcsolatban áll a hash függvény kimenetének mérete, amelyet korábban n-nel jelöltünk. Ahhoz, hogy ezt a kapcsolatot megértsük, először a **születésnapi paradoxon**nal kell megismerkednünk. Ezt a következőképpen vezethetjük be: Adott egy N elemű halmaz, melyből véletlenszerűen választunk k elemet visszatevéssel (azaz egy elemet többször is választhatunk). Kérdés, hogy mekkora valószínűséggel lesz a választott elemek között legalább két azonos?

Először annak a valószínűségét számoljuk ki, hogy minden választott elem különböző. Ehhez vegyük észre, hogy N elemből k különbözőt N alatt a k féleképpen választhatunk, és ezen k különböző elem lehetséges sorrendjeinek száma $k!$. Továbbá, ha ismétlést is megengedünk, akkor N elemből k-t pontosan N^k féleképpen választhatunk. Ebből a keresett P valószínűség:

$$\begin{aligned} P &= \frac{k! \binom{N}{k}}{N^k} \\ &= \frac{(N-k+1)\cdot(N-k+2)\cdot\ldots\cdot(N-1)\cdot N}{N^k} \\ &= \left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right)\ldots\left(1-\frac{k-1}{N}\right) \end{aligned}$$

Ha N nagy, akkor alkalmazhatjuk a következő közelítést:

$$\left(1-\frac{1}{N}\right) \approx e^{-\frac{1}{N}}.$$

Ennek segítségével P-re az alábbi közelítést kapjuk:

$$\begin{aligned} P &\approx e^{-\frac{1}{N}} \cdot e^{-\frac{2}{N}} \cdot \ldots \cdot e^{-\frac{k-1}{N}} \\ &= e^{-\frac{k(k-1)}{2N}} \end{aligned}$$

A fentiek alapján annak valószínűsége, hogy k elem visszatevéses választása esetén legalább két választott elem azonos a következő:

$$1 - e^{-\frac{k(k-1)}{2N}} \tag{3.26}$$

A (3.26) kifejezést felhasználva kiszámolhatjuk, hogy hány húzást kell végezni ahhoz, hogy valamilyen előre adott ε valószínűséggel legyen a választott elemek között legalább két azonos elem:

$$k \approx \sqrt{2N \ln \frac{1}{1-\varepsilon}} \tag{3.27}$$

ahol a $k(k-1) \approx k^2$ közelítést alkalmaztuk. Ha például $\varepsilon = 0.5$, akkor $k \approx 1.177\sqrt{N}$ adódik, míg $\varepsilon = 0.9$ esetén azt kapjuk, hogy $k \approx 2.146\sqrt{N}$.

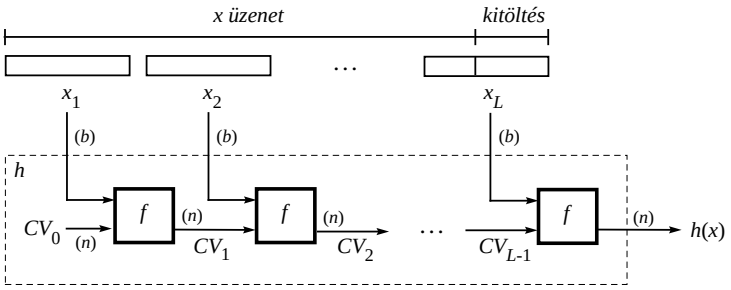
Ha a fenti eredményeket arra az esetre alkalmazzuk, amikor a halmaz elemei születésnapok (azaz $N = 365$), akkor azt kapjuk, hogy $1.177\sqrt{365} \approx 23$ véletlen választott ember között 0.5 valószínűséggel lesz legalább kettő akiknek egybe esik a születésnapja, míg $2.146\sqrt{365} \approx 41$ véletlen választott ember esetén egy közös születésnapi party lehetőségének valószínűsége már 0.9. Ez elsőre meglepőnek tűnik, mert intuitíve nem várnánk, hogy ilyen kis számú ember esetén ilyen nagy valószínűségeket kapunk. Ebből származik a születésnapi paradoxon elnevezés.

A születésnapi paradoxonnak mélyreható hatása van a hash függvények biztonságára vonatkozóan. A születésnapi paradoxon értelmében ugyanis kb. $\sqrt{2^n} = 2^{\frac{n}{2}}$ véletlen választott ősképtérbeli elem között nagy valószínűséggel lesz legalább egy ütköző pár. Azaz, ha $\frac{n}{2}$ túlságosan kicsi, akkor egyszerű véletlen választással hatékonyan lehet ütközéseket generálni. A mai technológia mellett ezért a hash függvény kimenetének méretét legalább $n = 128$ bitre érdemes választani.

A véletlen választáson alapuló ütközés-generálás hash függvények ellen hasonlít a rejtjelezőknél tárgyalt kimerető kulcskeresés támadáshoz. A rejtjelezőknél láttuk, hogy a megfelelő kulcsméret választása szükséges feltétele a rejtjelező biztonságának. Hasonlóképpen, a hash függvény kimenetének megfelelő méretezése szükséges feltétele az ütközés-ellenállóságnak. Ugyanakkor elképzelhető, hogy a hash függvény algebrai struktúrájában rejlő gyengeségek miatt, a hash függvény nem ütközés-ellenálló, annak ellenére, hogy kimenetének mérete megfelelően nagy.

Iteratív hash függvények

A gyakorlatban használt hash függvények iteratív módon állítják elő a bemenet hash értékét. Az itertív hash függvények működésének vázlata a 3.15. ábrán látható. Az iteratív hash függvény lelke az f tömörítő függvény. Az f tömörítő függvénynek két bemente van, ahol az egyik bemenet mérete b bit, a másik bemenet mérete pedig n bit, azaz megegyezik a hash függvény kimenetének méretével. Az f függvény kimenetének mérete n bit.



3.15. ábra. Az iteratív hash függvény működésének vázlata.

A bemeneti üzenet feldolgozása b bites blokkokban történik. Az i. iterációs lépés során, az f függvény egyik bementére az i. bemeneti blokk, a másik bemenetére pedig az előző iterációs lépés kimenete kerül, s az f függvény kimenete adja az i. iterációs lépés kimenetét, amelyet i. láncolási értéknek (chaining value) nevezünk, és CV_i-vel jelölünk. Az első iterációs lépésnél egy CV_0 kezdő láncolási értéket használunk, ami a hash függvény specifikációjában rögzített konstans. Az utolsó iterációs lépés kimenete adja a hash függvény kimenetét, azaz a hash értéket.

Ha a bemenet mérete nem egész számú többszöröse b-nek, akkor a bemeneti üzenetet a feldolgozás előtt ki kell tölteni. A célnak megfelelő az egyszerű, 0 bitekkel történő kitöltés. Ennél azonban biztonságosabb, ha olyan kitöltést alkalmazunk amely tartalmazza az üzenet hosszának bináris reprezentációját. Erre vonatkozik a következő tétel, amit bizonyítás nélül közlünk:

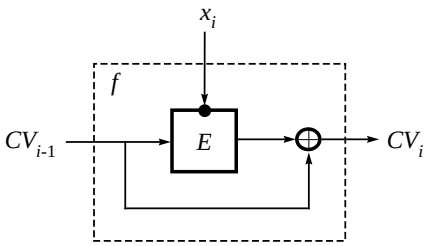
3.9. tétel (Merkle–Damgard- (MD) kiegészítés). *Tegyük fel, hogy a h iteratív hash függvényünk f kompressziós függvénye ütközés-ellenálló tulajdonsággal rendelkezik. Ha a h bementére kerülő üzenetet kiegészítjük egy blokkal, amely tartalmazza az üzenet bithosszát, akkor a h hash függvény is ütközésmentes lesz.*

Számos iteratív hash függvény konstrukció létezik, amely blokkrejtjelzőt használ tömörítő függvényként. Ilyen például a Davies–Meyer-séma, melynek tömörítő függvényét a 3.16. ábra szemlélteti. A Davies–Meyer tömörítő függvény esetén a hash függvény belső láncolási értéke a blokkrejtjelező bemenetére, a feldolgozandó üzenet aktuális blokkja pedig a blokkrejtjelező kulcsbementére kerül. A blokkrejtjelező kimenete és a bemenetként használt láncolási érték XOR összeg adja a következő láncolási értéket.

A Davies-Meyer séma előnyét a következő tétel foglalja össze:

3.10. tétel. *A Davies–Meyer-séma egyirányú hash függvényre vezet, ha az E transzformáció véletlen leképezéssel modellezhető.*

A Davies–Meyer-séma kapcsán megjegyezzük azonban, hogy kis blokkméretű (pl. 64 bites) blokkrejtjelező esetén az így nyert hash függvény nem lesz ütközés-



3.16. ábra. A Davies-Meyer tömörítő függvény.

ellenálló a születésnapi paradoxon miatt. Ezért tanácsos egy legalább 128 bites blokkrejtjelező használata.

A blokkrejtjelezőre épülő hash függvények hátránya, hogy nem feltétlenül gyorsak, hiszen a blokkrejtjelező nem az adott feladatra lett tervezve és optimalizálva. Ezért a gyakorlatban elterjedten használt iteratív hash függvények mind dedikált hash függvények, melyeket kifejezetten hash függvénynek terveztek. A két legelterjedtebben használt dedikált iteratív hash függvény az MD5 és a SHA-1 nevet viseli. Az MD5 128 bites, a SHA-1 pedig 160 bites hash értéket generál. A bemeneti üzenetet mindkét hash függvény 512 bites blokkokban dolgozza fel.

Üzenethitelesítő kódok

Az üzenethitelesítést és integritásvédelmet leggyakrabban üzenethitelesítő kódok (Message Authentication Code – MAC) alkalmazásával valósítjuk meg. Egy üzenethitelesítő kódra gondolhatunk úgy, mint egy kriptográfiai ellenőrzőösszegre, amelyet a küldő az üzenet elküldése előtt kiszámít és az üzenethez csatol. A csatornán átvitelre kerül az üzenet és az üzenet ellenőrző összege is. A vevő mindkettőt veszi, majd ellenőrzi az ellenőrző összeget. Ha az ellenőrzés sikerrel jár, akkor a vevő meg lehet győződve arról, hogy az üzenet sértetlen és valóban a vélt (pl. az üzenetben megjelölt) feladó küldte. Ellenkező esetben, az üzenet integritása az átvitel során megsérült, és mivel ez lehet rosszindulatú módosítás következménye is, ezért a vevő nem fogadja el az üzenetet.

Formáját és funkcióját tekintve tehát egy üzenethitelesítő kód egy hibadetektáló kódhoz (pl. CRC) hasonlítható. Van azonban egy nagyon fontos különbség az üzenethitelesítő és a hibadetektáló kódok között. Nevezetesen az, hogy a hibadetektáló kódok csak a zajos csatornán bekövetkezett véletlen hibák detektálására alkalmasak, és nem képesek egy rosszindulatú támadó által végrehajtott szándékos módosítások detektálására. Ez egyszerűen azért van, mert a támadó az üzenet módosítása után a módosított üzenethez ki tudja számolni az új hibadetektáló kódot. A vevő tehát a módosított üzenetet és a hozzá tartozó helyes hibadetektáló kódot kapja meg és így nem veszi észre a módosítást.

Ezzel szemben az üzenethitelesítő kódok nemcsak a véletlen hibákat, hanem a rosszindulatú módosításokat is képesek detektálni. Ezen képességük abból adódik, hogy a hibadetektáló kóddal ellentétben, az üzenethitelesítő kód értéke nemcsak magától az üzenettől függ, hanem egy a küldő és a vevő által megosztott titkos információtól, egy kulcstól is. A támadó ezen titok hiányában nem tudja kiszámítani a módosított üzenethez tartozó helyes üzenethitelesítő kódot, és így a módosítás nem maradhat észrevétlen. Ezen túlmenően a vevő tudja, hogy helyes üzenethitelesítő kód kiszámítására csak a küldő (és természetesen maga a vevő) alkalmas. Ezért meg lehet győződve arról, hogy minden helyes üzenethitelesítő kóddal vett (és nem saját magától származó) üzenet csakis a küldőtől származhat.

A fenti gondolatok az üzenethitelesítés következő modelljéhez vezetnek: Az A küldő és a B vevő rendelkezik egy közös K_{AB} kulccsal. K_{AB}-t rajtuk kívül más nem ismeri. Az m üzenet elküldése előtt A kiszámolja az m-hez tartozó $\mu = MAC_{K_{AB}}(m)$ üzenethitelesítő kódot, ahol $MAC_{K_{AB}}$ a K_{AB} kulccsal paraméterezett üzenethitelesítő függvény. A továbbiakban a μ üzenethitelesítő kódot röviden MAC értéknek, a $MAC_{K_{AB}}$ üzenethitelesítő függvényt pedig röviden MAC függvénynek fogjuk nevezni. A elküldi, B pedig megkapja a MAC értékkel kiegészített $m|\mu$ üzenetet. K_{AB} ismeretében B kiszámolja $MAC_{K_{AB}}(m)$-et, és az eredményt összehasonlítja μ-vel. Egyenlőség esetén B elfogadja az üzenetet, ellenkező esetben eldobja azt.

A MAC fügvénnyel szemben az alábbi követelményeket támasztjuk:

a) A MAC_K függvény olyan szűkítő transzformáció legyen, mely tetszőleges hosszúságú üzeneteket fix hosszúságú, n bites MAC értékbe képez.

b) A K kulcs ismeretében tetszőleges m üzenethez könnyű legyen kiszámolni $MAC_K(m)$-et.

c) A K kulcs ismeretének hiányában viszont legyen $MAC_K(m)$ kiszámítása nehéz feladat, még akkor is, ha nagy számú $(m_i, MAC_K(m_i))$ pár áll rendelkezésre, ahol természetesen $m \notin \{m_i\}$.

d) A K kulcs meghatározása legyen nehéz feladat még nagy számú $(m_i, MAC_K(m_i))$ pár ismerete esetén is.

Megjegyezzük, hogy ha egy MAC függvény eleget tesz a 3) követelménynek, akkor kielégíti a 4) követelményt is. Ha ugyanis a 4) követelményt nem elégítené ki, akkor egy támadó meg tudná határozni a K kulcsot és annak ismeretében tetszőleges üzenethez tudna MAC értéket generálni, azaz a MAC függvény nem elégíthetné ki a 3) követelményt sem. Fordítva azonban nem áll fenn az implikáció, ugyanis elméletileg elképzelhető, hogy a K kulcs ismerete nem szükséges ahhoz, hogy egy üzenethez helyes MAC értéket generáljon a támadó.

A CBC-MAC

A CBC-MAC függvény működésének gyors megértéséhez képzeljük el, hogy az üzenetet CBC módban rejtjelezzük egy blokkrejtjelezővel, azzal a módosítással,

hogy a rejtjeles blokkokat az utolsó kivételével mind eldobjuk. A MAC függvény kimenete az utolsó rejtjeles blokk lesz.

Formálisan, a CBC-MAC működését a következőképpen írhatjuk le: Adott egy m üzenet és egy K kulcs, ahol K mérete megegyezik a használni kívánt blokkrejtjelező kulcsméretével. Az m üzenetet először kitöltjük, hogy mérete a blokkrejtjelező n blokkméretének többszöröse legyen. Mivel magát az m-et is el fogjuk küldeni a vevőnek, ezért a kitöltés lehet nagyon egyszerű, például az üzenet megfelelő számú 0 bittel történő kiegészítése. Jelöljük a kiegészített üzenetet X-szel. X-et n bites blokkokra osztjuk, melyeket $X_1, X_2, \ldots, X_N$-nel jelölünk. Legyen $IV = 0$ (azaz a csupa 0 bitből álló blokk). A CBC mód működését definiáló (3.17) és (3.18) kifejezések, valamint a megadott IV és K felhasználásával számítsuk ki az utolsó rejtjeles blokkot, C_N-t. Ez lesz a CBC-MAC függvény kimenete, azaz $MAC_K(m) = C_N$. Ha rendelkezésre áll egy $K' \neq K$ másik kulcs is, akkor a MAC függvény kimenetére opcionálisan nem C_N-et vezetjük, hanem $E_K(D_{K'}(C_N))$-et, ahol E jelöli a blokkrejtjelező kódolót, D pedig a dekódoló függvényét.

Tekintsük most át a MAC függvénnyel szemben támasztott követelmények listáját. A CBC-MAC függvény tetszőleges méretű üzenethez egy n bites MAC értéket generál, ahol n az alkalmazott blokkrejtjelező blokkmérete. A K kulcs ismeretében a MAC számítása egyszerű, a CBC kódolással megegyező módon történik. A K kulcs meghatározása megfigyelt üzenet – MAC párokból lényegében a blokkrejtjelező feltörését jelenti, tehát megfelelő erősségű blokkrejtjelező esetén ez biztosan nehéz feladat. Hasonlóképpen nehéznek tűnik a MAC meghatározása egy adott üzenethez, ha a kulcs nem ismert.

Hash függvényre épülő MAC függvények

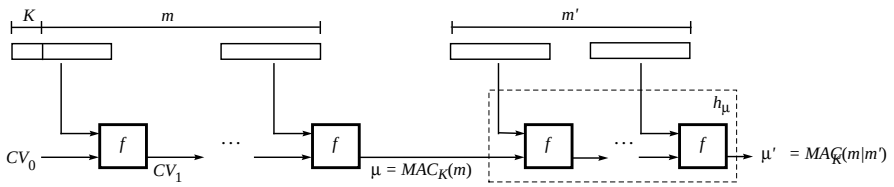
A legegyszerűbben úgy kovácsolhatunk egy hash függvényből MAC függvényt, hogy a kulcsot hozzáfűzzük az üzenethez, majd a kulccsal megtoldott üzenetnek kiszámítjuk a hash értékét és az eredményt tekintjük az eredeti üzenet MAC értékének. Attól függően, hogy a kulcsot az üzenet elé vagy mögé fűzzük, alapvetően két módszert különböztethetünk meg, melyeket **titok prefix** és **titok szuffix** módszereknek hívunk.

Titok prefix módszer. Nevéből adódóan a titok prefix módszer az üzenet elé fűzi a kulcsot. Az m üzenet MAC értékét tehát a következő módon számoljuk:

$$MAC_K(m) = h(K|m) \tag{3.28}$$

ahol h jelöli a hash függvényt, melyre a MAC konstrukció épül.

Az így nyert MAC függvény h tulajdonságaiból adódóan szűkítő transzformáció és a teljes input ismeretében (azaz a K kulcsot is ismerve) a MAC érték könnyen számolható. A hash függvény egyirányúsága miatt abban reménykedünk, hogy egy adott MAC érték ismeretében a K kulcs megfejtése nehéz feladat. Vegyük azonban észre, hogy a hash függvény egyirányúsága azt jelenti, hogy egy adott MAC érték



3.17. ábra. A titok prefix módszer elleni támadás.

ismeretében a teljes input (kulcs és üzenet) megfejtése nehéz feladat. A mi esetünkben azonban az input egy része, az üzenet ismert. Kérdéses tehát, hogy ez a konstrukció milyen garanciát biztosít a kulcs megfejtése ellen. További probléma, hogy ha h egy iteratív hash függvény, akkor a támadó egy m üzenet μ MAC értékének ismeretében bármely $m|m'$ üzenet μ' MAC értékét ki tudja számolni mint $\mu' = h_\mu(m')$, ahol h_μ ugyanaz a hash függvény mint h, csak a kezdeti láncváltozójának értéke μ. Ezt a 3.17. ábra szemlélteti.

Titok szuffix módszer. Egy másik lehetőség hash függvény alapú MAC függvény konstruálására a titok szuffix módszer. Ekkor a kulcsot az üzenet mögé fűzzük, majd a kulccsal így kiegészített üzenetnek kiszámoljuk a hash értékét és az eredményt tekintjük az üzenet MAC értékének:

$$MAC_K(m) = h(m|K) \tag{3.29}$$

A titok szuffix módszer tulajdonságai hasonlítanak a titok prefix módszer tulajdonságaihoz azzal a különbséggel, hogy a titok prefix módszernél említett támadás ebben az esetben nem kivitelezhető. A módszer egy gyengesége, hogy ha h egy iteratív hash függvény, akkor a kulcsot csak a MAC érték kiszámításának utolsó lépésében használjuk fel. Ez problémához vezethet, ha a hash függvény nem ütközésmentes (pl. az alkalmazás jellegéből adódóan a MAC érték hosszára vonatkozóan korlátozásaink vannak). Tegyük fel ugyanis, hogy a támadó (a születésnapi paradoxont használva) talált egy m és egy m' üzenetet, melyekre $h(m) = h(m')$. Könnyen látszik, hogy ekkor $MAC_K(m) = h(m|K) = h(m'|K) = MAC_K(m')$, hiszen m és m' feldolgozása után a hash függvény belső láncváltozójának értéke a két esetben megegyezik. Más szóval, ha a támadó megszerzi az egyik üzenet MAC értékét, akkor azt fel tudja használni a másik üzenet MAC értékeként is.

HMAC. A HMAC a gyakorlatban igen elterjedten használt, hash függvényre épülő MAC függvény. A HMAC függvényt a következő kifejezés definiálja:

$$MAC_K(m) = h(K^+ \oplus opad \mid h(K^+ \oplus ipad \mid m)) \tag{3.30}$$

ahol

- h egy iteratív hash függvény, ami az üzenetet B bájtos blokkokban dolgozza fel,
- *ipad* (inner pad) egy B bájt hosszú konstans blokk, melyben minden bájt értéke `36`,
- *opad* (outer pad) egy B bájt hosszú konstans blokk, melyben minden bájt értéke `5C`, és
- K^+ a K kulcs csupa 0 bittel kiegészítve, hogy hossza elérje a B bájtot.

A K kulcs tetszőleges hosszúságú lehet. B értéke tipikusan 64, így K általában rövidebb, mint B bájt. Ha mégis hosszabb lenne, akkor először kiszámoljuk $h(K)$-t és ezt használjuk kulcsként. A HMAC által definiált MAC érték mérete az alkalmazott h hash függvény kimenetének méretétől függ. Ha például h az MD5 hash függvény (HMAC-MD5), akkor a kiszámított MAC mérete 128 bit (16 bájt), míg a SHA-1 hash függvény használata esetén (HMAC-SHA1) a MAC mérete 160 bit (20 bájt).

Digitális aláírás

Az előzőekben tárgyalt üzenethitelesítő kódok hasznos szolgáltatásokat nyújtanak: lehetővé teszik a csatornán átküldött üzenetek (véletlen és szándékos) módosításának detektálását és az üzenetek küldőjének hitelesítését. Az üzenethitelesítő kódok hátránya azonban az, hogy ezeket a szolgáltatásokat csak a vevő számára biztosítják. A vevő egy kívülálló harmadik felet már nem tud meggyőzni arról, hogy egy vett üzenet sértetlen és a küldőtől származik. Ez azért van, mert az üzenethitelesítő kód értéke egy olyan titkos kulcstól függ, melyet a vevő is ismer. A harmadik fél tehát nem tudja biztosan eldönteni, hogy az adott üzenethitelesítő kódot a küldő vagy a vevő generálta. Ez azt jelenti, hogy a küldő bármikor letagadhatja, hogy egy üzenetet küldött a vevőnek, és a vevő nem tudja bebizonyítani, hogy a küldő hazudik. Más szóval, az üzenethitelesítő kódok nem biztosítanak letagadhatatlanság szolgáltatást.

A letagadhatatlanság szolgáltatás megvalósítására olyan aszimmetrikus mechanizmusra van szükség, melynek segítségével csakis az üzenet küldője állíthatja elő az üzenet eredetére vonatkozó bizonyítékot (így azt hamisítani nem lehet), de a rendszer bármely résztvevője (köztük a vevő is) ellenőrizni tudja azt. Ezt a mechanizmust digitális aláírásnak nevezzük, mivel tulajdonságai nagy mértékben hasonlítanak a hagyományos aláírás tulajdonságaihoz. Egy fontos különbség a digitális és a hagyományos aláírás között az, hogy a digitális aláírás nem az üzenet anyagi hordozójához (pl. papír) kötődik, hanem magához az üzenethez. Így nemcsak az üzenet eredetére vonatkozóan nyújt garanciát, hanem segítségével az üzenet tartalmában az aláírás generálása után bekövetkezett módosításokat is detektálni lehet. Összefoglalva tehát a digitális aláírás egy olyan mechanizmus, mely biztosítja az üzenetek integritását és hitelességét, valamint az üzenetek eredetének letagadhatatlanságát.

A digitális aláírás fent említett aszimmetrikus tulajdonsága miatt, a jól ismert digitális aláírás sémák mind nyilvános kulcsú kriptográfiára épülnek. Egy digitális aláírás séma két komponensből áll: egy S aláírás-generáló algoritmusból és egy V aláírás-ellenőrző algoritmusból. Az aláírás-generáló algoritmus az aláíró fél privát kulcsával van paraméterezve, míg az ellenőrző algoritmus az aláíró fél nyilvános kulcsát használja paraméterként. Jelölésben ezt úgy érzékeltetjük, hogy az aláíró fél azonosítóját alsó indexbe írjuk, azaz az A privát kulcsával paraméterezett aláíró algoritmust S_A-val, az A nyilvános kulcsával paraméterezett ellenőrző algoritmust pedig V_A-val jelöljük. Az S_A algoritmus bemenete az aláírni kívánt m üzenet, kimenete pedig a $\sigma = S_A(m)$ digitális aláírás. A V_A ellenőrző algoritmus bemenete egy m üzenet és egy σ aláírás, kimenete pedig *true* ha σ A érvényes aláírása m-en, és *false* egyébként:

$$V_A(m,\sigma) = \begin{cases} \textit{true} & \text{ha } \sigma = S_A(m) \\ \textit{false} & \text{egyébként} \end{cases} \tag{3.31}$$

Értelemszerűen, az ellenőrzést végző fél akkor fogadja el az aláírást hitelesnek, ha az ellenőrző algoritmus kimenete *true*.

Természetesen az ellenőrzés végrehajtásához az ellenőrző félnek ismenie kell az aláíró fél nyilvános kulcsát, mert ez szükséges az ellenőrző algoritmus helyes paraméterezéséhez (azaz V_A használatához). Az aláíró fél hiteles nyilvános kulcsának megszerzése külön feladat, amit a gyakorlatban legtöbbször valamilyen nyilvános kulcs infrastruktúra (Public Key Infrastructure – PKI) segítségével oldanak meg. Ezekkel a kérdésekkel később foglalkozunk részletesebben.

Tekintsünk egy nyilvános kulcsú rejtjelező rendszert. Az A résztvevő kódoló transzformációját (melyet A nyilvános kulcsa határoz meg) jelöljük E_A-val, az ehhez tartozó dekódoló transzformációt (melyet A privát kulcsa határoz meg) pedig D_A-val. Ha minden m üzenetre teljesül az $E_A(D_A(m)) = m$ egyenlőség, akkor az adott nyilvános kulcsú rendszerből a következőképpen készíthetünk digitális aláírás sémát. Az aláírás-generáló S_A algoritmus legyen a titkosító rendszer D_A dekódoló transzformációja ($S_A(m) = D_A(m)$), az aláírás-ellenőrző V_A algoritmust pedig definiálja a következő kifejezés:

$$V_A(m,\sigma) = \begin{cases} \textit{true} & \text{ha } E_A(\sigma) = m \\ \textit{false} & \text{egyébként} \end{cases} \tag{3.32}$$

Más szavakkal, az aláírást úgy generáljuk, hogy az m üzeneten a titkosító rendszer dekódoló transzformációját alkalmazzuk, az aláírást pedig úgy ellenőrizzük, hogy a σ aláíráson a kódoló transzformációt alkalmazzuk, és az eredményt összehasonlítjuk az eredeti m üzenettel. Ha az aláírás hiteles, azaz $\sigma = D_A(m)$, akkor a titkosító rendszer fenti tulajdonsága miatt $E_A(\sigma) = m$ egyenlőségnek teljesülnie kell.

Hangsúlyozzuk, hogy a fenti digitális aláírás konstrukció csak abban az esetben működik, ha a nyilvános kulcsú titkosító rendszer rendelkezik azzal a tulajdonsággal, hogy $E_A(D_A(m)) = m$ minden m-re teljesül. Ilyen például az RSA rendszer, melyre $E_A(D_A(m)) = (m^d)^e \bmod n = (m^e)^d \bmod n = D_A(E_A(m)) = m$. Ez azonban nem minden nyilvános kulcsú titkosító rendszernél van így.

Lenyomat aláírása

A fent bevezetett digitális aláírás séma hátránya, hogy az aláírás mérete függ az üzenet méretétől. Gyakorlati okokból azonban célszerű ezt a függést megszüntetni. Ezt úgy tehetjük meg, hogy egy alkalmas nyilvános ütközésellenálló hash függvény felhasználásával, nem az eredeti üzeneten, hanem annak hash értékén alkalmazzuk az aláírás-generáló algoritmust. Ekkor az aláírt üzenet $m \mid S_A(h(m))$ alakú lesz. A hash érték aláírása azért előnyös mert így az aláírás-generálásának ideje lényegében függetlenné válik az üzenet méretétől[6]. Ez azt jelenti, hogy nagy méretű üzeneteket is hatékonyan tudunk aláírni.

Ügyelnünk kell azonban arra, hogy ha egy támadó megfigyel egy $m \mid S_A(h(m))$ aláírt üzenetet, és talál egy olyan m' üzenetet melyre $h(m') = h(m)$, akkor az $S_A(h(m))$ aláírást felhasználhatja az m' üzenet hiteles aláírásaként, hiszen ekkor nyilván $S_A(h(m')) = S_A(h(m))$ is teljesül. Csakhogy ha h ütközésellenálló hash függvény, akkor a támadó gyakorlatilag nem találhat azonos hash értékre vezető m' üzenetet. Megjegyezzük, hogy a gyakorlatban, a támadó feladatát tovább nehezíti, hogy nemcsak a $h(m') = h(m)$ egyenlőségnek kell teljesülnie, hanem m'-nek értelmes, a támadó céljainak megfelelő csaló tartalmú üzenetnek kell lennie.

Nyilvános kulcs hitelesítés

Aszimmetrikus kulcsú rejtjelezés vagy digitális aláírás használata esetén szükséges, hogy a kommunikáló felek ismerjék egymás hiteles nyilvános kulcsát. Most azt vizsgáljuk, hogy hogyan lehet a hiteles nyilvános kulcsokhoz hozzájutni. Vegyük észre, hogy ha A szeretné megszerzni B nyilvános kulcsát, K_B-t, akkor nem jó megoldás az, ha B mindenféle védelem nélkül elküldi K_B-t A-nak. Igaz ugyan, hogy K_B nem titkos, ezért egy hallgatózó támadótól nem kell tartani. Azonban ha az X támadó aktív támadást is képes végrehajtani, akkor K_B-t az átvitel során saját nyilvános kulcsával, K_X-szel helyettesítheti. Ekkor A B-nek szánt üzeneteit K_X-szel fogja rejtjelezni, és így azokat X dekódolni tudja. Sőt, a dekódolt üzeneteket K_B-vel rejtjelezve és továbbítva B felé, X a támadást észrevétlenül tudja végrehajtani. Ezért nagyon fontos, hogy A meg tudjon győződni a kulcs hitelességéről, azaz arról, hogy a kulcs amit B nyilvános kulcsának hisz, valóban B nyilvános kulcsa.

Ezen probléma megoldására a gyakorlatban a **nyilvános kulcs tanúsítvány**ok használata terjedt el. A nyilvános kulcs tanúsítvány egy adatstruktúra, mely minimálisan a következő adatokat tartalmazza:

- a nyilvános kulcsot,
- a nyilvános kulcs tulajdonosának azonosítóját, és
- egy aláírást, mely az előző két mezőt elválaszthatatlanul összeköti.

[6] Az üzenet hash értékének kiszámítása továbbra is függ az üzenet méretétől, de a hash számítása még így is több nagyságrenddel gyorsabb, mint az aláírás generálása, ezért az aláírás-generáló algoritmus futási ideje dominál.

Az aláírást általában egy megbízható fél, a **hitelesítés szolgáltató** (Certification Authority – CA) generálja, aki kompetens annak meghatározásában, hogy a tanúsítvány által tartalmazott kulcs és azonosító valóban összetartoznak-e. A tanúsítvány lényegében ezen megbízható fél állítása, miszerint az adott kulcs az adott tulajdonoshoz tartozik. Kibocsájtása után a tanúsítvány bekerül egy nyilvános adatbázisba, ahonnan igény szerint letölthető.

A gyakorlatban a tanúsítvány további adatokat is tartalmaz, mint például a tanúsítvány aláírójának az azonosítóját, a kibocsájtás dátumát, a tanúsítvány érvényességi idejét, és a tanúsítvány használatára vonatkozó információt.

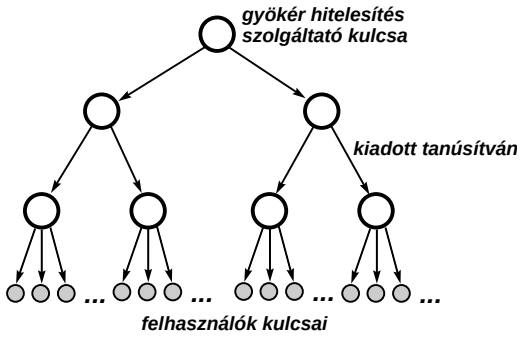
A legegyszerűbb esetben a rendszerben csak egy hitelesítés szolgáltató van, és az bocsájtja ki minden felhasználó tanúsítványát. Ekkor minden felhasználó minden tanúsítványt ellenőrizni tud, feltéve hogy ismeri a hitelesítés szolgáltató nyilvános kulcsát. Ehhez tipikusan akkor juthat hozzá biztonságosan mikor a saját tanúsítványát igényli a hitelesítés szolgáltatónál, és személyazonosságának igazolása érdekében személyesen megjelenik. Az egyetlen hitelesítés szolgáltatóra épülő rendszer hátránya, hogy nem skálázható, azaz a felhasználók számának növekedésével a rendszer használata egyre nehézkesebbé válik, a megbízható működés esetleg nem is garantálható.

Nagy rendszerekben több hitelesítés szolgáltató található, melyek valamilyen struktúra szerint szerveződnek. A skálázhatósággal kapcsolatos problémák megoldására általában célszerű a hitelesítés szolgáltatókat hierarchiába szervezni. Tiszta hierarchia esetén a hierarchia csúcsán egyetlen szolgáltató áll, amit **gyökér szolgáltatónak** (root CA) hívunk. Ez alatt helyezkednek el az első szintű szolgáltatók, majd alattuk a második szintűek, és így tovább. Tipikusan minden szolgáltató a közvetlenül alatta elhelyezkedő szolgáltatók számára bocsájt ki tanúsítványokat, azaz minden alatta elhelyezkedő szolgáltató azonosítóját és nyilvános kulcsát aláírja. A felhasználók tanúsítványát a legalsó szinten elhelyezkedő szolgáltatók bocsájtják ki. Ezt a fajta szerveződést a 3.18. ábra szemlélteti.

Csakúgy mint az egyetlen hitelesítés szolgáltatót tartalmazó rendszerben, itt is feltesszük, hogy minden felhasználó ismeri a gyökér szolgáltató hiteles nyilvános kulcsát. Ez azonban még nem elég egy adott felhasználó tanúsítványának ellenőrzéséhez, hiszen a felhasználók tanúsítványait nem a gyökér szolgáltató írja alá. Amire szükség van az egy **tanúsítvány-lánc**, mely a következő tulajdonságokkal rendelkezik:

- A lánc első eleme egy olyan tanúsítvány, amit a gyökér szolgáltató adott ki, és így a benne található nyilvános kulcs hitelessége bárki által ellenőrizhető.
- A lánc minden további tanúsítványára igaz, hogy ellenőrizhető a láncban őt közvetlenül megelőző tanúsítványban található nyilvános kulccsal.
- A lánc utolsó tanúsítványa tartalmazza a kérdéses, hitelesíteni kívánt felhasználói nyilvános kulcsot.

Könnyen látható, hogy egy ilyen tanúsítvány-lánc birtokában és a gyökér szolgáltató nyilvános kulcsának ismeretében bármely felhasználó bármely másik felhasz-



3.18. ábra. Hitelesítés szolgáltatók tiszta hierarchikus szerveződésének illusztrációja. Minden nyíl egy kibocsájtott tanúsítványt reprezentál, ahol a nyíl iránya a kibocsájtás irányára utal.

náló tanúsítványát ellenőrizni tudja. Ehhez a lánc tanúsítványait kell sorrendben ellenőrizni.

Fontos megjegyezni, hogy tanúsítvány-láncok használata esetén, a lánc minden egyes tanúsítványának kibocsájtójában meg kell bíznunk. Ha egy hitelesítés szolgáltató nem feltétlenül megbízható, akkor a lánc ellenőrzése az ezen szolgáltató által kibocsájtott tanúsítványnál megakad, hiszen az abban található nyilvános kulcs hitelességében nem lehetünk biztosak.

Előfordulhat, hogy egy tanúsítványt még a lejárati dátuma előtt vissza kell vonni. Ennek tipikus oka az lehet, hogy a tanúsítványban tárolt adatok megváltoztak vagy a tanúsítvány által hitelesített nyilvános kulcshoz tartozó privát kulcs kompromittálódott (azaz azt a jogos tulajdonosán kívül valaki más is megszerezhette). A visszavonás úgy történik, hogy a hitelesítés szolgálatató a tanúsítványt (annak azonosítóját) elhelyezi a **tanúsítvány visszavonási listán** (Certificate Revocation List – CRL). Ez egy rendszeres időközönként (pl. naponta) frissített, nyilvános lista, amit a szolgáltató aláírásával hitelesít.

Vizsgáljuk meg ezek után, hogy hogyan történik egy egyszerű digitális aláírás ellenőrzése. Először is meg kell szerezni az aláírás ellenőrzéséhez szükséges nyilvános kulcsot tartalmazó tanúsítványt, illetve a gyökér hitelesítő kulcsától az adott tanúsítványig vezető tanúsítvány-láncot. Minden így megszerzett tanúsítványt le kell ellenőrizni, ami egyrészt abból áll, hogy ellenőrizzük a tanúsítványban található digitális aláírást, másrészt ellenőrizzük, hogy a tanúsítvány nincs-e visszavonva. Utóbbi ellenőrzéshez meg kell szerezni a tanúsítvány kibocsájtójának legfrissebb tanúsítvány visszavonási listáját, ellenőrizni kell annak aláírását, és meg kell győződni arról, hogy a kérdéses tanúsítvány nem szerepel a listán. Mivel ezeket az ellenőrzéseket a tanúsítvány-lánc minden elemére végre kell hajtani, ezért a digitális aláírás ellenőrzése nagy komplexitású feladat, s az is látható, hogy komoly infrastruktúrát igényel.

Partnerhitelesítés

A partnerhitelesítő protokollok feladata, hogy lehetővé tegyék a kommunikáló felek számára egymás identitásának megbízható ellenőrzését. A megbízható jelző itt arra utal, hogy a felek valamilyen módon bebizonyítják állított identitásukat. Az identitás bizonyítása többféle módszerrel is történhet, melyek alapvetően három osztályba sorolhatók:

- **Biometriai módszerek:** Ezen módszerek elsősorban személyek identitásának bizonyítására használatosak. A bizonyítás alapját valamilyen személyes biológiai jellemző (pl. hang, ujjlenyomat, írisz mintázat, stb.) képezi.
- **Hardver alapú módszerek:** Ezen módszereknél a bizonyítás alapja valamilyen hardver token (badge, chip-kártya, stb.) birtoklása.
- **Algoritmikus módszerek:** Ezeknél a módszereknél a bizonyítás alapja valamilyen számítás elvégzésének képessége.

Az algoritmikus partnerhitelesítési módszerek maguk is többfélék lehetnek. Tipikus példák a következők:

- **Jelszavak:** Egy gyakran használt algoritmikus módszer valamilyen titok ismeretének bizonyítása magának a titoknak a felfedésével. Ezt jelszó alapú partnerhitelesítésnek nevezzük.
- **Kriptográfiai kihívás–válasz protokollok:** Ha a felek képesek bonyolultabb számítások elvégzésére, akkor kriptográfiai alapú módszereket is használhatnak. Ebben az esetben egy titkos kulcs használata (pl. egy véletlen kihívás digitális aláírása) képezi az identitás bizonyításának alapját, azzal a feltételezéssel, hogy a titkos kulcs valamilyen módon a bizonyítást végző félhez kötődik.
- **Zero-knowledge (ZK) protokollok:** Léteznek olyan kriptográfiai technikák, melyek garantálják, hogy az identitás bizonyítása során, a bizonyítás alapját képező titokról semmilyen információ nem szivárog ki azon kívül, hogy a bizonyítást végző fél ismeri a titkot. Ezeket zero-knowledge protokolloknak nevezzük.

Az alábbiakban kizárólag a kriptográfiai kihívás–válasz protokollokkal foglalkozunk.

Egy kihívás–válasz protokollban A úgy hitelesíti magát B felé, hogy bebizonyítja, birtokában van egy olyan kriptográfiai kulcsnak, amely valamilyen módon A-hoz van rendelve (azaz B tudja, hogy csak A ismerheti). A bizonyításhoz A használja a kulcsot, azaz rejtjelez, dekódol, vagy aláír valamit. A visszajátszásos támadások megakadályozása érdekében, A egy B által frissen generált elemen (kihíváson) alkalmazza a kulcsot, így A válasza nemcsak a kulcstól függ, hanem a kihívástól is.

Tekintsük a következő, építőelemként szolgáló protokoll-részleteket:

- **Partnerhitelesítés szimmetrikus kulcsú rejtjelezéssel:**

(1)	$B \to A$:	N_B
(2)	$A \to B$:	$E_{K_{AB}}(N_B)$

B generál egy N_B friss véletlenszámot, és nyíltan elküldi azt A-nak. A a K_{AB} kulccsal rejtjelezi N_B-t, ahol K_{AB} egy csak A és B számára ismert szimmetrikus kulcs. B dekódolja A válaszát, és ha visszakapta a korábban elküldött N_B számot, akkor A hitelesítette magát. A hitelesítés alapja, hogy (B-n kívül) csak A tud rejtjelezni K_{AB}-vel.

- **Partnerhitelesítés szimmetrikus kulcsú dekódolással:**

(1)	$B \to A$:	$E_{K_{AB}}(N_B)$
(2)	$A \to B$:	N_B

B generál egy N_B friss véletlenszámot, rejtjelezi azt a K_{AB} kulccsal, ahol K_{AB} egy csak A és B számára ismert szimmetrikus kulcs, és az eredményt elküldi A-nak. A dekódolja B üzenetét, majd visszaküldi a dekódolás eredményét B-nek. B ellenőrzi A válaszát, és ha az megegyezik az N_B számmal, akkor A hitelesítette magát. A hitelesítés alapja, hogy (B-n kívül) csak A tud dekódolni K_{AB}-vel.

- **Partnerhitelesítés digitális aláírással:**

(1)	$B \to A$:	N_B
(2)	$A \to B$:	$S_A(N_B)$

B generál egy N_B friss véletlenszámot, és nyíltan elküldi azt A-nak. A digitálisan aláírja N_B-t. B ellenőrzi A aláírását, és ha az aláírás hiteles, akkor A hitelesítette magát. A hitelesítés alapja, hogy csak A tud érvényes digitális aláírást generálni A nevében N_B-n.

- **Partnerhitelesítés nyilvános kulcsú rejtjelezéssel:**

(1)	$B \to A$:	$E_A(N_B)$
(2)	$A \to B$:	N_B

B generál egy N_B friss véletlenszámot, rejtjelezi azt A nyilvános kulcsával, és az eredményt elküldi A-nak. A dekódolja B üzenetét saját privát kulcsával, majd visszaküldi a dekódolás eredményét B-nek. B ellenőrzi A válaszát, és ha az megegyezik az N_B számmal, akkor A hitelesítette magát. A hitelesítés alapja, hogy csak A tud dekódolni A privát kulcsával.

Az alkalmazás jellegétől függ, hogy a fenti építőelemeket hogyan lehet felhasználni a gyakorlatban. Ha csak egyirányú partnerhitelesítésre van szükség, akkor a fenti protokollok önmagukban is megállják a helyüket. Ezzel ellentétben, egy olyan alkalmazásban, ahol kölcsönös partnerhitelesítésre van szükség a fenti protokollokat esetleg további elemekkel kell kiegészíteni és körültekintően kell kombinálni. Példaként próbáljunk meg a fenti, szimmetrikus kulcsú rejtjelezésre épülő egyirányú protokoll-részletből kölcsönös partnerhitelesítést végző protokollt konstruálni. Naivan úgy gondolhatnánk, hogy a protokollrészletet mindkét irányba lejátszva a feladatot megoldottuk:

(1)	$A \rightarrow B$:	N_A
(2)	$B \rightarrow A$:	$E_{K_{AB}}(N_A) \mid N_B$
(3)	$A \rightarrow B$:	$E_{K_{AB}}(N_B)$

Vegyük észre azonban, hogy a fenti protokoll hibás. Egy támadó A üzeneteit visszajátszva, meg tudja személyesíteni B-t A felé. A támadás menete a következő:

$$
\begin{array}{lll}
A \rightarrow X_B: & N_A & \\
\quad X_B \rightarrow A: & N_A & \\
\quad A \rightarrow X_B: & E_{K_{AB}}(N_A) \mid N'_A & \\
X_B \rightarrow A: & E_{K_{AB}}(N_A) \mid N_X & \\
A \rightarrow X_B: & E_{K_{AB}}(N_X) &
\end{array}
$$

A fenti támadásban az X_B támadó rejtjelező orákulumként használja A-t. Ezt úgy éri el, hogy visszajátsza A N_A kihívását A-nak B nevében. Ekkor A előállítja $E_{K_{AB}}(N_A)$-t. X-nek pontosan erre van szüksége az első protokoll példány befejezéséhez. Hasonló módon A is megszemélyesíthető B felé.

A protokollt a következő módon javíthatjuk ki:

(1)	$A \rightarrow B$:	N_A
(2)	$B \rightarrow A$:	$E_{K_{AB}}(A \mid N_A) \mid N_B$
(3)	$A \rightarrow B$:	$E_{K_{AB}}(B \mid N_B)$

A javítás lényege, hogy a rejtjelezett üzenetekben jelöljük, hogy azok kinek szólnak. Így A üzenete nem játszható vissza B nevében, s a fenti támadás nem működik.

3.4. Kulcscsere protokollok

Szimmetrikus kulcsú rejtjelezés vagy üzenethitelesítő kódok használata esetén szükséges, hogy a kommunikáló felek rendelkezzenek egy titkos kulccsal, amit rajtuk kívül más nem ismer. Ebben a szakaszban azt vizsgáljuk, hogyan lehet ezt a közös titkos kulcsot a kommunikáló felek birtokába juttatni.

A kulcscsere probléma koncepcionálisan legegyszerűbb megoldása az, mikor a kommunikáló felek fizikailag (pl. személyesen) találkoznak és megegyeznek egy

közös kulcsban. Ez a megoldás — amelyet manuális kulcscserének is hívnak — azonban csak korlátozott mértékben használható a gyakorlatban, mert drága és időigényes, továbbá sokszor az alkalmazás jellegénél fogva egyszerűen nem is használható.

A kulcscsere probléma gyakorlatban is jól használható megoldását a kulcscsere protokollok jelentik. Egy kulcscsere protokoll lehetővé teszi két (vagy több) fél számára egy közös titok létrehozását anélkül, hogy a két fél fizikailag találkozna.

A kulcscsere protkolloknak alapvetően két fajtája létezik: **kulcsszállító** és **kulcsmegegyezés** protokollok. Kulcsszállító protokollok esetében a kulcsot a protokoll valamelyik résztvevője (az egyik fél vagy egy megbízható harmadik fél) generálja, majd azt biztonságosan eljuttatja a többi résztvevőnek. A kulcs értéke tehát egy résztvevőtől függ. Ezzel szemben, kulcsmegegyezés protokollok esetében a kulcs értékéhez minden résztvevő hozzájárul. A résztvevők az általuk generált hozzájárulásokat kicserélik, majd minden résztvevő lokálisan generálja a közös kulcsot a másik résztvevőtől kapott hozzájárulást is felhasználva.

A kulcsmegegyezés protokollok előnye, hogy a kulcs értékét egyik fél sem tudja befolyásolni, hiszen az a másik fél hozzájárulásától is függ. Kulcsszállító protokollok esetében az a résztvevő, amelyik a kulcsot generálja, szándékosan választhat egy speciális tulajdonságokkal rendelkező (pl. valamilyen értelemben gyenge) kulcsot. A kulcsmegegyezés protokollok hátránya, hogy megvalósításuk általában nyilvános kulcsú kriptográfiára épül, így végrehajtásuk nagyobb számítási kapacitást igényel a résztvevőktől. Ennek ellenére, a fent említett kedvező tulajdonságuk miatt, gyakorlati alkalmazásokban gyakran használnak kulcsmegegyezés protokollokat.

A kulcscsere protokollok által nyújtott fontosabb szolgáltatások a következők:

- **Implicit kulcshitelesítés:** Egy kulcscsere protokoll akkor nyújt implicit kulcshitelesítés szolgáltatást valamely *A* fél számára, ha a protokoll sikeres lefutása után *A* meg lehet győződve arról, hogy rajta kívül csak a feltételezett másik fél, mondjuk *B*, és esetleg egy megbízható harmadik fél (pl. a kulcsszerver) férhet hozzá a protokoll során létrehozott kulcshoz. Ez egy olyan alapvető szolgáltatás, amit minden kulcscsere protokolltól elvárunk. Fontos megjegyezni azt, hogy implicit kulcshitelesítés esetén *A* nem feltétlenül biztos abban, hogy *B* ismeri a kulcsot. Csupán annyit követelünk meg, hogy *A* biztos legyen abban, hogy csak *B*-nek (és esetleg egy megbízható harmadik félnek) van meg a lehetősége arra, hogy a kulcshoz hozzáférjen.

- **Kulcskonfirmáció:** Ez az a szolgáltatás, melynek segítségével az egyik résztvevő, mondjuk *A*, meggyőződhet arról, hogy a másik résztvevő, mondjuk *B*, valóban birtokában van a protokoll futása során létrehozott kulcsnak. Ezen szolgáltatás megvalósítására több lehetőség is van: *B* elküldheti például *A*-nak a kulcs hash értékét, vagy egy, a kulccsal rejtjelezett publikus nyílt szöveget. Lenyomat küldése esetén, a hash függvény egyirányúsága miatt, egy támadó nem tudja megfejteni a kulcsot a megfigyelt hash érték-

ből. Ismert nyílt szöveg rejtjelezése esetén pedig a rejtjelező függvény ismert nyílt szövegű támadás elleni ellenállóképessége biztosítja ugyanezt.

- **Explicit kulcshitelesítés:** Explicit kulcshitelesítésről akkor beszélünk, ha a protokoll egyszerre biztosítja az implicit kulcshitelesítés és a kulcskonfirmáció szolgáltatásokat ugyanazon fél számára.

- **Kulcsfrissesség:** Ha a protokoll kulcsfrissesség szolgáltatást nyújt az A résztvevő számára, akkor a protokoll sikeres futása után A meg van győződve arról, hogy a létrehozott kulcs új, és nem egy korábban már használt és feltehetően azóta feltört kulcs.

A továbbiakban kulcsszállító és kulcsmegegyezés protokollokra mutatunk példákat.

A módosított Otway–Rees-protokoll

Első példánk az Otway–Rees-protokoll egy módosított változata. A protokoll a kulcsszállító protokollok családjába tartozik. Három résztvevője van, akiket A-val, B-vel és S-sel jelölünk. A és B szeretne a protokoll segítségével egy közös titkos kulcsot létrehozni; S egy megbízható kulcsszerver, aki a kulcsot generálja és eljuttatja A-hoz és B-hez. Feltesszük, hogy A és S, illetve B és S már rendelkezik egy közös kulccsal, amit K_{AS}-sel, illetve K_{BS}-sel jelölünk. Ezeket a kulcsokat használja a protokoll a frissen generált kulcs védelmére.

Módosított Otway-Rees protokoll		
(1)	$A \to B:$	$A \mid N_A$
(2)	$B \to S:$	$A \mid B \mid N_A \mid N_B$
(3)	$S \to B:$	$E_{K_{AS}}(N_A \mid A \mid B \mid K) \mid E_{K_{BS}}(N_B \mid A \mid B \mid K)$
(4)	$B \to A:$	$E_{K_{AS}}(N_A \mid A \mid B \mid K)$

A protokoll működése a következő: A kezdeményezi a protokoll futtatását azzal, hogy generál egy friss véletlen számot, N_A-t, s elküldi azt saját A azonosítójával együtt B-nek. B hasonlóan generál egy friss véletlenszámot, N_B-t, és elküldi azt N_A-val valamint az A és a B azonosítókkal együtt az S szervernek. S generál egy friss K kulcsot, majd előállít két rejtjeles üzenetrészt, az egyiket K_{AS}-sel kódolva A számára, a másikat pedig K_{BS}-sel kódolva B számára. Mindkét rejtjeles üzenetrész tartalmazza a K kulcsot, valamint A és B azonosítóját. Az A-nak szóló rész tartalmazza ezen kívül az N_A számot, míg a B-nek szóló rész tartalmazza az N_B számot. S mindkét üzenetrészt B-nek küldi el, majd B továbbítja az A-nak szóló részt A-nak. Ezután mindkét fél dekódolja a neki szóló részt, és ellenőrzi az azonosítókat, valamint azt, hogy visszakapta-e a korábban elküldött véletlenszámot. Amennyiben az ellenőrzések sikeresek, a felek elfogadják a K kulcsot.

A protokoll implicit kulcshitelesítést biztosít mindkét fél számára. Ez azért van így, mert a felek a K kulcsot rejtjelezve kapják, továbbá megbíznak a szerverben,

hogy az csak az üzenetben megjelölt feleknek (azaz A-nak és B-nek) küldte el a kulcsot. Nem lehetnek azonban biztosak abban, hogy a másik fél valóban a kulcs birtokába jutott, ezért a kulcshitelesítés nem explicit. A kulcsfrissességet az N_A és N_B véletlenszámok megjósolhatatlansága biztosítja. Pontosabban, A tudja, hogy a szerver csak azután küldhette üzenetét, hogy megkapta N_A-t, hiszen korábban nem sejthette N_A értékét. Azaz a kulcs nem lehet régebbi, mint N_A. Hasonlóképpen, B tudja, hogy a kulcs nem régebbi, mint N_B.

Rejtjelezett kulcs aláírása

Az előző protokoll a kulcsfrissességet nem-predikálható véletlenszámok segítségével biztosította és szimmetrikus kulcsú rejtjelezést használt a kulcs bizalmas átvitelére. Most egy olyan kulcsszállító protokollt mutatunk be, amely időpecsétet használ a frissesség biztosítására, és nyilvános kulcsú rejtjelezést alkalmaz a bizalmasság megőrzése érdekében. Megjegyezzük, hogy az időpecsét alkalmazása feltételezi, hogy a résztvevők órái szinkronizálva vannak.

Rejtjelezett kulcs aláírása
(1) $A \rightarrow B:\quad A \mid E_B(k) \mid T_A \mid S_A(B \mid E_B(k) \mid T_A)$

A protokollnak két (on-line) résztvevője van, A és B, akik egy kulcsot szeretnének létrehozni egymás között. Feltételezzük, hogy mindkét fél ismeri a másik fél nyilvános kulcsát. A generál egy friss K kulcsot, rejtjelezi azt B nyilvános kulcsával, majd aláírja a rejtjelezett kulcsot, B azonosítóját, és egy T_A időpecsétet. Végül, A elküldi B-nek a rejtjelezett kulcsot, az időpecsétet, és az aláírást. B először ellenőrzi az időpecsétet és az aláírást (utóbbihoz használja A nyilvános kulcsát), majd dekódolja a rejtjelezett kulcsot saját privát kulcsával.

A protokoll mindkét fél számára biztosítja a kulcsfrissességet. Ezen kívül, a protokoll mindkét fél számára implicit kulcshitelesítést biztosít. Ugyanis A tudja, hogy a rejtjelezett kulcsot csak B tudja dekódolni, de nem tudja biztosan, hogy B megkapta-e az üzenetet. B az aláírásból tudja, hogy az üzenetet A küldte, de mivel az aláírás csak a rejtjelezett kulcsot tartalmazza, ezért B nem lehet biztos abban, hogy A ismeri a kulcs értékét. Lehetséges, hogy A lehallgatott egy protokoll példányt, amit valamikor C futtatott B-vel, és szert tett $E_B(K)$-ra, de nem ismeri K-t. A később bármikor aláírhatja $E_B(K)$-t egy friss időpecséttel együtt.

A fenti protokollt egyszerűen módosíthatjuk úgy, hogy explicit kulcshitelesítést biztosítson B számra:

Rejtjelezett és aláírt kulcs
(1) $A \rightarrow B:\quad A \mid E_B(K) \mid T_A \mid S_A(B \mid K \mid T_A)$

Ebben a protokollban már nyilvánvaló B számára, hogy A a K kulcs birtokában van, hiszen az aláírásban szerepel K. Ekkor természetesen feltételezzük, hogy az aláírásból a K kulcs nem fejthető meg. Ez praktikusan azt jelenti, hogy a hash-és-aláír módszert alkalmazzuk. A protokoll tulajdonságainak vizsgálatát az olvasóra bízzuk.

A Diffie–Hellman-protokoll

A Diffie–Hellman-protokoll egy kulcsmegegyezés protokoll. Két résztvevője van, akiket A-val és B-vel jelölünk. A és B szerepe teljesen szimmetrikus. Feltesszük, hogy mindkét fél számára ismert egy nagy p prímszám, és a Z_p^* véges csoport egy g primitív eleme (a primitív elem definícióját lásd a 2.3 szakaszban), ahol Z_p^* a p által definiált véges multiplikatív csoportot jelöli, azaz az $\{1,2,\ldots,p-1\}$ halmazt a $\bmod p$ szorzással mint művelettel. A generál egy x véletlen számot, melyre $1 \leq x \leq p-2$, majd kiszámolja $g^x \bmod p$ értékét, és az eredményt elküldi B-nek. Hasonlóan, B generál egy y véletlen számot, melyre $1 \leq y \leq p-2$, majd kiszámolja $g^y \bmod p$ értékét, és az eredményt elküldi A-nak. Ezek után a B-től kapott $g^y \bmod p$ értéket és saját titkos x számát felhasználva, A kiszámolja $(g^y)^x \bmod p$ értékét. B hasonlóan kiszámolja $(g^x)^y \bmod p$ értékét. A hatványozás kommutativitása miatt azonban mindkét fél ugyanazt a $k = g^{xy} \bmod p$ számot kapja, és ez lesz (vagy további számítással ebből kapható) a közös kulcs.

Diffie–Hellman-protokoll		
(1)	$A \to B$:	$g^x \bmod p$
(2)	$B \to A$:	$g^y \bmod p$
(3a)	A :	$K = (g^y)^x \bmod p = g^{xy} \bmod p$
(3b)	B :	$K = (g^x)^y \bmod p = g^{xy} \bmod p$

A Diffie–Hellman-protokoll biztonsága a Diffie–Hellman-probléma és a diszkrét logaritmus probléma nehézségén alapszik. A Diffie–Hellman-probléma a következő: adott egy p prím, a Z_p^* csoport egy g generátor eleme, valamint a $g^x \bmod p$ és $g^y \bmod p$ számok; számoljuk ki $g^{xy} \bmod p$ értékét. Ehhez szorosan kapcsolódik a diszkrét logaritmus probléma, mely a következőképpen szól: adott egy p prím, a Z_p^* csoport egy g generátor eleme, és egy $X \in Z_p^*$ szám; adjuk meg azt az $x \in Z_p^*$ számot, melyre $g^x \bmod p = X$. A két probléma közötti kapcsolatról annyit tudunk, hogy ha tudnánk hatékonyan diszkrét logaritmust számolni, akkor a Diffie–Hellman-problémára is hatékony megoldást kapnánk: először $g^x \bmod p$-ből meghatározzuk x-et, majd kiszámítjuk $(g^y)^x \bmod p$ értékét. A fordított irány nem bizonyított, ezért nem tudjuk, hogy a két probléma ekvivalens-e. Mindenesetre, úgy sejtik, hogy mindkét probléma nehéz, abban az értelemben, hogy nem lehet polinom időben megoldani őket. A pontos komplexitása azonban egyik problémának sem ismert.

A Diffie–Hellman-protokoll egyszerű és elegáns. Sajnos azonban a protokoll nem biztosít kulcshitelesítést, ezért csak passzív támadóval szemben biztonságos. Ha az aktív támadás lehetőségét nem lehet kizárni, akkor A és B nem lehetnek biztosak abban, hogy egymással hozták létre a közös K kulcsot, mert egy támadó kettőjük közé ékelődve (man-in-the-middle), mindkét féllel futtatni tudja a protokollt. A Diffie–Hellman-protokollt többféleképpen is ki lehet egészíteni kulcshitelesítéssel. A protokoll ezen kiegészített változatai igen széleskörben használtak a gyakorlatban (pl. az SSH, az SSL és az IPSec protokollokban).

3.5. Alkalmazások

GSM

A mobil kommunikációs rendszerek adatbiztonsági tervezés szempontjából speciálisak a következő okok miatt:

- fizikailag hozzáférhető rádiós csatornán is folyik kommunikáció,
- a mobil készülék általában kis számítási és tárolási kapacitással rendelkezik,
- a mobil készülék időben változtatja helyét.

Ennek megfelelően a GSM rendszerben lehetőség van a rádiós csatornán keresztül zajló kommunikáció rejtjelezésére, valamint a rádiós interfészen keresztül elérhető szolgáltatásokhoz (pl. híváskezdeményezés) történő hozzáférés korlátozására. A tervezők figyelembe vették a mobil készülékek kis számítási kapacitását is, ezért a védelem teljes egészében szimmetrikus kulcsú kriptográfiára épül, ami általában nagyságrendekkel kisebb számítási kapacitást igényel, mint a nyilvános kulcsú algoritmusok. A mobilitás támogatását úgy oldották meg, hogy az előfizetők idegen hálózatok felé (azaz roaming közben) is hitelesíteni tudják magukat.

A GSM rendszer biztonsági architektúrájában fontos szerepet játszik a mobil készülékekben található SIM kártya (Subscriber Identity Module). A SIM kártya egy intelligens chip-kártya, mely képes adatokat tárolni és azokon számításokat végezni. A SIM kártyára úgy is gondolhatunk, mint egy, a mobil készülék belsejében elhelyezett mini-számítógépre.

A SIM kártya lényegében az előfizetőt képviseli a GSM rendszerben (azaz az előfizető helyett végez kriptográfiai műveleteket). A SIM kártya tárolja többek között az U előfizető és a HN hazai hálózat (home network) közötti, hosszú élettartamú K_U szimmetrikus kulcsot. A SIM kártya ezen kulcs segítségével hitelesíti az előfizetőt a hálózat felé, mikor az előfizető egy hívást kezdeményez. Továbbá, a SIM kártya a K_U kulcs felhasználásával számolja ki a kommunikáció rejtjelezésére használt kapcsolatkulcsot is. Mivel minden biztonsági algoritmus a SIM kártyán van implementálva, ezért a K_U kulcs soha nem hagyja el a SIM kártyát.

A GSM rendszer a következő kihívás–válasz alapú partnerhitelesítő protokollt használja az előfizető hitelesítésére:

GSM előfizető-hitelesítés (egyszerűsített)		
(1)	$SIM_U \rightarrow VN$:	$IMSI$
(2)	$VN \rightarrow HN$:	$IMSI$
(3)	$HN \rightarrow VN$:	$RAND \mid SRES \mid K_C$
(4)	$VN \rightarrow SIM_U$:	$RAND$
(5)	$SIM_U \rightarrow VN$:	$SRES'$

A fenti protokollban VN az idegen hálózatot (visited network) jelöli, melyben az U előfizető roaming közben szeretne hívást kezdeményezni. U SIM kártyája,

SIM_U, elküldi U nemzetközi *IMSI* azonosítóját (International Mobile Subscriber Identity) VN-nek. VN nem tudja közvetlenül hitelesíteni U-t, mert nem ismeri a K_U kulcsot. Ezért tovább küldi az *IMSI* azonosítót az előfizető hazai hálózatának, HN-nek. Ezzel lényegében U hitelesítéséhez szükséges információkat kér HN-től. HN generál egy *RAND* véletlenszámot, melyet kihívásnak szán U felé, kiszámítja a kihívásra adandó helyes $SRES = f_{K_U}(RAND)$ választ, és generál egy friss $K_C = g_{K_U}(RAND)$ kapcsolatkulcsot, ahol f és g két alkalmas függvény, melyet a hazai szolgáltató akár maga választhat. Ezek után HN elküldi a $RAND|SRES|K_C$ hármast VN-nek.

VN elküldi a *RAND* véletlenszámot SIM_U-nak, amely K_U ismeretében kiszámítja az $SRES' = f_{K_U}(RAND)$ választ és a K_C kapcsolatkulcsot. Végül, SIM_U elküldi az $SRES'$ választ VN-nek, amely összehasonlítja azt a HN-től kapott *SRES* értékkel. Egyezés esetén U hitelesítése sikeres, és VN engedélyezi a szolgáltatáshoz történő hozzáférést. A továbbiakban U és VN a K_C kulcsot használják a mobil készülék és a bázisállomás közötti forgalom rejtjelezésére. A rejtjelezés egy, a GSM szabványban specifikált kulcsfolyam rejtjelező segítségével történik.

SSL

Az SSL (Secure Socket Layer) protokoll lehetővé teszi biztonságos TCP kapcsolatok kiépítését tetszőleges, amúgy TCP-t használó alkalmazások (pl. HTTP, FTP, SMTP, stb.) között. Kifejlesztésének fő motivációja a Web-böngésző és a Web-szerver közötti kommunikáció biztonságossá tétele volt, ezért szokták a Web biztonsági protokolljának is nevezni.

Az SSL protokoll több alprotokollból áll, melyek közül a két legfontosabb alprotokoll a következő:

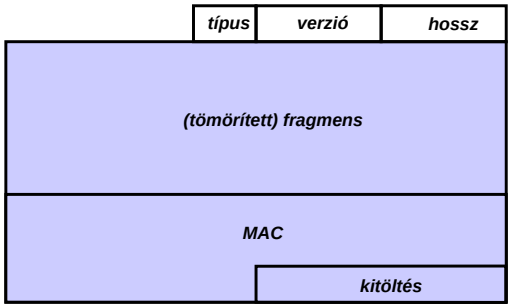
- **Record protokoll**: A Record protokoll feladata a kliens és a szerver (pl. egy Web-böngésző és egy Web-szerver) közötti kommunikáció védelme, mely titkosítást, integritásvédelmet és üzenetvisszajátszás elleni védelemet jelent.
- **Handshake protokoll**: A kliens és a szerver a Handshake protokoll segítségével egyezteti a Record protokollban használt kriptográfiai algoritmusokat és az algoritmusok paramétereit, beleértve a kulcsokat is. A Handshake protokoll további feladata a felek hitelesítése. Tipikusan a szerver mindig hitelesíti magát, a kliens hitelesítése azonban csak opcionális.

A továbbiakban tömören ismertetjük a Record és a Handshake protokoll működését.

Az SSL Record protokoll

Az SSL Record protokoll a felsőbb protokoll rétegektől érkező üzeneteket a következő lépéseken keresztül dolgozza fel:

- a hosszú üzeneteket fragmentálja,



3.19. ábra. Az SSL Record protokoll üzenetformátuma.

- a fragmenseket tömöríti,
- minden tömörített fragmenst fejléccel lát el,
- a fejléccel ellátott tömörített fragmensre üzenethitelesítő kódot (MAC) számol és azt a fragmenshez csatolja,
- majd az üzenethitelesítő kóddal ellátott tömörített fragmenst rejtjelezi.

A vevő a feldolgozást értelemszerűen ellentétes sorrendben végzi. A feldolgozás során használt tömörítő, üzenethitelesítő, és rejtjelezési algoritmusokban, paraméterekben, illetve kulcsokban az SSL Handshake protokoll végrehajtása során egyeznek meg a felek. A fenti lépések eredményeként előálló Record üzenet formátumát a 3.19. ábra mutatja.

Az üzenethitelesítő kód generálása a HMAC egy korai verziójával történik az alábbiak szerint:

$$MAC = H(K_{write}^{MAC} \mid pad_2 \mid H(K_{write}^{MAC} \mid pad_1 \mid seqnum \mid type \mid length \mid payload))$$

ahol:

- H az MD5 vagy a SHA-1 hash függvény, attól függően, hogy a Handshake során miben egyeztek meg a felek.
- K_{write}^{MAC} a MAC érték generálásához használt titkos kulcs, melyet csak a kliens és a szerver ismer. Az SSL különböző irányokban különböző kulcsot használ, azaz a klienstől a szerver felé tartó üzenetek integritását egy $K_{C\to S}^{MAC}$ kulccsal, a szervtől a kliens felé tartó üzenetek integritását pedig egy $K_{S\to C}^{MAC}$ kulccsal védi. A kliens a $K_{C\to S}^{MAC}$ kulcsra K_{write}^{MAC} néven hivatkozik (küldésre használt MAC kulcs), a $K_{S\to C}^{MAC}$ kulcsra pedig a K_{read}^{MAC} néven (vételre használt MAC kulcs). A szerver oldalon az egyes kulcsok szerepe nyilván fordított.
- pad_1 és pad_2 két konstans bájtsorozat.

- *seqnum* az üzenet sorszáma. Az SSL implicit üzenetsorszámot használ, ami azt jelenti, hogy a sorszám nem jelenik meg explicit mezőként az üzenetben (lásd 3.19. ábra), de a MAC számításban felhasználják aktuális értékét, melyet mindkét fél lokálisan számontart. Ezt azért lehet így megtenni, mert az SSL Record protokoll a TCP protkoll felett helyezkedik el, és a TCP normális körülmények között biztosítja az üzenetek sorrendhelyes vételét. Ezért általában igaz az, hogy mindig a várt sorszámú üzenetet veszik a felek. Ha valamilyen támadás vagy hiba folytán nem a várt sorszámú üzenet érkezik meg (ami tehát abnormális jelenség), akkor a MAC ellenőrzés sikertelen lesz és a vevő bontja a kapcsolatot.

- *type* és *length* az üzenet fejlécében található típus és hossz mezők értéke.

- *payload* az üzenetben található (tömörített) fragmens.

A 3.19. ábrán a Record üzenet rejtjelezett részét szürke színnel jelöltük. Mint látható, a fejléc kivételével az egész üzenet rejtjelezve van. Az SSL alapértelmezett rejtjelező algoritmusa az RC4 kulcsfolyam rejtjelező, de a Handshake protokoll végrehajtása során a felek más algoritmusban is megegyezhetnek. Az SSL támogatja még az RC2, a DES, a három kulcsos 3DES, az IDEA, és a Fortezza blokkrejtjelezők CBC módban történő használatát. Amennyiben a felek valamelyik blokkrejtjelező használatában egyeznek meg, úgy a rejtjelezés előtt minden Record üzenetet ki kell tölteni, hogy hossza a rejtjelező blokkméretének egész számú többszöröse legyen. A kitöltés bájtjai a MAC után kerülnek az üzenetbe. A MAC számításhoz hasonlóan, a felek különböző irányokban különböző kulcsokat használnak a rejtjelezéshez is. Ennek megfeleően a kliens egy $K_{C\to S}$ kulccsal rejtjelezi üzeneteit, a szerver pedig egy $K_{S\to C}$ kulcsot használ. A kliens a $K_{C\to S}$ kulcsra K_{write} néven, a $K_{S\to C}$ kulcsra pedig K_{read} néven hivatkozik. A szerver oldalon az egyes kulcsok szerepe nyilván fordított.

Blokkrejtjelező használata esetén, a CBC mód miatt, szükség van IV-re is. Az első üzenet rejtjelezéséhez használt IV-t a Handshake során generálják a felek (a kulcsokkal együtt). Minden további üzenetnél az előző üzenet utolsó rejtjeles blokkját használják IV-nek.

Az SSL Handshake protokoll

Az SSL Handshake protokoll egy komplex protokoll. A komplexitás legfőbb oka az, hogy a Handshake protokoll többféle kulcscsere módszer használatát is támogatja, és a Handshake üzenetek értelmezése az egyes módszerek esetén más és más. Ezért itt most nem is vállalkozunk az SSL Handshake protokoll részletes ismertetésére, csupán nagy vonalakban összefoglaljuk a főbb lépéseit.

Az SSL Handshake protokoll négy fázisra tagolódik. Az első fázisban történik meg a Record protokollban használandó algoritmusok egyeztetése, valamint a Handshake hátralevő részében használt kulcscsere módszer kiválasztása. Ezen kívül a felek az első fázisban kicserélnek két frissen generált véletlen számot is,

melyet a további számításoknál (pl. a kulcsok generálásában) használnak majd. A második fázisban a szerver a kiválasztott módszernek megfelelően végrehajtja a kulcscsere ráeső részét, míg a kliens a harmadik fázisban teszi meg ugyanezt. A harmadik fázis után, az addig kicserélt információt felhasználva, mindkét fél előállítja a közös mestertitkot, majd abból generálja az új kapcsolatkulcsokat. A negyedik fázisban a felek áttérnek az új algoritmusok és kulcsok használatára. Továbbá a mestertitok felhasználásával kiszámolnak egy kriptográfiai ellenőrzőösszeget az összes addig a pontig küldött és vett Handshake üzenet együttesére, és elküldik egymásnak ezeket az ellenőrzőösszegeket. Ily módon igyekeznek detektálni a Handshake során egy harmadik fél által esetlegesen végrehajtott aktív támadásokat.

Mint említettük, az SSL több kulcscsere módszert is támogat. Ezek közül a fontosabbak a következők:

- **RSA alapú kulcscsere**: RSA alapú kulcscsere esetén a kliens generál egy 48 bájt méretű véletlen blokkot, amelyet a szerver nyilvános RSA kulcsával kódolva elküld a szervernek. Ebből a véletlen blokkból aztán mind a kliens, mind a szerver előállítja a közös mestertitkot.

- **Fix Diffie–Hellman-kulcscsere**: Fix Diffie–Hellman-kulcscsere esetén a felek a Diffie–Hellman-algoritmust használják, és az ennek eredményeként kialakult közös értékből generálják a mestertitkot. A fix jelző arra utal, hogy a szerver Diffie–Hellman-paraméterei (p, g, $g^x \bmod p$) fixek, azokat egy hitelesítés szolgáltató aláírta, és az erről szóló tanúsítványt a kliens ellenőrizni tudja.

- **Egyszeri Diffie–Hellman-kulcscsere**: Az előző módszerhez hasonlóan Diffie–Hellman-algoritmust használnak a felek, de a szervernek nincsenek fix, aláírt Diffie–Hellman-paraméterei. Helyette a szerver egyszer használatos paramétereket generál, és azokat RSA vagy DSS aláíró kulcsával aláírva juttatja el a kliensnek. A kliens szintén egyszer használatos Diffie–Hellman nyilvános értéket generál a szerver p és g paramétereit használva.

- **Anonim Diffie–Hellman-kulcscsere**: Ezen módszer használata esetén a felek az eredeti, hitelesítés nélküli Diffie–Hellman-algoritmust hajtják végre. Ezen módszer használata nem tanácsos, ha az aktív támadások veszélyét nem lehet kizárni.

A kulcscsere végrehajtása után a kliens és a szerver előállít egy közös K mestertitkot. Ez a következő módon történik. Jöljük K'-vel a kliens által generált 48 bájtos véletlen blokkot, melyet RSA alapú kulcscsere esetén a kliens a szerver nyilvános RSA kulcsával rejtjelezve juttat el a szervernek. Fix, egyszer használatos, vagy anonim Diffie–Hellman-kulcscsere esetén pedig K' jelölje a Diffie–Hellman-algoritmus $g^{xy} \bmod p$ végeredményét. Ekkor

$$K = \mathrm{MD5}(K' \mid \mathrm{SHA1}(\text{„A”} \mid K' \mid N_C \mid N_S)) \mid$$

$$\text{MD5}(K' \mid \text{SHA1}(\text{„BB”} \mid K' \mid N_C \mid N_S)) \mid$$
$$\text{MD5}(K' \mid \text{SHA1}(\text{„CCC”} \mid K' \mid N_C \mid N_S))$$

ahol N_C és N_S a Handshake első fázisában kicserélt véletlen számok, „A”, „BB” és „CCC” pedig az „A”, a „B”, illetve a „C” ASCII karakterekből alkotott egy, kettő, illetve három hosszú karakterláncok. Mivel az MD5 hash függvény kimenetének mérete 16 bájt, ezért a három hash érték összefűzéséből nyert K mestertitok 48 bájt hosszú. Ebből a mestertitokból generálja mindkét fél a Record protokollban használt MAC kulcsokat, rejtjelező kulcsokat és IV-ket.

Mint korábban említettük, a Handshake során megtörténik a szerver, és opcionálisan a kliens hitelesítése is. A szerver hitelesítése tipikusan a szerver tanúsítvány ellenőrzésével történik. A kliens hitelesítéséhez a kliensnek is szüksége van tanúsítványra. Mivel azonban a gyakorlatban a legtöbb kliensnek (felhasználónak) nincsen tanúsítványa, ezért a kliens általában nem hitelesíti magát az SSL protokoll szintjén. Ha az alkalmazás jellege mégis szükségessé teszi a klienshitelesítést (pl. home banking), akkor azt maga az alkalmazás végzi el (pl. jelszó alapú hitelesítés).

DigiCash

A DigiCash egy elektronikus fizetési rendszer, mely a hagyományos készpénz modellje szerint működik. Ebben a rendszerben, az elektronikus készpénz (e-cash) egy bináris sorozat, melynek egyik számítógépről a másikra történő küldésével történik meg a fizetés. A DigiCash tervezésénél fontos szempont volt, hogy a hagyományos készpénzhez hasonlóan lehetővé tegye a nyomonkövethetetlen on-line vásárlást. Ez azt jelenti, hogy még az elektronikus érméket kibocsájtó bank sem képes nyomonkövetni, hogy hol, mit, és mennyiért vásárolnak az ügyfelei. A DigiCash ezen tulajdonsága élesen megkülönbözteti ezt a rendszert a hitelkártyás vásárlástól, ahol a bank pontosan ismeri ügyfelei vásárlási tranzakcióinak részleteit.

A DigiCash rendszer résztvevői a C vásárló, az M kereskedő, valamint a B on-line bank. Feltesszük, hogy a vásárlónak és a kereskedőnek számlája van a bankban. Feltesszük továbbá, hogy a bank rendelkezik (e_i, d_i) RSA kulcspárok egy sorozatával, ahol az egyes párok különböző pénzcímletekhez tartoznak (pl. $i \in \{1, 2, 5, 10, 20, 50, 100, \ldots\}$). Mikor a vásárló pénzt szeretne kivenni a számlájáról, hitelesíti magát a bank felé. Hasonlóképpen, mikor a kereskedő be akarja váltani a vásárló által elköltött érméket a banknál, akkor hitelesíti magát a bank felé. A vásárló és a kereskedő bank felé történő hitelesítése bármilyen parnerhitelesítési módszerrel történhet. Az alábbi DigiCash protokoll az egyszerűség kedvéért ezeket a hitelesítési lépéseket nem tartalmazza, csak azt szemlélteti, hogyan történik az elektronikus érme létrehozása, elköltése, és beváltása.

DigiCash protokoll (egyszerűsített)		
(1)	$C \to B$:	$v,\ b = s \cdot r^{e_v} \bmod n_v$
(2)	$B \to C$:	$b^{d_v} \bmod n_v$
(3)	$C \to M$:	$c = (v,\ s,\ s^{d_v} \bmod n_v)$
(4)	$M \to B$:	c
(5)	$B \to M$:	valid / invalid

A vásárló generál egy s (részben) véletlen sorozatszámot, amelyre teljesül, hogy az s-et reprezentáló bitsorozat bal oldali fele egyértelmű determinisztikus kapcsolatban van a jobb oldali felével. A vásárló kiválasztja a kívánt v címlethez tartozó e_v nyilvános kulcsot (és n_v modulust) a bank kulcsai közül, és választ egy csak általa ismert, titkos r véletlen számot, majd elküldi az (1) üzenetet a banknak.

A bank levonja a választott címletnek megfelelő összeget a vásárló számlájáról, majd az (1) üzenetben kapott b üzenetrészt aláírja az adott címletnek megfelelő d_v titkos kulcsával, s az eredményt a (2) üzenetben visszaküldi a vásárlónak. A vásárló a $b^{d_v} \bmod n_v = s^{d_v} \cdot r \bmod n_v$ és az r ismeretében egy osztással előállítja az $s^{d_v} \bmod n_v$ értéket, azaz a bank aláírását az s sorozatszámon. A $c = (v,\ s,\ s^{d_v} \bmod n_v)$ hármas az elektronikus érme. Ennek megfelelően leolvasható az értéke, a sorozatszáma, valamit hordozza a bank digitális aláírását.

A fizetés során a vásárló elküldi a megfelelő címletű elektronikus pénzek egy sorozatát a kereskedőnek. Az egyszerűség kedvéért tegyük fel, hogy a vásárló a (3) üzenetet küldi, mely csak egyetlen érmét tartalmaz. A kereskedő az érmét a (4) üzenteben elküldi a banknak, egyrészt ellenőrzésre, másrészt, hogy a bank a pénzt jóváírja a kereskedő számláján.

A bank először azt ellenőrzi, hogy az s sorozatszám nem szerepelt-e már egy korábbi ellenőrzésnél, azaz nincs-e szó pénzduplikálásról. Ha a bank duplikálást észlel, azaz a pénzt egyszer már elköltötték és beváltották, akkor jelezi a kereskedőnek, hogy az általa ellenőrzésre bemutatott érme másolat (invalid üzenet). Ellenkező esetben, a bank ellenőrzi, hogy az s és $s^{d_v} \bmod n_v$ összetartozó pár-e, azaz az érme a bank hiteles aláírását hordozza-e. Ezt a v címlethez tartozó nyilvános kulcs felhasználásával végzi. Ha az aláírás hiteles, akkor a bank tárolja az s sorozatszámot (későbbi ellenőrzések céljára), és azt ettől kezdve a forgalomból kivontnak tekinti. Ezután a bank jóváírja a megfelelő összeget a kereskedő számláján, és tájékoztatja a kereskedőt, hogy az érme beváltása sikeres volt (valid üzenet).

Figyeljük meg, hogy amikor a bank aláírását adja a b üzenetre, akkor b felépítése miatt nem tudja megállapítani a vásárló által éppen választott s sororozatszám értékét, hiszen az ismeretlen r véletlennel való moduláris szorzás elfedi azt. Tehát úgy ad aláírást a b üzenet aláírásra szánt s részére, hogy nem láthatja annak valódi értékét. Az ilyen módon történő aláírást **vak aláírás**nak nevezzük. A vak aláírás technika alkalmazása biztosítja, hogy a bank nem tudja nyomonkövetni a pénz mozgását, azaz mikor a kereskedő beváltja az s sorozatszámú érmét, akkor a bank nem tudja, hogy melyik vásárló számára írta alá ezt a sorozatszámot.

Láttuk, hogy a DigiCash rendszerben a pénzduplikálási kísérletek a banki ellenőrzésen fennakadnak. Ravaszabb csalási kísérlet lehet a banki aláírás hamisí-

tása. Ehhez a hamisítónak olyan értéket kellene találnia, amelyet a címletnek megfelelő banki nyilvános kitevőre emelve egy érvényes sorozatszámra jutna. Mivel azonban a sorozatszámoknak rögzített struktúrája van, megfelelő méretek mellett gyakorlatilag kizárható ezen támadás sikere.

3.6. Feladatok

3.1. feladat. Egy egyszerű lineáris rejtjelezőt konstruálunk az

$$y = (a \cdot x + b) \bmod n$$

lineáris transzformációval, ahol a és b a kulcs részei, x a nyílt szöveg, y a rejtett szöveg, továbbá n az ábécé mérete.

a) Adja meg a kulcstér méretét, ha $n = 256$.

b) A kulcsot szeretnénk megfejteni. Tegyük fel, hogy a forrás ideálisan tömörített, s így véletlen forrásként modellezhető. Rejtett szövegű támadásban is gondolkodhatunk?

c) Milyen információt kellene birtokolnunk egy véletlen forrás esetén a sikeres támadáshoz?

d) Mit mondhatunk az esetben, ha a forrás írott szöveg, s ismerjük a karakterek gyakoriságát?

3.2. feladat. Írott szövegek rejtjelezését betűpárokra alkalmazott lineáris rejtjelezéssel végezzük: $y = (a \cdot x + b) \bmod n^2$, ahol a és b a kódoló kulcs elemei, $(a, n^2) = 1$, továbbá n az ábécé mérete. Az ábécé elemeit a $\{0, 1, \ldots, n-1\}$ halmaz elemeinek feleltetjük meg. A nyílt és rejtett szöveg, azaz x illetve y a $\{0, 1, \ldots, n^2 - 1\}$, halmazból veszi értékét, olyan módon, hogy egy (u, v) forrás-betűpárt az $u \cdot n + v$ egészbe képezve áll elő egy x nyílt szöveg. Az y rejtett szöveget továbbítás előtt visszaképezzük betűpárba. A betűpárok összefogásától azt várjuk, hogy a rejtett szövegben az egyenletesebb betűpár gyakoriság érvényesül a gyakran nagyon inhomogén betűgyakoriság esetén is, ezáltal csökkentve egy statisztikai alapú támadás esélyét. Helyes-e ezen várakozásunk?

3.3. feladat. 8 bites karaktereket szeretnénk rejtjelezni Hill-rejtjelezővel. Definiáljon alkalmas kripto-rendszert a feladat megoldására! Adja meg a nyilt szövegek terét, a rejtett szövegek terét, és a kulcsteret. Válasszon egy alkalmas kulcsmátrixot, és számolja ki a dekódoláshoz szükséges inverz-kulcsot!

3.4. feladat. Tekintsük a következő rejtjel-rendszert: nyílt üzenetek tere $\mathcal{P} = \{a, b\}$, rejtett üzenetek tere: $\mathcal{C} = \{\alpha, \beta, \gamma, \delta, \varepsilon\}$, kulcstér: $\mathcal{K} = \{1, 2, 3, 4, 5\}$. A kódolást a következő táblázat írja le:

	1	2	3	4	5
a	α	β	γ	ε	δ
b	β	δ	α	γ	ε

Legyen a kulcs eloszlása $P_{\mathcal{K}} = \{\frac{2}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{10}, \frac{1}{10}\}$ és a nyílt szövegek eloszlása $P_{\mathcal{P}} = \{\frac{1}{3}, \frac{2}{3}\}$. Tökéletes-e a fenti rejtjelező? Miért?

3.5. feladat. Tekintsünk egy lineáris bináris blokk rejtjelezőt. Adja meg az ismert nyílt-rejtett párokon alapuló támadást a rejtjelező ellen! Van-e megkötés a párokra?

3.6. feladat. Legyen $\overline{x}$ az x bináris blokk bitenkénti komplemense. $E_K(x)$ jelölje x DES kódolását K kulcs mellett. Igazoljuk, ha $y = E_K(x)$, akkor $\overline{y} = E_{\overline{K}}(\overline{x})$, azaz egyszerre komplementálva az üzenet, rejtett szöveg és kulcs blokkokat, köztük a rejtjelező kódolási leképezés kapcsolat változatlanul fennáll!

3.7. feladat. Tegyük fel, hogy DES rejtjelezést használtunk 64 bites üzenet blokkok rejtjelezésére, amelyek 8 bites karakterekből állnak, s a 8. bit paritásbit. Elvben végrehajtható-e a kimerítő kulcskeresés támadás csak rejtett szövegek megfigyelésére alapozva (azaz várhatóan sikerül-e kiválasztani a keresett kulcsot)? Hány rejtjeles blokkot kellene megfigyelni ehhez? (A DES kódolást és dekódolást véletlen függvényként modellezheti.)

3.8. feladat. Fermat-álprím-e

a) $n = 87$ a $b = 5$ bázisra nézve?

b) $n = 33$ a $b = 32$ bázisra nézve?

3.9. feladat. Egy játék RSA algoritmus esetén: $p = 5, q = 11$ prímeket választottuk. Adja meg a legkisebb kódoló kulcsot, s az ehhez tartozó dekódoló kulcsot!

3.10. feladat. Alice publikus RSA kulcsa a következő: $(n = 11413, e = 9)$. Rejtjelezze az $m = 9726$ üzenetet Alice számára!

3.11. feladat. Támadjon egy RSA rejtjelezőt, amelyről a nyilvános $n = 4003997$ modulus valamint az $e = 379$ kulcs mellett megtudja a $\varphi(n) = 3999996$ értéket is.

a) Számítsa ki a d dekódoló kulcsot!

b) Adja meg az n prímfaktorait!

3.12. feladat. Ha két, RSA rejtjelezéssel kommunikáló pár közötti különböző lehetséges üzenetek száma kicsi halmaz, az támadásra ad lehetőséget. Tegyük fel, hogy a támadó ismeri az üzenethalmazt, de nem ismeri az RSA dekódoló kulcsot, s támadásra szánja el magát. Adja meg a támadás menetét!

3.13. feladat. Ha gyors RSA kódolást kívánunk megvalósítani, miért előnyös az $e = 3$, illetve általában az $e = 2^t + 1$ alakú választás? Adja meg a számításigényt a szükséges szorzások számában.

3.14. feladat. Legyen x_1 és x_2 két üzenet, továbbá y_1 és y_2 a megfelelő két rejtjeles blokk, amelyet RSA kódolással kaptunk. Érvényes a következő multiplikatív tulajdonság:

$$y_1 \cdot y_2 \equiv (x_1 \cdot x_2)^e \pmod{n}$$

Azaz, az üzenetek szorzatának rejtjeles képe a rejtjeles képek $\bmod n$ szorzata. Egy támadó dekódolni szeretne egy $x^e \bmod n = y$ rejtett szöveget. Tegyük fel, hogy a megtámadott dekódol tetszőleges neki küldött, számára nem bizalmas tartalmú üzenetet. Hogyan hajthat végre a támadó sikeres támadást az y dekódolására?

3.15. feladat. Egy n bites hash értéket képező iteratív hash függvényből $n' = 2n$ bites lenyomatot képzőt szeretnénk előállítani olyan módon, hogy a két utolsó iteráció eredményét konkatenáljuk, azaz $CV_{L-1}|CV_L$ $2n$ bites lenyomatot használunk az n bites CV_L helyett. Hatásos-e ezen dimenziónövelő megerősítési ötlet a születésnapi támadás sikerének csökkentésére?

3.16. feladat. Legyen H_1 és H_2 két ütközés-ellenálló hash függvény. Bizonyítsa be, hogy a konkatenáció művelettel kapott $H(x) = H_1(x)|H_2(x)$ hash függvény is ütközés-ellenálló. Tehát ez a dimenziónövelés hatásos.

3.17. feladat. Rejtjelezett üzeneteinket egy 10^{-4} bit hiba arányú csatornán továbbítjuk. Mekkora bit hiba arányt figyelhetünk meg a dekódoló kimenetén ha

a) AES-t használunk CBC módban?

b) AES-t használunk CFB módban, és 8 bites karaktereket rejtjelezünk?

c) AES-t használunk CTR módban, és teljes blokkokat rejtjelezünk?

3.18. feladat. Tekintsük az alábbi integritásvédelmi kódolást, ahol rejtjelezés és MAC kombinációját használjuk:

$$E_K(m|MAC_{K'}(m))$$

Legyen CBC módú mind a rejtjelezés, mind a MAC számítás, ahol az egyik funkcióra (IV, K), míg a másikra (IV', K') (inicializáló vektor, kulcs) páros kerül alkalmazásra. Van-e veszélye annak, ha $IV = IV', K = K'$ egyszerűsítő választással élünk? Mi a tanulság?

3.19. feladat. A Woo–Lam-protokollban A hitelesíti magát B-nek egy S megbízható harmadik fél segítségével. S-re azért van szükség, mert feltételezzük, hogy A és B nem rendelkezik közös kulccsal. Ezzel szemben feltesszük, hogy A és S valamint B és S rendelkezik egy közös K_{AS} illetve K_{BS} kulccsal. A protokoll lépései a következők:

Woo–Lam-protokoll		
(1)	$A \to B$:	A
(2)	$B \to A$:	N_B
(3)	$A \to B$:	$\{N_B\}_{K_{AS}}$
(4)	$B \to S$:	$\{A \mid \{N_B\}_{K_{AS}}\}_{K_{BS}}$
(5)	$S \to B$:	$\{N_B\}_{K_{BS}}$

Az első üzenetben A elküldi saját azonosítóját B-nek, aki egy frissen generált N_B kihívással válaszol. A rejtjelezi N_B-t a K_{AS} kulccsal, és az eredményt visszaküldi B-nek. B nem tudja dekódolni A válaszát, ezért továbbküldi azt S-nek, mellétéve A azonosítóját, és az egész üzenetet a K_{BS} kulccsal rejtjelezve. S dekódolja az üzenetet, sorban mindkét kulccsal, majd N_B-t rejtjelezi K_{BS}-sel, és az eredményt visszaküldi B-nek. Mostmár B is dekódolni tudja az üzenetet, és ellenőrzi, hogy visszakapta-e N_B-t. Ha igen, akkor B úgy gondolja, hogy A hitelesítette magát.

a) Mutassuk meg, hogy egy legális, de rosszindulatú X részvevő meg tudja személyesíteni A-t! (Segítség: Mivel X legális résztvevő, ezért rendelkezik egy K_{XS} kulccsal, és kezdeményezni tudja a protokollt B-vel.)

b) Javítsuk ki a protokollt!

3.7. Megoldások

3.1. megoldás.

a) Az a kulcselem multiplikatív inverzének léteznie kell $\bmod n$, aminek feltétele $(a,n) = 1$. $n = 256 = 2^8$ esetén $2^8 - 2^7 = 2^7$ ilyen szám van az $[1,255]$ tartományban. b kulcselem 2^8 féleképpen választható. Az $a = 1, b = 0$ esetet kizárva (identitás transzformáció), a kulcstér mérete $2^7 \cdot 2^8 - 1 = 32767$.

b) Nem. Ha a forrás véletlenszerűen sorsol az ábécé elemei közül, akkor az $a \cdot x$ szorzat is ilyen, következésképp az y rejtett szöveg és b kulcselem közötti kölcsönös információ zérus.

c) Két olyan (x,y), (x',y') ismert nyílt-rejtett pár elegendő, amelyre $(x-x')^{-1}$ $\bmod 256$ létezik.

d) A betűnkénti rejtjelezés miatt a karaktergyakoriságok az ábécé felett permutálódnak, de az értékek nem változnak. Ezért, ha van két szignifikánsan kiemelkedő gyakoriságú karakter az ábécében, akkor ezen gyakoriságok a rejtjelezés után is ugyanúgy léteznek. Ekkor statisztikai alapú támadással rejtett szöveg elegendő mennyiségben történő megfigyelése után végrehajtható a támadás: a legnagyobb gyakoriságú rejtett szöveg karakterekhez tartozó nyílt szöveg karakterekre sejtést teszünk, s minden ilyen sejtés egy újabb egyenletet ad az ismeretlen kulcselemekre. Két ilyen, lineárisan független egyenlet alapján kapunk sejtést a két kulcselemre, amelyek helyességét ismert szövegpáron teszteljük.

3.2. megoldás. Nem. Ugyanis $x = u \cdot n + v$ esetén, $y = (a \cdot (u \cdot n + v) + b) \bmod n^2 = ((a \cdot u) \cdot n + (a \cdot v + b)) \bmod n^2 = w \cdot n + z$, azaz a rejtett szöveg betűpár második karaktere (z) csak a nyílt szöveg betűpár második karakterétől (v) függ.

3.3. megoldás. Egy lehetséges megoldás, hogy a 8 bites karaktereket két 4 bites félre bontjuk, s az így nyert két elemű vektort szorozzuk egy 2×2-es kulcsmátrixszal. A műveleteket Z_{2^4} felett végezzük. Ekkor:

- A nyílt szövegek tere: $\mathcal{P} = Z_{16}^2$
- A rejtett szövegek tere: $\mathcal{C} = Z_{16}^2$
- A kulcstér: a Z_{16} feletti invertálható 2×2-es mátrixok halmaza.
- Minden invertálható mátrix alkalmas kulcsnak. Egy $\mathbf{K}$ mátrix akkor invertálható Z_{16} felett, ha $(\det(\mathbf{K}), 16) = 1$. Például a

  $$\mathbf{K} = \begin{bmatrix} 7 & 4 \\ 5 & 3 \end{bmatrix}$$

  mátrix megfelel, mert $\det(\mathbf{K}) = 7 \cdot 3 - 4 \cdot 5 = 1$.
- Az inverz-kulcs:

  $$\mathbf{K}^{-1} = \det(\mathbf{K})^{-1} \begin{bmatrix} k_{22} & -k_{12} \\ -k_{21} & k_{11} \end{bmatrix} = 1 \cdot \begin{bmatrix} 3 & -4 \\ -5 & 7 \end{bmatrix} = \begin{bmatrix} 3 & 12 \\ 11 & 7 \end{bmatrix}$$

3.4. megoldás. A rejtjelező nem tökéletes, mert pl. $\mathbf{P}\{Y = \alpha \mid X = a\} = P_{\mathcal{K}}(1) = \frac{2}{5}$ és $\mathbf{P}\{Y = \alpha \mid X = b\} = P_{\mathcal{K}}(3) = \frac{1}{5}$, azaz X és Y nem függetlenek.

3.5. megoldás. $y = A \cdot x + b$, ahol y, x, b n bites bináris vektorok, A egy $n \times n$ bites bináris mátrix, x a nyílt blokk, y a rejtett blokk, $K = (A, b)$ a kulcs. Lineáris egyenletrendszer megoldással történik a támadás. $n + 1$ párra van minimálisan szükség, amelyből n darab, lineárisan független nyílt szöveghez tartozik.

3.6. megoldás. A DES egy rétegének struktúrájára tekintve az igazolás nyilvánvaló: A DES S-dobozok bemenete nem változik, mivel az iterációs kulcs is és az adat input félblokk is komplementálva van, s XOR összegük kerül a bemenetekre, azaz $\overline{a} \oplus \overline{b} = a \oplus b$. Továbbá, a kimeneten ehhez a másik adat input félblokk komplementáltja adódik.

3.7. megoldás. Igen, végrehajtható, mivel ismerjük a rejtjeles szöveg redundáns struktúráját, amit adott esetben az egy blokkbeli 8 paritásbit jelent. Annak a valószínűsége, hogy egy téves kulccsal helyes paritásúra dekódolunk egy rejtett szöveg blokkot, 2^{-8} . Annak a valószínűsége, hogy például 7 rejtjeles blokk mindegyikét helyes paritásúra dekódoljuk téves kulcs mellett 2^{-56}, tehát ez esetben a ki nem szűrt téves kulcsok átlagos száma a kulcstér teljes végigkeresése után $2^{56} \cdot 2^{-56} = 1$ lenne. Ezért, például, 10 rejtjeles blokk megfigyelése elegendő lenne a gyakorlatilag egyértelmű kulcsazonosításhoz.

3.8. megoldás.

a) Nem, $5^{86} \equiv 25 \pmod{87}$

b) Igen, $32^{32} \equiv (-1)^{32} \equiv 1 \pmod{33}$.

3.9. megoldás. e számítása: $n = 55 \to \varphi(n) = 4 \cdot 10 = 40 = 2 \cdot 2 \cdot 2 \cdot 5 \to e = 3$. d számítása: $40 = 13 \cdot 3 + 1 \to 1 \cdot 40 + (-13) \cdot 3 = 1 \to d = -13 = 27 \bmod 40$.

3.10. megoldás. A $c = m^e \bmod n = 9726^9 \bmod 11413$ kifejezést kell kiszámolnunk. Az ismételt négyzetre emelés és szorzás módszerét használjuk. Ehhez felhasználjuk, hogy a 9 bináris alakja 1001.

i	b_i	$c_i = m^{b_i} \cdot c_{i+1}^2 \bmod n$
		$c_4 = 1$
3	1	$c_3 = 9726 \cdot 1^2 \bmod 11413 = 9726$
2	0	$c_2 = 9726^2 \bmod 11413 = 4132$
1	0	$c_1 = 4132^2 \bmod 11413 = 10989$
0	1	$c_0 = 9726 \cdot 10989^2 = 9726 \cdot 3663^2 \cdot 3^2 = 6950$

A rejtett szöveg tehát $c = 6950$.

3.11. megoldás.

a) $1 = (3999996, 379) = a \cdot 3999996 + b \cdot 379$. Az euklidészi algoritmus felhasználásával: $1 = (139) \cdot 3999996 + (-1467017) \cdot 379$, ahonnan $d = -1467017 = 2532979$.

b) $\varphi(n) = (p-1)(q-1) = pq - (p+q) + 1 = n - (p + n/p) + 1$ alapján másodfokú egyenletre jutunk p-ben, ahonnan $p = 2003$, $q = 1999$.

3.12. megoldás. A támadó elkészíti a lehetséges üzenetek kódolt megfelelőjét a nyilvános kódoló kulcs birtokában, s tárolja a {nyílt üzenet, rejtett üzenet} párok halmazát. Megfigyelve a csatornában az aktuális rejtett üzenetet a tárolt párok halmaza alapján rekonstruálni képes az üzenetet.

3.13. megoldás. Az ismételt négyzetre emelés és szorzás algoritmust hasznljuk a hatványozáshoz, ahol a szorzások száma a kitevőben található egyesek számával egyezik meg. Az $e = 2^t + 1$ kitevőben minimális számú, azaz kettő egyes található.

3.14. megoldás. A támadó elfedi az eredeti x tartalmat azáltal, hogy beszorozza y rejtjeles szöveget az üzenetek teréből választott r véletlen elem $r^e \bmod n$ rejtjelezettjével, s az $y \cdot r^e \bmod n$ szorzatot nyújtja be dekódolásra. A dekódolás eredménye $z = r \cdot x \bmod n$ lesz. Amennyiben r invertálható $\bmod n$ (ez gyakorlatilag biztos esemény), akkor a támadó könnyen kiszámítja az x üzenetet. Ezen támadás ugyanakkor könnyen kivédhető, ha a támadott csak olyan dekódolt üzeneteket ad ki, amelyek előre rögzített formátumnak megfelelők.

3.15. megoldás. Nem, mivel a kapott hash függvény ellen $O(2^{n'/2})$ -nél kisebb számításigényű születésnapi ütközéses támadás továbbra is végrehajtható. Ugyanis elegendő csak magára az n bites CV_{L-1} -re végrehajtani ezt a támadást, mert ha az $m = m_1|m_2|\ldots|m_{L-1}$ és $m' = m'_1|m'_2|\ldots|m'_{L-1}]$ üzenetpárra ütközés áll elő CV_{L-1}-re, akkor m_L blokkal meghosszabbítva m és m' üzeneteket, előáll az ütközés CV_L-re is, s így $CV_{L-1}|CV_L$-re is. Ezért a támadás $O(2^{n'/4})$ számításigényű maradt.

3.16. megoldás. A bizonyítás indirekt. Ha lenne (m,m') üzenetpár, amelyre ütközés állna elő a H hash függvény kimenetén, akkor az nyilván ütközést jelentene a komponens H_1 és H_2 hash függvények vonatkozásában is, ami ellentmondana a feltevésünknek.

3.17. megoldás.

a) CBC módban egy bit hiba az i. rejtjeles blokkban elrontja az i. nyílt blokk bitjeinek felét (átlagosan) és a következő nyílt blokk egy bitjét. Mivel az AES blokk mérete 128 bit, ezért a dekódoló kimenetén megfigyelt bit hiba arány: $(64+1)\cdot 10^{-4} = 65\cdot 10^{-4}$.

b) CFB módban egy bit hiba az i. rejtjeles karakterben elrontja az i. nyílt karakter egy bitjét majd az azt követő n/s nyílt karakter bitjeinek átlagosan felét, ahol n a rejtjelező blokkmérete és s a karakterek mérete bitben mérve. Jelen esetben $n = 128$, $s = 8$, ezért a megfigyelt bit hiba arány: $(1+\frac{128}{8}\cdot 4)\cdot 10^{-4} = 65\cdot 10^{-4}$.

c) CTR módban nincs hibaterjedés, ezért a megfigyelt bit hiba arány a dekódoló kimenetén 10^{-4}.

3.18. megoldás. A CBC mód lépéseit végiggondolva közvetlenül látható, hogy a feladatban adott kódolás eredménye $E_K(m)|E_K(0)$ az üzenettől függetlenül, azaz semmiféle integritásvédelmet nem kapunk. A tanulság az, hogy mindenképpen eltérő kulcsot kell alkalmazni. Például DES alkalmazása esetére K' kulcsra egy szokásos választás a K kulcs minden második félbájtjának komplementálása.

3.19. megoldás.

a) A támadás menete a következő:

$$
\begin{array}{ll}
X \to B\colon & X \\
 & X_A \to B\colon \quad A \\
B \to X\colon & N_{BX} \\
 & B \to X_A\colon \quad N_{BA} \\
X \to B\colon & \{N_{BA}\}_{K_{XS}} \\
 & X_A \to B\colon \quad \{N_{BA}\}_{K_{XS}} \\
B \to S\colon & \{X \mid \{N_{BA}\}_{K_{XS}}\}_{K_{BS}} \\
 & B \to S\colon \quad \{A \mid \{N_{BA}\}_{K_{XS}}\}_{K_{BS}} \\
 & S \to B\colon \quad \{N_{BA}\}_{K_{BS}} \\
S \to B\colon & \{*\}_{K_{BS}}
\end{array}
$$

ahol $*$ jelöli azt a pénzfeldobás sorozatot, amit S kap, amikor $\{N_{BA}\}_{K_{XS}}$-et dekódolja a K_{AS} kulccsal. B a szervertől kapott válaszokat helytelenül rendeli hozzá az éppen futó protokoll példányokhoz. Ez azért van, mert csak a szervertől visszakapott kihívás elem értékét tudja ellenőrizni, és mivel az megegyezik azzal az N_{BA}-val, amelyet B A-nak küldött, ezért azt hiszi, a protokoll A-val futott le sikeresen és valaki megpróbálta megszemélyesíteni X-et. Valójában azonban A egyáltalán nincs jelen a támadás alatt.

b) Egy lehetséges javítás a következő:

Javított Woo–Lam-protokoll		
(1)	$A \rightarrow B:$	A
(2)	$B \rightarrow A:$	N_B
(3)	$A \rightarrow B:$	$\{B,\ N_B\}_{K_{AS}}$
(4)	$B \rightarrow S:$	$A,\ \{B,\ N_B\}_{K_{AS}}$
(5)	$S \rightarrow B:$	$\{A,\ B,\ N_B\}_{K_{BS}}$

Figyeljük meg az azonosítók explicit használatát az üzenetekben!

3.8. Összefoglalás

Ebben a fejezetben bemutattuk a kommunikációs rendszerekben alkalmazott főbb biztonsági szolgáltatások — azaz a titkosítás, az integritásvédelem, a hitelesítés és a letagadhatatlanság — megvalósításához szükséges kriptográfiai eszközöket.

Megismerkedtünk két fontos kriptográfiai primitívvel, a rejtjelezéssel és a kriptográfiai hash függvényekkel. A rejtjelezésnek két fajtája létezik, a szimmetrikus kulcsú és az aszimmetrikus kulcsú, vagy más néven nyilvános kulcsú rejtjelezés. A szimmetrikus kulcsú rejtjelezőket tovább osztályoztuk kulcsfolyam rejtjelezőkre és blokkrejtjelezőkre. Megvizsgáltuk a primitívek konstrukciójának elvi kérdéseit és bemutattuk az ismertebb algoritmusok működését, úgy mint a DES, a 3DES, az AES és az RSA.

Bemutattuk továbbá, hogy a primitívek mint építőelemek segítségével hogyan konstruálhatók kriptográfiai protokollok. Részletesen tárgyaltuk a blokkrejtjelezők használatának módjait (ECB, CBC, CFB, OFB, CTR). Megismerkedtünk a hitelesítési feladatok megoldásának lehetőségeivel: az üzenethitelesítő kódokkal, a digitális aláírással, és a partnerhitelesítő protokollokkal. Röviden összefoglaltuk a nyilvános kulcsok hitelesítésének módszerét, azaz a nyilvános kulcs infrastruktúra (PKI) alapjait is. Foglalkoztuk továbbá a kulcscsere problémával, és a megoldásként használható kulcscsere protokollokkal.

Végül a bemutatott módszerek gyakorlati alkalmazásait a valós életből vett példákon, a GSM és a Web biztonsági mechanizmusainak ismertetésén keresztül szemléltettük. Bemutattunk továbbá egy elektronikus kereskedelmi alkalmazást, a DigiCash elektronikus készpénz protokollt, mely szintén kriptográfiai építőelemeket használ.

Az érdeklődő olvasó a kriptográfia itt bemutatott fejezeteinek részletesebb tárgyalását találja [3]-ban. Az angol nyelvű szakirodalomból ajánljuk Stinson tankönyvét [6], Schneier könyvét [5], mely számos kriptográfiai algoritmus leírását és C forráskódját tartalmazza olvasmányos formában, valamint Menezes, van Oorschot és Vanstone kézikönyvét [4], mely kimerítő jelleggel és matematikai alapossággal tárgyalja a mérnöki gyakorlatban használatos kriptográfiai módszereket. A partnerhitelesítő és a kulcscsere protokollok részletes tárgyalását tartalmazza Boyd és Mathuria könyve [2], továbbá ezen protokollok tervezésével kapcsolatos hasznos tanácsokat találhatunk Abadi és Needham cikkében [1].

Irodalomjegyzék

[1] Abadi, M., Needham, R., Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1), 1996.

[2] Boyd, C., Mathuria, A., *Protocols for Authentication and Key Establishment*. Springer, 2003.

[3] Buttyán L., Vajda I., *Kriptográfia és alkalmazásai*. TypoTEX Kiadó, Budapest, 2004.

[4] Menezes, A., van Oorschot, P., Vanstone, S., *Handbook of Applied Cryptography*. CRC Press, 1996.

[5] Schneier, B., *Applied Cryptography*. Wiley, 1996.

[6] Stinson, D., *Cryptography: Theory and Practice*. CRC Press, 1995.

4. fejezet

Adattömörítés

A 4. és 5. fejezetben a forráskódolásra fogunk koncentrálni. Feltételezzük, hogy a csatorna ideális, vagy másképp, a csatornakód olyan jó, hogy a forráskódolóból kilépő adatsorozat pontosan megegyezik a forrásdekódolóba belépő adatsorozattal.

A forráskódolás lehet veszteségmentes és veszteséges. Veszteségmentes forráskódolás, vagyis az ún. adattömörítés esetében megköveteljük, hogy az eredeti forrásadatok pontosan helyreállíthatók legyenek. Veszteséges forráskódolás esetén csak annyit várunk el, hogy a dekódoló által visszaállított adatok valamilyen értelemben eléggé közel legyenek az eredetihez. Az adott alkalmazástól függ, hogy a kettő közül melyik módszert használjuk. Például egy szöveg tömörítésekor azt szeretnénk, hogy a visszaállított szöveg ne tartalmazzon hibát. Egy kép tömörítésekor viszont megelégszünk azzal, ha a visszaállított kép látványa az eredetihez hasonló minőségű.

Ebben a fejezetben bevezetjük a veszteségmentes adattömörítés fogalmát. Ennek célja az, hogy egy üzenetet, amely egy véges halmaz (forrásábécé) elemeiből álló véges sorozat, egy másik véges halmaz (kódábécé) elemeiből álló sorozattal reprezentáljunk úgy, hogy ez a sorozat a lehető legrövidebb (és belőle az üzenet visszaállítható) legyen. A legismertebb példa erre az, amikor a kódábécé a $\{0,1\}$ halmaz, és egy üzenetet (pl. írott szöveg, fájl) bináris sorozatok formájában kódolunk tárolás, illetve átvitel céljából.

Az első tömörítő eljárás a 19. század közepén Samuel F. B. Morse (1791–1872) által kidolgozott Morse-kód volt. A távírón átküldött betűket ti–tá (rövid–hosszú, mai szóhasználattal bináris) sorozatokkal kódolta. Morse észrevette, hogy mivel az ábécé egyes betűi gyakrabban fordulnak elő, mint mások, ha ezekhez rövidebb sorozatokat rendel, akkor ezzel lerövidítheti az üzenetek átküldéséhez szükséges időt.

A tömörítés minőségét a tömörítési aránnyal jellemezhetjük, ami a tömörített hossznak és az eredeti adatsorozat hosszának az aránya. Mindenki számára világos, hogy a tömörítési aránynak, a tömöríthetőségnek van alsó határa. Az adattömörítés természettörvényét Claude E. Shannon (1916–2001) fedezte fel, amikor kiszá-

mította a tömörítési arány elvi alsó határát, a forrásentrópiát, és megadott olyan kódolási eljárásokat, amelyek ezt az elvi alsó határt elérik.

A következő szakaszokban megismerkedünk az üzenetek egy természetesen adódó kritérium szerinti kódolásával — az egyértelműen dekódolható kódolással —, mely később vizsgálataink középpontjában áll majd. Bevezetjük a diszkrét valószínűségi változó entrópiáját, és megmutatjuk, hogy az entrópia milyen alapvető szerepet játszik a kódolással kapcsolatban. Az elkövetkezőkben mindig azt tartjuk szem előtt, hogy az üzenetek kódja minél rövidebb legyen, ezért a kódok általunk vizsgált fő tulajdonsága az átlagos kódszóhossz lesz. A fejezetet az univerzális forráskódolással és annak gyakorlati alkalmazásaival zárjuk.

4.1. Prefix kódok

Jelöljön X egy diszkrét valószínűségi változót, amely az $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ véges halmazból veszi értékeit. Az X-et a továbbiakban véletlen betűnek, az $\mathcal{X}$ halmazt **forrásábécé**nek, elemeit pedig betűknek nevezzük.

$\mathcal{Y}$ jelöljön egy s elemű $\{y_1, y_2, \ldots, y_s\}$ halmazt. Ezt **kódábécé**nek nevezzük. $\mathcal{Y}^*$ jelölje az $\mathcal{Y}$ elemeiből álló véges sorozatok halmazát. $\mathcal{Y}^*$ elemeit **kódszavak**nak nevezzük. Egy

$$f : \mathcal{X} \to \mathcal{Y}^*$$

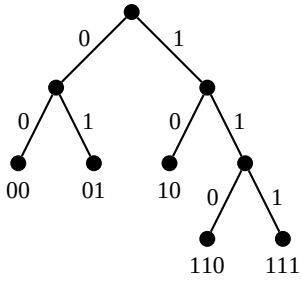
függvényt, amely megfeleltetést létesít a forrásábécé és a kódszavak között, **kód**nak nevezünk. Az $\mathcal{X}$ elemeiből alkotott véges sorozatok az **üzenet**ek vagy közlemények (értékeiket az $\mathcal{X}^*$ halmazból veszik). Amennyiben az f kód értékkészlete különböző hosszú kódszavakból áll, úgy változó szóhosszúságú kódolásról beszélünk.

4.1. definíció. *Az $f : \mathcal{X} \to \mathcal{Y}^*$ kód **egyértelműen dekódolható**, ha minden véges kódbetűsorozat legfeljebb egy közlemény kódolásával állhat elő, azaz ha $\mathbf{u} \in \mathcal{X}^*$, $\mathbf{v} \in \mathcal{X}^*$, $\mathbf{u} = u_1 u_2 \ldots u_k$, $\mathbf{v} = v_1 v_2 \ldots v_m$, $\mathbf{u} \neq \mathbf{v}$, akkor $f(u_1) f(u_2) \ldots f(u_k) \neq f(v_1) f(v_2) \ldots f(v_m)$. (Itt az $f(u) f(u')$ a két kódszó egymás után írását [konkatenáció] jelenti.)*

MEGJEGYZÉS:

a) Az egyértelmű dekódolhatóság több, mint az invertálhatóság. Ugyanis legyen $\mathcal{X} = \{a, b, c\}, \mathcal{Y} = \{0, 1\}$ és $f(a) = 0, f(b) = 1, f(c) = 01$. Ekkor az $f : \mathcal{X} \to \mathcal{Y}^*$ leképezés invertálható, viszont a 01 kódszót dekódolhatjuk $f(a) f(b) = 01$ szerint *ab*-nek, vagy $f(c) = 01$ szerint *c*-nek is.

b) Az előbbi definícióban szereplő kódolási eljárást, amikor egy közlemény kódját az egyes forrásbetűkhöz rendelt kódszavak sorrendben egymás után írásával kapjuk, betűnkénti kódolásnak nevezzük. Természetesen a kódolást teljesen általánosan egy $g : \mathcal{X}^* \to \mathcal{Y}^*$ függvénnyel is definiálhatnánk, de belátható, hogy



4.1. ábra. Egy prefix kód kódfája

a betűnkénti kódolás és annak természetes kiterjesztése, a blokk-kódolás elegendő számunkra, azaz a tömörítési arány minimuma elérhető blokkonkénti kódolással.

c) Az egyértelmű dekódolhatóság végtelen sok eset ellenőrzését igényli, ezért a gyakorlatban használhatatlan követelmény.

4.2. definíció. *Az f kód **prefix**, ha a lehetséges kódszavak közül egyik sem folytatása a másiknak, vagyis bármely kódszó végéből bármekkora szegmenst levágva nem kapunk egy másik kódszót.*

Egy prefix kód ábrázolható az ún. kódfával (4.1. ábra), amelynek levelei jelentik az üzeneteket. A gyökértől a levelekig vezető éleken szereplő kódbetűket összeolvasva kaphatók meg a kódszavak. A prefix kódok egyben egyértelműen dekódolhatóak is, mivel egy adott kódszó karakterei a kódfa gyökerétől indulva egyértelmű utat jelölnek ki egy levélig, s ez lesz a dekódolt üzenet.

Az eddig bevezetett fogalmak illusztrálására nézzük a következő példákat:

4.1. példa. $\mathcal{X} = \{a,b,c\}$; $\mathcal{Y} = \{0,1\}$ A kód legyen a következő: $f(a) = 0,\ f(b) = 10,\ f(c) = 11$. Könnyen ellenőrizhető, hogy a kód prefix.

Ha az *abccab* üzenetet kódoljuk, akkor a 0101111010 kódbetűsorozatot kapjuk. A kódból az üzenet visszafejtése nagyon egyszerű a prefix tulajdonság miatt; többek között ez a gyors dekódolási lehetőség teszi vonzóvá a prefix kódokat.

4.2. példa. $\mathcal{X} = \{a,b,c,d\}$; $\mathcal{Y} = \{0,1\}$; $f(a) = 0,\ f(b) = 01,\ f(c) = 011,\ f(d) = 0111$. Jól látható, hogy a kód nem prefix, de egyértelműen dekódolható, hiszen a 0 karakter egy új kódszó kezdetét jelzi.

Bebizonyítható, hogy minden egyértelműen dekódolható kódhoz létezik vele ekvivalens (azonos kódszóhosszú) prefix kód, tehát nem vesztünk semmit, ha az egyértelmű dekódolhatóság helyett a speciálisabb, és ezért könnyebben kezelhető prefix tulajdonságot követeljük meg.

4.2. Átlagos kódszóhossz, entrópia

Legyen X egy $\mathcal{X}$ értékű valószínűségi változó, amit kódolni akarunk. Vezessük be a következő $p:\ \mathcal{X}\to[0,1]$ függvényt:

$$p(x)=\mathbf{P}\{X=x\},\qquad x\in\mathcal{X}.$$

A $p(x)$ függvény tehát az x forrásbetűhöz annak valószínűségét rendeli.

4.3. definíció. *Az X valószínűségi változó **entrópiá**ját, $H(X)$-et, a*

$$H(X)=\mathbf{E}(-\log p(X))=-\sum_{i=1}^{n}p(x_i)\log p(x_i).$$

összeggel definiáljuk.

MEGJEGYZÉS: A $\log z$ jelölés a z pozitív szám kettes alapú logaritmusát jelenti. Mivel a 4.3. definíció megkívánja, a logaritmusfüggvénnyel kapcsolatban a következő „számolási szabályokat” vezetjük be ($a\geq 0, b>0$):

$$0\log\frac{0}{a}=0\log\frac{a}{0}=0;\qquad b\log\frac{b}{0}=+\infty;\qquad b\log\frac{0}{b}=-\infty.$$

(E szabályok az adott pontban nem értelmezett függvények folytonos kiterjesztései.) A definícióból közvetlenül látszik, hogy az entrópia *nemnegatív*. Vegyük észre, hogy az entrópia értéke valójában nem függ az X valószínűségi változó értékeitől, csak az eloszlásától.

4.4. definíció. *Egy f kód **átlagos kódszóhossz**án az*

$$\mathbf{E}|f(X)|=\sum_{i=1}^{n}p(x_i)|f(x_i)|$$

várható értéket értjük.

4.3. példa. A 4.1. példa kódja esetén legyen $p(a)=0.5,\ p(b)=0.3, p(c)=0.2$, ekkor az entrópia

$$H(X)=-0.5\cdot\log 0.5-0.3\cdot\log 0.3-0.2\cdot\log 0.2\approx 1.485,$$

az átlagos kódszóhossz pedig

$$\mathbf{E}|f(X)|=0.5\cdot 1+0.3\cdot 2+0.2\cdot 2=1.5.$$

Bebizonyítható, hogy az entrópiával alsó korlát adható az átlagos kódszóhosszra:

4.1. tétel. *Tetszőleges egyértelműen dekódolható $f:\ \mathcal{X}\to\mathcal{Y}^*$ kódra*

$$\mathbf{E}|f(X)|\geq\frac{H(X)}{\log s}. \tag{4.1}$$

Az entrópia néhány tulajdonsága: Legyenek X és Y valószínűségi változók, amelyek a véges $\mathcal{X}$ illetve $\mathcal{Y}$ halmazból veszik értékeiket.

4.2. tétel.

a) Ha az X valószínűségi változó n különböző értéket vehet fel pozitív valószínűséggel, akkor

$$0 \leq H(X) \leq \log n,$$

és a bal oldalon egyenlőség akkor és csak akkor áll fenn, ha X 1-valószínűséggel konstans, a jobb oldalon pedig akkor és csak akkor, ha X egyenletes eloszlású, azaz $p(x_i) = \frac{1}{n},\ i = 1,\ldots,n$.

b) X és Y diszkrét valószínűségi változókra

$$H(X,Y) \leq H(X) + H(Y),$$

és az egyenlőség szükséges és elégséges feltétele X és Y függetlensége. ($H(X,Y)$ az (X,Y) valószínűségi változópár együttes eloszlásához rendelt entrópiát jelöli.)

c) Az X tetszőleges $g(X)$ függvényére

$$H(g(X)) \leq H(X),$$

és itt az egyenlőség szükséges és elégséges feltétele az, hogy g invertálható legyen.

4.3. Shannon–Fano-kód

Miután láttuk, hogy egy X valószínűségi változó egyértelműen dekódolható kódjának átlagos kódszóhosszára a $\frac{H(X)}{\log s}$ mennyiség alsó korlátot ad, most megmutatjuk, hogy prefix kóddal ezt a korlátot jól meg lehet közelíteni.

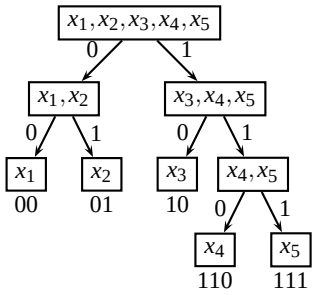
A **Shannon–Fano-kód** kostrukciójához feltehetjük, hogy

$$p(x_1) \geq p(x_2) \geq \cdots \geq p(x_{n-1}) \geq p(x_n) > 0,$$

mert különben az $\mathcal{X}$ elemeinek átindexelésével elérhetjük ezt. A konstrukció technikája miatt felhasználjuk az $y_i \mapsto i-1,\ i = 1,\ldots,s$ megfeleltetést. Legyenek a w_i számok a következők:

$$w_1 = 0, \qquad w_i = \sum_{l=1}^{i-1} p(x_l), \qquad i = 2,\ldots,n.$$

Kezdjük a w_j számokat felírni s-alapú számrendszerbeli tört alakban. A w_j-hez tartozó törtet olyan pontosságig írjuk le, ahol az először különbözik az összes többi w_i, $i \neq j$ szám hasonló alakbeli felírásától. Miután ily módon n darab véges hosszú



4.2. ábra. A 4.4. példa Shannon–Fano-kódjának előállítása

történt kapunk, az $f(x_j)$ legyen a w_j-hez tartozó tört az egészeket reprezentáló nulla nélkül.

A Shannon–Fano-kódot előállíthatjuk egy másik algoritmussal is. Osszuk fel a szimbólumok valószínűségeinek csökkenő sorozatát két részre úgy, hogy a sorozat első felében lévő valószínűségek összege és a sorozat második felében lévő valószínűségek összege a lehető legkevésbé térjen el egymástól. Legyen az első részben lévő szimbólumok kódszavának első bitje 0, a második részben lévőké pedig 1. Ezután alkalmazzuk ezt az eljárást rekurzívan az első és a második részben lévő szimbólumokra, s így megkapjuk a kódszavak további bitjeit. Az algoritmus akkor ér véget, ha már minden részben csak egyetlen szimbólum szerepel.

4.4. példa. Legyen $n = 5$ és $p(x_1) = 0.35,\ p(x_2) = 0.2,\ p(x_3) = 0.2,\ p(x_4) = 0.15,\ p(x_5) = 0.1$ és $\mathcal{Y} = \{0,1\}$. Keressük meg az ehhez tartozó Shannon–Fano-kódot. Használjuk a második algoritmust. A valószínűségek két részre osztása után az első részben az x_1 és x_2 szimbólumok lesznek ($p(x_1)+p(x_2) = 0.55$), a második részben pedig az x_3, x_4 és x_5 szimbólumok ($p(x_3)+p(x_4)+p(x_5) = 0.45$). Így az x_1 és x_2 szimbólumok kódszavának első bitje 0, míg az x_3, x_4 és x_5 szimbólumoké 1 lesz. Az x_1 és x_2 részt kettéosztva elkészültünk ezek kódszavaival: az x_1-é 00, az x_2-é pedig 01 lesz. A második részt ismét kettéosztva az x_3 egyedül marad, így ennek kódszava 10, az x_4 és x_5 kódszavát pedig egy újabb kettéosztás után kapjuk: x_4 kódszava 110, míg x_5-é 111 lesz.

Könnyedén belátható, hogy az így kapott kód prefix, mivel kódfával ábrázolható.

4.3. tétel. *Létezik olyan $f : \mathcal{X} \to \mathcal{Y}^*$ prefix kód, mégpedig a Shannon–Fano-kód, amelyre*

$$\mathbf{E}|f(X)| < \frac{H(X)}{\log s} + 1.$$

Blokk-kódolás

A betűnkénti kódolásnak van egy természetes általánosítása, a **blokk-kódolás**. Ezt formálisan egy $f : \mathcal{X}^m \to \mathcal{Y}^*$ leképezéssel definiálhatjuk, ahol tehát a forrásábécé betűiből alkotott rendezett m-eseket tekintjük forrásszimbólumoknak és ezeknek feleltetünk meg kódszavakat. A helyzet tulajdonképpen nem változik a betűnkénti kódolás esetéhez képest, hiszen egy új $\widehat{\mathcal{X}}$ forrásábécét definiálhatunk az $\widehat{\mathcal{X}} = \mathcal{X}^m$ jelöléssel, és az eddig elmondottak mind érvényben maradnak. Az egyértelmű dekódolhatóság definíciója ugyanaz marad, mint a betűnkénti kódolás esetében, az előbbieket szem előtt tartva. Legyen $\mathbf{X} = (X_1, \ldots, X_m)$ egy valószínűségi vektorváltozó, melynek koordinátái az $\mathcal{X}$-ből veszik értékeiket. Az entrópia 4.3. definíciójából jól látszik, hogy az csakis az eloszlástól függ. Mivel $\mathbf{X}$ is csak véges sok különböző értéket vehet fel, a 4.3. definíciót közvetlenül alkalmazhatjuk. Az $\mathbf{X}$ entrópiája tehát a

$$p(\mathbf{x}) = p(x_1, \ldots, x_m) = \mathbf{P}\{X_1 = x_1, \ldots, X_m = x_m\}, \qquad x_1, \ldots, x_m \in \mathcal{X},$$

jelölést bevezetve a következő:

$$\begin{aligned} H(\mathbf{X}) &= -\sum_{\mathbf{x} \in \mathcal{X}^m} p(\mathbf{x}) \log p(\mathbf{x}) = \\ &= -\sum_{x_1 \in \mathcal{X}} \cdots \sum_{x_m \in \mathcal{X}} p(x_1, \ldots, x_m) \log p(x_1, \ldots, x_m). \end{aligned}$$

Az $\mathbf{X} = (X_1, \ldots, X_m)$ entrópiájára a $H(\mathbf{X})$ jelölés mellett, ahol ez a célszerűbb, gyakran a $H(X_1, \ldots, X_m)$ jelölést fogjuk használni.

A betűnkénti átlagos kódszóhossz definíciója értelemszerűen módosul a blokk-kódolás esetében: a kódszóhossz várható értékét el kell osztani az egy blokkot alkotó forrásbetűk számával, vagyis a betűnkénti átlagos kódszóhosszon a következő mennyiséget értjük:

$$\frac{1}{m}\mathbf{E}|f(\mathbf{X})| = \frac{1}{m}\sum_{\mathbf{x} \in \mathcal{X}^m} p(\mathbf{x})|f(\mathbf{x})|$$

Értelemszerűen a 4.1. tétel állítása blokk-kódolás esetén is igaz:

$$\frac{1}{m}\mathbf{E}|f(\mathbf{X})| \geq \frac{\frac{1}{m}H(\mathbf{X})}{\log s}.$$

Másrészről a 4.3. tételből következik, hogy a Shannon–Fano-kódra

$$\frac{1}{m}\mathbf{E}|f(\mathbf{X})| \leq \frac{\frac{1}{m}H(\mathbf{X})}{\log s} + \frac{1}{m}.$$

Az m blokkhossz növelésével általában javul az adattömörítés hatékonysága, azaz csökken az egy betűre jutó átlagos kódszóhossz. Példaként tekintsük az irodalmi angol szöveg tömörítésének problémáját. Csak a kis betűket és a legfontosabb írásjeleket megtartva kaphatunk egy 32 elemű ábécét, vagyis tömörítés nélkül

egy betűt 5 bittel tudunk reprezentálni. Ha becsüljük az X_1 eloszlását, akkor a $H(X_1)$-re körülbelül 4.03 adódik, tehát $m = 1$ esetén a Shannon–Fano-kódra a

$$4.03 \leq \mathbf{E}|f(X_1)| \leq 4.03 + 1 = 5.03$$

adódik, ami még nem előrelépés a tömörítetlen 5 bithez képest. $m = 2$ esetén $H(X_1, X_2)/2 \approx 3.32$, tehát

$$3.32 \leq \frac{1}{2}\mathbf{E}|f(X_1, X_2)| \leq 3.32 + \frac{1}{2} = 3.82,$$

$m = 3$ esetén

$$3.1 \leq \frac{1}{3}\mathbf{E}|f(X_1, X_2, X_3)| \leq 3.43,$$

és $m = 4$ esetén

$$2.8 \leq \frac{1}{4}\mathbf{E}|f(X_1, X_2, X_3, X_4)| \leq 3.05.$$

4.5. definíció. *Az $\mathbb{X}$ információforrást az $X_1, X_2, \ldots$ valószínűségi változók végtelen sorozatával modellezzük, azaz a forrás az i-edik időpillanatban az X_i jelet bocsátja ki. Az X_i valószínűségi változók mindegyike ugyanabból a véges $\mathcal{X} = \{x_1, \ldots, x_n\}$ halmazból, a forrásábécéből veszi az értékét. Az $\mathbb{X}$ forrást statisztikai tulajdonságaival jellemezzük, vagyis adottnak vesszük, ha minden véges dimenziós eloszlását ismerjük.*

Információforrások egy tág osztályára megmutatható, hogy az $\frac{1}{m}H(X_1, \ldots, X_m)$ betűnkénti entrópia monoton csökken, tehát az m növelésével az alsó határ csökken.

Markov-láncok

A következőkben egy olyan struktúrát, az ún. Markov-láncot ismerjük meg, amellyel hatékonyan leírhatók egyes redundáns karaktersorozatok, és szemléltethető a blokkonkénti kódolás hatékonysága.

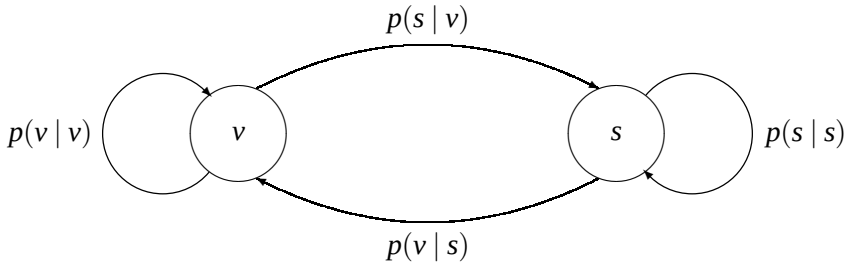
Az $X_1, X_2, \ldots$ valószínűségi változók Markov-láncot alkotnak, ha

$$\mathbf{P}\{X_k = x_k \mid X_1 = x_1, X_2 = x_2, \ldots, X_{k-1} = x_{k-1}\} = \mathbf{P}\{X_k = x_k \mid X_{k-1} = x_{k-1}\} \quad (4.2)$$

minden $k \geq 2$-re és minden lehetséges $x_1, x_2, \ldots, x_k$ sorozatra. A X_i értékeit a Markov-lánc állapotainak, az állapotok $\mathcal{X}$ halmazát pedig a Markov-lánc állapotterének nevezzük. Feltesszük, hogy $|\mathcal{X}| < \infty$, vagyis az állapottér véges. Az $X_1, X_2, \ldots$ Markov-lánc homogén, ha az ún. egylépéses átmenetvalószínűségek nem függenek a változók indexétől, vagyis

$$\mathbf{P}\{X_k = x_2 \mid X_{k-1} = x_1\} = \mathbf{P}\{X_2 = x_2 \mid X_1 = x_1\}$$

minden $x_1, x_2 \in \mathcal{X}$-re és minden $k \geq 2$-re.



4.3. ábra. Egy fekete-fehér kép Markov-lánc modellje

Egy Markov-lánc eloszlása az alábbi láncszabály alapján megadható a kezdeti eloszlás és az egylépéses átmenetvalószínűségek segítségével:

$$\begin{aligned}
&\mathbf{P}\{X_m = x_m, \ldots, X_1 = x_1\} = \\
&= \mathbf{P}\{X_m = x_m \mid X_{m-1} = x_{m-1}, \ldots, X_1 = x_1\} \cdots \mathbf{P}\{X_2 = x_2 \mid X_1 = x_1\}\mathbf{P}\{X_1 = x_1\} = \\
&= \mathbf{P}\{X_m = x_m \mid X_{m-1} = x_{m-1}\} \cdots \mathbf{P}\{X_2 = x_2 \mid X_1 = x_1\}\mathbf{P}\{X_1 = x_1\} = \\
&= \prod_{i=2}^{m} p(x_i \mid x_{i-1}) p(x_1).
\end{aligned}$$

A $\mathbf{P}\{X_k = x_2 \mid X_{k-1} = x_1\}$ átmenetvalószínűségek ismeretében a Markov-modell jól használható szövegek és képek tömörítésére. Angol nyelvű szövegek entrópiájának meghatározására elsőként Shannon végzett kísérleteket. A nyelvi redundanciát modellezte Markov-láncokkal. A 26 betűs angol ábécét használó szövegek esetén másodrendű Markov-lánc modell használatával 3.1 bit/betű tömörítési arányt ért el. Ez 2.4 bit/betű-re szorítható le, ha betűk helyett szavak szerepelnek a modellben. Az angol nyelv entrópiájának becslésére Shannon kísérleti személyeket is alkalmazott a statisztikai modellek helyett. A résztvevőknek a szöveg aktuális betűjét kellett kitalálniuk a megelőző 100 betűs környezet alapján. Így alsó határnak 0.6 bit/betű, míg felsőnek 1.3 bit/betű adódott. Minél hosszabb környezetet vizsgálunk, annál jobb a jóslás eredménye, azonban a környezet hosszával exponenciálisan növekszik a lehetséges megelőző állapotok száma.

4.5. példa. Egy fekete-fehér kép sorait modellezzük egy $\{X_i\}$ homogén Markov-lánccal a 4.3. ábrán látható módon. A v állapot jelöli azt, hogy az aktuális képpont világos, míg az s, hogy sötét. $p(v)$ a világos, $p(s)$ a sötét képpont kezdeti valószínűsége. Az átmenetvalószínűségek közül például $p(v \mid s)$ jelenti annak az eseménynek a valószínűségét, amikor egy sötét képpont után egy világos következik.

Számítsuk ki egy tetszőleges Markov-lánc entrópiáját:

$$H(X_1, \ldots, X_m) = -\sum_{x_1, \ldots, x_m} p(x_1, \ldots, x_m) \log p(x_1, \ldots, x_m) =$$

$$
\begin{aligned}
&= -\sum_{x_1,\ldots,x_m} p(x_1,\ldots,x_m)\log\prod_{i=2}^{m} p(x_i \mid x_{i-1})p(x_1) = \\
&= -\sum_{i=2}^{m}\sum_{x_1,\ldots,x_m} p(x_1,\ldots,x_m)\log p(x_i \mid x_{i-1}) \\
&\quad -\sum_{x_1,\ldots,x_m} p(x_1,\ldots,x_m)\log p(x_1) = \\
&= -\sum_{i=2}^{m}\sum_{x_{i-1},x_i} p(x_{i-1},x_i)\log p(x_i \mid x_{i-1}) - \sum_{x_1} p(x_1)\log p(x_1) = \\
&= -(m-1)\sum_{x_1,x_2} p(x_1,x_2)\log p(x_2 \mid x_1) - \sum_{x_1} p(x_1)\log p(x_1) = \\
&= (m-1)H(X_2 \mid X_1) + H(X_1).
\end{aligned}
$$

A $-\sum_{x_1,x_2} p(x_1,x_2)\log p(x_2 \mid x_1)$ mennyiséget feltételes entrópiának nevezzük, és $H(X_2 \mid X_1)$-gyel jelöljük. A betűnkénti entrópia ekkor így alakul:

$$\frac{1}{m}H(X_1,\ldots,X_m) = \frac{1}{m}H(X_1) + \frac{m-1}{m}H(X_2 \mid X_1).$$

Nagy m blokkhossz esetén az első tag elhanyagolható, ezért a betűnkénti entrópia közelítőleg megegyezik a $H(X_2 \mid X_1)$ feltételes entrópiával.

4.6. példa. Legyenek a 4.5. példa valószínűségei a következőek:

$$p(s \mid s) = 0.77, \quad p(v \mid s) = 0.23, \quad p(s \mid v) = 0.02, \quad p(v \mid v) = 0.98$$

Megmutatható, hogy ha

$$p(s) = 0.08, \quad p(v) = 0.92,$$

akkor a Markov-lánc azonos eloszlású. Ennek az eloszlásnak az entrópiája:

$$H(X_1) = -0.92\log 0.92 - 0.08\log 0.08 = 0.402$$

Ugyanez a modellünk szerint:

$$
\begin{aligned}
H(X_2 \mid X_1) &= -\sum_{x_1}\sum_{x_2} p(x_2 \mid x_1)p(x_1)\log p(x_2 \mid x_1) = \\
&= -p(s \mid s)p(s)\log p(s \mid s) - p(v \mid s)p(s)\log p(v \mid s) \\
&\quad -p(s \mid v)p(v)\log p(s \mid v) - p(v \mid v)p(v)\log p(v \mid v) = \\
&= -0.77\cdot 0.08\cdot\log 0.77 - 0.23\cdot 0.08\cdot\log 0.23 \\
&\quad -0.02\cdot 0.92\cdot\log 0.02 - 0.98\cdot 0.92\cdot\log 0.98 = \\
&= 0.192
\end{aligned}
$$

Látható, hogy kevesebb, mint a felére csökkent az entrópia az átmenetvalószínűségek ismerete miatt.

4.4. Optimális kódok, bináris Huffman-kód

A 4.1. tétel alsó korlátot ad az egyértelműen dekódolható kódok átlagos kódszóhosszára, a 4.3. tétel pedig mutat egy olyan kódkonstrukciót, ahol ezt a korlátot jól megközelíthetjük. A jó kód konstrukciójának problémáját persze ennél általánosabban is felvethetjük: konstruáljuk meg az optimális, azaz legkisebb átlagos kódszóhosszú kódot, ha adott az X valószínűségi változó eloszlása. Először is gondoljuk át, hogy optimális kód valóban létezik. Ugyan véges kódábécé esetén is az egyértelműen dekódolható vagy prefix kódok halmaza végtelen, de a bizonyos átlagos kódszóhossznál (pl. $\frac{H(X)}{\log s}+1$) jobb kódok halmaza véges. Másodszor, vegyük észre, hogy az optimális kód nem feltétlenül egyértelmű; egyenlő valószínűségekhez tartozó kódszavakat felcserélhetünk, csakúgy, mint az egyenlő hosszú kódszavakat, anélkül, hogy az átlagos kódszóhosszat ezzel megváltoztatnánk.

Egy optimális egyértelműen dekódolható kód konstruálását egy optimális prefix kód konstruálására lehet visszavezetni. Ezért a következőkben kimondjuk az optimális prefix kódok néhány tulajdonságát. A továbbiakban az egyszerűség kedvéért a bináris, $s=2$ esettel foglalkozunk; feltesszük, hogy $\mathcal{Y}=\{0,1\}$. Az általános, $s>2$ eset bonyolultabb, és a dolog lényege így is jól látható.

4.4. tétel. *Ha az $f:\mathcal{X}\to\{0,1\}^*$ prefix kód optimális, és $\mathcal{X}$ elemei úgy vannak indexelve, hogy $p(x_1)\geq p(x_2)\geq\cdots\geq p(x_{n-1})\geq p(x_n)>0$, akkor feltehető, hogy f-re a következő három tulajdonság teljesül:*

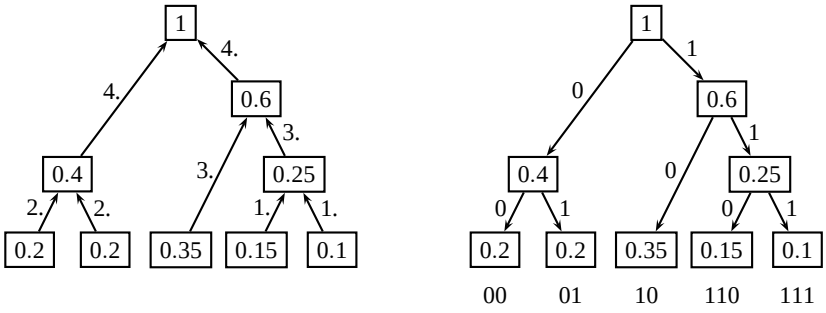
a) $|f(x_1)|\leq|f(x_2)|\leq\cdots\leq|f(x_{n-1})|\leq|f(x_n)|$, *vagyis nagyobb valószínűségekhez kisebb kódszóhosszak tartoznak.*

b) $|f(x_{n-1})|=|f(x_n)|$, *vagyis a két legkisebb valószínűségű forrásbetűhöz tartozó kódszó egyenlő hosszú.*

c) Az $f(x_{n-1})$ és az $f(x_n)$ kódszavak csak az utolsó bitben különböznek.

4.5. tétel. *Tegyük most fel, hogy a 4.4. tétel feltételei teljesülnek, és hogy a $\{p(x_1),p(x_2),\ldots,p(x_{n-1})+p(x_n)\}$ valószínűségeloszláshoz ismerünk egy g optimális bináris prefix kódot. (Az x_{n-1} és x_n forrásbetűket összevonjuk egy $\bar{x}_{n-1}$ szimbólumba; $p(\bar{x}_{n-1})=p(x_{n-1})+p(x_n)$). Ekkor az eredeti $\{p(x_1),p(x_2),\ldots,p(x_{n-1}),p(x_n)\}$ eloszlás egy optimális f prefix kódját kapjuk, ha a $g(\bar{x}_{n-1})$ kódszót egy nullával, illetve egy egyessel kiegészítjük (a többi kódszót pedig változatlanul hagyjuk).*

Az előző tétel alapján már megadhatjuk az optimális prefix kód Huffman-féle konstrukcióját: A két legkisebb valószínűség összevonásával addig redukáljuk a problémát, amíg az triviális nem lesz, vagyis amíg összesen két valószínűségünk marad. Ezt $n-2$ lépésben érhetjük el. Ezután az összevonások megfordításával, mindig a megfelelő kódszó kétféle kiegészítésével újabb $n-2$ lépésben felépítjük az optimális kódot, a Huffman-kódot. Vegyük észre, hogy ez egy ún. mohó algoritmus, azaz az egyes lépésekben mindig lokálisan optimális döntést hoz.



4.4. ábra. A 4.7. példa Huffman-kódjának előállítása

A Huffman-kód prefix, ezért azt bináris faként is ábrázolhatjuk. A leveleket a forrásszimbólumokkal címkézzük, az éleken pedig a kódábécé $(\{0,1\})$ elemei szerepelnek. A csúcsokban a szimbólumok valószínűségei állnak. Egy forrásszimbólumhoz tartozó kódszót úgy kapunk meg, hogy a fa gyökerétől a megfelelő levélig húzódó út élein szereplő kódbetűket sorban összeolvassuk (konkatenáljuk). A Huffman-kódolás szemléletesen úgy történik, hogy kiindulásként felvesszük a forrásszimbólumokhoz tartozó leveleket, a csúcsokba a forrásszimbólumok valószínűségeit írjuk, majd lépésenként mindig a két legkisebb értéket tartalmazó csúcs fölé teszünk egy új csúcsot (szülőt), s ebbe a két régi érték összegét írjuk. Az eljárás végén kialakul az összefüggő fa, melynek a legutolsó lépésben megkapott csúcsa lesz a gyökér.

```
INPUT: lista={az n féle szimbólumnak megfelelő
              izolált csúcsok}

FOR i=1 to n-1
   csúcs=Csúcsot_Felvesz()
   csúcs.bal=Legkisebb_Értékű_Csúcsot_Kivesz(lista)
   csúcs.jobb=Legkisebb_Értékű_Csúcsot_Kivesz(lista)
   csúcs.valószínűség=
      (csúcs.bal).valószínűség+(csúcs.jobb).valószínűség
   Beilleszt(lista,csúcs)
ENDFOR
RETURN Legkisebb_Értékű_Csúcsot_Kivesz(lista)
```

A következő példában nyomon követhetjük az algoritmus egyes lépéseit.

4.7. példa. Legyen $n = 5$ és $p(x_1) = 0.35,\ p(x_2) = 0.2,\ p(x_3) = 0.2,\ p(x_4) = 0.15,\ p(x_5) = 0.1$, és keressük meg az ehhez tartozó Huffman-kódot. A 4.4. ábra bal oldalán az egyes lépéseket az összevonásokat jelölő nyilak melletti sorszámozással szemléltettük. A 0.4 és 0.6 valószínűségek optimális prefix kódja nyilván a 0 és az 1 (vagy fordítva). Az összevonások megfordításával elvégezhetjük a Huffman-kód felépítését. A 4.4. ábra jobb oldalán a lefelé mutató nyilakon feltüntettük, hogy az adott lépésben hogyan különböztettük meg a kódszavakat a 0 vagy

az 1 bit hozzárendelésével. Végül a bináris fa gyökerétől indulva és a levelekig szintenként lefelé haladva kiolvashatjuk az így kapott kódszavakat, amelyeket a levelek alatt is feltüntettünk. Tehát $f(x_1) = 10,\ f(x_2) = 00,\ f(x_3) = 01,\ f(x_4) = 110,\ f(x_5) = 111$.

A bemenet néhány speciális eloszlása esetén azonnal meg tudjuk adni a Huffman-kódot, az algoritmus tényleges lefuttatása nélkül is.

4.8. példa. A bemenet ún. diadikus eloszlása esetén, vagyis amikor az egyes szimbólumok valószínűségei a 2 negatív egész kitevős hatványaiként írhatók fel

$$p(x_i) = 2^{-\alpha_i}, \quad \alpha \in \mathbb{Z},$$

az egyes kódszavak α_i hosszúak, így az átlagos kódszóhossz

$$\mathbf{E}|f(X)| = \sum_{i=1}^{n} p(x_i)|f(x_i)| = \sum_{i=1}^{n} p(x_i)\,(-\log p(x_i)) = \sum_{i=1}^{n} 2^{-\alpha_i}\alpha_i.$$

4.9. példa. A bemenet egyenletes eloszlása, azaz

$$p(x_i) = \frac{1}{n}$$

esetén a kódszavak hossza k vagy $k-1$, ahol $k = \lceil \log n \rceil$. A Huffman-algoritmus a k hosszú kódszavak számát a lehető legkisebbre választja meg. Ezt legegyszerűbben úgy kaphatjuk meg, ha kiindulunk egy $k-1$ mélységű teljes bináris fából, és ennek néhány leveléhez további 2-2 csúcsot kapcsolunk a k-adik szinten. Egy levél ily módon való kettéágaztatása eggyel növeli a levelek, s ezzel együtt a kódszavak számát. Ennek eredményeként $2n - 2^k$ darab k hosszú és $2^k - n$ darab $k-1$ hosszú kódszót kapunk.

Mivel a Huffman-kód optimális, azaz a legkisebb átlagos kódszóhosszú prefix kód, ezért a 4.3. tétel miatt az átlagos kódszóhossza az entrópiánál legfeljebb 1 bittel nagyobb. Valójában ennél erősebb állítás is igaz: a Huffman-kód átlagos kódszóhossza az entrópiánál legfeljebb $p_{\max} + 0.086$ értékkel nagyobb, ahol $p_{\max}$ a leggyakoribb szimbólum valószínűsége. A gyakorlatban, nagy bemeneti ábécé esetén $p_{\max}$ értéke kicsi, így a Huffman-kód átlagos kódszóhosszának eltérése az entrópiától szintén kicsi, különösen ha az eltérést az entrópia arányában nézzük. Ugyanakkor kis ábécéméret és nagyon eltérő valószínűségek esetén $p_{\max}$, s így az átlagos kódszóhossz entrópiától való eltérése is meglehetősen nagy lehet. Ezen részben segíthetünk a blokk-kódolással, de sajnos ez újabb problémák forrása lehet.

4.10. példa. Tegyük fel, hogy a bemeneten kapott tömörítendő adatsorozat karakterei egy háromelemű ábécéből veszik fel az értékeiket, és az egyes szimbólumok egymástól függetlenül a következő eloszlás szerintiek:

$$p(x_1) = 0.95, \quad p(x_2) = 0.03, \quad p(x_3) = 0.02.$$

Ekkor a bemenet entrópiája 0.335 bit/szimbólum, míg a betűnkénti Huffman-kód átlagos kódszóhossza 1.05 bit/szimbólum, amely az entrópia 213%-a. Ha két-két szimbólum alkotta blokkra végezzük el a Huffman-kódolást, akkor 0.611 bit/szimbólum átlagos kódszóhosszat kapunk. Ezt folytatva egészen 8 szimbólum méretű blokkokig kell elmennünk ahhoz, hogy az átlagos kódszóhossz és az entrópia különbsége elfogadható értékre csökkenjen. Vegyük észre, hogy ehhez $3^8 = 6561$ elemű ábécé tartozik. Ilyen nagy méretű kódok alkalmazása több szempontból is előnytelen. Egyrészt már a kód tárolása (illetve a vevőhöz való átküldése) is sok helyet foglal. Másrészt sok időt és erőforrást igényel a dekódolás. Harmadrészt, ha egy kicsit is megváltozik a szimbólumok valószínűsége, az jelentősen ronthatja a kód hatékonyságát.

Láthatjuk, hogy sokkal hatékonyabb, ha blokkokat kódolunk egyes szimbólumok helyett. Minél nagyobb blokkméretet választunk, annál kisebb lesz a veszteség. Ugyanakkor a Huffman-kód nagy blokkméret esetén a gyakorlatban kevésbé alkalmazható. Egyrészt a kódszó elküldése csak a teljes m hosszú blokk beolvasása után lehetséges, és emiatt nagy m esetén jelentős késleltetés keletkezhet. Másrészt m hosszú blokkok Huffman-kódolásához az összes lehetséges m-hosszú sorozathoz tartozó kódszót elő kell állítanunk, ami a kód méretének exponenciális növekedését jelenti, és ez praktikus szempontból is korlátot állít m növelése elé. Olyan eljárásra van szükségünk, amely az egyes blokkokhoz úgy rendel kódszavakat, hogy közben nem kell meghatároznunk az összes többi ugyanolyan hosszú blokk kódszavát. Ezt a követelményt teljesíti a 4.5. szakaszban ismertetett aritmetikai kódolás, amelyet kifejezetten blokkonkénti kódoláshoz alkalmaznak. Jellegzetessége, hogy valós időben történik a kódszó előállítása és visszafejtése, tehát nem lép fel jelentős késleltetés, akármilyen hosszú blokkokat is kódolunk.

4.5. Aritmetikai kódolás

Az aritmetikai kódolás a Shannon–Fano-kód természetes kiterjesztése. Alapötlete, hogy a valószínűségeket intervallumokkal ábrázolja, amihez a valószínűségek pontos kiszámítása szükséges. Egyetlen kódszót rendel a bemeneti adathalmaz egy blokkjához, vagyis blokkonkénti kód. Egy kódszó a $[0, 1)$ intervallum egy jobbról nyílt részintervallumának felel meg, és megadása annyi bittel történik, amennyi már egyértelműen megkülönböztethetővé teszi bármely más részintervallumtól. A rövidebb kódszavak hosszabb intervallumokat, vagyis nagyobb valószínűségű bemeneti blokkokat kódolnak. A gyakorlatban a részintervallumokat fokozatosan finomítjuk a blokk szimbólumainak valószínűségei szerint, és mihelyt eldől a kimenet egy bitje (elegendően kicsi lesz az intervallum aktuális hossza), továbbítjuk azt.

Az aritmetikai kódolást ideális, azaz végtelen pontosságú lebegőpontos aritmetikával mutatjuk be. A gyakorlatban természetesen csak véges pontosságú aritmetikák léteznek, de a módszer ezeken is implementálható.

A kódoló működése a következő. Induláskor az aktuális $[E_0, V_0)$ intervallumnak a $[0,1)$ intervallumot vesszük. Ezután minden egyes bemeneti szimbólumot két lépésben kódolunk. Először tovább osztjuk az aktuális $[E_k, V_k)$ intervallumot diszjunkt részintervallumokra úgy, hogy ezek közül egy-egy a szimbólum lehetséges értékeinek feleljen meg. A részintervallumok hosszát a szimbólumértékek feltételes valószínűségével arányosnak választjuk meg. Az ily módon partíciót alkotó részintervallumok közül azt választjuk, amelyik a szimbólum ténylegesen előforduló értékének felel meg, és ezután ez lesz az új aktuális $[E_{k+1}, V_{k+1})$ intervallum.

Valójában nem szükséges egy-egy szimbólum esetén az összes részintervallumot meghatároznunk, elegendő a szimbólum tényleges értékének megfelelőre szorítkoznunk. Ehhez kumulatív valószínűségeket vezetünk be, a k-adik bemeneti szimbólum feldolgozásához annak i-edik lehetséges értéke esetén a

$$w_i^k := \begin{cases} 0, & \text{ha } i = 0, \\ \sum_{j=1}^{i-1} p(x_j \mid x_{\alpha_1}, \ldots, x_{\alpha_{k-1}}), & \text{ha } 1 \le i \le n \end{cases}$$

valószínűséget, ahol $x_{\alpha_1}, \ldots, x_{\alpha_{k-1}}$ az eddig már feldolgozott $k-1$ bemeneti szimbólumot jelöli. Az $[E_{k-1}, V_{k-1})$ aktuális intervallum esetén, ha a bemeneten az i-edik lehetséges szimbólumérték érkezik, akkor az új $[E_k, V_k)$ intervallum a következő:

$$E_k = E_{k-1} + w_i^k (V_{k-1} - E_{k-1}), \quad V_k = E_{k-1} + (w_i^k + p(x_i))(V_{k-1} - E_{k-1}).$$

Látható, hogy a végül így kialakuló intervallum egyértelműen meghatározza a teljes egymásba skatulyázott intervallumsorozatot, és emiatt belőle a teljes blokk visszafejthető. Az intervallum hossza a bemeneten érkezett szimbólumok feltételes valószínűségeinek szorzata, azaz a blokk valószínűsége. Az utolsó intervallumot kódoljuk tehát, de a gyakorlatban nem az intervallum alsó és felső határa, hanem annak egy tetszőlegesen választott eleme adja a kódszót. Ebből is visszaállítható ugyanis az üzenet. Célszerű az intervallumból a lehető legkevesebb bittel leírható elemet kiválasztanunk. Belátható, hogy az $\mathbf{x}$ üzenetblokkhoz tartozó intervallumból alkalmasan kiválasztott elem leírásához elegendő $\left\lceil \log \frac{1}{p(\mathbf{x})} \right\rceil + 1$ bit. Ebből megkapható, hogy egy m hosszú blokkhoz tartozó kódszó hosszának várható értékére, azaz az átlagos kódszóhosszra igaz a következő:

$$\begin{aligned} E\,|f(\mathbf{X})| &\le \mathbf{E}\left(\left\lceil \log \frac{1}{p(\mathbf{X})} \right\rceil + 1\right) = \\ &= \sum p(\mathbf{x}) \left\lceil \log \frac{1}{p(\mathbf{x})} \right\rceil + 1 < \\ &< \sum p(\mathbf{x}) \left(\log \frac{1}{p(\mathbf{x})} + 1\right) + 1 = \\ &= -\sum p(\mathbf{x}) \log p(\mathbf{x}) + \sum p(\mathbf{x}) + 1 = \\ &= H(\mathbf{X}) + 2. \end{aligned}$$

Az aritmerikai kód átlagos kódszóhosszára tehát rosszabb korlátunk van, mint a Shannon–Fano-kód esetén. Ennek az az oka, hogy nem rendezzük sorba minden lépésben a szimbólumokat a feltételes valószínűségeik szerint csökkenő módon.

Ha az $\mathbf{X}$ üzenetvektor komponensei független azonos eloszlásúak, akkor a w_i^k valószínűségek nem függenek k-tól, és a kiszámításukhoz sem kell feltételes valószínűségeket használnunk:

$$w_i := \begin{cases} 0, & \text{ha } i = 0, \\ \sum_{j=1}^{i-1} p(x_j), & \text{ha } 1 \le i \le n \end{cases}$$

Ekkor $H(\mathbf{X}) = mH(X_1)$, tehát a betűnkénti átlagos kódszóhosszra

$$\frac{1}{m}\mathbf{E}|f(\mathbf{X})| \le H(X_1) + \frac{2}{m},$$

azaz az m blokkhossz növelésével tetszőlegesen megközelíti $H(X_1)$-et. Fontos, hogy ellentétben a blokkonkénti Huffman-kódolással itt a bonyolultság nem nő a blokkok hosszának növelésével.

Az alábbiakban az aritmetikai kódoló független azonos eloszlású bemeneti szimbólumokra működő pszeudokódját adjuk meg. Annyiban egyszerűsítjük a megoldást, hogy az algoritmus a végső intervallumból nem feltétlenül a lehető legkevesebb bittel ábrázolható elemet választja ki.

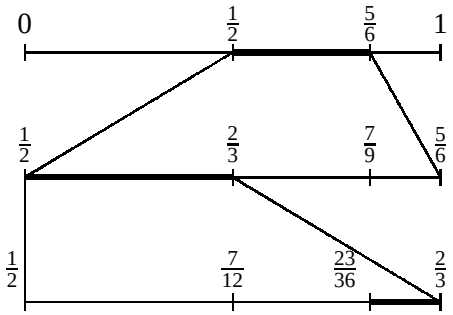
```
INPUT:  x[ ]={tömörítendő blokk}
        n={tömörítendő blokk hossza}
        p[ ]={szimbólumok valószínűségei}
        w[ ]={szimbólumok kumulált valószínűségei}

OUTPUT: c={kódszó}

E=0
hossz=1
FOR i=0 to n-1
   E=E+hossz*w[x[i]]
   hossz=hossz*p[x[i]]
ENDFOR
méret=FelsőEgészRész(-log(hossz))+1
c=FelsőEgészRész(E*Hatvány(2,méret))
RETURN c
```

A dekódoló pszeudokódja pedig a következő:

```
INPUT:  c={kódszó}
        n={dekódolandó blokk hossza}
        p[ ]={szimbólumok valószínűségei}
        w[ ]={szimbólumok kumulált valószínűségei}
        F[ ]={forrásábécé}
```



4.5. ábra. Az intervallum felosztása aritmetikai kódolás során

```
OUTPUT: x[ ]={dekódolt blokk}

FOR i=0 to n-1
   j=1
   WHILE c<w[F[j]]
      j++
   ENDWHILE
   x[i]=F[j-1]
   c=(c-w[x[i]])/p[x[i]]
ENDFOR
RETURN x
```

4.11. példa. Tegyük fel, hogy a bemeneti szimbólumok független azonos eloszlásúak, és háromféle értéket vehetnek fel. Eloszlásuk és az ehhez tartozó kumulatív valószínűségek a következők:

$$\begin{aligned} p(x_1) &= \tfrac{1}{2}, \qquad w_1 = 0, \\ p(x_2) &= \tfrac{1}{3}, \qquad w_2 = \tfrac{1}{2}, \\ p(x_3) &= \tfrac{1}{6}, \qquad w_3 = \tfrac{5}{6}. \end{aligned}$$

Kódoljuk aritmetikai kódolóval az x_2, x_1, x_3 sorozatot. A kezdeti intervallum az

$$[E_0, V_0) = [0, 1)$$

(alsó indexszel az eddig már feldolgozott szimbólumok számát jelöljük). Az első, vagyis az x_2 szimbólum után három részre vágjuk az intervallumot, és a középső részt választjuk ki (4.5. ábra), tehát

$$[E_1, V_1) = \left[0 + \tfrac{1}{2}(1-0),\ 0 + \left(\tfrac{1}{2} + \tfrac{1}{3}\right)(1-0)\right) = \left[\tfrac{1}{2}, \tfrac{5}{6}\right).$$

Ismét három részre osztjuk az aktuális intervallumot, s mivel a második szimbólum az x_1, így az első részintervallumot választjuk:

$$[E_2, V_2) = \left[\tfrac{1}{2} + 0\left(\tfrac{5}{6} - \tfrac{1}{2}\right),\ \tfrac{1}{2} + \left(0 + \tfrac{1}{2}\right)\left(\tfrac{5}{6} - \tfrac{1}{2}\right)\right) = \left[\tfrac{1}{2}, \tfrac{2}{3}\right).$$

Végül a harmadik részintervallumokra bontás után az x_3 szimbólumot a három közül az utolsó intervallummal kódoljuk, tehát

$$[E_3, V_3) = \left[\tfrac{1}{2} + \tfrac{5}{6}\left(\tfrac{2}{3} - \tfrac{1}{2}\right),\ \tfrac{1}{2} + \left(\tfrac{5}{6} + \tfrac{1}{6}\right)\left(\tfrac{2}{3} - \tfrac{1}{2}\right)\right) = \left[\tfrac{23}{36}, \tfrac{2}{3}\right).$$

Láthatjuk, hogy az $[E_3, V_3)$ intervallum hossza valóban megegyezik a szimbólumok valószínűségeinek szorzatával, azaz a $p(x_2)p(x_1)p(x_3) = \frac{1}{36}$ értékkel. Ebből rögtön felső becslést is adhatunk a kimenet hosszára: $\left\lfloor -\log \frac{1}{36} \right\rfloor + 2 = 7$ bit. A kimenet előállításához írjuk fel az utolsó intervallumot bináris alakban:

$$[E_3, V_3) = [0.101000111\ldots_2,\ 0.101010101\ldots_2).$$

Mivel az összes, $0.101001\ldots_2$ számjegyekkel kezdődő bináris szám benne van ebben az intervallumban, és ez a legrövidebb ilyen tulajdonságú prefix, így a kimenetre az 101001 sorozatot küldhetjük.

Az aritmetikai kódolás előzőekben ismertetett legegyszerűbb megvalósítása két problémát vet fel. Az egyre csökkenő méretű intervallumok kezelése igen nagy pontosságú aritmetikát tesz szükségessé, másrészt a teljes bemenet vagy bemeneti blokk beolvasásáig egyetlen bit sem jelenik meg a kimeneten. Mindkét problémára megoldást jelent, hogy a kódszó elején lévő bitek (tehát a legutolsó intervallum alkalmasan kiválasztott elemének bináris tört ábrázolásban vett első néhány bitje) már az első néhány forráskarakterből meghatározhatók. Ennélfogva a kódszó bitjeit, amint ismertté válnak, rögtön elküldhetjük a kimenetre. Ezzel együtt felskálázzuk az aktuális intervallum hosszát, amely így már csak a végső intervallum eddig még ismeretlen részét fogja reprezentálni. Abban az esetben, amikor az aktuális intervallum teljes egészében az $\frac{1}{2}$ bal vagy jobb oldalára esik, a megoldás viszonylag egyszerű. A bemutatott eljárás igazi ötlete annak kezelése, amikor az aktuális intervallum közrefogja az $\frac{1}{2}$-et. Ehhez egy számlálót fogunk bevezetni.

A korábban ismertett algoritmusba az új aktuális $[E_k, V_k)$ intervallum kiválasztása után be kell szúrnunk az alábbi lépéseket:

- Ha az aktuális intervallum a $\left[0, \frac{1}{2}\right)$ szakaszra esik, akkor egy 0-t, majd a számláló értékének megfelelő darab 1-est írunk a kimenetre. Ezután megduplázzuk az intervallum hosszát úgy, hogy a $\left[0, \frac{1}{2}\right)$ szakaszt lineárisan kiterjesztjük a $[0,1)$ szakaszra.

- Ha az aktuális intervallum az $\left[\frac{1}{2}, 1\right)$ szakaszra esik, akkor egy 1-est, majd a számláló értékének megfelelő darab 0-t írunk a kimenetre. Ezután megduplázzuk az intervallum hosszát úgy, hogy az $\left[\frac{1}{2}, 1\right)$ szakaszt lineárisan kiterjesztjük a $[0,1)$ szakaszra.

- Ha az aktuális intervallum az $\left[\frac{1}{4}, \frac{3}{4}\right)$ szakaszra esik, akkor nem írunk semmit a kimenetre, de megnöveljük a számláló értékét. Ezután megduplázzuk az intervallum hosszát úgy, hogy az $\left[\frac{1}{4}, \frac{3}{4}\right)$ szakaszt lineárisan kiterjesztjük a $[0,1)$ szakaszra.

Ezeket a lépéseket mindaddig ismételjük, ameddig már egyik feltétel sem teljesül az aktuális intervallumra.

```
INPUT:  x[ ]={tömörítendő blokk}
        n={tömörítendő blokk hossza}
        p[ ]={szimbólumok valószínűségei}
        w[ ]={szimbólumok kumulált valószínűségei}

OUTPUT: c={kódszó}

E=0
V=1
hossz=1
számláló=0

FOR i=0 to n-1
   E=E+w(x[i])*hossz
   hossz=hossz*p(x[i])
   V=E+hossz
   WHILE V-E<0.5 && !(E<0.25 && V>=0.75)
      IF V<0.5 THEN
         c=c,0,számláló db 1
         számláló=0
         E=E*2
         V=V*2
      ELSE IF E>=0.5 THEN
         c=c,1,számláló db 0
         számláló=0
         E=E*2-1
         V=V*2-1
      ELSE IF E>=0.25 && V<0.75 THEN
         számláló=számláló+1
         E=E*2-0.5
         V=V*2-0.5
      ENDIF
   ENDWHILE
ENDFOR
IF számláló>0 THEN
   c=c,0,számláló db 1
ENDIF
RETURN c
```

Végezetül még egy dologra szeretnénk rámutatni. Az aktuális intervallum a már feldolgozott bemenetről hordoz olyan információt, amely eddig még nem jelent meg a kimeneten, ezért a kódoló állapotának is tekinthetjük. Egy h hosszú intervallum $-\log h$ bites állapotot jelent. A szakasz elején bemutatott legegyszerűbb aritmetikai kódoló esetén az állapot az összes információt tartalmazza, hiszen a kimeneten egyetlen bit sem jelenik meg a bemenet teljes feldolgozásáig. A módosított algoritmus esetén az állapot mindig kettőnél keveseb bit információt tar-

talmaz, ugyanis a szabály szerint az aktuális intervallum hossza mindig legalább $\frac{1}{4}$.

4.6. Adaptív tömörítés

A veszteségmentes tömörítés alapkérdése, hogy hogyan bontsuk a bemeneti adathalmazt elemi összetevőkre, ún. szimbólumokra, majd hogyan kódoljuk ezeket a lehető legkevesebb bittel. Az alapötlet az, hogy rövid kódszavakat rendelünk a gyakran előforduló szimbólumokhoz, és hosszabb kódszavakat a ritkábbakhoz. Az adatok akkor tömöríthetők, ha a szimbólumok egy része gyakrabban fordul elő, mint a többi (vagy az egymás utáni szimbólumok összefüggőek). Az alapján, hogy ezt milyen módon valósítják meg a forráskódolók, két csoportot különböztethetünk meg. A statisztikus kódolók megpróbálnak jó becslést adni a szimbólumok valószínűségi eloszlására. Ezt a lépést a bemeneti adatok (forrás) modellezésének nevezzük. Ide tartozik például a Huffman-kódoló is. A szótár alapú algoritmusok vagy más néven univerzális forráskódolók viszont olyan mintákat gyűjtenek egy szótárba, amelyek korábban előfordultak a bemeneten, és egy minta következő előfordulását a szótárban elfoglalt helyével kódolják. A módszer legelterjedtebb képviselői a Lempel–Ziv-algoritmusok (4.8. szakasz). Meg kell említenünk egy harmadik típusú módszert, amelyik igazából egyik előző csoportba sem esik, ez az ún. Burrows–Wheeler-transzformáción alapuló tömörítési eljárás (4.9. szakasz).

A statisztikus kódolónak egy modellel kell kiegészülnie, amely a kódolás minden lépésében megbecsüli az egyes szimbólumok előfordulási valószínűségeit. A valószínűségi modellnek nem szükséges leírnia a bemeneti adatokat előállító folyamatot, elegendő ha megadja a szimbólumok eloszlását. A kódoló működéséhez a valószínűségeknek még csak nem is kell különösebben pontosaknak lenniük. Persze minél pontosabbak, annál jobb tömörítésre számíthatunk. Szélsőséges esetben, ha a valószínűségek meg sem közelítik a valóságot, a kimenet akár hosszabb is lehet, mint a bemenet. A hatékony tömörítés eléréséhez egyrészt jó valószínűségi modellre van szükségünk, másrészt minél pontosabban kell ismernünk (vagy a bemenet alapján a modellnek megtanítanunk) a valószínűség-értékeket.

A dekódolhatóság érdekében a tömörítéskor csak olyan információ használható fel, amely a dekódolónak is rendelkezésére áll. Ettől eltekintve nincs más megszorítás a modellre, sőt, a bemeneti adatok feldolgozása közben még változhat is az. A modell lehet adaptív (dinamikusan, az előző szimbólumok figyelembe vételével határozza meg az adott szimbólum valószínűségét), fél-adaptív (előzetesen végigolvassa a teljes bemenetet azért, hogy statisztikát készítsen) és nem adaptív (rögzített statisztikát használ a teljes bemeneti adathalmazra). A nem adaptív modell rendkívül rosszul is teljesíthet. Az adaptív kódok a bemenetet csak egyszer olvassák végig, de bonyolult adatszerkezeteket igényelhetnek. A fél-adaptív kódok a bemenet kétszeri beolvasását és a modell adatainak átküldését teszik szükségessé, ami hosszú üzenet esetén jelentős késleltetést okoz. Amennyiben ez utóbbit hatékonyan tudjuk megtenni, a fél-adaptív kódok az adaptívaknál kicsivel jobb ered-

ményt adhatnak, de általában a modellparaméterek átküldése illetve megtanulása azonos költséget jelent.

Fél-adaptív modell esetén a forrásbetűk eloszlását a relatív gyakoriságokkal kell becsülnünk. A gyakorlati problémák során általában adott a $Z_1,\ldots,Z_N$ bemeneti sorozat (szöveg, fájl), amelyet szeretnénk optimálisan tömöríteni. Feladatunk a $\sum_{i=1}^{N}|f(Z_i)|$ minimalizálása, vagyis az ilyen értelemben optimális f kódolófüggvény megválasztása. Ez egyenértékű a kódszóhosszak átlagának, $\frac{1}{N}\sum_{i=1}^{N}|f(Z_i)|$-nek a minimalizálásával.

Belátjuk, hogy $\frac{1}{N}\sum_{i=1}^{N}|f(Z_i)| = \mathbf{E}_N|f(Z)|$, ha a várható értéket az empirikus eloszlás szerint vesszük, tehát a forrásbetűk valószínűségét a relatív gyakoriságukkal definiáljuk:

$$p_N(x_j) = \frac{1}{N}\sum_{i=1}^{N} I_{\{Z_i=x_j\}},$$

ahol $I_{\{\}}$ az indikátor függvény. Nyilván

$$\begin{aligned}\mathbf{E}_N|f(Z)| &= \sum_{j=1}^{n} p_N(x_j)|f(x_j)| = \\ &= \sum_{j=1}^{n}\frac{1}{N}\sum_{i=1}^{N} I_{\{Z_i=x_j\}}|f(x_j)| = \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n} I_{\{Z_i=x_j\}}|f(x_j)| = \\ &= \frac{1}{N}\sum_{i=1}^{N}|f(Z_i)|\end{aligned}$$

A fél-adaptív kódolás két problémája, hogy egyrészt magát az empirikus eloszlást is le kell írni, át kell vinni a dekódolóhoz, és ez a „fejléc” így a fentinél nagyobb átlagos kódszóhosszat eredményez. Az aszimptotikus vizsgálat során azonban ettől a konstans költségtől eltekinthetünk. Másrészt az algoritmust csak két lépésben tudjuk végrehajtani. Először meghatározzuk a forrásbetűk relatív gyakoriságát, ami az előzőek értelmében megegyezik a valószínűségekkel, majd ennek felhasználásával elvégezzük a tényleges kódolást. Nem mindig engedhető meg azonban olyan nagy mértékű késleltetés, hogy csak az összes bemeneti adat megérkezése után kezdünk hozzá a kimenet előállításához, másrészt a kétmenetes beolvasás akkor is lassítja az algoritmust (bár kétségkívül optimális kódot eredményez), ha a bemenet már rendelkezésre áll. A gyakorlatban ezért sokszor érdemes adaptív kódot, vagyis egymenetes algoritmust használni. Így az optimalitás rovására időt takaríthatunk meg. Egy forrásbetűt az előző forrásbetűk előfordulásai alapján kódolunk, s ezzel együtt lépésenként változik maga a kód is. Tehát az aktuális forrásbetű kódolását egy, az előzőleg feldolgozott forrásbetűkre nézve optimális kóddal hajtjuk végre.

Jó tömörítés eléréséhez fel kell tárnunk a bemeneti adatok struktúráját. Ez például képek esetén azt jelenti, hogy egy adott képpont intenzitás értékét a szomszédainak intenzitásaiból becsüljük meg és az így elkövetett hibát egy alkalmas valószínűségi eloszlással írjuk le, amely figyelembe veszi a kép különböző területeinek jellegzetességeit. Szövegfájlok esetén a megelőző betűk alapján egy Markov-lánc segítségével következtethetünk az aktuális karakterre.

4.7. Adaptív Huffman-kód

A 4.4. szakaszban vizsgált (nem adaptív) Huffman-kódnál feltételeztük, hogy eleve ismerjük a bemenet eloszlását. Ez azonban nincs mindig így. Fél-adaptív Huffman-kód esetén az eloszlást a relatív gyakoriságokkal kell becsülnünk. Amennyiben nem engedhető meg a bemenet kétszeri beolvasásából eredő késleltetés, adaptív Huffman-kódot használunk.

A bináris Huffman-fában a valószínűségek illetve a relatív gyakoriságok szerepeltek (lásd 4.4. ábra). Adaptív Huffman-kódolás esetén ezek helyett a gyakoriságokat használjuk (hiszen utóbbiakat a bemenet hosszával elosztva megkapjuk a relatív gyakoriságokat, és számunkra csak az értékek egymáshoz való aránya érdekes).

Az adaptív Huffman-kódoláshoz is egy bináris kódfát fogunk használni, viszont ezt nem építjük újra minden egyes forrásbetű után, hanem csak fokozatosan finomítjuk. Ugyan egy új betű olvasásakor megváltoznak a relatív gyakoriságok, de ez ritkán eredményezi a kódfa megváltozását. Algoritmikus szempontból két feladat van: egyrészt meg kell oldanunk, hogy a kódoló és a dekódoló észrevegye, ha meg kell változtatnia a kódfát, másrészt egyszerű algoritmust kell adnunk a változtatásra. Ennek egyszerű kezeléséhez bevezetjük a kódfa egy új tulajdonságát, az ún. testvér tulajdonságot (sibling property). Belátható, hogy egy prefix kód pontosan akkor Huffman-kód, ha teljesíti a testvér tulajdonságot.

4.6. definíció (testvér tulajdonság). *Egy bináris kódfa rendelkezik a testvér tulajdonsággal, ha a gyökér kivételével minden csúcsának van testvére (azaz a szülőjének egy másik gyereke), és a csúcsok felsorolhatók a hozzájuk rendelt valószínűségek (általánosabban: értékek) nemnövekvő sorrendjében úgy, hogy a testvérek egymás mellé kerüljenek ebben a listában.*

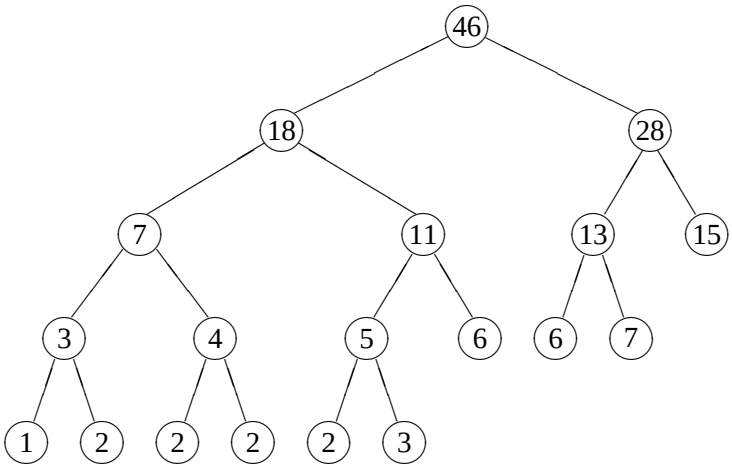
4.12. példa. A 4.6. ábrán látható fára teljesül a testvér tulajdonság. A nemnövekvő sorrend a következő:

$$(28,18);\ (15,13);\ (11,7);\ (7,6);\ (6,5);\ (4,3);\ (3,2);\ (2,2);\ (2,1)$$

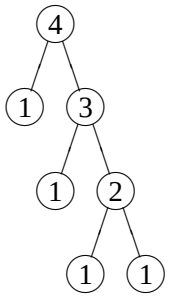
A 4.7. ábra kódfájának testvérei:

$$(3,1);\quad (2,1);\quad (1,1)$$

Ezeket nem tudjuk megfelelően sorba rendezni, így nem teljesül a fára a testvér tulajdonság.

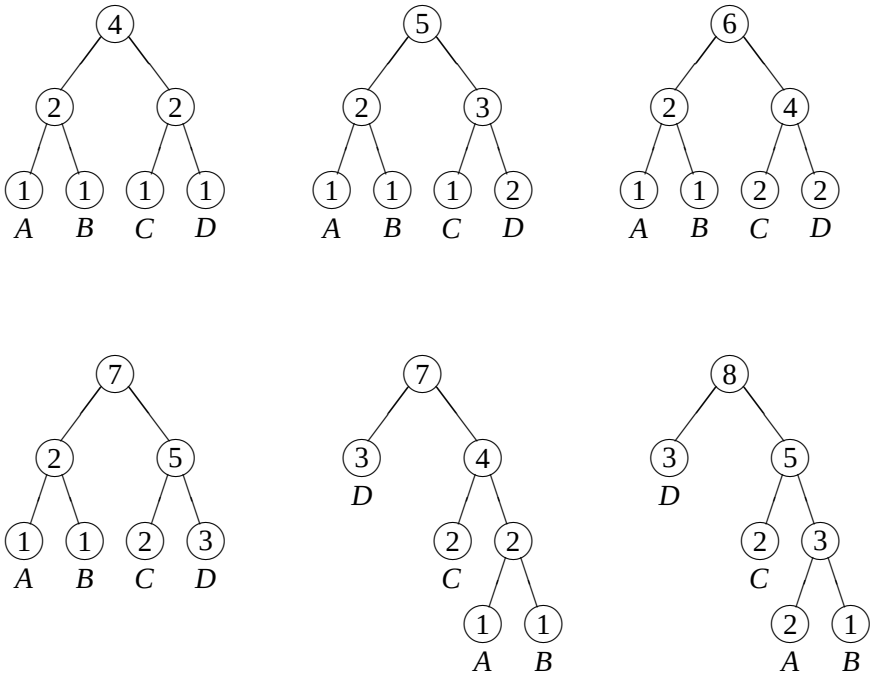


4.6. ábra. Kódfa, amelyre teljesül a testvér tulajdonság



4.7. ábra. Kódfa, amelyre nem teljesül a testvér tulajdonság

Az adaptív algoritmus lépései a következők:
Ismerjük a forrásszimbólumokat: $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$. Kiindulásként felépítünk egy olyan Huffman-fát, amelyben — ha nincs információnk a betűk előfordulásának valószínűségéről — minden forrásszimbólum azonos, 1 gyakorisággal szerepel. Így egy kiegyensúlyozott bináris fát kapunk. (Ha ismerjük a kezdeti eloszlást, megtehetjük, hogy erre építjük fel az adaptív algoritmus kiindulási fáját.) Elküldjük a dekódolónak sorrendben (pl. a gyökértől a levelek felé, szintenként, balról jobbra) a lehetséges forrásszimbólumokat, amelyből az szintén felépíti a kiindulási Huffman-fát. (Ügyelni kell arra, hogy a kódoló és a dekódoló azonos algoritmust használjon.) A kódolás során beolvassuk a k-adik forrásszimbólumot. Ezt a korábban beolvasott $k-1$ betű feldolgozásával kialakult, lokálisan optimális kódfával kódoljuk, majd eggyel növeljük a forrásszimbólum gyakoriságszámlálóját. Aktualizáljuk a fát, vagyis megvizsgáljuk, hogy fennáll-e még a testvér tulajdonság. Amennyiben nem, helyreállítjuk azt a kód optimalitásának érdekében: a most be-



4.8. ábra. Az 4.13. példa kódfájának alakulása

olvasott szimbólumhoz tartozó levelet felcseréljük a tőle a fában legtávolabb lévő (vagyis a leghosszabb úton elérhető), nála eggyel kisebb értékű csúccsal az abból esetlegesen kiinduló részfával együtt. A felcserélt csúcsok szülőinek gyakoriságszámlálóit megfelelően módosítjuk (eggyel csökkentjük vagy növeljük). Ezeket a lépéseket a dekódoló is elvégzi, tehát a következő kódbetű dekódolásakor ugyanazt a fát fogja használni, mint amelyet a kódoló használt annak előállításakor. A $(k+1)$-edik betű beolvasása után az előzőekhez hasonlóan folytatjuk.

Ügyelni kell arra, hogy hosszú bemenet esetén előfordulhat a szimbólumok gyakoriságszámlálójának túlcsordulása. Ezt megfelelő technikával orvosolni kell.

4.13. példa. A bemeneti ábécénk: $\mathcal{X} = \{A,B,C,D\}$. Adaptív Huffman-kóddal kódoljuk a beérkező forrásszimbólumokat, melyek legyenek a következők: *DCDA*. A 4.8. ábrán követhetjük nyomon a kódfa alakulását. Látható, hogy a testvér tulajdonság az *DCD* feldolgozása után sérül, a *D* gyakoriságszámlálójának 3-ra növelésével. Ekkor az *A* és *B* levelek szülőjét a hozzá tartozó részfával együtt felcseréljük a *D* csúccsal, s így helyreállítjuk a kódfa testvér tulajdonságát, majd ezzel kódolható az utolsó, *A* szimbólum.

4.8. Univerzális forráskódolás: Lempel–Ziv-típusú módszerek

A '70-es évek második felében Abraham Lempel és Jacob Ziv az addig alkalmazott statisztikus kódolóktól gyökeresen eltérő forráskódoló technika ötletével állt elő. Az általuk javasolt módszer nem használ valószínűségi modellt. Alapötlete az, hogy egy előzőleg már látott karaktersorozatot egy pufferbeli mutatóra (LZ77) illetve egy szótárbeli indexre (LZ78) cserél le. A statisztikus kódolók egy gyakori blokkhoz rövid kódszót rendelnek, a Lempel–Ziv-típusú kódolók pedig egy, a múltban látott leghosszabb egyezéshez rendelnek lényegében fix hosszúságú kódszót.

Az eddig vizsgált kódok alkalmazásakor az adó és a vevő között átvitelre kerülő bitek két csoportot alkotnak. Először átvisszük a blokk-kódot leíró információt. Ez egy állandó költséget jelent, független az üzenet tényleges hosszától. Majd következnek az üzenet kódszavai. Elméleti vizsgálataink során azzal a feltételezéssel éltünk, hogy a továbbítandó üzenetünk végtelen hosszú. Ilymódon, a kódok aszimptotikus viselkedését tekintve, az állandó költség fajlagosan nullához tart, tehát elhanyagolható. A gyakorlatban azonban véges hosszúságú forrásszegmensekkel van dolgunk. Ebben az esetben az állandó költség akár nagyobb is lehet, mint az üzenet kódszavainak összhosszúsága. Ezt elkerülendő, jó lenne, ha rendekezésünkre állna egy olyan technika, amelynek nincs állandó költsége, de aszimptotikusan ugyanolyan jó tömörítési arányt ér el, mint a blokk-kódok. Az állandó költség abból adódik, hogy a kódot a forráson előzetesen elvégzett statisztikai vizsgálatok (a forrásszimbólumok gyakorisága) alapján hozzuk létre, tehát ezek az adatok szükségesek a kód leírásához. Ehelyett járjunk el úgy, hogy menet közben gyűjtünk információt a forrásszimbólumokról, vagyis az aktuális szimbólumot az ezt megelőző szimbólumok alapján kódoljuk. Az ilyen kódokat **adaptív kódok**nak nevezzük, alkalmazásuk során nincs állandó költség. Korábban találkoztunk már ilyen módszerrel az adaptív Huffman-kód kapcsán. A most tárgyalásra kerülő Lempel–Ziv-kódok is ebbe a családba tartoznak.

Az első Lempel–Ziv-algoritmus az 1977-ben publikált LZ77.

Az LZ77 algoritmus

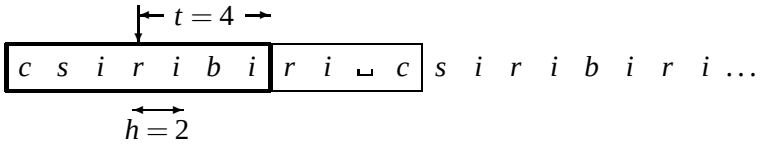
A kódoló a forrásszimbólumok sorozatát egy h_a hosszú csúszóablakon keresztül vizsgálja. Az ablak két részből áll: egy keresőpufferből, amely a legutóbb kódolt h_k darab forrásszimbólumot tartalmazza, és egy előretekintő pufferből, amely a következő h_e darab kódolandó szimbólumot tartalmazza ($h_a = h_k + h_e$ és h_k általában sokkal nagyobb, mint h_e). A kódoló a keresőpufferben megkeresi az előretekintő pufferben lévő karaktersorozattal leghosszabban egyező részt, majd elküld egy $\langle t, h, c \rangle$ hármast, ahol t a keresőpufferben megtalált karaktersorozat távolsága az előretekintő puffertől (offset), h a kereső- és az előretekintő puffer egyező szimbólumainak legnagyobb hosszúsága, c pedig az első, az előretekintő pufferben lévő

nem egyező karakter kódszava. Azért küldjük el az első nem egyező karakter kódját is, hogy kezeljük azt az esetet, amikor az előretekintő puffer szimbólumait nem találjuk meg a keresőpufferben. Ilyenkor t és h értéke 0. Egy hármas kódolásához állandó hosszúságú kód használatával $\lceil \log h_k \rceil + \lceil \log h_e \rceil + \lceil \log |\mathcal{X}| \rceil$ bit szükséges, ahol $|\mathcal{X}|$ a forrásábécé mérete. Figyeljük meg, hogy az egyező szimbólumok hosszúságának átviteléhez nem $\lceil \log h_k \rceil$, hanem $\lceil \log h_e \rceil$ bit szükséges. Ennek oka, hogy az egyezés hossza meghaladhatja a keresőpuffer hosszát, vagyis az egyező rész átlóghat az előretekintő pufferbe.

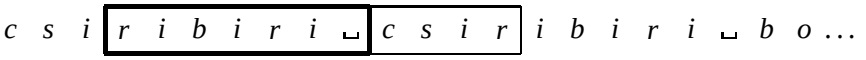
4.14. példa. Kódoljuk Weöres Sándor: Varázsének (1934) című versének alábbi részletét az LZ77 algoritmussal.

csiribiri␣csiribiri␣bojtorján–
lélek␣lép␣a␣lajtorján

Legyen $h_a := 11$, $h_k := 7$, $h_e := 4$. Tegyük fel, hogy az első néhány karaktert már kódoltuk. Ekkor:



Látható, hogy az előretekintő puffer tartalmával a leghosszabb egyezést a keresőpufferben $t = 4$ távolságra találjuk meg. Ekkor az egyezés hosszúsága $h = 2$. Tehát az *ri*␣ karaktereket a $\langle 4, 2, f(␣)\rangle$ hármassal kódoljuk, ahol $f(␣)$ a ␣ karakter kódját jelöli. Az ablakot hárommal jobbra mozgatjuk, így:



Az előretekintő puffer első karaktere, *c*, nem található meg a keresőpufferben. Átküldjük a $\langle 0, 0, f(c)\rangle$ hármast. Az ablakot eggyel jobbra mozgatjuk, és így tovább. Az elküldött hármasok a következők:

$\langle 4,2,f(␣)\rangle$ $\langle 0,0,f(c)\rangle$ $\langle 0,0,f(s)\rangle$ $\langle 6,3,f(b)\rangle$ $\langle 4,3,f(␣)\rangle$ $\langle 5,1,f(o)\rangle$ $\langle 0,0,f(j)\rangle$

$\langle 0,0,f(t)\rangle$ $\langle 3,1,f(r)\rangle$ $\langle 4,1,f(á)\rangle$ $\langle 0,0,f(n)\rangle$ $\langle 0,0,f(-)\rangle$ $\langle 0,0,f(l)\rangle$ $\langle 0,0,f(é)\rangle$

$\langle 2,1,f(e)\rangle$ $\langle 0,0,f(k)\rangle$ $\langle 0,0,f(␣)\rangle$ $\langle 6,2,f(p)\rangle$ $\langle 4,1,f(a)\rangle$ $\langle 6,2,f(a)\rangle$ $\langle 0,0,f(j)\rangle$

$\langle 0,0,f(t)\rangle$ $\langle 0,0,f(v)\rangle$ $\langle 0,0,f(r)\rangle$ $\langle 4,1,f(á)\rangle$ $\langle 0,0,f(n)\rangle$

Az alábbiakban bejelöltük a kereső és az előretekintő puffer határát az egyes kódolási lépések során:

csiribi|ri␣|c|s|irib|iri␣|bo|j|t|or|já|n|-|
l|é|le|k|␣|lép|␣a|␣la|j|t|o|r|já|n

Elképzelhető, hogy a leghosszabb egyező karaktersorozat átnyúlik az előretekintő pufferbe, tehát $h > t$. A dekódolás során ez az „átlógás” nem okoz gondot, mert az első néhány karaktert könnyen megkapjuk a már előzőleg dekódolt karakterekből, a maradékot pedig az előbbi lépés során megkapott néhány karakter segítségével nyerjük.

Láthatjuk, hogy az LZ77 egy rendkívül egyszerű adaptív algoritmus, amely nem igényel előzetes ismeretet vagy feltevést a forrásról. Megmutatható, hogy az eljárás hatékonysága aszimptotikusan ($h_k, h_e \to \infty$) megközelíti az optimális algoritmusét, amely előzetesen ismeri a bemenet eloszlását.

Az LZ77 alkalmazása során a bemeneti szimbólumok legutóbb kódolt sorozatát használjuk, így azzal a feltételezéssel élünk, hogy a minták egymáshoz közeli intervallumokban visszatérnek (a mozgó ablakon belül). Szélsőséges esetben, ha az ismétlődés hossza éppen eggyel hosszabb a keresőpuffer méreténél, nem tudunk tömöríteni. A Lempel–Ziv-algoritmus következő, 1978-as veriójánál (LZ78) ezt a problémát egy másfajta, adaptív szótár alapú rendszerrel oldják fel.

Az LZ78 algoritmus

A kódoló és a dekódoló is szótárt épít az előzőleg előfordult sorozatokból. A kódoló megkeresi a forrásszimbólumok aktuális pozíciójától kezdődő leghosszabb egyezést a szótárban. Átküld egy $\langle i, c\rangle$ párt, ahol i az egyező karaktersorozat szótárbeli indexét jelöli, c pedig az első nem egyező karakter kódja, majd felveszi a szótárba az i indexű egyező karaktersorozat és a c karakter összefűzésével kapott sztringet (a következő szabad indexet adja neki). Ha nem talál egyező karaktersorozatot a szótárban, akkor a $\langle 0, c\rangle$ párost küldi át, c itt is az első nem egyező karakter kódja, amely ebben az esetben természetesen az első feldolgozandó szimbólum.

4.15. példa. Kódoljuk a korábbi Weöres-idézetet az LZ78 algoritmussal.

csiribiri␣csiribiri␣bojtorján–
lélek␣lép␣a␣lajtorján

Kezdetben a szótár üres, ezért az első 4 szimbólumot egyenként felvesszük a szótárba, és a 0 indexszel átküldjük: $\langle 0, f(c)\rangle, \langle 0, f(s)\rangle, \langle 0, f(i)\rangle, \langle 0, f(r)\rangle$. Az ötödik szimbólum az *i*, amely szerepel a szótárban, a következővel együtt (*ib*) viszont már nem, ezért átküldjük a $\langle 3, f(b)\rangle$ párost, amelyből a 3 jelöli az *i* indexét, $f(b)$ pedig a következő karakter, vagyis a *b* kódját. Az *ib* sorozatot felvesszük a szótárba, indexe 5 lesz. Így folytatjuk az eljárást, az eredményt a 4.9. ábrán látható táblázatban foglaltuk össze. Látható, hogy a szótárbeli bejegyzések egyre hosszabbak, és ha a bemeneti sorozat ismétlődik, akkor előbb-utóbb az egész sztring szerepelni fog a szótárban. Az egyes lépéseket bejelöltük a bemeneten:

c|s|i|r|ib|ir|i␣|cs|iri|b|iri␣|bo|j|t|o|rj|á|n|-|
l|é|le|k|␣|lé|p|␣a|␣l|a|jt|or|já|n

a kódoló kimenete	szótár index	bejegyzés
$\langle 0, f(c)\rangle$	1	c
$\langle 0, f(s)\rangle$	2	s
$\langle 0, f(i)\rangle$	3	i
$\langle 0, f(r)\rangle$	4	r
$\langle 3, f(b)\rangle$	5	ib
$\langle 3, f(r)\rangle$	6	ir
$\langle 3, f(\text{␣})\rangle$	7	i␣
$\langle 1, f(s)\rangle$	8	cs
$\langle 6, f(i)\rangle$	9	iri
$\langle 0, f(b)\rangle$	10	b
$\langle 9, f(\text{␣})\rangle$	11	iri␣
$\langle 10, f(o)\rangle$	12	bo
$\langle 0, f(j)\rangle$	13	j
$\langle 0, f(t)\rangle$	14	t
$\langle 0, f(o)\rangle$	15	o
$\langle 4, f(j)\rangle$	16	rj
$\langle 0, f(á)\rangle$	17	$á$
$\langle 0, f(n)\rangle$	18	n
$\langle 0, f(-)\rangle$	19	$-$
$\langle 0, f(l)\rangle$	20	l
$\langle 0, f(é)\rangle$	21	$é$
$\langle 20, f(e)\rangle$	22	le
$\langle 0, f(k)\rangle$	23	k
$\langle 0, f(\text{␣})\rangle$	24	␣
$\langle 20, f(é)\rangle$	25	$lé$
$\langle 0, f(p)\rangle$	26	p
$\langle 24, f(a)\rangle$	27	␣a
$\langle 24, f(l)\rangle$	28	␣l
$\langle 0, f(a)\rangle$	29	a
$\langle 13, f(t)\rangle$	30	jt
$\langle 15, f(r)\rangle$	31	or
$\langle 13, f(á)\rangle$	32	$já$
$\langle 18, \ \rangle$		

4.9. ábra. Az 4.15. példa LZ78 kódolásának menete

Megmutatható, hogy az LZ78 is aszimptotikusan optimális.

Az LZ78 algoritmus egyik hibája, hogy a szótár folyamatosan, korlát nélkül növekszik. A gyakorlatban egy bizonyos határon túl gátat szabunk a növekedésnek: vagy rendszeresen eltávolítjuk a felesleges illetve ritkán használt bejegyzéseket, vagy egy idő után fix szótárasként működik tovább az eljárás.

Az LZW algoritmus

Terry Welch az LZ78 módosításával egy olyan technikát dolgozott ki, amellyel megtakarítható az $\langle i, c\rangle$ párból a c karakterkód átküldése. Ez az ún. LZW algoritmus. A kódoló tehát csak szótárbeli indexeket küld át. Ehhez szükséges, hogy a szótárban már a kiinduló állapotban is szerepeljen az összes egybetűs szimbólum a forrásábécéből. A kódolás során az aktuális pozíciótól kezdve addig olvassuk be

a forrásszimbólumokat a pufferbe, amíg az *s* sorozatuk szerepel a szótárban. Ha az *a* karakter az első olyan, amelyre *sa* nincs benne a szótárban (az egymás után írással a konkatenációt jelöltük), akkor átküldjük az *s* sorozat indexét, az *sa* sorozatot felvesszük a szótárba és az *a* karaktertől kezdve folytatjuk az eljárást.

4.16. példa. Kódoljuk az LZW algoritmussal az idézetünket.

csiribiri␣csiribiri␣bojtorján–
lélek␣lép␣a␣lajtorján

Az egyszerűség kedvéért legyen a forrásábécé az idézetben ténylegesen előforduló karakterek halmaza, vagyis $\mathcal{X} = \{a, á, b, c, e, é, i, j, k, l, n, o, p, r, s, t, ␣, -\}$. Kezdetben ez a 18 bejegyzés szerepel a szótárban. (Általános esetben felvesszük a magyar ábécé összes betűjét és a legfontosabb írásjeleket, ami 64 bejegyzéssel, vagyis 6 bittel megoldható.) A kódoló először veszi a *c* karaktert. Ez benne van a szótárban, így hozzáilleszti a következő, az *s* karaktert. A *cs* sorozat már nem szerepel a szótárban, ezért átküldi a *c* indexét, vagyis a 4-et, felveszi a szótárba a *cs* sorozatot a 19. helyre, és megy tovább az *s*-sel kezdve. Az *s* szerepel a szótárban, így hozzáveszi az *i*-t. *si* nincs bent, tehát átküldi *s* indexét, a 15-öt, felveszi *si*-t, és folytatja az eljárást *i*-től, stb. A 4.10. ábrán látható táblázat tartalmazza a kódolás végeztével a szótárban található indexeket és karaktersorozatokat. A kódoló kimenete a következő:

$$4, 15, 7, 14, 7, 3, 21, 7, 17, 19, 25, 24, 22, 17, 3, 12, 8, 16, 12, 14,$$

$$8, 2, 11, 18, 10, 6, 10, 5, 9, 17, 43, 13, 17, 1, 48, 1, 35, 37, 39, 11$$

Az egyes lépéseket pedig bejelöltük a bemeneten:

c|s|i|r|i|b|ir|i|␣|cs|iri|bi|ri|␣|b|o|j|t|o|r|j|á|n|-|
l|é|l|e|k|␣|lé|p|␣|a|␣|l|a|jt|or|já|n

4.17. példa. Dekódoljuk a 4.16. példában kapott LZW algoritmussal tömörített $\mathcal{X} = \{a, á, b, c, e, é, i, j, k, l, n, o, p, r, s, t, ␣, -\}$ forrásábécé feletti szöveget. A kódoló kimenetéről a dekódoló bemenetére a $4, 15, 7, 14, 7, 3, 21, 7, 17, 19, \ldots$ sorozat jut el. A kiindulási szótár tartalmazza a forrásábécé betűit. Így a $4, 15$ sorozatot dekódoljuk *c* illetve *s* karakterként. A szótárba felvesszük a *cs* bejegyzést 19-nek, és a következő sorozat, amely a szótárba kerül majd, az *s* karakterrel fog kezdődni. A 7 kódszó érkezik a bemeneten, ezt dekódoljuk *i*-ként, a szótárban a 20-ik helyre felvesszük az *si*-t, és feljegyezzük, hogy a következő szótárba kerülő sorozat *i*-vel kezdődik majd. A 14 kódszót *r*-ként dekódoljuk, és *ir* kerül a szótárba, a 7-et *i*-ként, és *ri* kerül a szótárba, a 3-at pedig *b*-ként, *ib* kerül a szótárba, és *b*-vel fog kezdődni a következő bejegyzés. Ekkor a 21 kódszó következik, amelyet *ir*-ként dekódolunk. Először az *i* betűt illesztjük a készülő új szótárbejegyzés kezdő *b* betűjéhez. Mivel a *bi* sorozat nincs a szótárban, felvesszük azt 24-ként. A következő

index	bejegyzés	index	bejegyzés	index	bejegyzés
1	a	20	si	39	já
2	á	21	ir	40	án
3	b	22	ri	41	n-
4	c	23	ib	42	-l
5	e	24	bi	43	lé
6	é	25	iri	44	él
7	i	26	i␣	45	le
8	j	27	␣c	46	ek
9	k	28	csi	47	k␣
10	l	29	irib	48	␣l
11	n	30	bir	49	lép
12	o	31	ri␣	50	p␣
13	p	32	␣b	51	␣a
14	r	33	bo	52	a␣
15	s	34	oj	53	␣la
16	t	35	jt	54	aj
17	␣	36	to	55	jto
18	-	37	or	56	orj
19	cs	38	rj	57	ján

4.10. ábra. Az 4.16. példa szótára

új bejegyzés *i* betűvel fog kezdődni. Mivel az *ir* párnak csak a kezdő *i* betűjét használtuk fel, a maradék *r* betűt hozzáillesztjük a készülő új szótárbejegyzéshez, így *ir*-t kapunk. Ez már szerepel a szótárban, ezért tovább folytatjuk a dekódolást. Az első 24 bejegyzés készen van, míg az 25-ik éppen készülőfélben, a következő bemenet pedig a 7 kódszó, amelyet *i*-ként dekódolunk. Illesszük az *i* karaktert a készülőben lévő új bejegyzéshez. Mivel az így kapott *iri* még nem szerepel a szótárban, ez lesz a 25-ik bejegyzés. A következő bejegyzés *i* betűvel fog kezdődni, stb. A dekódoló által épített szótár természetesen megegyezik a kódoló szótárával, amely a 4.10. ábrán látható.

4.18. példa. Dekódoljuk az LZW algoritmussal tömörített, $\mathcal{A} = \{a, b\}$ forrásábécé feletti *abababab*... karaktersorozatot. A kódoló kimenetéről a dekódoló bemenetére az $1, 2, 3, 5, 4, 7, 6, 9, 8, \ldots$ sorozat jut el. A kiindulási szótár tartalmazza az *a* és a *b* bejegyzést. Így az $1, 2$ sorozatot dekódoljuk *a* illetve *b* karakterként. A szótárba felvesszük az *ab* bejegyzést harmadiknak, és a következő sorozat, amely a szótárba kerül majd, a *b* karakterrel fog kezdődni. A 3 kódszó érkezik a bemeneten, ezt dekódoljuk *ab*-ként. Először az *a* betűt illesztjük a készülő új szótárbejegyzés kezdő *b* betűjéhez. Mivel a *ba* sorozat nincs a szótárban, felvesszük azt. A következő új bejegyzés *a* betűvel fog kezdődni. Mivel az *ab* párnak csak a kezdő *a* betűjét használtuk fel, a maradék *b* betűt hozzáillesztjük a készülő új szótárbejegyzéshez,

így *ab*-t kapunk. Ez már szerepel a szótárban, ezért tovább folytatjuk a dekódolást. Az első 4 bejegyzés készen van, míg az 5. éppen készülőfélben, a következő bemenet pedig az 5 kódszó, amely a még nem teljesen kész bejegyzésre hivatkozik. Ennek ellenére tovább tudjuk folytatni a dekódolást! Ha ismernénk az 5. bejegyzést, annak első két karaktere *ab* lenne. Illesszük az *a* karaktert a készülőben lévő új bejegyzéshez. Mivel az így kapott *aba* még nem szerepel a szótárban, ez lesz az 5. bejegyzés. A következő bejegyzés *a* betűvel fog kezdődni, és még megmaradt a *ba* sorozat az előző dekódolásból, stb. Az alábbi táblázat tartalmazza a kódoló és a dekódoló által épített szótárt:

index	bejegyzés
1	*a*
2	*b*
3	*ab*
4	*ba*
5	*aba*

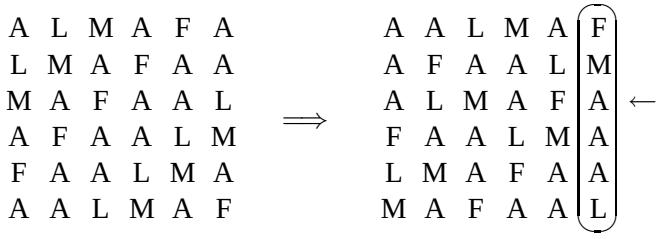
index	bejegyzés
6	*abab*
7	*bab*
8	*baba*
9	*ababa*
⋮	

4.9. Burrows–Wheeler-transzformáció

A tömörítendő adatok statisztikai elemzésével a széles körben elterjedt Lempel–Ziv-algoritmusoknál jobb tömörítési arány érhető el, de ez lényegesen nagyobb futási időt igényel. A következőkben egy olyan eljárást mutatunk be, amelynek tömörítési aránya jól megközelíti a statisztikai módszerekét, viszont a Lempel–Ziv-algoritmusokkal összemérhető sebességgel működik.

A Burrows és Wheeler által javasolt eljárás egy blokk-kódolás. Alapötlete az, hogy *invertálható módon* úgy rendezzük át a blokk szimbólumait, hogy az azonos karakterek nagy valószínűséggel egymás mellé kerüljenek, s így már egyszerű tömörítési eljárásokkal is jó hatásfokot érhetünk el. Minél nagyobb a blokkméret, annál jobb tömörítési arány érhető el. (Természetesen a műveletek memóriaigénye gátat szab a blokkméret határtalan növelésének.) Az eljáráshoz szükséges, hogy a lehetséges bemeneti szimbólumokon rendezést definiáljunk.

Vegyük a bemeneti szimbólumok egy m hosszú blokkját. Képezzük rendre az összes ciklikus eltoltját, és ezeket rendezzük egy táblázatba. (Valójában nem szükséges ezeket az eltoltakat fizikailag létrehoznunk, elegendő ha a bemeneti blokk memóriaterületének mutatóival dolgozunk.) Ennek az m soros táblázatnak a sorait rendezzük lexikografikusan, majd ebben a sorrendben olvassuk ki az utolsó oszlopot, vagyis a blokkok utolsó szimbólumát. Az így kapott, szintén m hosszú sorozatból helyreállítható az eredeti blokk, amennyiben még azt is tudjuk, hogy a rendezett táblázat hányadik sorában szerepel az eredeti blokk. A 4.11. ábrán egy $m = 6$ hosszú blokkra szemléltetjük a módszert (a gyakorlatban természetesen ennél lényegesen nagyobb blokkméretet célszerű választani). A transzformáció az ALMAFA sorozatból az FMAAAL sorozatot állítja elő, és az eredeti sorozat pedig a táblázat harmadik sorában jelenik meg.



4.11. ábra. A Burrows–Wheeler-transzformáció

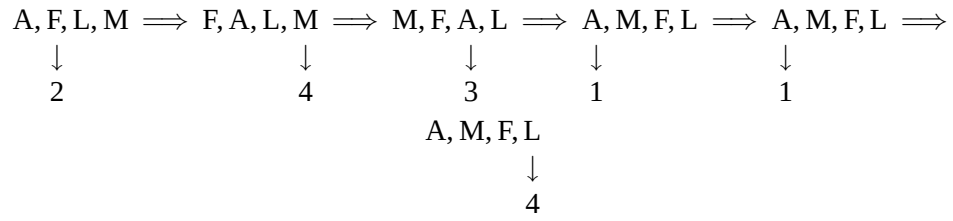
```
INPUT:  x[ ]={tömörítendő blokk}
        n={tömörítendő blokk hossza}

OUTPUT: L={rendezett táblázat utolsó oszlopa}
        i={az eredeti blokk sorszáma}

FOR i=0 to n-1
   FOR j=0 to n-1
      T[i,j]=x[(i+j) mod n]
   ENDFOR
ENDFOR
R=Lexikografikusan_Rendez(T)
FOR i=0 to n-1
   L[i]=T[i,n-1]
ENDFOR
i=0
WHILE {R i-edik sora} != {T i-edik sora}
   i=i+1
ENDWHILE
RETURN L,i
```

Az inverz transzformáció is nagyon egyszerűen működik. A rendezett táblázatnak azon sorát kell helyreállítanunk, amelyben az eredeti sorozat szerepel. Ennek sorszámát tudjuk. Ismerjük még a táblázat utolsó oszlopát, s ez rögtön megadja az első oszlopot is, hiszen abban szintén az utolsó oszlop karakterei szerepelnek, csak rendezett sorrendben. Ezután megkeressük, hogy a kérdéses sor első szimbóluma melyik sor utolsó elemeként szerepel, majd e sor első elemét írjuk a kérdéses sor második helyére, ugyanis a ciklikus eltolások miatt bármely sorban a karakterek sorrendje azonos. Általában pedig a kérdéses sor i-edik helyére azon sor első szimbólumát írjuk, amelynek utolsó helyén a kérdéses sor $i-1$-edik szimbóluma szerepel. Ha az utolsó oszlopban a keresett szimbólum többször is előfordul, akkor az annyiadik előfordulását vesszük, ahányadikként az első oszlopban szerepelt ezen szimbólumok közül.

Vegyük észre, hogy a bemeneti blokkot eddig még nem tömörítettük, hiszen a BW-transzformáció az eredetivel megegyező hosszúságú blokkot állít elő. A bemenet szimbólumai közötti függőségek miatt viszont a transzformált blokkban hosszú, azonos szimbólumokból álló sorozatok jelennek meg, amelyek jól tömö-



4.12. ábra. A „mozgasd az elejére” eljárás

ríthetők az ún. „mozgasd az elejére” (move to front) módszerrel. Kiindulásként vegyünk fel egy listát, amely a bemeneten előforduló szimbólumokat tartalmazza tetszőleges sorrendben. A szimbólumokat egyesével olvasva kódoljuk azokat a listában elfoglalt helyük sorszámával, és minden lépésben mozgassuk a lista elejére az aktuális szimbólumot (a többit eggyel hátrébb tolva). A BW-transzformáció által szolgáltatott speciális sorozat miatt azt várhatjuk, hogy kis sorszámok sorozatát eredményezi ez az algoritmus, amely jól tömöríthető például Huffman-kódolással.

```
INPUT:  L[ ]={BW-transzformált blokk}
        n={blokk hossza}
        F[ ]={forrásábécé}

OUTPUT: c[ ]={sorszámok sorozata}

FOR i=0 to n
   j=0
   WHILE F[j]!=L[i]
      j=j+1
   ENDWHILE
   c[i]=j
   tmp=F[j]
   FOR k=j to 1
      F[k]=F[k-1]
   ENDFOR
   F[0]=tmp
ENDFOR
RETURN c
```

Alkalmazzuk a 4.11. ábra példájára a mozgasd az elejére módszert. Az egyszerűség kedvéért a kezdeti listán csak a blokkban ténylegesen előforduló szimbólumok szerepelnek, mégpedig ábécé sorrendben. A lista alakulása és a kimeneten megjelenő sorszámok a 4.12. ábrán nyomonkövethetők. Az FMAAAL sorozatból a $2,4,3,1,1,4$ sorozatot kapjuk.

4.10. Alkalmazások

Faxkódolás

A 4.6. példa átmenetvalószínűségei jellegzetesek egy túlnyomóan szöveget tartalmazó (pl. üzleti) dokumentum esetén. Megfigyelhető, hogy a következő képpont színe sokkal nagyobb valószínűséggel lesz azonos az előzőével, mint eltérő (különösen világos esetében). Ahelyett, hogy a képpontok színét külön-külön kódolnánk, kódoljuk egyszerűen az azonos színű képpontok (vagyis a futamok) hosszát, tehát azt a hosszt, amíg a Markov-lánc azonos állapotában maradunk. Ezt a technikát **futamhossz kódolás**nak nevezzük. (A futamhossz kódolás optimalitására vonatkozóan lásd a 4.14. feladatot.) Például, ha 190 világos pixelt 30 sötét követ, majd 210 világos jön, a 430 képpont egyenkénti kódolása helyett a $190, 30, 210$ sorozatot fogjuk kódolni, valamint jeleznünk kell azt, hogy az első pontsorozat milyen színű volt.

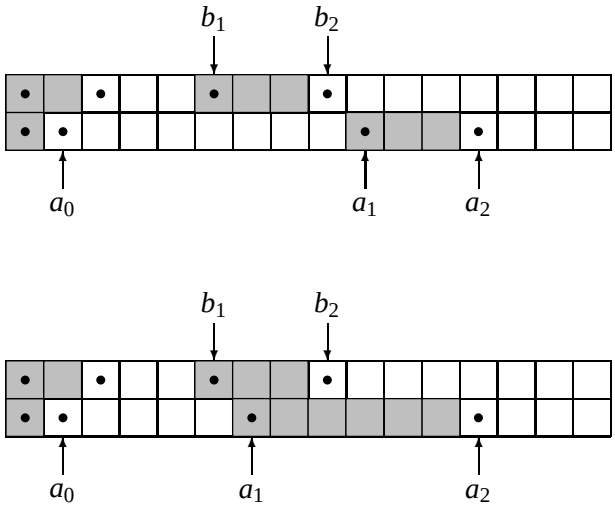
Futamhossz kódolást alkalmaznak az ITU-T fax-szabványaiban is. Számunkra a Group 3 illetve Group 4 ajánlások érdekesek, ugyanis a korábbi Group 1 és 2 technikák csak analóg módszereket használtak, ennélfogva nem tömörítettek.

Az 1980-ban megjelent Group 3 szabvány egydimenziós futamhossz kódolással dolgozik. Ez azt jelenti, hogy az egymás alatti vízszintes sorokat egymástól függetlenül kódolja, a futamok pedig az egy soron belül váltakozó fehér és fekete képpontokból állnak. Minden sor első futama fehér képpontokból áll; ha egy sor fekete pixellel kezdődik, akkor az első futamot egy 0 hosszúságú fehér futamnak kell tekinteni. A különböző hosszúságú futamok eltérő valószínűséggel fordulnak elő egy dokumentumban, ezért ezeket változó szóhosszúságú kóddal, mégpedig a szabvány szerint Huffman-kóddal kódolják. A futamok összhossza, vagyis egy sor hosszúsága 1728 képpont. Ez túl sok lehetséges futamhosszt eredményez, nincs értelme ilyen nagyméretű kódot alkalmazni. Ezért a h hosszt 1 vagy 2 jegyű, 64-es számrendszerben felírt (vagyis 6 bites) számként kódolják:

$$h = 64m + t \qquad t = 0, 1, \ldots, 63 \qquad m = 0, 1, \ldots, 26$$

Külön kódtáblázat vonatkozik az m, vagyis a kiegészítő kód (make up code, MUC) illetve a t, vagyis a befejező kód (terminating code, TC) értékeire, külön a fekete és külön a fehér képpontok esetére. Egy futam kódját a színnek megfelelő MUC illetve TC táblázatból kiolvasott kódszavak konkatenációja adja. Azonos színhez tartozó MUC és TC táblázatok prefix tulajdonságúak (együttesen is). Mivel a fekete és fehér futamok mindig váltakozva szerepelnek, ezért a fekete táblázatokban álló kódszavak lehetnek a fehér táblázatokban állók prefixei és fordítva. A sor végét a speciális EOL (end of line, sorvége) kódszó jelöli: 000000000001. Ez biztosítja az adó és vevő közötti szinkronizációt is.

1984-ben publikálta a ITU-T a Group 4 ajánlást, amely kihasználja a függőleges irányú redundanciát is, emellett felülről kompatíbilis a Group 3-mal. Egy soron



4.13. ábra. Az a_0, a_1, a_2, b_1, b_2 mutatók lehetséges elhelyezkedése

belül a futamokat nem csak a már megismert módon, a futamhosszak felsorolásával kódolhatjuk, hanem a színátmenetek helyének pozícióival is.

A kétdimenziós kódolás megértéséhez vezessük be az alábbi jelöléseket:

a_0: Az utolsó pixel, amelynek értékét a kódoló és a dekódoló egyaránt ismeri. Egy sor kódolásának megkezdésekor az a_0 egy képzeletbeli fehér képpontra mutat, amely az első aktuális pixel bal oldalán áll.

a_1: Az első színátmenetet jelentő képpont a_0 jobb oldalán. a_1 színe ellentétes a_0 színével. a_1 helye csak a kódoló számára ismert.

a_2: Az második színátmenetet jelentő képpont a_0 jobb oldalán. a_2 színe ellentétes a_1 színével, ami azt jelenti, hogy megegyezik a_0 színével.

b_1: Az első színátmenetet jelentő képpont az aktuálisan kódolandó sort megelőző sorban a_0 jobb oldalán, amelynek színe ellentétes a_0 színével. Mivel a megelőző sor és a_0 értéke ismert a kódoló és a dekódoló számára is, ezért b_1 helye is ismert mindkettőjük előtt.

b_2: Az első színátmenetet jelentő képpont az aktuálisan kódolandó sort megelőző sorban b_1 jobb oldalán.

A Group 4 algoritmusa az első sort ugyanúgy kódolja, mint a Group 3 esetében láttuk. A további sorok kódolásához felhasználja az azt megelőzőt, így a másodikhoz az elsőt, a harmadikhoz a másodikat, stb. A 4.13. ábra azt a helyzetet ábrázolja, amikor éppen a második sornál tartunk a feldolgozásban, és a képpontokat a második pixelig már feldolgoztuk. A színátmenetet jelentő pixeleket ponttal jelöltük. Két esetet kell megkülönböztetnünk:

Ha b_1 és b_2 a_0 és a_1 között fekszik, a kódoláshoz az átadó (pass) módot használjuk. A kódoló egy speciális kódszó kiküldésével értesíti a dekódolót erről. Ebből a dekódoló tudja, hogy az a_0-tól a b_2 alatti pixelig a képpontok színe azonos. Ha ez nem lenne igaz, akkor közben lenne egy színátmenetet jelentő képpont, vagyis

a_1 és b_2 viszonyára nem lenne igaz a feltételünk. Ekkor a kódoló és a dekódoló által egyaránt ismert legutolsó képpont a b_2 által mutatott lesz. Így ez lesz az a_0 új helye, a másik négy mutató új pozícióját pedig a már ismert módon jelöljük ki.

Ha a_1 megelőzi b_2-t, ismét két eset lehetséges: Ha a_1 és b_1 távolsága nem nagyobb 3-nál, elküldjük ezt a távolságértéket (ezt függőleges módnak nevezzük). (A kódszó természetesen függ attól, hogy a_1 jobbra vagy balra van-e b_1-től, így itt összesen 7 eset lehetséges azt is beleértve, hogy a_1 éppen b_1 alatt helyezkedik el.) a_0-t a_1-re állítjuk, módosítjuk a másik négy mutatót is, és folytatjuk a kódolást az algoritmus elejétől. Ha a_1 és b_1 távolsága nagy, lényegében visszatérünk az egydimenziós technikához. Egy speciális kódszóval jelezzük a dekódolónak, hogy vízszintes módban vagyunk, majd elküldjük a_0 és a_1 illetve a_1 és a_2 távolságát Huffman-kódolva. a_0-t a_2 helyére állítjuk, és aktualizáljuk a másik négy mutatót is. A kódolást az algoritmus elejétől folytatjuk.

A kétdimenziós algoritmus használatával egy sor kódolása a megelőző soron alapul, így elképzelhető, hogy egy sorban bekövetkező hiba kiterjed a többire is. Ezt megelőzendő, a szabvány rögzíti, hogy minden egydimenziós eljárással kódolt sort normál függőleges felbontás esetén legfeljebb 1, nagy felbontás esetén legfeljebb 3 kétdimenziós algoritmussal kódolt sor követhet.

A sorvégeket a Group 3 szabványban megismert EOL szimbólum jelzi, azonban attól függően, hogy vízszintes módban, vagyis az egydimenziós Group 3 szerint kódoltuk a sort, vagy az új kétdimenziós módszerrel, egy 1 illetve 0 bit követi. Ez lehetővé teszi, hogy mindkét módszerrel elvégezve a kódolást az adó a rövidebb hosszúságot eredményezőt választhassa ki, majd a megfelelő EOL jellel jelezze a vevőnek, hogy melyik algoritmust használta.

Tömörítő programok

Bár az LZ77 algoritmus aszimptotikusan optimális, a gyakorlatban számos továbbfejlesztése ismeretes, amelyek célja a hatékonyság növelése. Például a népszerű PKZIP és ARJ tömörítőkben a $\langle t,h,c\rangle$ hármasokat nem fix, hanem változó hosszúságú kóddal kódolják. Egy másik variáció változtatható méretű kereső és előretekintő ablakot használ. Az LZ77 legegyszerűbb módosítása annak kiküszöbölése, amikor egyetlen karaktert kódolunk egy hármassal. Ez egy jelzőbittel oldható meg. Ezzel jelezzük, hogy nem egy hármast, hanem csak egy kódszót küldünk át. Szintén az LZ77 algoritmuson alapul a Unix GZIP tömörítője, a WINZIP, az LHARC, a PNG (Portable Network Graphics) képtömörítő formátum és a PDF dokumentumformátum belső tömörítése is.

A Unix COMPRESS parancsa és a GIF (Graphics Interchange Format) képtömörítő eljárás az LZW algoritmust használja, mégpedig adaptív szótárérettel. A COMPRESS esetében kezdetben a szótárban 512 bejegyzésnek van hely, ez azt jelenti, hogy a kódszavak 9 bit hosszúak. Amikor a szótár betelik, méretét megduplázzuk, 1024 bejegyzésre. Ettől kezdve 10 bites kódszavakat viszünk át, és így tovább. A kódszavak lehetséges maximális hosszát a felhasználó állíthatja be 9

és 16 bit között (alapértelmezés: 16 bit). Ezt elérve, a COMPRESS eljárás statikus szótár alapú technikává válik. Ilyenkor a program figyeli a tömörítési arányt. Amennyiben ez egy bizonyos küszöb alá esik, a szótár már nem felel meg céljainknak, ezért a szótárépítő folyamat kezdődik elölről. Így a szótár mindig tükrözi a forrás lokális jellemzőit.

Szintén az LZW algoritmusra épül az ITU-T V.42bis tömörítési szabványa, amely a telefonhálózaton modemmel történő adatátvitelről szóló V.42 ajánlás kiegészítése. Az algoritmus két üzemmódot definiál. Az egyik a transzparens mód, amikor az adatok tömörítetlen formában kerülnek átvitelre, a másik pedig a tömörített mód. A két üzemmódra azért van szükség, mert lehetséges, hogy az átvitelre kerülő adatokban nincs redundancia, ezért nem tömöríthető az LZW algoritmussal. Ebben az esetben a tömörítő eljárás még hosszabb kimenetet eredményezne, mint a bemenet (ez a helyzet például, ha egy előzőleg már tömörített fájlt akarunk átvinni a telefonvonalon). Tömörített üzemmódban a rendszer LZW algoritmust használ változó méretű szótárral. A szótár kezdeti méretét a kapcsolat létrejöttekor egyezteti az adó- és a vevőberendezés. A V.42bis ajánlás minimum 512 bejegyzés méretű szótárat tartalmaz, de 2048 méretűt tart ideálisnak. Az összes bejegyzés nem használható fel szabadon, mert van 3 kitüntetett szerepű kódszó. Ezek illetve jelentésük a következő: Enter Transparent Mode (üzemmódváltás: ettől kezdve a transzparens mód érvényes), Flush Data (a szótárépítést előről kezdjük), Increment Codeword Size (megduplázzuk a szótár méretét, s ezzel együtt eggyel növeljük a kódszavak méretét is). Az adatátvitel során bekövetkező hibák hatásának csökkentésére az ajánlás meghatározza a maximális sztringméretet, amely szerepelhet a szótárban. Ezt 6 és 250 között az adó- és a vevőberendezés határozza meg a kapcsolat felépítésekor (alapértelmezés: 6).

A Burrows–Wheeler-transzformáción alapul többek között a BZIP tömörítőprogram.

4.11. Feladatok

4.1. feladat. Nevezzünk egy $f : \mathcal{X} \to \mathcal{Y}^*$ kódot egyértelműen dekódolhatónak, ha az $\mathbf{u} = u_1 \cdots u_k$ és $\mathbf{v} = v_1 \cdots v_k$ üzenetekre (itt $u_1, \ldots, u_k, v_1, \ldots, v_k \in \mathcal{X}$)

$$f(u_1)f(u_2)\cdots f(u_k) = f(v_1)f(v_2)\cdots f(v_k)$$

esetén $u_i = v_i$ minden i-re. Tehát a 4.1. definícióval ellentétben csak azt követeljük meg, hogy bármely két különböző, azonos hosszúságú üzenet kódja is különbözzön. Bizonyítsa be, hogy a két definíció ekvivalens!

4.2. feladat. Legyen az $\mathcal{X} = \{x_1, \ldots, x_n\}$ forrásábécén adott valószínűség-eloszlás olyan, hogy minden egyes elem valószínűsége 2^{-i} alakú, ahol i egy pozitív egész szám. Bizonyítsa be, hogy ilyen esetekben a bináris Shannon–Fano-kód optimális! Mutassa meg, hogy a bináris Huffman-kód átlagos kódszóhossza akkor és csak akkor egyezik meg az entrópiával, ha az eloszlás ilyen alakú!

4.3. feladat. Legyen az $\mathcal{X}$ forrásábécé ötelemű, a következő valószínűségekkel: $0.4; 0.35; 0.1; 0.1; 0.05$. Mennyi az eloszlás entrópiája? Konstruálja meg a bináris Shannon–Fano-kódot erre az eloszlásra, illetve konstruáljon bináris prefix kódot az $l_i = \lceil -\log p(x_i) \rceil$ kódszóhosszakkal. Mekkora az átlagos kódszóhossz?

4.4. feladat. Egy szabályos pénzérmét addig dobunk fel, amíg írást nem kapunk. Jelölje az X valószínűségi változó a dobások számát. Mennyi az X entrópiája?

4.5. feladat. Tekintsük a

$$\mathbf{p} = (p_1, \ldots, p_i, \ldots, p_j, \ldots, p_n)$$

illetve

$$\mathbf{q} = (p_1, \ldots, \frac{p_i + p_j}{2}, \ldots, \frac{p_i + p_j}{2}, \ldots, p_n)$$

eloszlásokat. Mutassa meg, hogy a $\mathbf{p}$ eloszlás entrópiája nem lehet nagyobb, mint a $\mathbf{q}$ eloszlás entrópiája! Mutassa meg, hogy a $\mathbf{p}$ eloszláshoz tartozó optimális (minimális átlagos kódszóhosszúságú) kód átlagos kódszóhossza nem lehet nagyobb, mint a $\mathbf{q}$ eloszláshoz tartozó optimális kód átlagos kódszóhossza!

4.6. feladat. Bizonyítsa be, hogy ha X és Y független valószínűségi változók, akkor

$$H(X,Y) = H(X) + H(Y).$$

4.7. feladat (Információs divergencia). Definiáljuk a $\mathbf{p} = (p_1, \ldots, p_n)$ és a $\mathbf{q} = (q_1, \ldots, q_n)$ valószínűségi eloszlás közötti „információs távolságot” a

$$D(\mathbf{p} \mid \mathbf{q}) = \sum_{i=1}^{n} p_i \log \frac{p_i}{q_i}$$

kifejezéssel. (A mennyiséget gyakran információs divergenciának, relatív entrópiának, vagy Kullback–Leibler-távolságnak nevezik.) Lássuk be, hogy bármely két eloszlásra $D(\mathbf{p} \mid \mathbf{q}) \geq 0$, és egyenlőség pontosan akkor teljesül, ha $\mathbf{p} = \mathbf{q}$.

4.8. feladat. Bizonyítsa be, hogy $H(\mathbf{p}) = \log n - D(\mathbf{p} \mid \mathbf{u})$, ahol $H(\mathbf{p})$ jelöli a $\mathbf{p}$ eloszlás entrópiáját, $\mathbf{u}$ pedig az egyenletes eloszlást az $\{1, \ldots, n\}$ halmazon.

4.9. feladat. Bizonyítsa be a 4.2. tétel a) részét!

4.10. feladat. Bizonyítsa be a 4.2. tétel b) részét!

4.11. feladat (Shannon–Fano- és Huffman-kód). Legyen az X valószínűségi változó eloszlása $\left(\frac{1}{3}; \frac{1}{3}; \frac{1}{4}; \frac{1}{12}\right)$. Konstruáljon Huffman-kódot ehhez az eloszláshoz. Mutassa meg, hogy két különböző optimális kód is van, azaz, hogy az $(1;2;3;3)$, és a $(2;2;2;2)$ kódszóhosszúságokkal adott mindkét kód optimális. Vonja le a következtetést, hogy létezik olyan optimális kód, amelynek van a megfelelő Shannon–Fano-kódénál hosszabb kódszava is.

4.12. feladat (Huffman-kód kódszóhosszai). Tegyük fel, hogy a $(p_1,\dots,p_n)$ eloszláshoz készítünk optimális bináris prefix kódot, ahol $p_1 > p_2 > \cdots > p_n > 0$. Bizonyítsa be, hogy

a) Ha $p_1 > \frac{2}{5}$, akkor a hozzá tartozó kódszó egy hosszúságú;

b) Ha $p_1 < \frac{1}{3}$ akkor a hozzá tartozó kódszó legalább kettő hosszúságú.

4.13. feladat. Legyen $X_1, X_2, \dots$ egy független, azonos eloszlású, bináris valószínűségiváltozó-sorozat, amelyre $\mathbf{P}\{X_i = 1\} = 10^{-6}$. Adja meg a sorozat egy olyan változó szóhosszúságú blokk-kódját, melynek betűnkénti átlagos kódszóhossza kisebb, mint $\frac{1}{100}$. Legyen a blokkméret $m = 1024$.

4.14. feladat (Futamhossz kódolás). Legyenek $X_1,\dots,X_n$ bináris valószínűségi változók. Jelölje $\mathbf{R} = (R_1,\dots,R_k)$ az egyes szimbólumok előfordulásainak futamhosszait az n hosszon. Tehát például az 1110010001111 sorozathoz $\mathbf{R} = (3,2,1,3,4)$ tartozik. Hogyan viszonyul egymáshoz $H(X_1,\dots,X_n), H(\mathbf{R})$ és $H(\mathbf{R},X_n)$?

4.15. feladat (Bináris entrópiafüggvény tulajdonságai). Legyen a $[0,1]$ intervallumon értelmezett h függvény (bináris entrópiafüggvény) értéke

$$h(x) = -x\log x - (1-x)\log(1-x),$$

ha $x \in (0,1)$, és $h(0) = h(1) = 0$. Mutassuk meg, hogy h rendelkezik a következő tulajdonságokkal:

a) szimmetrikus az $\frac{1}{2}$ pontra, és $h\left(\frac{1}{2}\right) = 1$;

b) $[0,1]$ minden pontjában folytonos;

c) $\left(0,\frac{1}{2}\right)$-ben szigorúan monoton növekvő;

d) $(0,1)$-ben szigorúan konkáv.

4.16. feladat. Adjuk meg a 36 darab a betűből álló karaktersorozat LZ78-kódját!

4.17. feladat. Legyen X egy bájt értékű valószínűségi változó úgy, hogy

$$\mathbf{P}\{X = i\} = p_i, \quad i = 0,1,\dots,255,$$

és Y egy bájt értékű, egyenletes eloszlású valószínűségi változó. Ha X és Y függetlenek, akkor számítsa ki az $(X+Y) \mod 256$ entrópiáját!

4.18. feladat. A bajnoki döntő az A és B csapat között addig tart, amíg az egyik meg nem nyer két meccset. Jelölje X valószínűségi változó a döntő mérkőzéseinek győzteseit (tehát X lehetséges értékei $AA, ABA, BAA, BB, BAB, ABB$), és Y a lejátszott összes meccs számát, azaz Y értéke 2 vagy 3. Feltéve, hogy a két csapat egyformán erős és a meccsek függetlenek, határozza meg $H(X)$ és $H(Y)$ értékét!

4.19. feladat. Tegyük fel, hogy egy **p** eloszláshoz tartozó Shannon–Fano-kód leghosszabb kódszava $l_{\max}^{\mathrm{SF}}$ hosszúságú, és ugyanezen eloszlás Huffman-kódjának leghosszabb kódszava $l_{\max}^{\mathrm{H}}$ hosszúságú. Mutasson olyan eloszlást, amelyre $l_{\max}^{\mathrm{SF}} > l_{\max}^{\mathrm{H}} + 800$.

4.20. feladat. Legyen a forrásábécé $\mathcal{X} = \{A, B, Z, Y\}$. Az adaptív Huffman-algoritmus használatával adja meg egy bináris kódját a BABY szónak! Minden lépésben adja meg a Huffman-fát a súlyokkal!

4.21. feladat (Maximális entrópia). Tekintsük a pozitív egész számok halmazára koncentrálódó, m várható értékű diszkrét eloszlásokat. Mutassa meg, hogy a geometriai eloszlásnak maximális az entrópiája.

4.22. feladat. Milyen kódolási következménye van a 4.21. feladatnak?

4.23. feladat. Alkalmazzuk a Burrows–Wheeler-transzformációt a KALEVALA szóra mint blokkra.

4.12. Megoldások

4.1. megoldás. A 4.1. definícióból triviálisan következik a feladat definíciójának teljesülése, vagyis az azonosság igazolásához a másik irányt kell belátnunk. Indirekt bizonyítást alkalmazunk. Tegyük fel, hogy az állítás nem igaz, vagyis létezik olyan kód, amely megfelel a feladat definíciójának, viszont nem tesz eleget a 4.1. definíciónak. Ekkor léteznie kell két különböző $\mathbf{u} = u_1, \ldots, u_k$ és $\mathbf{v} = v_1, \ldots, v_m$ üzenetnek ($\mathbf{u} \neq \mathbf{v}$, $k \neq m$), amelyek kódszavai megegyeznek $f(\mathbf{u}) = f(u_1) \cdots f(u_k) = f(v_1) \cdots f(v_m) = f(\mathbf{v})$. Kódoljuk most a két üzenet mindkét lehetséges sorrendű konkatenáltját, vagyis **uv**-t és **vu**-t.

$$f(\mathbf{uv}) = f(u_1) \cdots f(u_k) f(v_1) \cdots f(v_m) = f(v_1) \cdots f(v_m) f(u_1) \cdots f(u_k) = f(\mathbf{vu}).$$

Bár **uv** és **vu** hossza megegyezik ($k+m$), kódszavaik mégis azonosak, ellentmondva ezzel a feladat definíciójának, amelyből kiindultunk. Indirekt feltevésünk hamisnak bizonyult, így a feladat definíciója valóban ekvivalens a 4.1. definícióval.

4.4. megoldás. Az X valószínűségi változó geometriai eloszlású $\frac{1}{2}$ paraméterrel, vagyis

$$\mathbf{P}\{X = k\} = \frac{1}{2}\left(\frac{1}{2}\right)^{k-1} = 2^{-k}, \qquad k \in \mathbb{Z}^+.$$

Az entrópia definíciója szerint

$$H(X) = -\sum_{k=1}^{\infty} \mathbf{P}\{X = k\} \log \mathbf{P}\{X = k\} = -\sum_{k=1}^{\infty} 2^{-k} \log 2^{-k} = \sum_{k=1}^{\infty} k 2^{-k} = 2.$$

4.5. megoldás. Az entrópia definíciója szerint

$$H(\mathbf{p}) = -\sum_{k=1}^{n} p_k \log p_k = -\sum_{\substack{k=1\\k\neq i\\k\neq j}}^{n} p_k \log p_k - p_i \log p_i - p_j \log p_j,$$

$$H(\mathbf{q}) = -\sum_{\substack{k=1\\k\neq i\\k\neq j}}^{n} p_k \log p_k - 2\,\frac{p_i+p_j}{2} \log \frac{p_i+p_j}{2}.$$

Ennek felhasználásával és az $f(x) := -x\log x$ bevezetésével

$$\begin{aligned} H(\mathbf{p}) &\overset{?}{\leq} H(\mathbf{q}) \\ -p_i \log p_i - p_j \log p_j &\overset{?}{\leq} -2\,\frac{p_i+p_j}{2} \log \frac{p_i+p_j}{2} \\ \frac{f(p_i)+f(p_j)}{2} &\overset{?}{\leq} f\left(\frac{p_i+p_j}{2}\right), \end{aligned}$$

ami viszont éppen az $f(\cdot)$ függvény konkávitását jelenti. Mivel $f''(x) = -\frac{1}{\ln 2}\,\frac{1}{x} \leq 0$ minden $x \geq 0$-ra, az állítás igaz, hiszen a p_i és p_j valószínűségek pozitívak.

4.6. megoldás.

$$\begin{aligned} H(X,Y) &= -\sum_{x,y} p(x,y) \log p(x,y) = \\ &= -\sum_{x}\sum_{y} p(x)p(y) \log p(x)p(y) = \\ &= -\sum_{x}\sum_{y} p(x)p(y) \log p(x) - \sum_{x}\sum_{y} p(x)p(y) \log p(y) = \\ &= -\sum_{x} p(x) \log p(x) - \sum_{y} p(y) \log p(y) = \\ &= H(X) + H(Y), \end{aligned}$$

ahol kihasználtuk az X és Y változók függetlenségét, hiszen ekkor $p(x,y) = p(x)p(y)$.

4.7. megoldás.

$$\begin{aligned} D(\mathbf{p} \mid \mathbf{q}) &= \sum_{i=1}^{n} p_i \log \frac{p_i}{q_i} = \\ &= -\sum_{i=1}^{n} p_i \log \frac{q_i}{p_i} \geq \\ &\geq -\sum_{i=1}^{n} p_i \frac{1}{\ln 2}\left(\frac{q_i}{p_i} - 1\right) = \\ &= -\frac{1}{\ln 2}\sum_{i=1}^{n} (q_i - p_i) = \\ &= 0, \end{aligned}$$

ahol kihasználtuk a $\log x \leq \frac{1}{\ln 2}(x-1)$ egyenlőtlenséget.

4.8. megoldás.

$$\begin{aligned}\log n - D(\mathbf{p} \mid \mathbf{u}) &= \log n - \sum_{i=1}^{n} p_i \log \frac{p_i}{u_i} = \\ &= \log n - \sum_{i=1}^{n} p_i \log \frac{p_i}{\frac{1}{n}} = \\ &= \log n - \sum_{i=1}^{n} p_i (\log n + \log p_i) = \\ &= -\sum_{i=1}^{n} p_i \log p_i = \\ &= H(\mathbf{p}).\end{aligned}$$

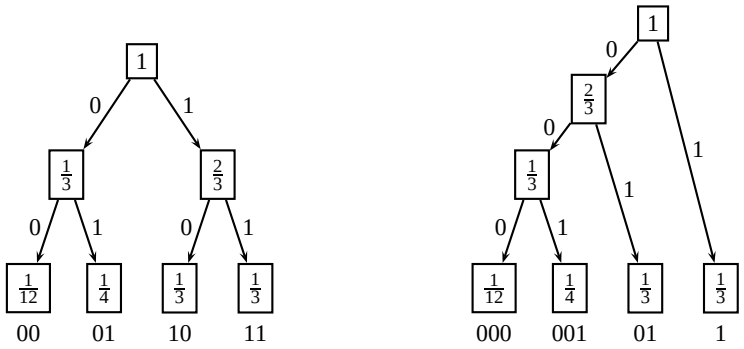
4.9. megoldás. Az, hogy a $H(X)$ entrópia nemnegatív, egyszerűen következik a definícióból. A felső korlát igazolásához pedig kombináljuk a 4.7. és a 4.8. feladatok állításait. Ha az X valószínűségi változó eloszlása $\mathbf{p}$, akkor

$$H(X) = H(\mathbf{p}) = \log n - D(\mathbf{p} \mid \mathbf{u}) \leq \log n,$$

ahol kihasználtuk, hogy az információs divergencia nemnegatív.

4.10. megoldás. Alkalmazza a 4.7. feladat eredményét úgy, hogy $\mathbf{p}$ legyen az X,Y együttes eloszlása, $\mathbf{q}$ pedig az X és Y szorzateloszlása.

4.11. megoldás. A Huffman-fa előállításához az első lépésben összevonjuk az $\frac{1}{4}$ és az $\frac{1}{12}$ valószínűségű szimbólumokat, s ezzel egy új, $\frac{1}{3}$ valószínűségű csúcsot kapunk. A második lépésben három darab $\frac{1}{3}$ valószínűségű csúcs közül kell kettőt összevonnunk. Attól függően, hogy ezek között szerepel-e az előző lépésben kapott csúcs vagy sem, két különböző fát kapunk eredményül:



Mindkét kód optimális, kódszóhosszuk 2. A második kódnak van 3 hosszú kódszava is, míg a Shannon–Fano-kódnak nincs, amint azt mindjárt látni fogjuk. Ehhez a kumulált valószínűségek, bináris alakjuk és az ebből leolvasható kódszavak

a következők:

$$\begin{aligned} w_1 = 0 &= 0.00000000_2 &\Longrightarrow\quad & 00 \\ w_2 = \tfrac{1}{3} &= 0.01010101\ldots_2 &\Longrightarrow\quad & 01 \\ w_3 = \tfrac{2}{3} &= 0.10101010\ldots_2 &\Longrightarrow\quad & 10 \\ w_4 = \tfrac{11}{12} &= 0.11101010\ldots_2 &\Longrightarrow\quad & 11 \end{aligned}$$

4.12. megoldás.

a) Indirekt tegyük fel, hogy a p_1-hez tartozó kódszó egynél hosszabb. Ez csak úgy lehet, ha a Huffman-fa konstruálásánál a p_1 csúcsot nem az utolsó lépésben vonjuk össze egy másik csúccsal. Legyen ez a másik csúcs p_j. Ebben a közbülső összevonási lépésben kell, hogy legyen egy p_1-nél nagyobb valószínűségű csúcs (máskülönben a p_1-hez tartozó csúcsot ebben a lépésben nem vonnánk össze a p_j csúccsal), legyen ez p_i, amelyre tehát $p_i > p_1 > \frac{2}{5}$. A valószínűségek összegének 1-et kell adnia, így a $p_j \le 1 - p_1 - p_i < 1 - 2 \cdot \frac{2}{5}$-nek is teljesülnie kell. A p_i csúcs nem lehet levél, azaz nem tartozhat szimbólumhoz (hiszen ekkor nem p_1 lenne a legvalószínűbb szimbólum), vagyis egy korábbi lépés során összevonással keletkezett p_{i1} és p_{i2} csúcsokból. Az utóbbi két valószínűségre $p_j > p_{i1}$ és $p_j > p_{i2}$ teljesül, máskülönben nem a p_{i1}-hez és p_{i2}-höz tartozó csúcsokat vontuk volna össze. Az eddigieket összevetve:

$$\tfrac{2}{5} < p_1 < p_i = p_{i1} + p_{i2} < p_j + p_j < 2\left(1 - 2 \cdot \tfrac{2}{5}\right) = \tfrac{2}{5}$$

ellentmondásra jutottunk, vagyis hamis az indirekt feltevésünk.

b) Indirekt tegyük fel, hogy a p_1-hez tartozó kódszó egy hosszú. Ez csak úgy lehet, ha a Huffman-fa konstruálásánál a p_1 csúcsot az utolsó lépésben vonjuk össze egy másik csúccsal. Legyen ez a másik csúcs p_j. A valószínűségek összegének 1-et kell adnia, így $p_j = 1 - p_1 > 1 - \frac{1}{3} = \frac{2}{3}$. A p_j csúcs nem lehet levél, azaz nem tartozhat szimbólumhoz (hiszen ekkor nem p_1 lenne a legvalószínűbb szimbólum), vagyis egy korábbi lépés során összevonással keletkezett p_{j1} és p_{j2} csúcsokból. Az utóbbi két valószínűség egyike, mondjuk $p_{j1} > \frac{p_j}{2} > \frac{1}{3}$. Ezzel viszont ellentmondásra jutottunk, hiszen ekkor $p_{j1} > p_1$, vagyis ebben a lépésben biztosan nem p_{j1}-et vontuk volna össze, tehát hamis az indirekt feltevésünk.

4.13. megoldás. Egy 1024 hosszú blokkban lévő 1-esek száma $(1024, 10^{-6})$ paraméterű binomiális eloszlást követ, vagyis nagy valószínűséggel nem esik 1-es egy blokkba:

$$\mathbf{P}\{\text{nincs 1-es a blokkban}\} = \left(1 - 10^{-6}\right)^{1024} \approx 0.99898$$

Így elegendően kis átlagos kódszóhosszúságú kódot kapunk, ha a csupa 0 blokkot 1 biten, például a 0-val kódoljuk, a többi üzenetet pedig nem tömörítjük, hanem egy 1-es után változatlanul átküldjük. Ez a kód egyértelműen dekódolható, mert ha egy kódszó 0-val kezdődik, akkor azt a csupa 0 blokként dekódoljuk. A többi

kódszó pedig 1-essel kezdődik és 1025 bit hosszú, dekódolásuk pedig az 1-es után álló 1024 bit változatlanul hagyásával történik. A kód átlagos kódszóhossza:

$$\frac{1}{m}\mathbf{E}|f(\mathbf{X})| = \frac{1}{1024}\left(1\cdot\left(1-10^{-6}\right)^{1024}+1025\cdot\left(1-\left(1-10^{-6}\right)^{1024}\right)\right)$$
$$\approx 2\cdot 10^{-3} < \frac{1}{100}.$$

4.14. megoldás. Létezik egy invertálható g függvény úgy, hogy

$$(\mathbf{R}, X_n) = g(X_1,\dots,X_n),$$

ugyanis X_n-ből és $\mathbf{R}$-ből $(X_1,\dots,X_n)$ visszaállítható. A 4.2. tétel c) része miatt

$$H(X_1,\dots,X_n) = H(\mathbf{R}, X_n)$$

és

$$H(\mathbf{R}) < H(X_1,\dots,X_n).$$

4.15. megoldás.

a) Ez közvetlenül a definícióból következik, ugyanis egy $[0,1]$-beli x pont $\frac{1}{2}$-re vett tükörképe éppen $1-x$, és $h(1-x)=h(x)$.

b) Mivel a $h(x)$ függvény a $(0,1)$-ben folytonos függvények összetételével kapható, ezért ott maga is folytonos. A végpontokban pedig:

$$\lim_{x\to 0} h(x) = \lim_{x\to 0}\left(-x\log x-(1-x)\log(1-x)\right) = \lim_{x\to 0} -\frac{\log x}{\frac{1}{x}} \overset{\text{L'H}}{=} \lim_{x\to 0} -\frac{\frac{1}{x}}{-\frac{1}{x^2}} =$$
$$= \lim_{x\to 0} x = 0 = h(0),$$

ahol a L'Hospital-szabályt alkalmaztuk. Mivel a határérték megegyezik a helyettesítési értékkel, ezért a függvény folytonos a végpontokban is. ($h(1)$-re hasonlóan kapható az állítás.)

c) Ehhez határozzuk meg a függvény deriváltját:

$$h'(x) = -\log x - x\frac{1}{\ln 2}\frac{1}{x} + \log(1-x) + (1-x)\frac{1}{\ln 2}\frac{1}{1-x} = \log\left(\frac{1}{x}-1\right).$$

Mivel $\frac{1}{x}-1 \geq 1$ a $\left(0,\frac{1}{2}\right)$-ben, ezért itt $h'(x)$ is pozitív, vagyis $h(x)$ szigorúan monoton növekedő.

d) Ehhez pedig a második derivált vizsgálata szükséges:

$$h''(x) = -\frac{1}{\ln 2}\frac{x}{1-x}\frac{1}{x^2} = -\frac{1}{\ln 2}\frac{1}{x(1-x)}.$$

Mivel mind az x, mind az $1-x$ pozitív a $(0,1)$ intervallumban, ezért a $h''(x)$ negatív, vagyis $h(x)$ konkáv.

4.16. megoldás. Az LZ78-kódolás egyes lépéseit bejelöltük a bemeneten:

a|aa|aaa|aaaa|aaaaa|aaaaaa|aaaaaaa|aaaaaaaa

a kódoló kimenete és a szótárépítés pedig az alábbiakban követhető nyomon:

a kódoló kimenete	szótár index	szótár bejegyzés
$\langle 0, f(a)\rangle$	1	*a*
$\langle 1, f(a)\rangle$	2	*aa*
$\langle 2, f(a)\rangle$	3	*aaa*
$\langle 3, f(a)\rangle$	4	*aaaa*
$\langle 4, f(a)\rangle$	5	*aaaaa*
$\langle 5, f(a)\rangle$	6	*aaaaaa*
$\langle 6, f(a)\rangle$	7	*aaaaaaa*
$\langle 7, f(a)\rangle$	8	*aaaaaaaa*
$\langle 8, f(a)\rangle$	9	*aaaaaaaaa*

4.17. megoldás. Az $(X+Y) \bmod 256$ valószínűségi változó eloszlása megegyezik az Y valószínűségi változó eloszlásával, így az entrópiájuk is azonos:

$$H((X+Y) \bmod 256) = H(Y) = -\sum_{i=0}^{255} \frac{1}{256} \log \frac{1}{256} = \log 256 = 8.$$

4.18. megoldás. Abból, hogy mindkét csapat egyenlő eséllyel nyer meg egy mecscset, és a meccsek egymástól függetlenek, kiszámíthatjuk a két valószínűségi változó eloszlását:

$$\mathbf{P}\{X = AA\} = \tfrac{1}{4}, \quad \mathbf{P}\{X = ABA\} = \tfrac{1}{8}, \quad \mathbf{P}\{X = BAA\} = \tfrac{1}{8},$$
$$\mathbf{P}\{X = BB\} = \tfrac{1}{4}, \quad \mathbf{P}\{X = BAB\} = \tfrac{1}{8}, \quad \mathbf{P}\{X = ABB\} = \tfrac{1}{8},$$

és

$$\mathbf{P}\{Y = 2\} = \mathbf{P}\{X = AA\} + \mathbf{P}\{X = BB\} = \tfrac{1}{2},$$
$$\mathbf{P}\{Y = 3\} = 1 - \mathbf{P}\{Y = 2\} = \tfrac{1}{2}.$$

Ezekből pedig az entrópia definíciója szerint:

$$H(X) = -2 \cdot \tfrac{1}{4} \log \tfrac{1}{4} - 4 \cdot \tfrac{1}{8} \log \tfrac{1}{8} = \tfrac{5}{2},$$
$$H(Y) = -2 \cdot \tfrac{1}{2} \log \tfrac{1}{2} = 1.$$

4.19. megoldás. A Shannon–Fano-kódnak úgy keletkezhetnek hosszú kódszavai, ha két kumulált valószínűség nagyon közel esik egymáshoz, s így sokjegyű bináris tört alakban kell felírnunk őket ahhoz, hogy megkülönböztethetők legyenek egymástól. A **p** eloszlásnak legalább háromeleműnek kell lennie ahhoz, hogy a

Shannon–Fano-kódlás ne a triviális $\{0,1\}$ kódot eredményezze. Ez elegendő is: mutatunk egy alkalmas háromelemű eloszlást.

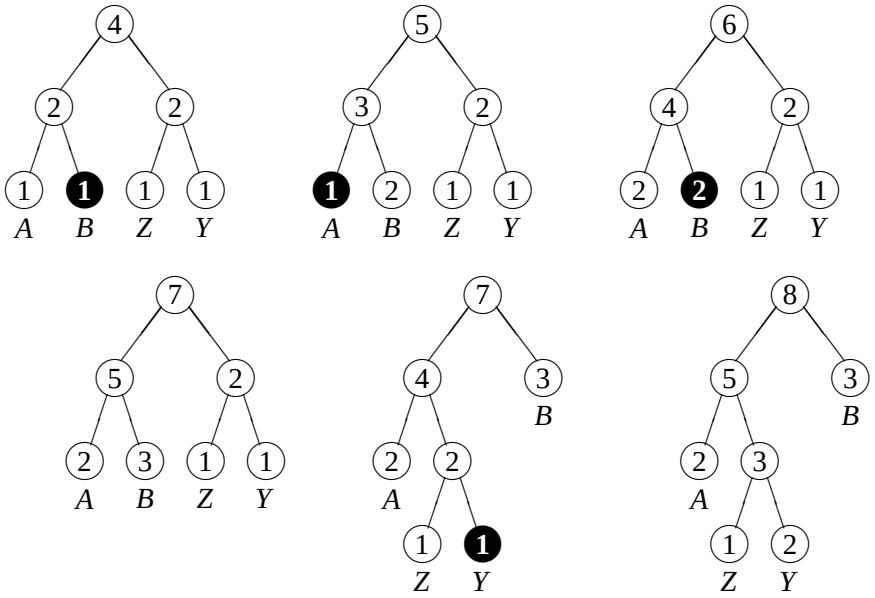
$$\mathbf{p} := \left(1-2^{-k+1}, 2^{-k}, 2^{-k}\right),$$

ahol $k \geq 2$ egész szám. Ekkor a kumulált valószínűségek bináris tört alakban:

$$w_0 = 0, \quad w_1 = 0.\underbrace{11\ldots1}_{k-1}0_2, \quad w_2 = 0.\underbrace{11\ldots1}_{k}{}_2$$

A megkülönböztethetőség érdekében az utóbbi két bináris törtet k jegy pontossággal kell leírni, így végül két k hosszú és egy 1 hosszú kódszót kapunk, tehát $l_{\max}^{\mathrm{SF}} = k$. A Huffman-kódolás algoritmusa szerint először a két 2^{-k} valószínűséget vonjuk össze, majd ezt az $1-2^{-k+1}$ valószínűséggel, így két 2 hosszú és egy 1 hosszú kódszót kapunk, tehát $l_{\max}^{\mathrm{H}} = 2$. A feltétel azt kívánja, hogy legyen $l_{\max}^{\mathrm{SF}} - l_{\max}^{\mathrm{H}} = k-2 > 800$, vagyis bármely $k > 802$ egész szám esetén a fenti $\mathbf{p}$ eloszlás ilyen tulajdonságú.

4.20. megoldás. A kódfa alakulását az alábbiakban követhetjük nyomon:



Feleltessük meg a bal éleknek a 0-t, a jobbaknak pedig az 1-et. A négy betűt minden lépésben az aktuális kódfa szerint kódoljuk. A kódfák megfelelő csúcsait feketével jelöltük. Ekkor a bináris kód: 01 00 01 011.

4.21. megoldás. Az m várható értékű, vagyis $\frac{1}{m}$ paraméterű geometriai eloszlású valószínűségi változó eloszlása

$$p_k = \frac{1}{m}\left(1-\frac{1}{m}\right)^{k-1}, \qquad k \in \mathbb{Z}^+.$$

Legyen $\mathbf{q}$ tetszőleges eloszlás m várható értékkel. Ekkor

$$\begin{aligned}
H(\mathbf{p}) - H(\mathbf{q}) &= -\sum_{k=1}^{\infty} p_k \log p_k + \sum_{k=1}^{\infty} q_k \log q_k = \\
&= -\sum_{k=1}^{\infty} p_k \log \left(\frac{1}{m} \left(1 - \frac{1}{m} \right)^{k-1} \right) + \sum_{k=1}^{\infty} q_k \log q_k = \\
&= \sum_{k=1}^{\infty} p_k \left(-\log m + (k-1) \log \left(1 - \frac{1}{m} \right) \right) + \sum_{k=1}^{\infty} q_k \log q_k = \\
&= \sum_{k=1}^{\infty} q_k \left(-\log m + (k-1) \log \left(1 - \frac{1}{m} \right) \right) + \sum_{k=1}^{\infty} q_k \log q_k = \\
&= -\sum_{k=1}^{\infty} q_k \log p_k + \sum_{k=1}^{\infty} q_k \log q_k = \\
&= -\sum_{k=1}^{\infty} q_k \log \frac{q_k}{p_k} \geq \\
&\geq 0,
\end{aligned}$$

ahol az utolsó lépésben a 4.7. feladat eredményét használtuk fel.

4.22. megoldás. Adott várható értékű és pozitív egész értékű véletlen betűk esetén a legnagyobb átlagos kódszóhossz geometriai eloszlása esetén adódik.

4.23. megoldás. A KALEVALA szó ciklikus eltoltjainak lexikografikus rendezése után a következőt kapjuk:

K	A	L	E	V	A	L	A			A	K	A	L	E	V	A	L
A	L	E	V	A	L	A	K			A	L	A	K	A	L	E	V
L	E	V	A	L	A	K	A			A	L	E	V	A	L	A	K
E	V	A	L	A	K	A	L			E	V	A	L	A	K	A	L
V	A	L	A	K	A	L	E	$\Longrightarrow$		K	A	L	E	V	A	L	A $\leftarrow$
A	L	A	K	A	L	E	V			L	A	K	A	L	E	V	A
L	A	K	A	L	E	V	A			L	E	V	A	L	A	K	A
A	K	A	L	E	V	A	L			V	A	L	A	K	A	L	E

vagyis a transzformáció eredménye az LVKLAAAE blokk és az 5, hiszen ebben a sorban jelenik meg az eredeti blokk.

4.13. Összefoglalás

Ebben a fejezetben a veszteségmentes tömörítési módszereket tekintettük át. [2, 3, 6, 8, 11] Megállapítottuk, hogy elegendő az egyértelműen dekódolható kódok egy könnyen kezelhető osztályával, a prefix kódokkal foglalkoznunk. Definiáltuk a tömöríthetőség elvi korlátját, a Shannon által bevezetett entrópiát. [4, 5] Kimondtuk, hogy az egyértelműen dekódolható kódok átlagos kódszóhossza nem lehet kisebb

az entrópiánál. A Shannon–Fano-kód segítségével felső korlátot adtunk az átlagos kódszóhosszra. Kiterjesztettük a betűnkénti kódolást blokk-kódolássá. Megismerkedtünk egy optimális tömörítési módszerrel, a Huffman-kódolással. Ezután az aritmetikai kódolás következett, amely nagy blokkméret esetén is hatékonyan alkalmazható, valamint lehetővé teszi a valós idejű dekódolást. [12]

Az adaptív tömörítési eljárások közül a Lempel–Ziv-algoritmusokat (LZ77, LZ78, LZW) és az adaptív Huffman-kódot vettük sorra. Az egyre népszerűbb Burrows–Wheeler-transzformáción alapuló tömörítési módszer is terítékre került. Végül a faxkódolás példája és néhány, a gyakorlatban használt tömörítőprogram ismertetése zárta a fejezetet. [12]

Irodalomjegyzék

[1] Ash, R.B. *Information Theory*. Interscience Publishers, 1965.

[2] Cover, T., Thomas, J. *Elements of Information Theory, Second Edition*. John Wiley and Sons, New Jersey, 2006.

[3] Csibi S. (szerk.) *Információ közlése és feldolgozása*. Tankönyvkiadó, Budapest, 1986.

[4] Csiszár I., Fritz J. *Információelmélet*. ELTE TTK jegyzet, Budapest, 1986.

[5] Csiszár I., Körner J. *Information Theory: Coding Theorems for Discrete Memoryless Systems*. Akadémiai Kiadó, Budapest, 1981.

[6] Gallager, R.G. *Information Theory and Reliable Communication*. Wiley, 1968.

[7] Géher K. (szerk.) *Híradástechnika*. Műszaki Könyvkiadó, Budapest, 1993.

[8] Györfi L., Győri S., Vajda I. *Információ- és kódelmélet*. TypoTEX Kiadó, Budapest, 2002.

[9] Linder T., Lugosi G. *Bevezetés az információelméletbe*. Műegyetemi Kiadó, Budapest, 1993.

[10] Massey, J.L. *Applied Digital Information Theory*. Course Notes, ETH Zürich, 1984.

[11] McEliece, R.J. *The Theory of Information and Coding*. Addison-Wesley, 1977.

[12] Sayood, K. *Introduction to Data Compression*. Morgan Kaufmann Publishers, San Francisco, 1996.

[13] Shannon, C.E. A mathematical theory of communication. *Bell System Technical Journal*, 1948.

5. fejezet

Veszteséges forráskódolás

Az eddigi vizsgálataink során megköveteltük, hogy a kódolt üzenet egyértelműen visszaállítható legyen. Ezt a követelményt sok gyakorlati probléma esetén fel kell adnunk, illetve jobb, ha feladjuk. Veszteséges forráskódolás esetén nem cél a tökéletes reprodukció, vagyis megengedünk torzítást, de továbbra is gazdaságos, tömör reprezentációt szeretnénk. A mindennapi alkalmazások közé tartozik a beszéd, zene, kép, videó tömörítése. Kép tömörítése esetén például nyilván felesleges megkövetelni, hogy a reprodukált kép képpontról képpontra megegyezzen az eredeti képpel, csupán azt szeretnénk, szemmel ne érzékeljünk romlást.

Ebben a fejezetben olyan forráskódolási eljárásokat vizsgálunk, ahol az üzenet tökéletes reprodukciója helyett csak azt várjuk el, hogy a dekódolt üzenet az eredetit valamilyen értelemben hűen — de nem feltétlenül pontosan — adja visszsza. Ebben a feladatban két célfüggvényünk van. Az egyikkel mérjük a tömörítést, a másikkal a torzítást, vagyis azt, hogy a tömörítés utáni reprodukció mennyire hasonlít az eredetire. Két, egymásnak ellentmondó célunk van, nevezetesen alacsony értéken tartani mind a tömörítési arányt, mind a torzítást. A probléma úgy kezelhető, ha az egyiket, például a torzítást egy előírt értéken rögzítjük, és emellett minimalizáljuk a tömörítési arányt. Az elvi határ ekkor is tisztázható, de az elvi határt közelítő kódok ma még nem ismertek. Ugyanakkor léteznek a gyakorlatban hatékony veszteséges tömörítő eljárások. Ilyennel találkozhatunk például a mobiltelefóniában. Az emberi beszéd digitális átvitele esetén például a folytonos jelből mintavételezéssel és kvantálással olyan jelet kapunk, amely már véges értékkészletű. Mégis azt mondhatjuk, hogy ezzel semmit sem vesztettünk, hiszen például a digitális központon keresztülhaladó telefonkapcsolat ugyanolyan jó minőségű (vagy jobb), mintha az analóg/digitális – digitális/analóg átalakítást elhagynánk. A lényeg az, hogy a forrásnak csak a számunkra lényeges jellemzőit tartjuk meg, és így — megelégedve a közelítő visszaállítással — úgy kódolhatjuk, hogy a kapott jel továbbítása már kisebb költséggel megoldható legyen. (Vagyis pl. bináris kódot használva, a forrás kevesebb biten reprezentálható.)

A következőkben tárgyalandó kódok közös jellemzője lesz, hogy ún. blokkból-blokkba kódok, azaz a forrásábécé betűinek állandó hosszú blokkjait állandó hosz-

szú kódszavakkal kódoljuk. Feltesszük, hogy adott az üzenetek és a kódjaik között egy ún. hűségmérték, ami azt méri, hogy egy adott kódszót milyen mértékben tekinthetünk egy adott üzenet reprodukciójának. Vizsgálataink középpontjában az a kérdés áll, hogy kódolással milyen mértékben tömöríthetjük a forrás által kibocsátott jelet, ha azt akarjuk, hogy a kód a forrást adott átlagos hűséggel reprezentálja.

Sajnos torzítást megengedő forráskódolás esetén a tömöríthetőség elvi határainak a jellemzése általában nem konstruktív, ugyanakkor a gyakorlati feladatokban mégis kell tömöríteni, tehát megemlítünk néhány gyakorlati eljárást is: kvantálást, prediktív kódolást, beszéd-, hang-, kép- és videotömörítést.

5.1. Kvantálás

A digitális módszereket a hírközlés szinte minden területén alkalmazzák. Minden esetben, amikor az adatok feldolgozása digitálisan történik, a folytonos értékkészletű jelet véges értékkészletűvé kell alakítani. Az alkalmazott, tehát konstruktív digitalizálás legegyszerűbb módja a skalár (egydimenziós) kvantálás.

Skalárkvantálás

Legyen X egy valós értékű véletlen jel, az egydimenziós kvantáltján egy véges értékkészletű $Q : \mathbb{R} \to \mathbb{R}$ leképezéssel kapott $Q(X)$ diszkrét valószínűségi változót értünk. A $Q(\cdot)$ függvényt **kvantáló**nak nevezzük. A kódolás hűségét egy speciális hűségmértékkel, a $D(Q)$ négyzetes torzítással mérjük:

$$D(Q) = \mathbf{E}\left((X - Q(X))^2\right). \tag{5.1}$$

Természetesen más hűségmértéket (torzításmértéket) is választhattunk volna, de a gyakorlatban ez a legelterjedtebb.

Legyen a Q kvantáló értékkészlete az $\{x_1, x_2, \ldots, x_N\}$ halmaz, ahol az x_i-k valós számok. Az x_i számokat kvantálási szinteknek nevezzük. Vegyük észre, hogy Q-t egyértelműen leírják az $\{x_1, x_2, \ldots, x_N\}$ kvantálási szintek és a $\mathcal{B}_i = \{x \in \mathbb{R} : Q(x) = x_i\},\ i = 1, \ldots, N$ kvantálási tartományok. A $\mathcal{B}_i$ halmazok persze diszjunktak és egyesítésük kiadja az egész valós egyenest. A kvantáló működése ezért a következőképp írható le:

$$Q(x) = x_i, \quad \text{ha } x \in \mathcal{B}_i.$$

Tegyük most fel, hogy a kvantálót leíró adatok közül most csak az $\{x_1, x_2, \ldots, x_N\}$ kvantálási szinteket ismerjük. Ekkor az ilyen kvantálási szinteket használó kvantálók közül a legkisebb torzítású az a Q kvantáló, amelyre

$$\mathcal{B}_i = \{x :\ |x - x_i| \le |x - x_j|,\ j = 1, 2, \ldots, N\} \tag{5.2}$$

(legközelebbi szomszéd feltétel), ahol a döntési szabályt úgy tehetjük egyértelművé (vagyis a $\mathcal{B}_i$-ket diszjunktakká), hogy ha egy adott x kettő vagy több $\mathcal{B}_i$-be

tartozna, akkor a legkisebb indexűhöz soroljuk. Az adott kvantálási szintekkel Q valóban legkisebb négyzetes torzítású, hiszen ha Q^* egy másik kvantáló ugyanezen kvantálási szintekkel, akkor egy tetszőleges x-re $Q(x) = x_i$ (tehát $x \in \mathcal{B}_i$) és $Q^*(x) = x_j$ valamely $1 \le i,\ j \le N$ indexekre, de ekkor (5.2) szerint

$$|x - x_i| \le |x - x_j|,$$

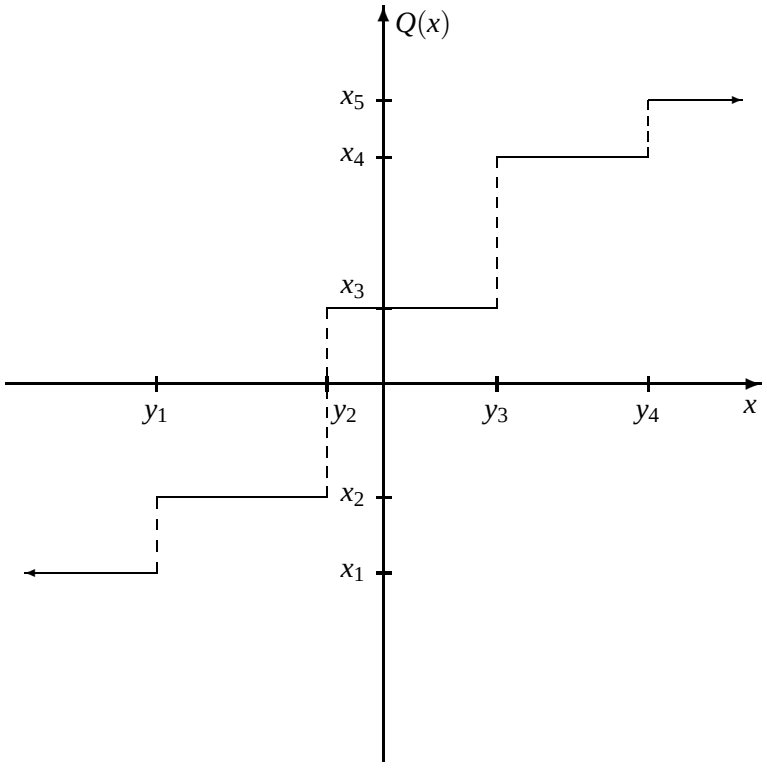
tehát

$$(x - Q(x))^2 \le (x - Q^*(x))^2$$

teljesül minden $x \in \mathbb{R}$-re, amiből $D(Q) \le D(Q^*)$ következik. Ezért a következőkben csak az (5.2) szerinti kvantálókkal foglalkozunk. Az ilyen kvantálók $\mathcal{B}_i$ kvantálási tartományai nagyon egyszerűen néznek ki. Az általánosság megszorítása nélkül tegyük fel most, hogy a kvantálási szintek nagyság szerint rendezve vannak, vagyis $x_1 < x_2 < \cdots < x_N$. Ekkor, bevezetve az $y_i = \frac{x_i + x_{i+1}}{2}$, $i = 1, \ldots, N-1$ jelölést, az (5.2) szerinti $\mathcal{B}_i$ halmazok a következő intervallumok lesznek:

$$\mathcal{B}_1 = (-\infty, y_1], \qquad \mathcal{B}_i = (y_{i-1}, y_i],\ i = 2, \ldots, N-1, \qquad \mathcal{B}_N = (y_{N-1}, \infty).$$

Egy ilyen, $N = 5$ szintű kvantálót ábrázol az 5.1. ábra.



5.1. ábra. Kvantáló $N = 5$ kvantálási szinttel

Tekintsük most azt a feladatot, hogy adottak a $\{\mathcal{B}_i\}$ tartományok, és ehhez hogyan válasszuk meg az x_i kvantálási szinteket. A teljes valószínűség tétele szerint

$$\mathbf{E}\left((X-Q(X))^2\right) = \sum_{i=1}^{N} \mathbf{E}\left((X-Q(X))^2 \mid X \in \mathcal{B}_i\right) \mathbf{P}\{X \in \mathcal{B}_i\} = \\ = \sum_{i=1}^{N} \mathbf{E}\left((X-x_i)^2 \mid X \in \mathcal{B}_i\right) \mathbf{P}\{X \in \mathcal{B}_i\},$$

amiből következik, hogy az optimális x_i kvantálási szint a $\mathcal{B}_i$ tartomány súlypontja, ugyanis a Steiner-tétel miatt tetszőleges c konstansra

$$\mathbf{E}\left((X-c)^2 \mid X \in \mathcal{B}_i\right) = \mathbf{E}\Big((X-\mathbf{E}(X \mid X \in \mathcal{B}_i))^2 \mid X \in \mathcal{B}_i\Big) + (\mathbf{E}(X \mid X \in \mathcal{B}_i) - c)^2.$$

Jelölje $f(x)$ az X sűrűségfüggvényét, ekkor a négyzetes torzítás a $\mathcal{B}_i$ tartományon, azaz

$$\frac{1}{\int\limits_{\mathcal{B}_i} f(x)\,\mathrm{d}x} \int\limits_{\mathcal{B}_i} (x-x_i)^2 f(x)\,\mathrm{d}x = \mathbf{E}\left((X-x_i)^2 \mid X \in \mathcal{B}_i\right)$$

akkor minimális, ha

$$x_i = \mathbf{E}(X \mid X \in \mathcal{B}_i) = \frac{\int\limits_{\mathcal{B}_i} x f(x)\,\mathrm{d}x}{\int\limits_{\mathcal{B}_i} f(x)\,\mathrm{d}x},$$

ami épp a súlypont.

A Lloyd–Max-algoritmus

Optimális kvantáló tervezésénél azt a technikát alkalmazzuk, hogy az átlagos torzítás csökkentéséhez a bemenetet pontosabban közelítjük a nagyobb valószínűségű tartományokban, míg a kis valószínűségű tartományokban rosszabb közelítést engedünk meg, mint egyenletes kvantálás esetén. Ezt úgy tehetjük meg, hogy a nagyobb valószínűségű helyeken a kvantálási intervallumokat kisebbnek választjuk. Amennyiben az intervallumok száma konstans, ez egyben azt is jelenti, hogy a kisebb valószínűségű helyeken nagyobb intervallumhosszakkal dolgozunk. (Ez hasonlatos ahhoz, hogy veszteségmentes tömörítés esetén az átlagos tömörítési arány javításához a kisebb valószínűséggel előforduló forrásszavakhoz hosszabb kódszavakat rendelünk.)

Egy adott X valószínűségi változóhoz keressük az N szintű, optimális Q kvantálót. Feladatunk az $x_1 < x_2 < \cdots < x_N$ kvantálási szintek és a $Q(X)$ függvény meghatározása úgy, hogy a $D(Q)$ négyzetes torzítás minimális legyen. Bebizonyítható, hogy egy optimális kvantáló kielégíti az alábbi két szükséges feltételt, amelyeket együtt Lloyd–Max-feltételnek nevezünk:

1. Legközelebbi szomszéd feltétel:
   A kvantálási intervallumok határait éppen a kvantálási szintek által kijelölt szakaszok felezőpontjai adják, vagyis

$$y_i = \frac{x_i + x_{i+1}}{2}, \qquad i = 1,2,\ldots,N-1.$$

2. Súlypont feltétel:
   Minden x_i kvantálási szint a $\mathcal{B}_i$ kvantálási intervallumnak a súlypontja, azaz

$$x_i = \frac{\int\limits_{\mathcal{B}_i} x f(x)\,\mathrm{d}x}{\int\limits_{\mathcal{B}_i} f(x)\,\mathrm{d}x}, \qquad i = 1,\ldots,N.$$

A Lloyd–Max-feltételt kielégítő kvantálót Lloyd–Max-kvantálónak nevezzük. Ha egy kvantáló nem elégíti ki a Lloyd–Max-feltétel bármelyik részét, akkor lehetséges egy olyan kvantálót készíteni, amelynek négyzetes torzítása kisebb, tehát egy nem Lloyd–Max-kvantáló nem lehet optimális, de létezik nem optimális Lloyd–Max-kvantáló is.

5.1. példa. Legyen az X valószínűségi változó az $\{1,2,3,4\}$ halmazon egyenletes eloszlású. Pontosan 3 féle 2-szintű Lloyd–Max-kvantálót alkalmazhatunk erre:

$$Q_2^1(1) = 1; \qquad Q_2^1(2) = Q_2^1(3) = Q_2^1(4) = 3$$

$$Q_2^2(4) = 4; \qquad Q_2^2(1) = Q_2^2(2) = Q_2^2(3) = 2$$

$$Q_2^3(1) = Q_2^3(2) = 1.5; \qquad Q_2^3(3) = Q_2^3(4) = 3.5$$

Q_2^1 és Q_2^2 négyzetes torzítása egyaránt 0.5, míg Q_2^3-é 0.25. Annak ellenére, hogy mindhárom Lloyd–Max-kvantáló, csak Q_2^3 optimális.

A Lloyd–Max-algoritmus:
A kvantálót egyértelműen jellemzik az x_i kvantálási szintek és a $\mathcal{B}_i = (y_{i-1}, y_i]$ tartományok. Célunk a szintek és az intervallumhatárok javítása lépésről lépésre.

1. Vegyünk fel egy közelítést a kvantálási szintekre.

2. Optimalizáljuk a kvantálót a kvantálási szintek szerint, vagyis határozzuk meg az intervallumhatárokat a legközelebbi szomszéd feltétel kielégítésével.

3. Számítsuk ki, hogy mennyivel csökkent a torzítás, és ha ez egy előre meghatározott küszöbértéknél kisebb, akkor készen vagyunk.

4. Optimalizáljuk a kvantálót az imént kapott intervallumokhoz, vagyis alkalmazzuk a súlypont feltételt, és folytassuk az algoritmust a 2. ponttól.

Egyenletes kvantáló

A technikai nehézségek elkerülése végett a további vizsgálataink során feltesszük, hogy a kvantált X valós valószínűségi változó eloszlása abszolút folytonos f sűrűségfüggvénnyel, valamint azt is feltesszük, hogy f a $[-A,A]$ intervallumban folytonos, a $[-A,A]$ intervallumon kívül nulla értékű függvény. Az f-et felhasználva a kvantáló négyzetes torzítása a következőképp írható fel:

$$D(Q) = \int_{-\infty}^{\infty} (x - Q(x))^2 f(x)\,\mathrm{d}x = \sum_{i=1}^{N} \int_{\mathcal{B}_i} (x - x_i)^2 f(x)\,\mathrm{d}x.$$

A legegyszerűbb kvantáló az **egyenletes kvantáló**. A Q_N N-szintű egyenletes kvantálót úgy kapjuk, hogy az X lehetséges értékeinek halmazát, vagyis a $[-A,A]$ intervallumot, N egyenlő nagyságú intervallumra osztjuk (ezek a $\mathcal{B}_i$ intervallumok), és a kvantálási szinteket ezen intervallumok közepén helyezzük el. Formálisan tehát a kvantálási szintek

$$Q_i(x) = -A + (2i-1)\frac{A}{N},$$

ha

$$-A + 2(i-1)\frac{A}{N} < x \le -A + 2i\frac{A}{N}, \qquad i = 1,\ldots,N,$$

a kvantálási intervallumok hossza pedig

$$\Delta_N = \frac{2A}{N}.$$

A következő tétel megmutatja, hogy az N szintű egyenletes kvantáló négyzetes torzítása nagy N esetén közelítőleg $\frac{\Delta_N^2}{12}$.

5.1. tétel. *Ha az X valószínűségi változó f sűrűségfüggvényére a fenti feltételek teljesülnek, akkor az X N-szintű egyenletes kvantálásának $D(Q_N)$ torzítására a*

$$\lim_{N\to\infty} \frac{D(Q_N)}{\Delta_N^2} = \frac{1}{12}$$

aszimptotikus összefüggés teljesül.

A kvantáló kimenete egy valószínűségi változó, amely függvénye a bemenetnek. Ezért a 4.2. tétel c) pontja szerint a kvantáló kimenetének entrópiája (vagy röviden: a kvantáló entrópiája) nem lehet nagyobb, mint a bemenet entrópiája, azaz $H(Q_N(X)) \le H(X)$. A következő tétel egy aszimptotikus összefüggést ad az egyenletes kvantáló szintjeinek száma és entrópiája között.

5.2. tétel. *Tegyük fel, hogy az X valószínűségi változó f sűrűségfüggvényére a fenti feltételek teljesülnek, valamint a*

$$H(f) = -\int_{-A}^{A} f(x)\log f(x)\,\mathrm{d}x$$

integrál véges. Ekkor az X N-szintű egyenletes kvantálásának $H(Q_N(X))$ entrópiájára a

$$\lim_{N\to\infty}\left(H(Q_N(X)) + \log\Delta_N\right) = H(f).$$

aszimptotikus összefüggés teljesül.

A tétel tehát azt állítja, hogy a kvantálási szintek N számának növekedésével az egyenletes kvantáló entrópiájára a $H(Q_N(X)) \approx H(f) - \log\Delta_N$ közelítés érvényes.

A $H(f)$ mennyiséget **differenciális entrópiának** hívjuk, mely a finom, egyenletes kvantáló kimenetének a tömöríthetőségét méri, azaz valamilyen értelemben az eloszlás terjedelmét. Ha az 5.1. és 5.2. tételeket összevetjük, akkor könnyen belátható a

$$\lim_{N\to\infty}\left(H(Q_N(X)) + \log\sqrt{12D(Q_N)}\right) = H(f)$$

aszimptotikus összefüggés az egyenletes kvantálás torzítása és entrópiája között. Ezt úgy is fogalmazhatjuk, hogy

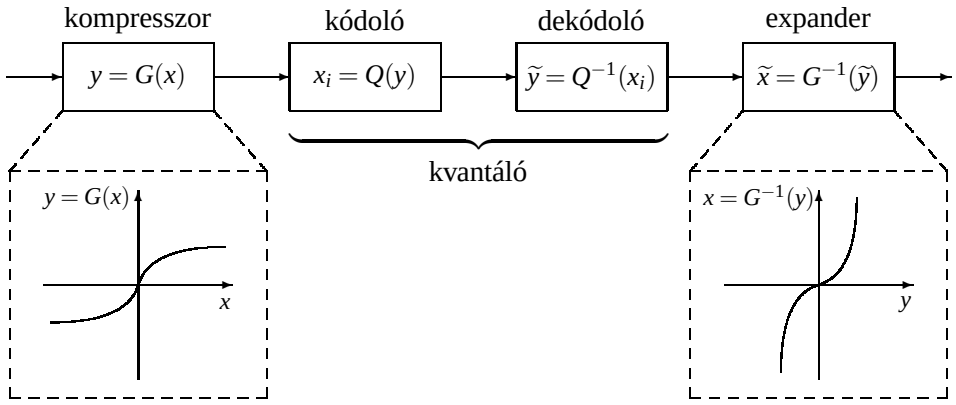
$$H(Q_N(X)) \approx H(f) - \log\sqrt{12D(Q_N)}$$

nagy N-ek, vagyis finom (kis lépésközű) kvantálás esetén, ami egy hasznos közelítés lehet egyenletes kvantáló tervezésénél.

Kompanderes kvantáló

A kvantálásra a gyakorlatban általában olyan alkatrészek érhetők el, melyekkel egyenletes kvantálást valósíthatunk meg. Ebben az esetben a kvantált értéket kell tömöríteni. Gyakran ezt nem akarjuk megtenni, sokkal inkább azt szeretnénk, hogy rögzítve a kvantálási szintek számát, N-et, minimalizáljuk a négyzetes torzítást. Ezt úgy tesszük, hogy a kvantálandó jelet egy monoton növekedő függvénnyel, a kompresszorral a $\left[-\frac{1}{2},\frac{1}{2}\right]$-be képezzük, ott alkalmazunk egy egyenletes kvantálót, és a kvantált értéket a kompresszor inverzével (az expanderrel) visszatranszformáljuk (5.2. ábra). (kompander = kompresszor + expander)

A leggyakrabban alkalmazott nemegyenletes kvantálók a logaritmikus kompresszor karakterisztikát használó eszközök, amelyek a távközlő hálózatokban a beszédkódolást végzik a ’60-as évek óta. Azért alkalmaznak logaritmikus kvantálást, mert az emberi beszédben a kis amplitúdójú jelek az érthetőség szempontjából rendkívül fontosak, ezért a lehető legnagyobb pontossággal kell ezeket kvantálni, míg a nagy amplitúdójú jelek esetében a túl nagy jelszintet kell megakadályozni. A beszédkódolásban kétféle kompander használatos: a μ-law és az A-law. Az előbbi



5.2. ábra. Kompanderes kvantáló

az Egyesült Államokban, Kanadában és Japánban elterjedt. Kompresszor ill. expander függvénye:

$$G_\mu(x) = \frac{\ln(1+\mu|x|)}{\ln(1+\mu)}\operatorname{sgn} x \qquad -1 \le x \le 1$$

$$G_\mu^{-1}(x) = \frac{1}{\mu}\left((1+\mu)^{|x|}-1\right)\operatorname{sgn} x \qquad -1 \le x \le 1$$

A μ paraméter értékét általában 255-nek választják.

Az Európában, Afrikában, Ausztráliában és Dél-Amerikában alkalmazott A-law kompandere az alábbi ($A = 87.6$):

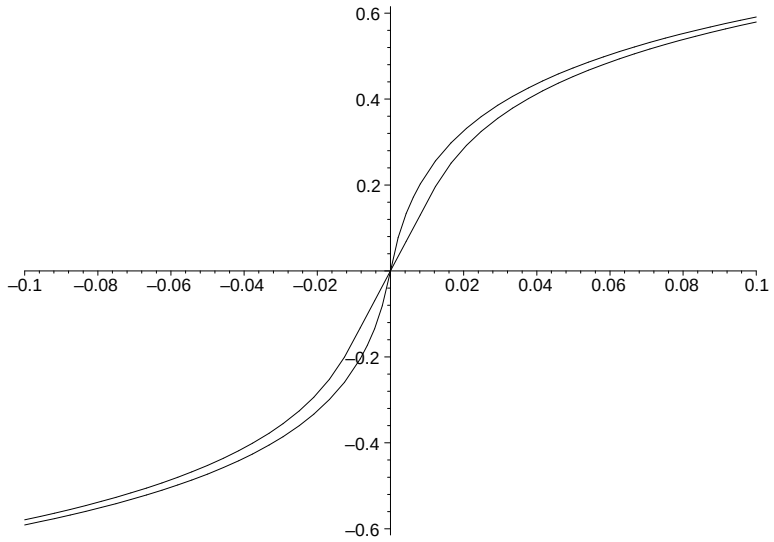
$$G_A(x) = \begin{cases} \frac{Ax}{1+\ln A}, & 0 \le |x| < \frac{1}{A} \\ \frac{1+\ln|Ax|}{1+\ln A}\operatorname{sgn} x, & \frac{1}{A} \le |x| \le 1 \end{cases}$$

$$G_A^{-1}(x) = \begin{cases} \frac{x|1+\ln A|}{A}, & 0 \le |x| < \frac{1}{1+\ln A} \\ \frac{e^{|x|(1+\ln A)-1}}{A}\operatorname{sgn} x, & \frac{1}{1+\ln A} \le |x| \le 1 \end{cases}$$

Az alapvető különbség az A-law és a μ-law karakterisztika között, hogy az A-lawnak kicsit szélesebb a dinamikatartománya (értékkészletének terjedelme), míg a μ-lawnak egy kicsit kisebb az alapzaja. Az 5.3. ábrán látható a két kompresszor függvény. A különbség csak a 0-hoz közeli tartományban figyelhető meg, ezért a $[-0.1, 0.1]$ intervallumban ábrázoltuk a őket (a vízszintes tengelyhez közelebb haladó görbe az A-law, míg a távolabbi a μ-law).

A finom, egyenletes kvantáló négyzetes torzítása kiterjeszthető a kompanderes esetre. Jelölje $G(x)$ a kompresszort és

$$g(x) = G'(x)$$



5.3. ábra. A-law és μ-law kompresszor függvények

a deriváltját, pontosabban

$$G(x) = -\frac{1}{2} + \int_{-\infty}^{x} g(z)\,\mathrm{d}z,$$

és $g(x)$ egy sűrűségfüggvény. Legyenek

$$-\frac{1}{2} = y'_0 < y'_1 < \cdots < y'_N = \frac{1}{2}$$

a transzformált intervallumban a kvantálási határok, $y'_i - y'_{i-1} = \Delta' = \frac{1}{N}$, és

$$y_i = G^{-1}(y'_i), \qquad i = 1,2,\ldots,N-1$$

a valós kvantálási határok. Ekkor az i-edik kvantálási lépcső nagysága

$$\Delta_i = y_i - y_{i-1}.$$

Megmutatható, hogy a közel optimális kompresszor függvényt az X változó $f(x)$ sűrűségfüggvényéből a következő módon lehet kiszámítani:

$$G^*(x) = -\frac{1}{2} + \frac{\int_{-\infty}^{x} f^{1/3}(z)\,\mathrm{d}z}{\int_{-\infty}^{\infty} f^{1/3}(z)\,\mathrm{d}z}. \tag{5.3}$$

Vektorkvantálás

A forrás kimenetének többdimenziós eloszlását felhasználva kisebb torzítást érhetünk el ugyanakkora bit/minta arány mellett vektorkvantálással. A forrásszimbólumok egyenkénti kvantálása helyett (ahogyan azt skalár kvantáló esetén tettük) sorozatokat, vektorokat képezünk belőlük, s ezeket együtt kvantáljuk. Tekintsük például a kétdimenziós esetet. Skalár kvantáló alkalmazása esetén — egymásra merőleges tengelyeket feltételezve — téglalap alakú tartományokat kapunk a síkban, míg vektorkvantálóval bármilyen szabálytalan síkidom alakú (például kör) tartományok kialakíthatók.

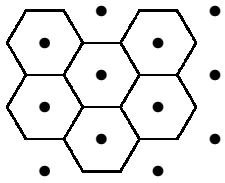
Tegyük fel, hogy emberek magasság- és tömegadatait szeretnénk feldolgozni. Ezek például 100 és 200 cm, illetve 20 és 120 kg közé esnek. Ha összesen 6 biten szeretnénk kvantálni az adatainkat, és skalár kvantálót alkalmazunk, akkor 3-3 bit jut a magasságra és a tömegre egyaránt. Az intervallumokat 8 egyenlő részre osztva, azok közepén kijelölve a szinteket (a szemléletesség kedvéért kétdimenziós koordinátarendszerben ábrázolva az így lehetséges kvantálási pontokat: magasság–testtömeg koordinátájú pontok), egy (22 kg, 197 cm) adatpárt ugyanolyan torzítással kvantálunk, mint egy (70 kg, 180 cm)-est. Holott az előbbi adatpár nyilvánvalóan nem fog előfordulni, míg az utóbbi gyakori lesz. Vektorkvantálás alkalmazásával megtehetjük, hogy például a statisztikai vizsgálatok szerint leggyakoribb magasság–testtömeg egyenes környezetében biztosítunk kis torzítást, míg a gyakorlatilag lehetetlen adatpárok esetében megengedünk nagyobb torzítást is.

Legyen $\mathbf{X}$ egy d-elemű forrásvektor $f(\mathbf{x})$ sűrűségfüggvénnyel. A d-dimenziós vektorkvantáló egy $Q(\mathbf{x})$ függvény, amely az $\mathbf{x} \in \mathbb{R}^d$ bemenetet az $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^d$ vektorok egyikébe képezi le. A kvantálót egyértelműen jellemzi az N kimeneti vektor, és a hozzájuk tartozó $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_N$ tartományok, amelyek $\mathbb{R}^d$ egy partícióját alkotják (diszjunktak, és lefedik $\mathbb{R}^d$-t), tehát $Q(\mathbf{x}) = \mathbf{x}_i$, ha $\mathbf{x} \in \mathcal{B}_i$, $i = 1, 2, \ldots, N$. A torzítás vizsgálatára továbbra is a négyzetes torzítási mértéket fogjuk használni:

$$D(Q) = \mathbf{E}\|\mathbf{X} - Q(\mathbf{X})\|^2 = \sum_{i=1}^{N} \int\limits_{\mathbf{x}\in\mathcal{B}_i} \|\mathbf{x} - \mathbf{x}_i\|^2 f(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

ahol $\|\cdot\|$ az euklidészi norma.

Vektorkvantáló tervezése adott torzítási mértékre ismert bemeneti eloszlás esetén a $\mathcal{B}_i$ tartományok és az $\mathbf{x}_i$ kimeneti vektorok meghatározását jelenti. Míg egydimenziós esetben ez viszonylag egyszerű problémát jelentett, hiszen csak a valós egyenes intervallumai jöhettek számításba, addig most még $\mathcal{B}_i$-k alakjára is végtelen sok lehetőségünk van. A torzítás szempontjából optimális megoldás $\mathcal{B}_i$-k alakjára a kör, a gömb, illetve magasabb dimenziókban a hipergömb lenne. Sajnos ezekkel az alakzatokkal azonban nem lehet hézagmentesen lefedni (csempézni) a teret, pedig minden lehetséges bemeneti vektorhoz pontosan egy kimeneti vektort kell hozzárendelnünk. Kétdimenziós esetben jó közelítést jelent a síkot lefedő szabályos hatszögminta, amely az 5.4. ábrán látható.



5.4. ábra. Hatszögminta

Egy optimális vektorkvantáló kielégíti az alábbi két szükséges feltételt, melyek az egydimenziós eset kézenfekvő általánosításai:

1. $\mathbb{R}^d$ partíciója Dirichlet-partíció, vagy más néven tartományai Voronoi-tartományok, azaz

$$\mathcal{B}_i = \{\mathbf{x} : \ \|\mathbf{x}-\mathbf{x}_i\| \leq \|\mathbf{x}-\mathbf{x}_j\|, \quad \forall j \neq i\}.$$

2. A kvantált vektorok a hozzájuk tartozó tartományok súlypontjai:

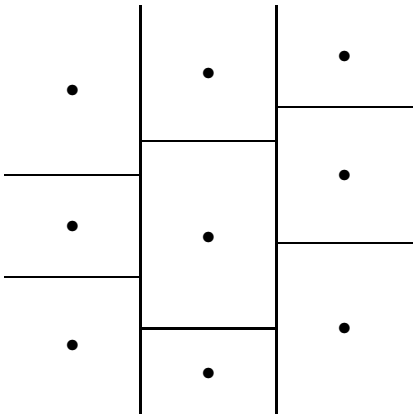
$$\mathbf{x}_i = \frac{\int\limits_{\mathcal{B}_i} \mathbf{x} f(\mathbf{x})\,\mathrm{d}\mathbf{x}}{\int\limits_{\mathcal{B}_i} f(\mathbf{x})\,\mathrm{d}\mathbf{x}}, \qquad i = 1,\ldots,N.$$

A Lloyd–Max-algoritmus általánosításaként vektorkvantálás esetén a **Linde–Buzo–Gray-algoritmus** használatos:

1. Vegyünk fel egy közelítést a kvantálási vektorokra.
2. Optimalizáljuk a kvantálót a kvantálási vektorok szerint, vagyis határozzuk meg a tartományokat a Voronoi-tartományokra vonatkozó feltétel kielégítésével.
3. Számítsuk ki, hogy mennyivel csökkent a torzítás, és ha ez egy előre meghatározott küszöbértéknél kisebb, akkor készen vagyunk.
4. Optimalizáljuk a kvantálót az imént kapott tartományokhoz, vagyis alkalmazzuk a súlypont feltételt, és folytassuk az algoritmust a 2. ponttól.

A Linde–Buzo–Gray-algoritmussal tervezett vektorkvantálónak általában nincs megfigyelhető belső struktúrája. Ez a „véletlenszerű” szerkezet megnehezíti ugyan a kvantálási folyamatot, de biztosítja a vektorkvantáló közel optimális torzítását. Több megoldás létezik struktúrált, de ugyanakkor kedvező torzítással rendelkező vektorkvantáló tervezésére.

A **fa-struktúrájú** vektorkvantáló alapelve az, hogy az $L = K^d$ kimeneti vektor közül a közel optimálisat egy d mélységű és K-adfokú fa segítségével keressük meg. Egy szint egy bejegyzése a következő szint egy K darab vektort tartalmazó



5.5. ábra. Fa-struktúrájú vektorkvantáló

halmazára mutat. Így összesen legfeljebb $d\lceil \log K \rceil$ összehasonlítást kell végrehajtanunk a teljes keresés K^d összehasonlítása helyett. Az 5.5. ábra a $d = 2, K = 3$ esetet szemlélteti, ahol az első koordináta szerint alkalmazunk egy skalár kvantálót, majd részintervallumonként külön tervezünk egy skalár kvantálót a második koordinátára.

Néhány alkalmazásban, mint például a beszédfeldolgozásban, a bemeneti jel dinamikatartománya széles skálán változhat. Az ehhez szükséges nagy kódtáblában való keresés helyett először normáljuk a vektort, majd külön-külön kvantáljuk a normalizált vektort, és a normalizáló faktort. A normalizáló faktor a dinamikatartománybeli helyet, vagyis az energiát határozza meg, míg a normalizált vektor a jel formáját, így ezt a technikát **energia–forma vektorkvantáló**nak nevezzük.

Osztott vektorkvantálás esetén a forrásvektorokat független osztályokba soroljuk térbeli tulajdonságuk alapján. A különböző osztályokhoz különböző vektorkvantálókat tervezünk. Ez a technika előnyös például a képtömörítésben, ahol az éleket tartalmazó illetve nem tartalmazó tartományok két eltérő osztályt alkotnak.

Többszintű vektorkvantálás esetén több menetben dolgozunk. Először egy alacsony bitarányú kvantálóval előállítjuk a bemenet durva közelítését, majd lépésről lépésre mindig az eredeti bemenet és az előző szint által előállított közelítés eltérését (a kvantálási hibát) kvantáljuk.

A képek több nagyobb, közel egyszínű „foltot” tartalmaznak, így jól működik az a megoldás, hogy a képpontok (blokkonkénti) átlagát egy skalár kvantálóval kvantáljuk, majd a képpontokból kivonva ezt az átlagot egy vektorkvantálót alkalmazunk.

5.2. Transzformációs kódolás

Ebben a szakaszban olyan technikát mutatunk be, amelynek segítségével a forrás kimenetét összetevőire bonthatjuk, és ezen összetevőket saját, jellemző karakterisztikájuk szerint kódoljuk.

A **transzformációs kódolás** a vektorkvantálás egy speciális esete. Az optimális vektorkvantáló nagy számítási bonyolultságú, míg a struktúrált vektorkvantáló nem optimális. A transzformációs kódoló megkísérli a kettőt ötvözni: először transzformálja a jelsorozatot, majd a transzformált sorozat komponenseit kvantálja skalár kvantálókkal. Ez így együtt egy struktúrált vektorkvantáló. Ugyanakkor az egyes komponenseket más és más finomságú skalár kvantálóval kvantáljuk, tehát a vektorkvantáló reprodukciós vektorai egy olyan többdimenziós rácsot alkotnak, melyet megkaphatunk egy többdimenziós téglatest eltolásaival.

A transzformációs kódolás lépései a következők:

1. Osszuk fel a kódolandó $\{x_j\}$ jelsorozatunkat k hosszú diszjunkt blokkokra. Minden egyes blokkra alkalmazzunk egy invertálható transzformációt, mely az $\{y_i\}$ sorozatot eredményezi.

2. Kvantáljuk a transzformált sorozatot skalárkvantálóval. Az alkalmazott kvantálási stratégia függ a kívánt bitsebességtől, a transzformált sorozat különböző részeinek eltérő statisztikai tulajdonságaitól és a megengedhető torzítástól.

3. Kódoljuk a kvantált értékeket valamilyen kóddal.

A továbbiakban csak lineáris transzformációval foglalkozunk. Ekkor a transzformált sorozat a következő alakban áll elő:

$$y_i = \sum_{j=1}^{k} a_{i,j} x_j$$

Az $\{y_i\}$ sorozat elemeinek eloszlása a sorozaton belül elfoglalt pozíciótól, i-től függ. A transzformált sorozat elemei eloszlásának terjedelmét a $\boldsymbol{\sigma}_i^2$ szórásnégyzettel mérjük. Ez fogja befolyásolni, hogy hogyan kódoljuk a transzformált sorozatot.

Az eredeti sorozat helyreállítható a transzformált sorozatból az inverz transzformációval:

$$x_j = \sum_{i=1}^{k} a_{j,i}^{-1} y_i$$

Ugyanezek mátrixos alakban:

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$$

ahol $\mathbf{A}$ és $\mathbf{A}^{-1}$ $k \times k$-as mátrixok, melyek i, j-edik eleme $a_{i,j}$ illetve $a_{i,j}^{-1}$.

A képtömörítésnél használt kétdimenziós esetben a transzformáció a következőképp néz ki:

$$\mathbf{Y} = \mathbf{A}\mathbf{X}\mathbf{A}^T$$

és

$$\mathbf{X} = \mathbf{A}^T\mathbf{Y}\mathbf{A},$$

ami akkor igaz, ha ún. ortonormált transzformációt alkalmazunk, amelynél a transzformáló mátrix inverze megegyezik a transzponáltjával: $\mathbf{A}^{-1} = \mathbf{A}^T$.

Szerencsés esetben az $\mathbf{y}$ egyes komponenseinek nagyságrendje különböző, tehát az egyes kvantálók különböző számú kvantálási szintet használnak. Ez a **bitallokáció** problémája. Tegyük fel, hogy a k hosszú blokk kódolására M bitet szánunk úgy, hogy a j-edik kvantáló M_j bitet használhat, azaz kvantálási szintjeinek száma 2^{M_j}. A skalár kvantáló entrópiája a $H(f)$ differenciális entrópiától függ, amely normális eloszlás esetén $\frac{1}{2}\log(2\pi e\sigma^2)$ (lásd az 5.6. feladatot). Az $\mathbf{X}$ lineáris transzformáltjának a komponensei nagy k blokkméret esetén a centrális határeloszlás tétel miatt közelítőleg normális eloszlásúak, tehát indokolt, hogy a j-edik kvantáló

$$M_j = c + \log\boldsymbol{\sigma}(Y_j) \tag{5.4}$$

bitet kapjon. Ekkor

$$\sum_{j=0}^{k-1}(c + \log\boldsymbol{\sigma}(Y_j)) = M,$$

ahonnan a c konstans kiszámolható:

$$c = \frac{M - \sum_{j=0}^{k-1}\log\boldsymbol{\sigma}(Y_j)}{k}.$$

A gyakorlatban a **diszkrét koszinusz transzformáció** (DCT) a legkedveltebb, felhasználja a JPEG és az MPEG álló- illetve mozgóképtömörítési szabvány is. $k \times k$-as $\mathbf{A}$ mátrixának első sorában csupa $\frac{1}{\sqrt{k}}$ elem áll, míg az ez alatti elemek az

$$a_{i,j} = \sqrt{\frac{2}{k}}\cos\left(\frac{(2j-1)(i-1)\pi}{2k}\right)$$

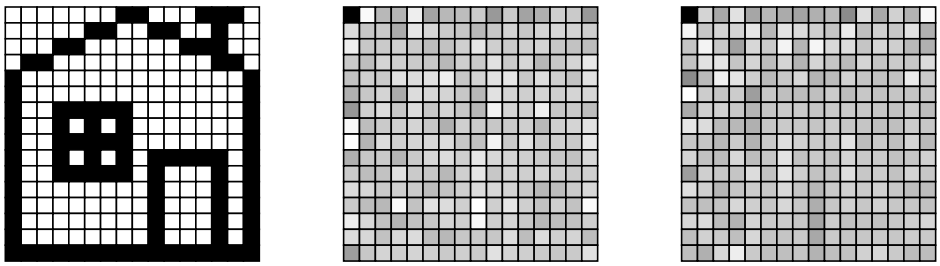
képlettel számolhatók.

Az egyszerűbb **diszkrét Walsh–Hadamard-transzformáció** (DWHT) kisebb számításigényű, de általában nem biztosít olyan jó tömörítési hatásfokot, mint a DCT. A transzformáló mátrixot a következő rekurzióval kapjuk:

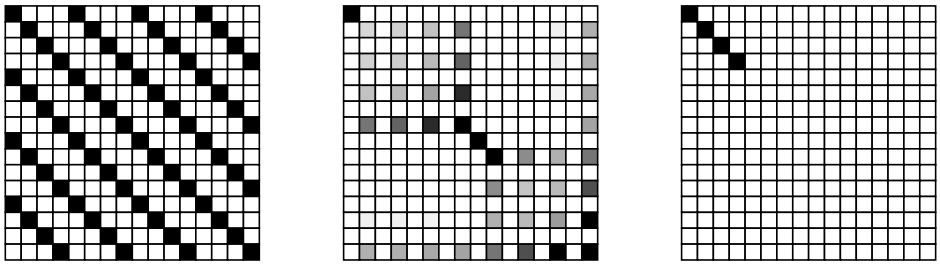
$$\mathbf{A}_{2^k} = \begin{pmatrix} \mathbf{A}_{2^{k-1}} & \mathbf{A}_{2^{k-1}} \\ \mathbf{A}_{2^{k-1}} & -\mathbf{A}_{2^{k-1}} \end{pmatrix}$$

A kiinduló mátrix pedig:

$$\mathbf{A}_1 = \begin{pmatrix} 1 \end{pmatrix}.$$



5.6. ábra. Egy 16×16-os fekete-fehér kép, valamint DCT és DWHT transzformáltja



5.7. ábra. Egy 16×16-os fekete-fehér szimmetrikus ábra, valamint DCT és DWHT transzformáltja

Így például:

$$\mathbf{A}_2 = \begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_1 \\ \mathbf{A}_1 & -\mathbf{A}_1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

A mátrixok kielégítik az $\mathbf{A}_{2^k}\mathbf{A}_{2^k}^T = 2^k\mathbf{I}$ feltételt, és a transzformáció során ennek a normalizáltjával dolgozunk. A 8×8-as DWHT transzformáló mátrixa így a következő:

$$\frac{1}{\sqrt{2^3}}\mathbf{A}_{2^3} = \frac{1}{\sqrt{8}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

Az 5.6. ábrán egy 16×16 képpontból álló fekete-fehér kép és annak DCT illetve DWHT transzformáltja látható. Míg az eredeti mátrix csak 0 és 1 értékeket tartalmaz, a transzformált mátrixokban valós számok szerepelnek, amelyeket szürkeárnyalatokkal jelenítettünk meg. Az 5.7. ábrán egy átlós vonalakból álló képen láthatjuk ugyanezt. Megfigyelhető, hogy mindkét transzformáció jól kihasználja a kép ismétlődéseit, szimmetriáit, hiszen a transzformáltakban sok az egyforma elem

(fehér négyzet). Ebben a speciális példában a DWHT sokkal jobban teljesít, mint a DCT, hiszen a transzformáltnak mindössze 4 eleme különbözik a többitől.

A transzformációs kódolás leggyakrabban használt megvalósítása a **részsávos kódolás**. Alapötlete, hogy a jelet nem a teljes sávszélességében kódoljuk, hanem frekvenciasávonként.

A részsávos kódolás egy gyakorlati megvalósítása látható az 5.8. ábrán. A forrás kimenetét egy szűrőbankon visszük keresztül, amely a jelet az összetevő frekvenciasávokra bontja szét. A szűrők kimenetén kapott frekvenciasávok lehetnek átlapoltak és nem átlapoltak (a teljes sávot egyszeresen lefedők). Ezután mintavételezzük az egyes szűrők kimeneteit. A mintavételi törvény szerint az elemi sávszélességek kétszeresével kell ezt megtennünk a visszaállíthatóság érdekében. Tehát a minták számát csökkenthetjük a szűrők kimenetén, hiszen itt kisebb a sávszélesség, mint a szűrőbank bemenetén volt. Ezt decimálásnak vagy alulmintavételezésnek (downsampling) nevezzük. A decimálás mértéke a szűrő bemenetén illetve kimenetén lévő sávszélességek arányától függ. A decimálás után következik a kódolás. A kódolt jelet átküldjük a csatornán, majd annak túloldalán dekódoljuk. Felülmintavételezzük (upsampling), vagyis a minták közé megfelelő számú 0-t illesztünk, így állítva vissza az eredeti másodpercenkénti mintaszámot. Végül egy szűrőbankon vezetjük keresztül a jelet, amely elvégzi a fordított transzformációt, s így a kimeneteit összegezve kapjuk a végleges jelet.

Felmerülhet a kérdés, hogy miért jó ez az egész? Az emberi érzékszervek, akár a hang, akár a (mozgó)kép érzékelés terén frekvenciafüggőek. Ezt a tényt úgy használhatjuk ki, hogy az érzékelés szempontjából fontos frekvenciasávok pontosabb rekonstrukciójára törekszünk, míg a kevésbé fontos sávokban nagyobb torzítást engedünk meg. Másrészt az egyes szűrők kimenetének szórása eltérő, ennek megfelelően rendre különböző számú bitet allokálhatunk a kvantáláshoz az (5.4) egyenlet szerint, és ezzel lényeges tömörítést érhetünk el.

5.3. Prediktív kódolás

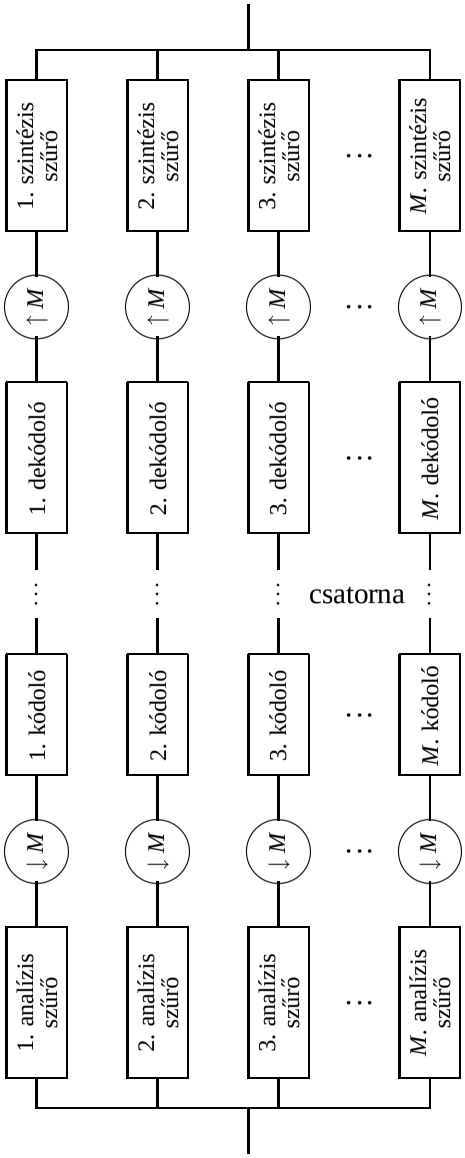
A prediktív kódolás alapelvét először egy speciális esetben vezetjük be, amikor a predikció az előző minta. Ezt hívjuk **különbségi kódolás**nak.

A különbségi kódolás alkalmazása egy adatforrásra akkor előnyös, ha a szomszédos minták közti eltérés nem túl nagy. Például digitális képek esetén a szomszédos képpontok közötti eltérés viszonylag kicsi, hacsak nem vagyunk egy él közelében.

A következő példa rávilágít, hogy mit nyerhetünk a különbségi kódolással a memóriamentes kódoláshoz képest, vagyis amikor a mintákat egymástól függetlenül kódoljuk.

5.2. példa. Legyen egy 8 bites intenzitású digitalizált kép 8 egymást követő pixelének értéke:

$$147,\ 145,\ 141,\ 146,\ 149,\ 147,\ 143,\ 145.$$



5.8. ábra. Részsávos kódoló

Ha ezeket pontról pontra kódoljuk, egyenként 8 biten, akkor ehhez 64 bit szükséges. Vegyük azonban a kódolás előtt ezen intenzitások különbségét oly módon, hogy az első képpont értékét változatlanul hagyjuk, majd minden további pixel esetén azt az értéket tároljuk, amelyet az előző képpont intenzitásához hozzáadva a saját intenzitását kapjuk. Esetünkben ez a következő lesz:

$$147,\ -2,\ -4,\ 5,\ 3,\ -2,\ -4,\ 2.$$

Világos, hogy az eredeti sorozat helyreállítható a második sorozatból. A kapott sorozatot egyszerűen vezessük át egy összegzőn.

Először 8 biten átküldjük a dekódolónak, hogy a különbségeket hány biten fogjuk ábrázolni (ez függ a legnagyobb megjelenítendő értéktől), majd szintén 8 biten átküldjük az első (változatlan) adatot. Esetünkben a legnagyobb különbségi érték az 5, ezért 4 biten fogjuk átküldeni a differenciákat (1 előjelbit + 3 adatbit). Így összesen $8+8+7\cdot 4 = 44$ bitet használunk fel, ez 30 %-kal jobb tömörítési arányt jelent.

A különbségi kódolás hatékonyságát növelhetjük predikció alkalmazásával, vagyis a múltból megbecsüljük a jelent, majd a jelen és a predikció különbségét kódoljuk. Ezt az elvet alkalmazza a veszteségmentes JPEG képtömörítő (lásd később).

Sajnos veszteséges tömörítés esetén nem ilyen egyszerű a helyzet: nem igaz, hogy adott hibahatáron belül helyreállítható ily módon az eredeti sorozat. Legyen a forrás kimenete:

$$6.2,\ 9.7,\ 13.2,\ 5.9,\ 8,\ 7.4,\ 4.2,\ 1.8.$$

Képezzük a különbségeket:

$$6.2,\ 3.5,\ 3.5,\ -7.3,\ 2.1,\ -0.6,\ -3.2,\ -2.4.$$

A veszteséges tömörítéshez használjunk egy 7 szintű kvantálót a következő kimeneti szintekkel: $-6,\ -4,\ -2,\ 0,\ 2,\ 4,\ 6$. A kvantált értékek:

$$6,\ 4,\ 4,\ -6,\ 2,\ 0,\ -4,\ -2.$$

Ha most megpróbáljuk az eredeti sorozatot helyreállítani a szokásos módon, a

$$6,\ 10,\ 14,\ 8,\ 10,\ 10,\ 6,\ 4$$

értékeket kapjuk. Az eredeti és a rekonstruált sorozat között az eltérések:

$$0.2,\ -0.3,\ -0.8,\ -2.1,\ -2,\ -2.6.$$

Látható, hogy egyre hosszabb sorozatokat vizsgálva a hiba folyamatosan nőhet.

Vizsgáljuk meg ezt az észrevételt általános esetben! Legyen a bemeneti sorozatunk $\{x_n\}$. A különbségi sorozat $\{d_n\}$, ahol $d_n = x_n - x_{n-1}$. A különbségi sorozatot kvantáljuk, s így kapjuk $\{\widehat{d_n}\}$-ot:

$$\widehat{d_n} = Q(d_n) = d_n + q_n$$

ahol q_n a kvantálási hiba. A vevő oldalán a rekonstruált sorozatot, $\{\widehat{x}_n\}$-ot úgy kapjuk, hogy a kvantált különbséget hozzáadjuk az előző értékhez:

$$\widehat{x}_n = \widehat{x}_{n-1} + \widehat{d_n}$$

Tételezzük fel, hogy az adó és a vevő ugyanarról az értékről indul, tehát $x_0 = \widehat{x}_0$. A kvantálás és a rekonstrukció első néhány lépése:

$$\begin{aligned}
d_1 &= x_1 - x_0 \\
\widehat{d_1} &= Q(d_1) = d_1 + q_1 \\
\widehat{x}_1 &= \widehat{x}_0 + \widehat{d_1} = x_0 + d_1 + q_1 = x_1 + q_1 \\
d_2 &= x_2 - x_1 \\
\widehat{d_2} &= Q(d_2) = d_2 + q_2 \\
\widehat{x}_2 &= \widehat{x}_1 + \widehat{d_2} = x_1 + q_1 + d_2 + q_2 = x_2 + q_1 + q_2.
\end{aligned}$$

Tovább folytatva, az n-edik lépés után:

$$\widehat{x}_n = x_n + \sum_{k=1}^{n} q_k.$$

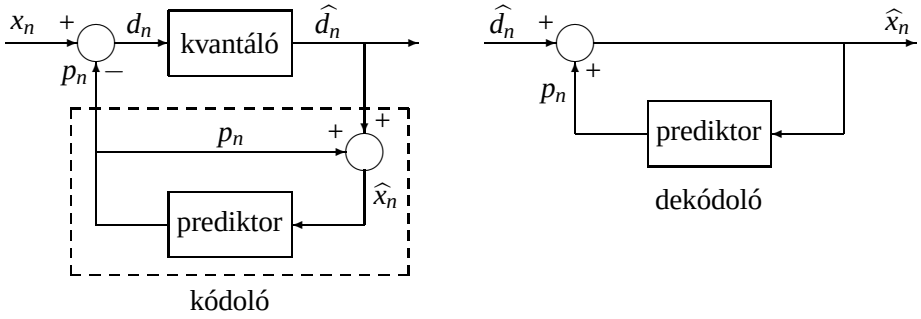
Tehát a kvantálási hiba felhalmozódik.

Vegyük észre, hogy a kódoló és a dekódoló különböző adatokon hajt végre műveleteket! A kódoló az eredeti sorozat különbségeit generálja, míg a dekódoló a kvantált különbségekkel dolgozik. Áthidalhatjuk ezt a problémát, ha az adót és a vevőt rákényszerítjük, hogy azonos adatokon dolgozzanak. A vevő az eredeti $\{x_n\}$ sorozatról csak a rekonstruált $\{\widehat{x}_n\}$ sorozatból szerezhet információt. Ez utóbbi viszont az adónak is rendelkezésére áll, így a különbségképzés műveletét a következőképpen módosíthatjuk:

$$d_n = x_n - \widehat{x}_{n-1}.$$

Ezt felhasználva az előző lépéseink így módosulnak:

$$\begin{aligned}
d_1 &= x_1 - x_0 \\
\widehat{d_1} &= Q(d_1) = d_1 + q_1 \\
\widehat{x}_1 &= \widehat{x}_0 + \widehat{d_1} = x_0 + d_1 + q_1 = x_1 + q_1 \\
d_2 &= x_2 - \widehat{x}_1 \\
\widehat{d_2} &= Q(d_2) = d_2 + q_2 \\
\widehat{x}_2 &= \widehat{x}_1 + \widehat{d_2} = \widehat{x}_1 + d_2 + q_2 = x_2 + q_2.
\end{aligned}$$



5.9. ábra. DPCM

Általánosítva:

$$\widehat{x}_n = x_n + q_n,$$

tehát a kvantálási hiba nem akkumulálódik.

Az a célunk, hogy minél kisebb különbségi értékeket kapjunk. Ehhez x_n értékét nem csak $\widehat{x}_{n-1}$ segítségével becsülhetjük, hanem a rekonstruált sorozat előző értékeinek függvényével is. Az ezt megvalósító eszköz a **prediktor**:

$$p_n = f(\widehat{x}_{n-1}, \widehat{x}_{n-2}, \ldots, \widehat{x}_0).$$

Itt a különbségképzés a

$$d_n = x_n - p_n$$

kifejezéssel történik. Továbbá a kvantált különbség

$$\widehat{d}_n = d_n + q_n,$$

és a reprodukció

$$\widehat{x}_n = \widehat{d}_n + p_n.$$

Az eredő torzítás ebben az esetben sem halmozódik. Az eddig elmondottakat megvalósító rendszer az 5.9. ábrán látható, az eljárás neve **különbségi impulzuskód moduláció** (differential pulse code modulation, **DPCM**). Az eljárást a Bell Laboratóriumban dolgozták ki a '40-es évek végén, és főként a beszédkódolásban terjedt el, így széleskörűen használják a telekommunikációban.

A különbségi kódoló rendszerek, mint a DPCM, azzal érik el céljukat, vagyis a tömörítést, hogy csökkentik a bemeneti sorozat szórását és dinamikatartományát. A szórás csökkentése attól függ, hogy a prediktor mennyire jól tudja becsülni a következő szimbólumot az előzőleg rekonstruáltakból.

A bemeneti jelek tulajdonsága változhat az időben. Ehhez tud alkalmazkodni az adaptív DPCM (**ADPCM**) rendszer. Az alkalmazkodás történhet a kódoló bemenetére érkező jel (x_n) alapján, ekkor előre adaptív módszerről beszélünk, vagy a kimenet ($\widehat{x}_n$) alapján, ez a hátra adaptív módszer. Előre adaptív esetben, mivel a dekódolónak nem áll rendelkezésére a kódoló bemeneti jele, ezért a rendszer módosított paramétereit is át kell vinni. Hátra adaptív esetben ilyen probléma

nem merül fel, hiszen a kódoló kimenetét megkapja a dekódoló. Az előre adaptív módszer megköveteli, hogy pufferelјük a bemenetet, ezzel késleltetést viszünk a rendszerbe, ami például beszédkódolás esetén nem szerencsés. Ugyanis egyetlen kódoló–dekódoló pár esetén még elviselhető a késleltetés mértéke, azonban egy telefonkapcsolatban számos DPCM kódoló és dekódoló vehet részt (pl. a nagy távolság miatt), s ekkor az eredő késleltetés már jelentős lehet.

Előre adaptív kvantáló esetén a bemenetet blokkokra bontjuk, minden lépésben kiszámoljuk az aktuális blokkra optimális kvantáló paramétereit, és átküldjük a dekódolónak kiegészítő információként. Beszédkódolás esetén a tipikus blokkhossz 16 ms, ami 8000 minta/másodperc esetén 128 mintát jelent, míg képtömörítés során általában 8×8 pixeles blokkokat használnak.

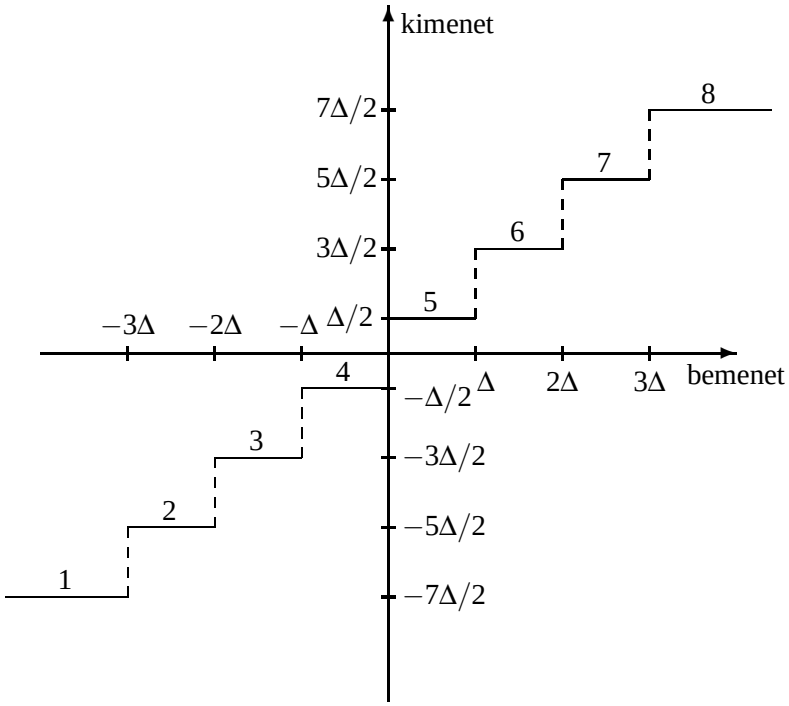
A gyakorlatban inkább a **Jayant-kvantáló** néven ismert hátra adaptív módszert használják. Itt minden egyes kvantálási intervallumhoz egy szorzótényezőt rendelünk, amely a kvantáló belső (origóhoz közeli) tartományainál 1-nél kisebb, míg a külső részeken 1-nél nagyobb. A tényezők általában az origóra szimmetrikusan helyezkednek el. Annak függvényében, hogy az aktuálisan kvantált érték melyik intervallumba esik, ezen intervallum szorzótényezőjével korrigáljuk a lépésköz értékét. Ha a kvantált érték kicsi, akkor finomítjuk a lépésközt, míg nagy értékek esetén nagyobb lépésközt fogunk alkalmazni.

5.3. példa. Az 5.10. ábra egy 8 szintű kvantálót ábrázol. Legyenek az egyes intervallumok szorzótényezői: $M_1 = M_8 = 1.2$, $M_2 = M_7 = 1$, $M_3 = M_6 = 0.9$, $M_4 = M_5 = 0.8$, a kezdeti lépésköz pedig $\Delta_0 = 0.5$. A kvantálandó sorozat: 0.1, -0.2, 0.2, 0.1, 0.2, 0.5, 0.9, 1.5, 1.0, 0.9, ... Az első bemeneti adatot a kezdeti 0.5 lépésközzel az 5. szintre kvantáljuk, a kimenet értéke 0.25 lesz, a hiba 0.15. Így az új lépésköz $\Delta_1 = M_5\Delta_0 = 0.8 \cdot 0.5 = 0.4$ lesz. A következő adat a 4. intervallumba esik, most a lépésköz 0.4, tehát a kimeneten -0.2 jelenik meg, és a lépésköz új értéke $\Delta_2 = M_4\Delta_1 = 0.32$. Így folytatva az eljárást, az eredmény táblázatos formában az 5.11. ábrán látható.

Vegyük észre, hogyan alkalmazkodik a kvantáló az inputhoz. Kezdetben a bemeneti értékek kicsik, ezért a lépésköz folyamatosan csökken, ezzel egyre jobb közelítést biztosítva. Majd nagyobb bemeneti értékek következnek, így a lépésköz is növekszik. Megfigyelhetjük, hogy a hiba értéke viszonylag nagy az átmeneti szakaszban.

A kvantálási lépésközt a konkrét megvalósítások természetesen véges pontossággal ábrázolják, ezért el kell kerülnünk azt a szituációt, hogy a lépésköz folyamatos csökkentésével, az nullává váljon. Be kell vezetnünk egy $\Delta_{\min}$ értéket, amelynél nem választjuk kisebbnek a lépésközt. Hasonló okok miatt egy $\Delta_{\max}$ is szükséges.

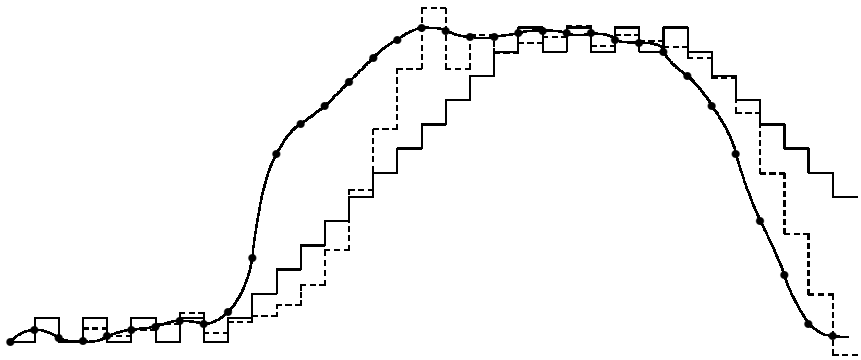
A DPCM egyszerűbb formája, a **delta moduláció** (DM). Ez egy 1 bites (2 szintű) kvantálóval rendelkező DPCM. A 2 szintű kvantálóval csak $\pm\Delta$ kimeneti értékeket állíthatunk elő, vagyis (egy folytonos jel mintavételezésével adódó) két minta közötti eltérést csak ezzel reprezentálhatjuk. Amennyiben egy adott bemeneti sorozatban a minták közötti különbség nagyon eltér Δ-tól, a kimeneten egy



5.10. ábra. Jayant-kvantáló 8 kvantálási szinttel

n	Δ_n	bemenet	szint száma	szint értéke	hiba	új lépésköz
0	0.5	0.1	5	0.25	0.15	$\Delta_1 = M_5\Delta_0$
1	0.4	−0.2	4	−0.2	0.0	$\Delta_2 = M_4\Delta_1$
2	0.32	0.2	5	0.16	0.04	$\Delta_3 = M_5\Delta_2$
3	0.256	0.1	5	0.128	0.028	$\Delta_4 = M_5\Delta_3$
4	0.2048	−0.3	3	−0.3072	−0.0072	$\Delta_5 = M_3\Delta_4$
5	0.1843	0.1	5	0.0922	−0.0078	$\Delta_6 = M_5\Delta_5$
6	0.1475	0.2	6	0.2212	0.0212	$\Delta_7 = M_6\Delta_6$
7	0.1328	0.5	8	0.4646	−0.0354	$\Delta_8 = M_8\Delta_7$
8	0.1594	0.9	8	0.5578	−0.3422	$\Delta_9 = M_8\Delta_8$
9	0.1913	1.5	8	0.6696	−0.8304	$\Delta_{10} = M_8\Delta_9$
10	0.2296	1.0	8	0.8036	0.1964	$\Delta_{11} = M_8\Delta_{10}$
11	0.2755	0.9	8	0.9643	0.0643	$\Delta_{12} = M_8\Delta_{11}$

5.11. ábra. A 5.3. példa végeredménye



5.12. ábra. Lineáris delta moduláció

állandó torzítás jelentkezik. Ezt gyakoribb mintavételezéssel ellensúlyozhatjuk. A DM rendszerekben a legnagyobb előforduló frekvenciának nem csak a kétszeresével, hanem akár a százszorosával is mintavételeznek a torzítás alacsony szinten tartása végett (például jó minőségű A/D átalakítókban, „1 bites átalakítók").

A fix lépésközű kvantálót tartalmazó DM rendszereket **lineáris delta modulátor**nak nevezzük. Az 5.12. ábrán (folytonos vonal) megfigyelhetjük, hogy torzítás két okból jelentkezik. Azokon a részeken, ahol a bemenet hozzávetőleg konstans, a kimenet oszcillál Δ-val (granular regions). Ezt a hibát Δ csökkentésével kompenzálhatjuk. Azokban a tartományokban, ahol a bemenet túl gyorsan növekszik vagy csökken, a kimenet nem tud lépést tartani (slope overload regions), ezért ezen a Δ lépésköz növelésével segíthetünk. Látható, hogy a kétféle torzítás egyidejű javítása éppen ellentétes feltételeket támaszt Δ-ra vonatkozóan, ezért fix lépésközzel nem oldható meg.

Ezen segít a lépésköz adaptív megválasztása. A közel konstans tartományokban kis lépésközt választunk, míg gyors változások esetén növeljük Δ-t. A bemenet lokális tulajdonságához alkalmazkodó rendszerek közül az egyik legkedveltebb a konstans faktorral alkalmazkodó delta modulátor (constans factor adaptive delta modulation, CFDM). A legfontosabb feladat annak megállapítása, hogy éppen milyen jellegű tartományban vagyunk. A közel állandó részeken a kvantáló kimenete majdnem minden mintánál előjelet vált, míg a gyorsan változó intervallumokban a kimenet előjele állandó. A legegyszerűbb esetben csak a megelőző mintáig tekintünk vissza a tartomány jellegének eldöntésére. Jelöljük s_n-nel a delta modulátor n-edik időegységbeli „lépését" és Δ_n-nel az itt érvényes lépésközt ($s_n = \pm\Delta_n$). Ekkor az $n+1$-edik időegységbeli lépésközt a következőképpen kapjuk:

$$\Delta_{n+1} = \begin{cases} M_1\Delta_n, \text{ ha } \operatorname{sgn} s_n = \operatorname{sgn} s_{n-1} \\ M_2\Delta_n, \text{ ha } \operatorname{sgn} s_n \neq \operatorname{sgn} s_{n-1} \end{cases}$$

ahol $1 < M_1 = \frac{1}{M_2} < 2$. Az 5.12. ábrán szaggatott vonallal ábrázoltuk az adaptív delta modulátor kimenetét, amelynek paraméterei: $M_1 = 1.8$, $\Delta_{\min} = 0.25\Delta_0$, $\Delta_{\max} = 2.5\Delta_0$. A memória növelésével, vagyis például 2 mintára visszatekintve

tovább csökkenthető a torzítás:

$$\Delta_{n+1} = \begin{cases} M_1\Delta_n, \text{ ha } \operatorname{sgn} s_{n-2} \neq \operatorname{sgn} s_{n-1} \neq \operatorname{sgn} s_n \\ M_2\Delta_n, \text{ ha } \operatorname{sgn} s_{n-2} = \operatorname{sgn} s_{n-1} \neq \operatorname{sgn} s_n \\ M_3\Delta_n, \text{ ha } \operatorname{sgn} s_{n-2} \neq \operatorname{sgn} s_{n-1} = \operatorname{sgn} s_n \\ M_4\Delta_n, \text{ ha } \operatorname{sgn} s_{n-2} = \operatorname{sgn} s_{n-1} = \operatorname{sgn} s_n \end{cases}$$

ahol pl. $M_1 = 0.4 < M_2 = 0.9 < M_3 = 1.5 < M_4 = 2.0$.

A beszédkódolásban kívánatos, lassabb alkalmazkodást tesz lehetővé a folytonosan változó meredekségű delta moduláció (continuously variable slope delta modulation, CVSD). Ez csökkenti a közel konstans esetben elkövetett hibát, viszont növeli a gyors változások esetén bekövetkező hibát. Az adaptív lépésköz számítására szolgáló képlet:

$$\Delta_{n+1} = \beta\Delta_n + \alpha_n\Delta_0,$$

ahol β egy 1-nél alig kisebb konstans, α_n értéke pedig 1, ha a kvantáló legutóbbi K darab kimeneti értékéből J darabnak azonos volt az előjele, egyébként 0. Tehát egy K hosszú ablakon keresztül figyeljük a bemenetet, és ebből következtetünk annak lokális viselkedésére. Jellemző értékek: $J = K = 3$.

5.4. Alkalmazások

Beszédtömörítés

A beszédtömörítő eljárások területén erőteljes fejlődés volt megfigyelhető az elmúlt évtizedekben. A minél alacsonyabb bitsebesség mellett minél jobb minőségű hang átvitelének igénye főleg a vezetékes távközlés, majd újabban a mobil távközlés részéről fogalmazódott meg. Kezdetben az eddigiekben megismert általános célú eljárásokat alkalmazták.

Az ismertetésre kerülő megoldásokat a nyilvános távközlőhálózatokban használt **impulzuskód moduláció**hoz (Pulse Code Modulation, PCM) hasonlítjuk. Határozzuk meg ennek bitsebességigényét! A „telefon-minőség” viszonylag kis sávszélességgel is beéri. Az analóg 3.4 kHz sávszélességű beszédjelből (némi rátartással) másodpercenként 8000 mintát veszünk, és 8 bittel kvantáljuk, így $8000 \cdot 8 = 64$ kbps-ot kapunk. Az 1972-ben megjelent **G.711**-es távközlési szabvány logaritmikus PCM kódolást használ, azaz kompanderes kvantálót (mint erről az 5.1. szakaszban szó volt: μ-law, A-law).

A **beszédtömörítés**sel szemben támasztott megnövekedett kommunikációs igény elkerülhetetlenné tette egy új eljárás kifejlesztését, amelynek kisebb a bitsebességigénye, de emellett jó minőségű, jól érthető beszédátvitelt tesz lehetővé. A kívánalmaknak megfelel az 1984-es **G.721** szabvány, amely 32 kbps-os átviteli sebességével megduplázta a kiépített vonalakon folytatható beszélgetések számát. A G.721 adaptív különbségi kódoláson (ADPCM) alapul. Kihasználja az emberi

beszéd különböző időpontokban megfigyelt mintái közötti hosszú- és rövidtávú korrelációt (2–20 ms). A kvantálási lépcsők méretét az előző minták alapján változtatja. A kódoló késleltetése kisebb mint 2 ms, és számos központon keresztülhaladó jel esetén is megfelelő minőséget biztosít.

A mobiltelefóniában további tömörítésre van szükség. Ezt részben a fenti eljárások finomításával, részben más technikával érhetjük el. Ezek a módszerek az emberi hangképzés modelljét használják fel. A hangszalagok zöngét (alaphangot) képeznek, majd a hangképző út modulálja azt. A mesterséges hangképzéshez tehát elő kell állítanunk egy gerjesztő jelet, és ezt kell modulálnunk a hangképző út szerepének megfelelően.

A kódoló a beszédet szegmensekre osztja, és minden szegmensre meghatározza a gerjesztő jelet, valamint a hangképző utat modellező szűrő paramétereit. A **szintetizált** beszédtömörítők esetén a gerjesztő jelet nem küldjük át, csak néhány paraméterét, és ebből a dekódoló előállítja a megfelelő gerjesztő jelet, majd ezzel hajtja meg a saját szűrőbankját, amelynek paramétereit a kapott adatoknak megfelelően állította be.

A tömörítendő beszéd minden szegmensét egy sávszűrő bankon vezetjük keresztül, amelyet analízis szűrőnek nevezünk. Az analízis szűrő kimenetének energiáját meghatározott időközönként (általában másodpercenként 50-szer) mintavételezzük, és átküldjük a dekódolónak. (Az emberi beszéd kb. 20 ms-os egységen belül stacionáriusnak tekinthető.) Digitális esetben az energiát közelíthetjük a szűrő kimeneti jelének átlagos négyzetösszegével, míg analóg esetben a jel burkológörbéjének mintavételezésével érhetünk célt. Az energia meghatározásával egyidejűleg azt is el kell dönteni, hogy az adott beszédszegmens zöngés vagy zöngétlen hangot tartalmaz-e. A zöngés hangok álperiodikus viselkedést mutatnak, az alapharmonikus frekvenciáját hangmagasságnak hívjuk. A kódoló a becsült hangmagasság értékét szintén átküldi a dekódolónak. A zöngétlen hangok zajszerű szerkezetet mutatnak. A szintetizált beszédtömörítést tekinthetjük egy olyan vektorkvantálásnak, ahol a kvantálást az egyes szegmensek Fourier-transzformáltjainak a terében végezzük.

A dekódolóban a hangképző szerveket sávszűrő bankkal modellezzük, ez a szintézis szűrő, amely megegyezik az analízis szűrővel. Annak megfelelően, hogy zöngés vagy zöngétlen hangot kell-e visszaállítani, a szintézis szűrő bemeneteként egy, a hangmagasságnak megfelelő periodikus jelgenerátort vagy egy álzajgenerátort használunk. A jel amplitúdóját a becsült energiának megfelelően állítjuk be.

A hangképző út egy változtatható méretű rezonátorüregként fogható fel, ezért számos rezonanciafrekvencia lehetséges, de nem az összes frekvenciaösszetevő egyformán fontos. A rezonanciafrekvenciákat formánsoknak nevezzük. A formáns frekvenciája hangonként eltérő, de férfiak esetén 200–800 Hz, míg nők esetén 250–1000 Hz közötti tartományba esik. Ezt a jellemzőt használja fel a **formáns beszédkódoló**, amely meghatározza a formánsok értékének közelítését (általában 4 formáns megkülönböztetése elegendő), valamint ezek sávszélességét, majd a vevő oldalon a gerjesztő jelet hangolható szűrőkön vezetik keresztül, amelyeket a formánsok frekvenciájára és sávszélességére hangolnak.

Lineáris prediktív kódolás (LPC) esetén egy lineáris szűrőt használunk, mely az $\{a_i\}$ együtthatók optimális megválasztásával az előző minták alapján próbál becslést adni az aktuális mintára úgy, hogy a négyzetes torzítás minimális legyen:

$$x_n = \sum_{i=1}^{M} a_i x_{n-i} + G\varepsilon_n, \tag{5.5}$$

ahol G a szűrő energiája (gain), ε_n pedig a különbségi jel.

Az LPC-nek két változata van. Az egyszerűbb, nem-szintetizált esetben (ez még „hagyományos" eljárásnak tekinthető abban az értelemben, hogy nem használja fel az emberi beszéd modelljét) közvetlenül a különbségi jel mintavételezett értékét (ε_n) küldjük át a vevőnek. Ekkor (5.5)-ben x_n helyén az 5.3. szakaszban megismert módon a reprodukált $\widehat{x}_n$ értékek állnak.

A jobb tömörítésű módszer a vevőoldalon szintetizált beszédet állít elő, vagyis csak a szűrő $\{a_i\}$ paramétereit és a G energiát kell átküldenünk, és ebből generálja a vevő a saját szűrőjével a beszédet úgy, hogy az ε_n sorozatot helyettesíti egy mesterséges gerjesztő jelsorozattal.

Az **LPC–10** algoritmus esetén a kódolónak a következő adatokat kell átküldenie: zöngés vagy zöngétlen-e a hang (1 bit); hangmagasság, amelyet 60 lehetséges értékre kvantálnak egy logaritmikus karakterisztikájú kompanderes kvantáló segítségével (6 bit); hangképző utat modellező szűrő paraméterei, amely zöngés esetben 10-edrendű, míg zöngétlen esetben 4-edrendű predikciót jelent (31 ill. 10 bit a szűrő együtthatóira és 5 bit az energiára). A szűrő nagyon érzékeny az 1-hez közeli együtthatók hibájára. Mivel az első néhány együttható általában 1-hez közeli, ezért a szabvány nemegyenletes kvantálást ír elő ezekre. A zöngétlen esetben fennmaradó 21 bitet hibavédelemre használják. A szinkronizációhoz szükséges 1 további bittel együtt ez összesen 54 bitet tesz ki. Egy szegmens hossza 22.5 ms, tehát az LPC–10 algoritmus 2.4 kbps átvitelét teszi szükségessé.

A dekódoló zöngés szegmens esetén a szűrő gerjesztőjeleként egy előre tárolt hullámformát használ. A hullámforma 40 minta hosszúságú (a mintavételezés 8 kHz-cel történik), ezért a hangmagasságtól függően csonkolja vagy nullákkal tölti ki. Ha a szegmens zöngétlen, akkor a szűrőt egy álvéletlen generátor jelével hajtja meg (fehér Gauss-zaj). Az LPC–10 kódoló érthető, de nem túl jó minőséget biztosít 2.4 kbps bitsebességen. Az, hogy csak kétféle gerjesztőjelet alkalmaz a szűrő bemeneteként, gépi jellegű hangot eredményez. Ez különösen zajos környezet esetén jelent problémát, ugyanis a kódoló a környezeti zaj miatt a zöngés szegmenseket is zöngétleneknek fogja nyilvánítani.

A természetesen hangzó beszéd előállításának egyik legfontosabb összetevője a gerjesztőjel, ugyanis az emberi fül különösen érzékeny a hangmagasság hibájára. Az LPC gyenge pontját küszöbölik ki a **szinuszos kódoló**k. Ezek gerjesztőjelként szinuszos összetevőket használnak. Egy elemi összetevőt három paraméter határoz meg: az amplitúdó, a frekvencia és a fázis. Ezek közül, ha a jelet egy lineáris szűrőn vezetjük keresztül, a frekvencia nem változik. Mivel a hangképző út modellezésére használt szintézis szűrő lineáris, ezért ezen egy szinuszos jelet átvezetve csak az amplitúdó és a fázis változik. Megállapíthatjuk, hogy a gerjesztő jel

leírásához szükséges paraméterek száma megegyezik a szintetizált beszédet reprezentáló paraméterek számával. Így ahelyett, hogy külön-külön kiszámítanánk és átvinnénk a gerjesztő jel és a hangképző út szűrőjének paramétereit és a gerjesztőjelet átvezetnénk a szűrőn, megtehetjük, hogy közvetlenül a beszédet állítjuk elő a vevő oldalon a szinuszos összetevőkből.

A szinuszos kódolók is szegmensekre bontják a beszédet. Amennyiben a vevő oldalon egy-egy szegmensen belül a többitől függetlenül szintetizáljuk a beszédet, úgy az a határokon nem lesz folytonos, ami jelentősen rontja a minőséget. Ezért a szinuszos kódolók különböző interpolációs algoritmusokat használnak a szegmensek közötti átmenetek finomítására.

Azonban ez még nem elég, mert hiába közelítjük jól a hangmagasságot, amíg a hangképző utat modellező szűrő bemenetére periodikus jelként csak egyetlen frekvenciaösszetevőből álló jelet adunk (ahogyan LPC esetén tesszük), addig csak zümmögő orrhangot kapunk. Ezt a problémát oldják meg a **szintézis általi analízis** alapú kódolók, amelyek a gerjesztő jelet egy optimalizáló hurokban határozzák meg. A kódoló egyben dekódolót is tartalmaz, amely a szintézist végzi. Az így szintetizált jelből kivonja a bemenő jelet, s az előálló hibajelet minimalizálja. A szintézis általi analízis alapú kódolók egyike a **többfrekvenciás lineáris prediktív kódoló** (MultiPulse Linear Predictive Coding, MultiPulse Excitation, MPE) eljárás, amely minden szegmensben egyidejűleg több frekvenciaösszetevőt használ. A lehetséges frekvenciaösszetevő-mintát egy kódtáblából választja ki olymódon, hogy a valódi és a szintetizált beszédszegmens közötti, az emberi hallás tulajdonságainak megfelelően súlyozott eltérés minimális legyen (ez egy speciális vektorkvantálónak tekinthető). Az MPE 6.4 kbps átviteli sebességgel jó minőségű beszédet produkál. Ennek továbbfejlesztett változata a **kóddal gerjesztett lineáris prediktív kódoló** (Code Excited Linear Prediction, CELP), ahol a lehetséges mintákat tartalmazó (szűk) kódtábla helyett számos gerjeszetőjelet engedünk meg (egy nagyon nagy méretű sztochasztikusan kitöltött kódtáblát használunk). Minden szegmens esetén a kódoló megkeresi azt a gerjesztővektort, amelynek alkalmazásával a legtökéletesebb szintetizálás lehetséges. A CELP jó minőségű beszédet biztosít 4.8 kbps-os sebességgel, annak árán, hogy kimerítő keresést kell végezni egy tipikusan 1024 méretű kódtáblában.

A **GSM** rendszer teljes sebességű kódolása az MPE-hez hasonló módszeren, a szabályos impulzus gerjesztésen (Regular Pulse Excitation, RPE) alapul. MPE esetén egy rövid perióduson belül (5–15 ms) meghatározott számú (M db) impulzust adunk. Egy adott impulzus amplitúdóját és helyét az előző M darab impulzus alapján határozzuk meg. RPE esetén az impulzusok száma szintén rögzített, és az MPE-vel szemben ezek helye sem választható szabadon. Ezáltal szuboptimális megoldáshoz jutunk, de egyben egyszerűsödik is a kódoló. Az RPE-n alapuló rövid idejű szintézis szűrőt a GSM rendszer egy hosszú idejű becslő hurokkal (Long Term Prediction) egészíti ki. (Ez a közepes bitsebességű kódolók, mint pl. az RPE esetén nem létfontosságú, de általában alkalmazzák a minőség növelése érdekében.) 8000 minta/s mintavételezési sebesség mellett 13 kbps-os kódsebességet

biztosít, ugyanis 20 ms-onként 260 bitnyi paramétert (tömörített információt) állít elő.

A GSM félsebességű kódolása a CELP családba tartozó **vektorösszeggel gerjesztett lineáris prediktív kódoló** (Vector-Sum Excited Linear Prediction, VSELP) megoldáson alapul. A gerjesztésre a kódtáblából származó értékek szolgálnak, melyek részben rögzítettek, részben adaptív jellegűek. A zöngés jelleg mértékének vizsgálata alapján a kódoló elrendezése négyféle lehet. A 0 kategória jelenti a zöngétlen, az 1–3 pedig az egyre növekvő mértékben zöngés jellegű beszédszegmenst. A struktúra 5 ms-os ún. alkeretenként változhat. A kódtáblában kimerítő keresést alkalmaznak az alkeretenkénti gerjesztősorozat meghatározásához, és a minimális hibajelet adó gerjesztő kódot választják ki. Ha a beszéd zöngés, akkor hosszú idejű becslőt is alkalmaznak. (Ez a kis bitsebességű kódolóknál, mint pl. a VSELP nélkülözhetetlen.) A félsebességű kódoló a teljes sebességű változat 260 bitje helyett szegmensenként csak 112 bitet használ, ez 5.6 kbps-nak felel meg.

Hangtömörítés

Az előzőekben az emberi beszéd minél gazdaságosabb, minél kisebb bitsebességet igénylő átvitelére koncentráltunk. Azonban a továbbítandó vagy tárolandó hanginformáció természetesen nemcsak beszéd lehet, hanem például zene is. Az ezen a területen használható módszereket tekintjük át az alábbiakban.

A hangtömörítésben mércének számító CD-minőség azt jelenti, hogy 44100 Hz-cel mintavételezünk. Ez a mintavételi tétel értelmében a legfeljebb 22050 Hz frekvenciájú hangok visszaállítását teszi lehetővé (valójában mintavételezés előtt a 20 kHz-es sávra szűrnek). A mintákat 16 bites kvantálóval kvantáljuk. Ez $44100 \cdot 16 \cdot 2 \approx 1.4$ Mbps átviteli sebességet igényel (a 2-es szorzó a sztereó átvitel 2 külön csatornája miatt kell).

A veszteségmentes tömörítési módok (pl. Huffman-kódolás, LZW) itt nem igazán hatékonyak, az eredmény általában az eredeti méretének 90 %-a körül van.

A modern, **hangtömörítés**re kifejlesztett eljárások az emberi hallás sajátosságait kihasználva érnek el az általános célú módszereknél jobb tömörítési arányt, így amellett, hogy veszteséges eljárások, a veszteség mértéke nem is egyenletes sem a frekvencia-, sem az időtartományban. Mivel itt a közel tökéletes hang visszaállítása a cél, ezért a hangtömörítésben nem megengedett a szintetizálás.

Az emberi hallás tartománya 20 Hz – 20 kHz, a legnagyobb érzékenysége 2 és 4 kHz között van, felbontása (a kvantálási lépcső) frekvenciafüggő. Ebből következően két azonos intenzitású, de különböző frekvenciájú hang különböző hangosságérzetet kelt a hallgatóban, és azokban a tartományokban, amelyekben fülünk kevésbé érzékeny, jobban elviseljük a torzítást. Hallásunkra jellemző két elfedési jelenség. Az egyik a frekvenciatartománybeli elfedés, melynek lényege, hogy egy adott frekvenciájú, nagy intenzitású hang (maszkoló hang) a vele egy időben szóló és közeli frekvencián lévő kisebb intenzitású hangokat elfedi, vagyis azok nem hallhatóak. A másik az időtartománybeli elfedés: egy adott frekvenciájú nagy

intenzitású hang a közeli frekvencián lévő kisebb intenzitású hangokat nemcsak akkor fedi el, amikor együtt szólnak, hanem kis ideig még a nagy intenzitású hang bekapcsolása előtt (kb. 2 ms-ig) vagy kikapcsolása után (kb. 15 ms-ig) sem halljuk a kisebb intenzitásúakat.

Mivel az emberi hallás legjobban a frekvenciatartományban modellezhető, ezért a hangtömörítő eljárások frekvenciatartománybeli analízissel dolgoznak. A fentiek alapján nem minden frekvenciaösszetevőt kell átvinni, és a kódolandókat sem azonos pontossággal kell kvantálni.

Az eredetileg mozgóképtömörítésre kifejlesztett MPEG–1 szabvány hangtömörítő része jól kihasználja az emberi hallás sajátosságait. A hangtömörítésre három hasonló eljárást definiál, és ezeket Layer 1,2,3-nak nevezi.

Az egyszerűbb Layer 1 és 2 eljárás analízis szűrőbankja a bemeneti jelet 32 részsávba képezi le. A mintavételezés 32, 44.1 vagy 48 kHz-cel történhet. Ezután részsávonként egy transzformációs kódolás következik. A maszkoló és a maszkolt hang(ok) eltérését (Signal-to-Mask Ratio, SMR) Layer 1 esetén 512 pontos, míg Layer 2 esetén 1024 pontos gyors Fourier-transzformációval (FFT) számítja. Mindkét módszer 12 mintát (keret) kódol együtt minden sávban, de a Layer 2 a megelőző és a követő 12 mintát is megvizsgálja az időbeli maszkoláshoz, valamint a skálafaktorokat is hatékonyabban kódolja. Dinamikus bitallokációval kiválasztja a lehetséges 15 kvantáló egyikét minden egyes részsáv számára úgy, hogy az a skálafaktor (a legnagyobb amplitúdójú jel) és az SMR közötti bitmegosztást tekintve optimális legyen. A CD-minőségű hang átviteléhez az MPEG–1 Layer 1 eljárásnak 384 kbit/s-ra, míg a Layer 2-nek 256 kbit/s-ra van szüksége.

A napjainkban oly nagy népszerűségnek örvendő MPEG–1 Layer 3 eljárás a Layer 1,2-höz képest lényeges, a hangminőséget javító eltéréseket tartalmaz. A 32 részsáv mindegyikét további 6 vagy 18 frekvenciaösszetevőre bontja módosított diszkrét koszinusztranszformáció (MDCT) felhasználásával. A frekvenciakomponensekre ezután egy nemegyenletes kvantálót alkalmaz, majd a kimenetet a lehetséges 18 Huffman-kódtábla egyikével kódolja. Míg rögzített bitsebesség mellett minden keret azonos számú bájtot tartalmaz, addig a Layer 3 esetében meg lehet azt tenni, hogy az egyik keretet kevésbé töltjük fel (ha nincs rá igény), és a maradék helyre a következő keret bitjeit tesszük. Így egy keret adatai adott határokon belül átcsúszhatnak a szomszédos keretekbe.

Az MPEG szabványok csak a dekódolót szabványosítják, a kódolót nem. Persze az adott dekódoló-részegységhez sok esetben létezik optimális kódolórész (pl. IDCT a dekódolóban $\longrightarrow$ DCT a kódolóban, ismertek a kódpontok a dekódolóban $\longrightarrow$ kiszámíthatóak a kvantálási tartományok a kódolóban). Bizonyos esetekben (pl. az emberi hallás modelljének felhasználása vagy a bitallokáció esetén) már nagy a szabadsági fok, ebből következnek a különböző MPEG kódolók közötti minőségi eltérések.

Képtömörítés

Képtömörítés esetén alkalmazhatunk egyértelműen dekódolható (veszteségmentes), vagy hűségkritériumon alapuló (veszteséges) módszereket. A veszteséges módszerek a veszteségmenteseknél egyes esetekben akár egy nagyságrenddel is jobban tömöríthetnek, észrevehető minőségromlás nélkül. Amennyiben a minőségi megkötések nem nagyon szigorúak, még ennél is többet nyerhetünk a veszteséges módszerek alkalmazásával. Ahhoz, hogy a minőségromlás és a tömörítés mértéke között meg tudjuk találni az egyensúlyt, szükségünk van e két jellemző kvantitatív mérésére.

A tömörítés mértékének megállapításához mérnünk kell a tömörítetlen és a tömörített információ méretét, majd ezek hányadosát kell vennünk. A tömörített információ a legtöbbször bitfolyamként jelenik meg, ennek mérete a benne levő bitek száma. A tömörítetlen információ mennyiségének mérése nehézségekbe ütközhet folytonos értékkészlet vagy értelmezési tartomány esetén. Ilyenkor az információ mennyiségét vehetjük a kódoló bementetén megjelenő jel — ami általában egy bitfolyam — méretének. Célszerűen ez a bitfolyam az eredeti jellel azonos vizuális élményt adó, de már mintavételezett és kvantálással véges értékkészletűvé alakított jel bináris ábrázolása. A tömörítetlen információ mennyiségének becsléséhez kulcsfontosságú a megfelelő mintavételezés és kvantálás kiválasztása. Ezt a legtöbb esetben szabványok határozzák meg. Ilyen szabványokkal találkozhattunk már a hangtömörítésnél is: a CD-minőségű hang a maga 44.1 kHz-es mintavételezésével és 16 bites lineáris kvantálásával pontosan ilyen volt. Képtömörítésnél a felbontást és a színmélységet kell meghatároznunk. Videotömörítés esetén emellett meg kell adnunk a képfrekvenciát is. Képtömörítésnél gyakorlatilag bármilyen felbontás használható, míg videotömörítésnél a nemzetközi televíziós szabványokhoz alkalmazkodva 720×480 (NTSC), 768×576 (PAL) a szokásos felbontás, de a HDTV esetén akár 1920×1080 is lehet. Az ezekhez tartozó képfrissítési frekvenciák pedig 30 Hz, 25 Hz illetve 60 Hz. A színtérbeli kvantálást a színmélység, vagyis az egy képpontra jutó bitek száma jellemzi. Általánosan alkalmazott a 8 és a 24 bites színmélység.

A ma használt képernyők legnagyobb része a három alapszín additív keverésével dolgozik. Ezek a piros, a zöld és a kék. Ezért egy képpont színét — az angol színelnevezések kezdőbetűi alapján — az ún. (R,G,B) értékekkel szokás megadni, amelyek azt mutatják, hogy a három alapszínt hogyan kell keverni a kívánt fényerő és színérzet eléréséhez. Ezek értéktartománya a 0 és 255 közötti egész számokból áll. Ilyen színkezelés esetén 2^{24} különböző színt tudunk reprodukálni, ami a legtöbb esetben kielégítő.

A veszteséges kép- és videotömörítési módszereket az emberi látásról meglevő ismereteket felhasználva alakították ki. Az információmennyiség csökkentése érdekében a kép azon részleteit kódoljuk kis pontossággal, amelyekre a szemünk kevéssé érzékeny, és azokat kódoljuk majdnem eredeti minőségben, amelyekre a szemünk igen érzékeny. Mivel szemünk érzékenysége a fényerőre jóval nagyobb, mint a színre, érdemes ezeket az adatokat elkülöníteni egymástól. Az (R,G,B) ér-

tékekből az alábbi transzformációval kaphatóak az ún. (Y,C_b,C_r) színkoordináták.

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ C_b &= B - Y \\ C_r &= R - Y \end{aligned}$$

Y-t **luminanciá**nak nevezzük, és a fényességérzetet írja le, C_b-t és C_r-t **krominanciá**nak nevezzük, ezek a színérzetet írják le. Bár szemünk nem egyenletesen érzékeny a fényességérzetre, ennek a színkoordináta-rendszernek az Y együtthatójában többé-kevésbé állandó a szemünk érzékenysége. További előny, hogy ezek az értékek is 0 és 255 közötti egész számok, és belőlük az (R,G,B) számértékek egyszerűen előállíthatóak.

Érdekességként megemlítjük, hogy az (R,G,B) színinformáció transzformálását az eredeti (analóg) televíziós szabványokban az tette szükségessé, hogy az adás fekete-fehér vevőkészüléken is nézhető lehessen. A luminancia jelet ugyanazon a frekvencián szórják, mint a régi fekete-fehér jelet, így azt egy fekete-fehér televíziókészülékkel is venni lehet. A két krominanciajelet pedig nagyobb frekvencián és szűkebb sávban (hiszen ennek torzítására kevésbé érzékeny a szemünk) adják.

Szemünknek sok egyéb olyan tulajdonsága is van, amit kép- és videotömörítéskor kihasználhatunk. Megfigyelhetjük, hogy a nagyobb térbeli frekvenciájú összetevőkre kevésbé érzékeny a szemünk mint az alacsonyabb frekvenciájúakra. Például egy halvány sűrű mintázatot sokkal kevésbé veszünk észre, mint egy ugyanolyan halvány, de ritkás mintát. Tömörítésnél, ha a kép kétdimenziós Fourier-transzformáltját vesszük, akkor ebben a magasabb frekvenciás összetevők torzítása nem okoz észrevehető hibát, míg az alacsonyabb frekvenciájú összetevők torzítása igen feltűnő a képen. Pontosabban, a kétdimenziós Fourier-transzformált frekvenciatartományban a szem érzékenysége a két frekvencia összegének növekedésével monoton csökken.

Kísérleti tény továbbá, hogy a térbeli és időbeli frekvenciák érzékelése összefügg: egy rövid ideig látott képen csak az igen kicsi térbeli frekvenciájú összetevőket figyeljük meg, a finomabb mintázatokat csak hosszabb ideig látott képen vagyunk képesek érzékelni. Veszteséges videotömörítésnél ennek megfelelően megengedjük, hogy egy hirtelen megjelenő képrészletnek az első néhány képkockán csak az alacsonyabb térbeli frekvenciás összetevői jelenjenek meg. A magasabb frekvenciás összetevőket elég csak ezután megjeleníteni.

A GIF (Graphics Interchange Format) formátumot, amelyet grafikus képek tömörítésére lehet használni, a Compuserve Information Service foglalta szabványba. A GIF formátum a képpontok színét úgynevezett **indexelt** formában tárolja. Amikor minden képpont színét az előzőekben bemutatott színkoordináta-rendszerek valamelyikében adjuk meg, **folytonos tónusú tárolás**ról beszélünk. Megtehetjük azonban, hogy a képen használt színek valamilyen leírását egy palettába gyűjtjük ki, és a képpontok megadásakor a színekre a palettabeli indexükkel hivatkozunk. Ezt nevezzük **indexelt tárolás**nak.

Ilyen tárolással igen nagy tömörítést érhetünk el. Gondoljuk meg, hogy ha egy képet az (R,G,B) színkoordinátákkal, koordinátánként 8–8 bitet felhasználva folytonos tónusúan tárolunk, akkor minden képponthoz 24 bitet kell megadnunk. Ha ugyanezt a képet egy 256 színű palettával indexelten tároljuk, akkor a palettán kívül képpontonként csak egy 8 bites indexet kell megadnunk. Ez — amennyiben a képméret nem túl kicsi — körülbelül $1:3$ tömörítést jelent.

Természetesen bármilyen kép tárolható indexelt formában, legfeljebb igen nagy méretű paletta szükséges. Az indexelt tárolás lényege viszont épp a viszonylag kis méretű paletta alkalmazása. A képek folytonos tónusú tárolással kerülnek a tömörítőalgoritmusok bemenetére, gondoljunk például szkennelt fényképre vagy digitalizált videojelre. (Ez alól csak a számítógépes grafikák alkotnak néha kivételt, ugyanis ezek lehet, hogy azonnal indexelt formában készülnek.) Ezért szükséges lehet, hogy egy folytonos tónusú képet indexeltté alakítsunk. Ennek során minél kisebb palettaméretet szeretnénk elérni, annál többet kell az eredeti kép színei közül másikkal helyettesíteni. Felfoghatjuk ezt úgy is, hogy az indexelt tárolású formává való átalakítás egy vektorkvantálás a három dimenziós (R,G,B) színtérben. Az optimális, tehát a lehető legkevésbé látható torzulást okozó paletta (vagyis az optimális kvantálási vektorok) kiválasztására speciális — az emberi színérzékelés tulajdonságait kihasználó — algoritmusok állnak rendelkezésre.

A GIF formátumban a paletta mérete általában 256, de bármely kisebb 2 hatvány használata is megengedett. A palettában a színek megadása az (R,G,B) színkoordinátákkal történik. A paletta tömörítésének mikéntjével nem foglalkozunk. A képpontokat a GIF tömörítés sorfolytonosan tapogatja le, és a sorozatot LZW algoritmussal tömöríti.

A tömörített kép első bájtjának értéke a képpontonkénti bitszám (b). Ez határozza meg a palettaméretet: 2^b. Az LZW szótár kezdeti mérete 2^{b+1}, tehát minden kódszó $b+1$ bites. Az első néhány képpont szempontjából (amíg nincs a képpontok sorozatában ismétlődés) ez azt jelenti, hogy képpontonként egy plusz bit képződik, hisz a b bites képpont $b+1$ biten lesz tárolva. Közben azonban épül az LZW szótár, és ha a képpontértékek sorozatában valahol ismétlődés van, azt az LZW azonnal kihasználja. Ha a 2^{b+1} méretű szótár betelt (ezt a pillanatot természetesen a kitömörítő algoritmus is meg tudja állapítani), a szótár méretét az algoritmus megduplázza, és ettől kezdve $b+2$ bites kódszavakkal dolgozik. Ha ez a szótár is betelik, akkor ismét duplázódik a méret, amíg el nem éri a 4096 bejegyzést. Innentől már nem növekszik tovább a szótár, hanem az eddig elkészült szótárat fogja a továbbiakban statikus szótárként használni az algoritmus.

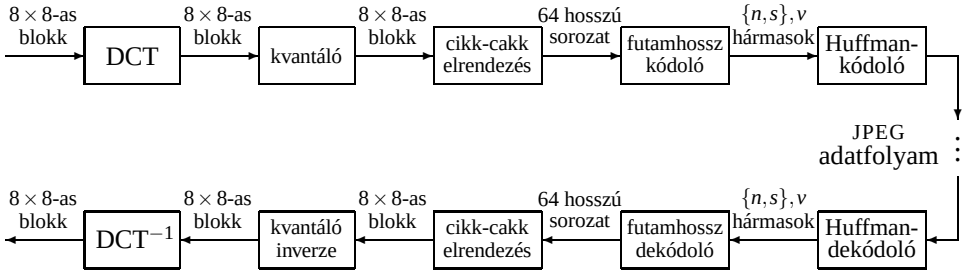
A GIF formátum nagyon elterjedt, leginkább kis ikonok, ábrák tömörítésére alkalmazzák, ugyanis ezeket képes igen jól tömöríteni. Ennek egyik oka, hogy ezek sok esetben nagy kiterjedésű, egyszínű területeket tartalmaznak, és ezek a sorfolytonos letapogatással hosszú egyszínű sorozatokként jelennek meg. A hosszú sorozatok az LZW szótárban egy kódszóval reprezentálhatók, így igen hatékony a tömörítés. Ugyancsak jellegzetessége az ilyen képeknek, hogy ismétlődő részleteket tartalmaznak, és ezek megintcsak kedveznek az LZW algoritmusnak. Azonban például fotók esetén az egyszínű területek igen ritkák, és méretük is viszonylag

kicsi. Ennek eredményeképpen a GIF tömörítési aránya ilyen képnél $(1 : 1.2 - 1.7)$ elmarad sok más, viszonylag egyszerű módszer mögött. (Például, ha predikcióként az előző képpont értékét használjuk, és a kapott különbségi sorozatot adaptív aritmetikai kódolóval tömörítjük, akkor a tömörítési arány $1 : 1.3 - 2.2$ lesz.)

A különbség másik oka a színkezelésben keresendő. Az ikonok és ábrák általában viszonylag kevés színt tartalmaznak, ezért igen előnyös az indexelt tárolásuk. Egy olyan tömörítés, amely nem tud indexelt képeket tömöríteni, egy ilyen ábrát kénytelen lenne folytonos tónusúvá alakítani a tömörítés előtt. Így kétszer-háromszor akkora adatmennyiséget kellene, hogy tömörítsen, mint a GIF, amely közvetlenül az indexelt formával képes dolgozni. Sok módszer azért nem tud közvetlenül indexelt képeken dolgozni, mert a szomszédos képpontok értékeinek kis különbségére alapoz. Mivel folytonos tónusú tárolásnál a szomszédos képpontokban tárolt értékek a színkoordináták, ez a feltételezés helyes, és általában teljesül is. Indexelt tárolás esetén azonban a képpontokban tárolt értékek a palettaindexek, és ezek tetszőlegesen távoliak lehetnek (csak a nekik megfelelő palettabeli színkoordináták értékei vannak egymáshoz közel). Fokozottan jelentkezik ez akkor, ha valamilyen palettaoptimalizáló eljárást alkalmaztunk. Eléggé jól látható, hogy a GIF-nek miért nem probléma ez: a GIF ismétlődő mintázatokat keres, és ilyen szempontból teljesen mindegy, hogy a közeli képpontok értékei megközelítőleg egyenlőek-e. Mindössze az számít, hogy vannak-e ismétlődések. Így már érthető, hogy miért is olyan hatásos módszer a GIF számítógépes ábrák tömörítésénél.

A JPEG (Joint Photographic Experts Group) a képtömörítés világában szinte egyeduralkodó a veszteséges tömörítések területén. Az egész szabványcsomagot a Joint Photographic Experts Group (ISO/IEC JTC 1 / SC 29 / WG 1) dolgozta ki folytonos tónusú képek tömörítésére. Valójában a JPEG szabványcsomag támogat veszteséges és veszteségmentes képtömörítést is, bár tény, hogy a gyakorlatban szinte kizárólag az elsőt használják. Mindezek ellenére a prediktív kódolás jó példája a veszteségmentes JPEG, ezért az alábbiakban ezt is bemutatjuk, továbbá részletesen tárgyaljuk a veszteséges JPEG egy egyszerű változatát, a baseline JPEG-et.

A **veszteségmentes** JPEG képtömörítés alapja a predikciós kódolás. Predikcióra az éppen tömörítésre kerülő képpont környezetét használjuk fel. Mivel a képpontok letapogatása itt is sorfolytonosan történik, az adott képponttól balra vagy felfelé eső képpontok értékei azok, amelyek a dekódoló rendelkezésére állnak az adott képpont értékének visszaállításakor, így a kódoló is csak ezeket használja fel a predikcióhoz. Nyolcféle predikciós séma létezik, és ezek közül egy adott kép tömörítéséhez bármelyik használható, de egy képen belül végig ugyanaz a séma érvényes. Jelölje $X_{i,j}$ az (i, j) koorinátájú képpont értékét, és $\widehat{X}_{i,j}$ az (i, j) koordinátájú képpont becsült értékét. Az (i, j) koordinátájú képpont becsült értéke az egyes predikciós sémákban:



5.13. ábra. Veszteséges baseline JPEG tömörítés

$$
\begin{aligned}
0 \quad & \widehat{X}_{i,j} = 0 \\
1 \quad & \widehat{X}_{i,j} = X_{i-1,j} \\
2 \quad & \widehat{X}_{i,j} = X_{i,j-1} \\
3 \quad & \widehat{X}_{i,j} = X_{i-1,j-1} \\
4 \quad & \widehat{X}_{i,j} = X_{i,j-1} + X_{i-1,j} - X_{i-1,j-1} \\
5 \quad & \widehat{X}_{i,j} = X_{i,j-1} + \frac{X_{i-1,j} - X_{i-1,j-1}}{2} \\
6 \quad & \widehat{X}_{i,j} = X_{i-1,j} + \frac{X_{i,j-1} - X_{i-1,j-1}}{2} \\
7 \quad & \widehat{X}_{i,j} = \frac{X_{i,j-1} + X_{i-1,j}}{2}
\end{aligned}
$$

Látható, hogy az első lehetőség maga a predikciómentes tömörítés. Megfigyelhető, hogy az összes többi predikciós séma mellett az egyszínű területeken a predikciós hiba 0, azaz $\widehat{X}_{i,j} = X_{i,j} = X_{i-1,j} = X_{i,j-1}$. Javítható a tömörítés mértéke, ha a képet blokkokra osztjuk, és minden blokkra külön adjuk meg a használni kívánt predikciós sémát.

A **veszteséges** baseline JPEG tömörítés algoritmusa az 5.13. ábrán látható. A JPEG a képeket először képsíkokra bontja: minden képsík egy-egy színkoordináta értékeit tartalmazza. Színkoordinátának az előzőekben bemutatott (Y, C_b, C_r) hármast használjuk, mindegyiket 8-8 biten tárolva. Így egy képpontra összesen 24 bit jut. A JPEG tömörítő algoritmus számára a képsíkok gyakorlatilag egymástól független képekként jelennek meg, külön-külön vannak kódolva.

Megengedett, hogy a képsíkok felbontása eltérjen. Mindössze annyi a kikötés, hogy a felbontások aránya racionális legyen. A kép kitömörítésekor a felbontások legkisebb közös többszörösének megfelelő felbontású képet állítunk vissza, ahol az egyes képsíkok képpontbeli értékeit lineáris interpolációval állítjuk elő a tárolt felbontással megegyező felbontásban rekonstruált képsík értékeiből. A különböző felbontású képsíkok alkalmazásának lehetőségét úgy szokás kihasználni, hogy a C_b és C_r képsíkok vízszintesen és függőlegesen is felére csökkentett felbontással

vannak tárolva. Például 400×400 pontos kép esetén az Y képsík felbontása 400×400, míg a C_b és C_r képsíkok felbontása 200×200. Kitömörítésnél visszaállítjuk a 400×400-as Y képsíkot, és a két 200×200-as C_b és C_r képsíkot. Ezután a C_b és C_r képsíkokat lineáris interpolációval a kétszeresükre nagyítjuk, majd az így kapott két 400×400-as képsíkot és a 400×400-as Y képsíkot „vetítjük egymásra”, hogy az eredeti képet kapjuk.

Az algoritmus első lépése a képsíkok 8×8-as **négyzetekre bontás**a. Ha a képméret nem osztható 8-cal, akkor a jobb szélső oszlop és/vagy a legalsó sor többszörözésével azt 8-cal oszthatóvá tesszük. Az így keletkező felesleges széleket a dekódoló levágja a képméret alapján.

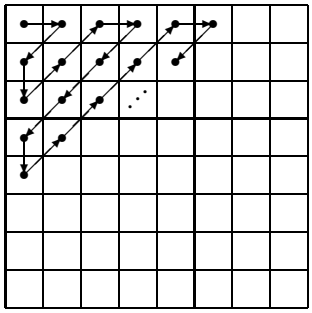
A 8×8-as négyzeteket kétdimenziós **diszkrét koszinusz transzformáció**nak (DCT) vetjük alá. Így ugyancsak 8×8-as négyzeteket kapunk, csak azok most már nem egész értékekből, hanem valós számokból állnak. Ezeket a frekvenciatartománybeli valós együtthatókat a következő lépés, a kvantálás fogja ismét egészekké alakítani.

A **kvantálás** egyenletes, viszont a 8×8-as négyzet minden egyes elemére más lépésközzel hajtjuk végre. A 8×8-as DCT transzformált négyzet együtthatói különböző frekvenciájú felharmonikusoknak felelnek meg. A kis frekvenciás együtthatók a négyzet bal felső részében vannak. Mint tudjuk, a szem ezekre a kisebb frekvenciához tartozó értékekre sokkal érzékenyebb, mint a nagyobb frekvenciához tartozókra, ezért ezeket finomabb lépésközzel fogjuk kvantálni. A szabvány nem köti meg, hogy milyen kvantálási lépésközöket használjunk. A használt lépésközöket táblázatba szokás foglalni, ahol egy táblázatelem a 8×8-as négyzet megfelelő együtthatójának kvantálási lépésközét tartalmazza. Bár a szabvány nem teszi kötelezővé a használatát, de ajánl egy kvantálási táblát:

$*$							
16	19	22	26	27	29	34	
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	48	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

A 0 frekvenciához tartozó együttható (neve DC komponens) a DCT transzformáció során a bal felső sarokba került. Kihasználva az egymás melletti képrészek viszonylag hasonló színét, a DC komponenst nem kvantáljuk, ehelyett az egymás utáni 8×8-as blokkok DC komponenseinek különbségét vesszük. Ezért nem tartozik a DC komponenshez érték a kvantálási táblában.

Látható, hogy a nagyobb frekvenciás együtthatókra nagyobb lépésközt ad meg a táblázat. Ennek eredményeképpen a nagyobb frekvenciás együtthatók kvantált értékei majdnem mind 0-k lesznek. Ezeket a 0-kat egy későbbi lépés (a futamhossz kódolás) során jól tudjuk majd tömöríteni. A kvantálási tábla változtatása



5.14. ábra. Cikk-cakk elrendezés

tehát lehetőséget ad a tömörítés mértékének és minőségének befolyásolására: a tömörítést növelni lehet a kvantálás lépésközének növelésével, a kép minőségének rontása árán.

A kvantált együtthatókat ezután az úgynevezett **cikk-cakk elrendezés** szerint sorbarendezzük (5.14. ábra). Így az első helyre a különbségi DC együttható kerül, azt követi a vízszintes alapharmonikus, majd a függőleges alapharmonikus, majd pedig az egyre növekvő frekvenciás felharmonikusok következnek. Köszönhetően annak, hogy a kvantálás során a táblázat magasabb frekvenciákhoz tartozó együtthatói nagyrészt kinullázódtak, a kapott sorozat a vége felé túlnyomórészt 0-kból áll.

A **futamhossz kódolási** lépésben a cikk-cakk elrendezéssel kapott, javarészt 0-kból álló sorozatot részekre bontjuk úgy, hogy minden részsorozat eleje tetszőleges számú 0-ból álljon, és ezt a végén egyetlen nem 0 elem zárja. Minden így kapott részsorozathoz egy $(\{n,s\},v)$ számhármast rendelünk. n az adott részsorozat elején levő 0-futam hossza, s jelzi, hogy a részsorozat végén található nem 0 értéket hány biten kódoljuk, v pedig ezen nem 0 érték bináris ábrázolása. Az s változóra a tömörítés hatékonyságának növelése miatt van szükség, mert ugyan előfordulnak néha nagy abszolút értékű elemek, de ezek ritkák, így nem érdemes minden elemet azonos bithosszon kódolni.

Így például a $23,0,0,0,0,-7,0,-19,4,0,0,0,\ldots$ sorozat első futamhossz-kódhármasa: $(\{0,6\},010111)$, hiszen az első elem, a 23 előtt nincs 0, így $n=0$. 23 bináris számként 6 biten kódolható előjelesen, így $s=6$. A további kódszavak: $(\{4,4\},1000)$; $(\{1,6\},101100)$; $(\{0,4\},0100)$;... (Az előjeles bináris számokat egyes komplemens ábrázolásban tüntettük fel. Megfelelő technikával ezek 1 bittel rövidebben is kódolhatóak.)

Az így kapott $(\{n,s\},v)$ hármasok $\{n,s\}$ elemeit statikus Huffman-kóddal tömörítjük tovább, a v értékeket pedig egyszerűen elválasztójel nélkül egymás után írjuk. A statikus Huffman-kód azt jelenti, hogy nem az $\{n,s\}$ párok adott képben levő gyakorisága alapján választjuk a kódszavakat, hanem azok a JPEG szabványban le vannak rögzítve. (A statikus kódot az $\{n,s\}$ párok sok képre vett átlagos gyakoriságai alapján tervezték meg.) Alkalmazhatnánk a Huffman-kódolást ma-

gukra az $(\{n,s\},v)$ hármasokra is, de akkor a kód annyira sok kódszóból állna, hogy kezelhetetlen lenne, nagyon lassú lenne vele dolgozni. A v értékek leválasztása jó döntés, mert ezzel a kód mérete kezelhető lesz, viszont a hatékonysága csak alig romlik: a v értékek bitjei a tapasztalat szerint gyakorlatilag függetlenek és egyenletes eloszlásúak, így nem tömöríthetőek, hiába is vonnánk be őket a Huffman-kódba.

A fenti példán bemutatva ezt az utolsó lépést, a tömörített képbe a $\{0,6\},\{4,4\}$, $\{1,6\}$ és $\{0,4\}$ Huffman-kódja, és a 01011110001011000100 sorozat kerül.

A JPEG egyszerűsége ellenére meglepően alacsony bitsebességeket tud elérni. A képpontonkénti 2 bites tömörítés egy 24 bites színmélységű kép esetén az eredetitől megkülönböztethetetlen képet eredményez — legalábbis az emberi szem számára nem látható a különbség. Igen kis, 1 bit/képpontos bitsebesség esetén kezd el zavaróvá válni a torzítás, és 0.5 bit/képpont alatt a kép nem élvezhető. A felismerhetőség még 0.086 bit/képpontos bitsebesség mellett is biztosítható. Természetesen ilyen alacsony bitsebességek esetén különböző kiterjesztések is szükségesek a szabványhoz. Mindenesetre ezek az eredmények nagyon jónak számítanak. A JPEG alacsony bitsebességek esetén legelőször „kockásodni” kezd. Ezt az alacsonyfrekvenciás együtthatók értékének kvantálás miatti torzulása okozza.

Videotömörítés

Az MPEG (Moving Picture Experts Group) a JPEG-hez hasonlóan az ISO egyik munkacsoportja (ISO/IEC JTC 1 / SC 29 / WG 11). Feladatuk olyan videotömörítési szabványok kidolgozása volt, melyek széleskörű ipari konszenzuson alapulnak. Az ilyen jellegű szabványokra a digitális konvergencia időszakában mind a tartalom előállítóknak, mind a felhasználóknak, mind a közöttük álló szolgáltatóknak szükségük van. Az MPEG egy öt szabványból álló szabványcsomag kidolgozását tűzte ki célul, mely a veszteséges videotömörítés (beleértve a hangtömörítést is), a digitális televíziózás és a multimédia alkalmazások széles körét fedi le. Az öt szabvány: MPEG-1, MPEG-2, MPEG-4, MPEG-7 és MPEG-21.

A CD-olvasók első generációja a hifi minőségű tömörítetlen zenéhez szükséges, 1.4 Mbit/s lejátszási sebességre volt képes. Az MPEG-1 szabvány kialakításakor azt tűzték ki célul, hogy az ilyen, egyszeres ($1\times$) CD-meghajtók olvasási sebességén használható formátumot hozzanak létre. Ez nehéz feladat volt, hiszen egy 8 biten mintavételezett video (NTSC vagy PAL) tömörítetlenül nagyságrendileg 200 Mbit/s átviteli sebességet igényel. A szükséges tömörítés tehát nagyjából $1:140$ kell hogy legyen, sőt még a hangcsatorná(k)nak és a hibajavító kódnak is helyet kell szorítani. A nagy tömörítési igény mellett a „véletlen elérés” lehetőségét is teljesíteni kellett, vagyis azt, hogy a tárolt video bármelyik részét (akár a közepét is) gyorsan el lehessen érni. Amennyiben prediktív algoritmussal tömörítünk, problémát okozhat ennek a kritériumnak a megvalósítása, hiszen ilyenkor csak a kezdeti, biztos ponttól elindulva lehetséges a rekonstrukció. Viszont a prediktív algoritmusok kizárása szóba sem jöhetett, hiszen ezek nélkül remény sincs

ilyen tömörítés elérésére. Az MPEG által választott megoldás a rövid, de egymástól független prediktív blokkok alkalmazása volt. Az MPEG-1 az 1×-es CD-olvasók sebességének megfelelő, igen erős tömörítést csak viszonylag gyenge képminőség mellett tudja megvalósítani. Azonban 1992-ben, amikor a szabvány megjelent, az ipar ezzel a minőséggel is megelégedett. A szabvány sikeres alkalmazásai többek között a CD-I és a Video-CD technológiák.

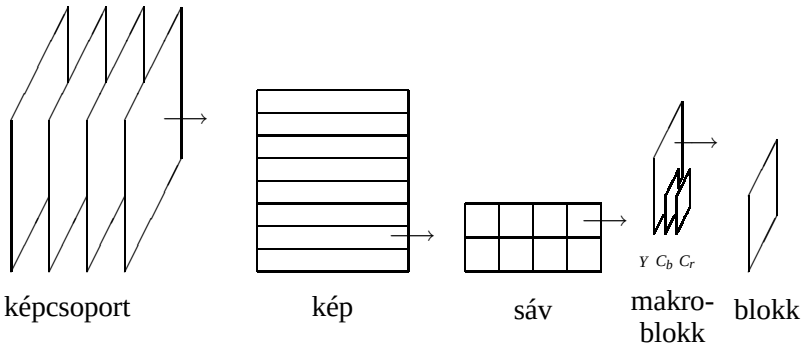
Az MPEG-2 szabványt a digitális televíziózáshoz fejlesztették ki. Ehhez az MPEG-1 segítségével elérhetőnél jobb minőségre volt szükség, támogatja például a szabvány a HDTV video tömörítését is. A HDTV technológia, a High Definition TeleVision a szokványos televízióminőségnél nagyobb felbontású és színmélységű, CD-minőségű hanggal kísért televíziós szabvány. Az MPEG-2 által igényelt bitsebesség (a minőség függvényében) 1–40 Mbit/s között alakul. Jellegzetes értékek a 4–6 Mbit/s hagyományos (PAL) videojel esetén, és a 12–20 Mbit/s HDTV videojel esetén. Digitális televíziózás során gondoskodni kell a tömörítésen kívül a jelátvitelről, multiplexelésről is. Az MPEG-2 szabvány egyik sikeres alkalmazása, az Európában szabványosított DVB (Digital Video Broadcasting) technológia. Ez a digitális műsorszórást teszi lehetővé, hagyományos földi (8 MHz sávszélességű) televíziócsatornában illetve műholdas (36 MHz sávszélességű) csatornában. Mindkét esetben 3–6 televízióműsor multiplex átvitelére van lehetőség. Az MPEG-2 további sikeres alkalmazása a VoD (Video on Demand) és a DVD is.

Az MPEG-4 a multimédia alkalmazások szabványa. Közös technológiai alapot nyújt a műsorszórásos, az interaktív és a beszélgetés-jellegű szolgáltatásokhoz. Sokrétű interaktivitást tesz lehetővé a hagyományos lejátszás–megállítás–előre/visszatekerés mellett. Képes szintetikus (pl. számítógép-animáció) és természetes forrásból származó (pl. videokamera által rögzített) információtípusok együttes kezelésére. A bitsebességek igen széles skáláját támogatja, egészen 10 kbit/s-tól kezdve. Többek között lehetséges MPEG-4-gyel egy videokonferencia 64 kbit/s-os tömörítése és a professzionális HDTV tömörítése is 40 Mbit/s sebességgel. Az MPEG-4 egyik sikeres alkalmazása a DivX.

Az MPEG-7 a többi MPEG szabvány által kódolt audiovizuális tartalmak katalogizálására, leírására, keresésére ad lehetőséget. Az MPEG-21 szabvány egy egységes multimédia keretrendszer, melynek feladata a felhasználók támogatása a digitális tartalmak hozzáférésében, cseréjében, megvásárlásában, módosításában.

A továbbiakban az MPEG-1 szabvány videotömörítési részének vázát fogjuk áttekinteni. Fontos megjegyezni, hogy az MPEG-1 emellett foglalkozik a hangtömörítéssel, valamint a video- és hangfolyamok multiplexelésével is. A hangtömörítési részről az előzőekben már volt szó.

A szabvány kidolgozása során kiemelt figyelmet fordítottak az implementáció szabadságára. A videotömörítési részében az MPEG-1 szabvány csak a tömörített bitfolyam szintaxisát és értelmezését írja le. A kódoló tetszőleges algoritmussal dolgozhat, de természetesen csak szintaktikailag helyes bitfolyamot állíthat elő. Nagyon sok múlik azon — mind képminőség, mind pedig bitsebesség tekintetében — hogy egy kódoló mennyire jól tudja kihasználni az MPEG-1 szabvány adta lehetőségeket. A dekódoló implementálása terén a bitfolyam kötött értelmezése



5.15. ábra. Az MPEG bitfolyam szintaktikai felépítése

miatt nincs nagy szabadság. A szabvány ad is egy referencia dekódolót, bár ennek használata nem kötelező.

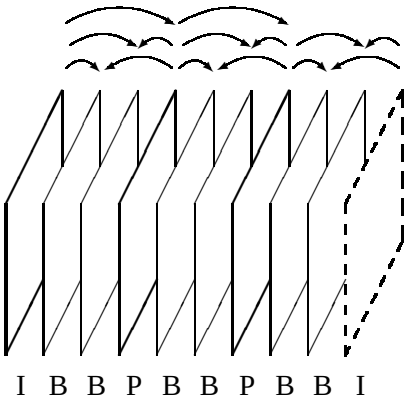
Az 5.15. ábrán a bitfolyam szabvány által definiált szintaktikai felépítése látható. A legkülső elem a **videoszekvencia** (ezt nem tüntettük fel az ábrán). Egy videoszekvenciának adott képmérete, képsebessége és egyéb jellemző paraméterei vannak. A videoszekvencia fejléce ezek leírását tartalmazza.

A videoszekvencia belsejében **képcsoportok** (Group Of Pictures, GOP) vannak, amelyek néhány kép egymásutánjából állnak. A képcsoportok az egymástól függetlenül kódolt prediktív egységek. A képcsoporton belül **képek** helyezkednek el. A kép háromféle lehet, I, P vagy B. Ez az alkalmazott predikció fajtájával van összefüggésben. A képek **sávok**ra, a sávok **makroblokkok**ra (16×16 képpont), a makroblokkok pedig **blokkok**ra vannak bontva.

A tömörítési algoritmus lelke a mozgásbecslés. Ez tulajdonképpen egy predikciós kódolás, ahol a video adatfolyam egymás utáni képeinek hasonlóságát használjuk ki. A predikciót makroblokk szinten végezzük: minden makroblokkhoz kikeressük az időben megelőző és az időben következő képeken található, hozzájuk leginkább hasonló részleteket. A megelőző és a következő képeken a hasonló részek helyét az úgynevezett mozgásvektorral határozzuk meg. A makroblokknak csak ezen referencia-képrészektől való eltérését (vagyis a predikció hibáját) kódoljuk a JPEG módszernél megismertek szerint.

Egy makroblokk tömörítése során ugyanis négyféle módon járhatunk el. Tömöríthetjük predikció nélkül, vagy predikcióval hátulról, elölről, és mindkét irányból. Amennyiben nem alkalmazunk predikciót, úgy az eredeti makroblokk blokkjait kell JPEG-szerűen kódolnunk. Amennyiben egyirányú predikciót alkalmazunk (időben megelőző vagy következő kép egy részét jelöljük ki), akkor a makroblokkból levonjuk a kijelölt részt, és a különbséget tömörítjük. Amennyiben kétirányú predikciót hajtunk végre (időben megelőző és következő kép egy-egy részletét is felhasználjuk), akkor a makroblokkból a JPEG tömörítés előtt a két kijelölt rész átlagát vonjuk le.

Az elölről és a kétirányból alkalmazott predikciónál gondoskodni kell az egyértelmű dekódolhatóságról. Ezt az MPEG-1 szabvány azzal oldja meg, hogy kizárja



5.16. ábra. Az MPEG különböző predikciótípusai

a képek körkörös egymásrahivatkozását. Ennek biztosítására minden egyes képben meg van kötve, hogy a benne szereplő makroblokkok kódolása során milyen típusú predikciót használhatunk. Ezt mutatjuk be az 5.16. ábrán.

Az I típusú képben egyáltalán nem alkalmazhatunk predikciót, vagyis ez egy önálló kép, JPEG típusú tömörítéssel. Így egy I típusú képet minden egyéb kép nélkül tudunk dekódolni.

A P típusú képek makroblokkjai lehetnek predikció nélkül kódolva, vagy pedig predikcióval hátulról, a hozzájuk legközelebb levő I vagy P típusú képből. Egy P típusú kép dekódolásához ezek szerint legfeljebb az előtte levő utolsó I képig kell visszaszaladnunk, és innentől kezdve P képről P képre tudunk lépkedni.

A harmadik fajta kép, a B kép bármilyen kódolású makroblokkokat tartalmazhat, de a makroblokkok csak a legközelebb eső megelőző vagy következő P vagy I képekre hivatkozhatnak. Sem B képekre, sem a legközelebbi I vagy P képnél távolabbi képekre nem lehet hivatkozni. A lényegi különbség a B és P képek között az, hogy B képeknél jövőbeli P vagy I képre is hivatkozhatunk.

Természetesen, a szabvány koncepciójának megfelelően a képek sorrendje nem kötött, akárhogyan jöhetnek egymás után a P, I és B képek, ahogy a kódoló akarja. A fenti szabályok betartásával lehetséges a dekódolás, mert egy I típusú kép önmagában, egy P típusú az előtte levő, hozzá legközelebbi I képtől indulva, míg egy B kép az őt körülvevő I vagy P képek alapján dekódolható.

Mindemellett hogy nem kötelező, kialakultak szokványos képsorrendek. A legsűrűbben egy I képre két P kép épül, és közöttük két-két B kép van. Az 5.16. ábrán egy ilyen képsorrend látható.

A képek sorrendje a bitfolyamban nem egyezik meg a képek idősorrendjével. A bitfolyambeli sorrend úgy lett meghatározva, hogy ha sorban olvassuk be a bitfolyamból a képeket, akkor egy kép beolvasásának időpontjában az összes szükséges információ rendelkezésre álljon a dekódoláshoz. Ezt úgy oldja meg a szabvány, hogy hátraküldi a B képeket az utánuk következő I vagy P kép mögé. Ezáltal a

B képek időben előre mutató referenciája a bitfolyamban hátramutató referencia lesz. Természetesen átrendezéskor az I és P képek a helyükön maradnak, és a B képek egymáshoz viszonyított sorrendjén sem változtatunk. Tehát, ha az időbeli sorrendet az indexekkel jelöljük, akkor a

$$\ldots B_{-2}B_{-1}I_0B_1B_2P_3B_4B_5P_6B_7B_8I_9B_{10}B_{11}P_{12}\ldots$$

sorozatból az átrendezés után az

$$\ldots I_0B_{-2}B_{-1}P_3B_1B_2P_6B_4B_5I_9B_7B_8P_{12}B_{10}B_{11}\ldots$$

sorozat lesz.

Egy képcsoport mindig egy I képpel kezdődik, méghozzá a bitfolyambeli sorrend szerint. Emiatt egy képcsoport bitfolyamban első képe önmagában dekódolható, vagyis nem szükséges régebben dekódolt képek ismerete a dekódolásához. A képcsoport további képei is mind dekódolhatóak a bitfolyambeli sorrendben. (Természetesen ezek dekódolása során szükségünk lesz az ezen képcsoportból előttük dekódolt képekre.) Kivételt képeznek az első P képet megelőző B képek, ugyanis ezek a megelőző képcsoport utolsó P vagy I képére is hivatkozhatnak. (Ne felejtsük el, hogy a bitfolyambeli sorrendben definiáljuk a képcsoportot!) Ezek a B képek a megelőző képcsoport dekódolása nélkül nem dekódolhatóak. Emiatt az első néhány nem dekódolható B kép miatt hívják nyíltnak az ilyen képcsoportot. (A szabvány definiál egy zárt képcsoportot is, természetesen ehhez más képsorrend szükséges.) Ezáltal a képcsoport a bevezetőben említett viszonylag rövid prediktív egységet testesíti meg, hiszen (nyílt csoport esetén néhány B kép kivételével) a képcsoportok egymástól függetlenül dekódolhatóak.

Az MPEG videokódolás, amennyiben ezt nem szabályozzuk külön, változó bitsebességet fog eredményezni. Gondoljunk például arra, hogy ahol egy teljesen új kép jelenik meg a videofolyamban (vágás), ott az első kép makroblokkjait nem tudjuk predikcióval tömöríteni, és így gyakorlatilag egy teljes képet kell JPEG-gel kódolnunk. Ez igen sok bitet igényelhet. Ott viszont ahol egy állókép van a videofolyamban, szinte nem is lesz predikciós maradék, vagyis nagyon rövid lesz a kód. Az átvitelhez használt csatorna bitsebességét a kódolás során természetesen nem léphetjük át. Ezen a problémán pufferek felhasználásával valamennyit lehet segíteni, de meg kell oldani, hogy a dekódoló puffere sohase ürüljön ki, és sohase csorduljon túl, továbbá a kódoló is csak a csatorna átviteli képességének megfelelő mennyiségű adatot hozzon létre. Ennek eléréséhez a kódoló állandóan figyeli a saját pufferét, és nyomon követi a dekódolóét is. (Ez utóbbihoz egyes esetekben a dekódoló szimulációjára kényszerül.) Ha például a kódoló puffere kezd megtelni csökkentenie kell a képenként átvitt bitek mennyiségét. Ezt a JPEG kvantálási lépcső megemelésével érheti el, természetesen a minőség rovására. Durvább megoldás, ha bizonyos makroblokkok predikciós maradékát egyszerűen elhagyja a kódból, és csak a mozgásvektorokat adja meg. Még durvább lehetőségek is rendelkezésére állnak a kódolónak. Ilyen például a bemenetén érkező kép információtartalmának korlátozása, például a képfrekvencia vagy a felbontás csökkentésével.

5.5. Feladatok

5.1. feladat (Normális eloszlás egy bites kvantálása). Legyen X nulla várható értékű, σ szórású normális eloszlású valószínűségi változó. Határozza meg az optimális egy bites (két szintű) kvantálót! Mennyi a torzítás?

5.2. feladat (Exponenciális eloszlás kvantálása). Legyen X valós valószínűségi változó, melynek sűrűségfüggvénye

$$f(x) = \begin{cases} c\,\mathrm{e}^{-\frac{1}{2}x}, & \text{ha } x \in [0,2] \\ 0, & \text{ha } x \notin [0,2] \end{cases},$$

ahol c olyan konstans, hogy f valószínűségi sűrűségfüggvény. Kvantáljuk X-et egy $[0,2]$-re illeszkedő 4 bites (16 szintű) egyenletes Q_1 kvantálóval. A tanult közelítéseket használva számolja ki a kvantáló négyzetes torzítását és a $H(Q_1(X))$ entrópiát!

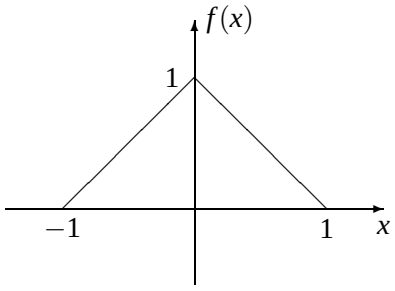
5.3. feladat. Az ábrán látható $f(x)$ sűrűségfüggvényű X valószínűségi változót 2 bites egyenletes kvantálóval kvantáljuk, mely illeszkedik a $[-1,1]$ intervallumra.

a) Számolja ki a kvantálási szinteket!

b) Számolja ki *pontosan* a kvantáló négyzetes torzítását és kimenetének entrópiáját!

c) Adjon közelítést a kvantáló négyzetes torzítására és entrópiájára! Mennyire egyeznek a pontos és közelítő értékek?

(Segítség: $\int_0^x t \ln t \,\mathrm{d}t = x^2\left(\frac{\ln x}{2} - \frac{1}{4}\right)$.)



5.4. feladat (A Lloyd–Max-algoritmus nem optimális). Mutasson példát arra, hogy a Lloyd–Max kvantálótervező algoritmus nem mindig a minimális torzítású kvantálóhoz konvergál. Azaz adjon meg egy olyan „rossz” sűrűségfüggvényt és kiindulási kvantálót, hogy az algoritmus biztosan ne az optimumhoz konvergáljon.

5.5. feladat. Számítsa ki a σ szórású, m várható értékű normális eloszlás differenciális entrópiáját.

5.6. feladat (Maximális differenciális entrópia). Mutassa meg, hogy az adott várható értékű és szórású valószínűségi változók közül a normális eloszlásúnak maximális a differenciális entrópiája.

5.7. feladat (Differenciális entrópia). Legyen X abszolút folytonos eloszlású valószínűségi változó, és jelöljük $H(X)$-szel a differenciális entrópiáját. Mutassa meg, hogy bármely $c > 0$ számra

$$H(cX) = H(X) + \log c.$$

5.8. feladat. Legyen X egyenletes eloszlású a $[0,60]$ intervallumon. Egy Δ lépcsőjű egyenletes kvantálóval kvantálva a négyzetes torzítás 0.03. Körülbelül mekkora Δ értéke? Mekkora a kvantáló kimenetének entrópiája?

5.9. feladat. Egy standard normális eloszlású X valószínűségi változót a következő kvantálóval kvantálunk:

$$Q(X) = \begin{cases} a, & \text{ha } x \geq 0 \\ -a, & \text{ha } x < 0 \end{cases}.$$

Számítsa ki a kvantáló négyzetes torzítását! Hogyan válasszuk meg a-t a torzítás minimalizálásához?

5.10. feladat. Legyen X egy 1 várható értékű, $\frac{1}{3}$ szórásnégyzetű, normális eloszlású valószínűségi változó. Írja fel az optimális kompresszorfüggvényt a $\Phi(x)$ standard normális eloszlásfüggvény segítségével.

5.11. feladat. A veszteséges baseline JPEG algoritmus mely lépésénél történik a tényleges tömörítés? Hogyan eredményezhet ugyanaz az algoritmus különböző tömörítési arányokat?

5.12. feladat. Egy MPEG videoszekvencia egy képcsoportja legyen $I_0B_1P_2B_3B_4P_5$ $B_6B_7P_8I_9$, ahol az indexekkel az időbeli sorrendet jelöltük. A B képek a hozzájuk legközelebb eső I és P képek alapján mindkét irányú predikciót használnak. A P képek csak a hozzájuk legközelebbi előző I képet használják a becsléshez.

a) Adja meg a képek helyes sorrendjét, amely megfelelő a dekódoláshoz.

b) Ha a P_5 kép megsérül, lehetséges-e a B_1, B_3, B_7, P_8 képek bármelyikét dekódolni?

5.6. Megoldások

5.1. megoldás. Az X valószínűségi változó sűrűségfüggvénye

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{x^2}{2\sigma^2}}.$$

Kövessük a Lloyd–Max-algoritmust. Ehhez először vegyünk fel egy tetszőleges két szintű kvantálót, például

$$Q_2^0(x) = \begin{cases} 2, \text{ ha } x \geq 1, \\ -2, \text{ ha } x < 1. \end{cases}$$

Optimalizáljuk a kvantálót a kvantálási szintek szerint, vagyis határozzuk meg az intervallumhatárokat a legközelebbi szomszéd feltétel kielégítésével:

$$Q_2^1(x) = \begin{cases} 2, \text{ ha } x \geq 0, \\ -2, \text{ ha } x < 0. \end{cases}$$

Alkalmazzuk a súlypont feltételt:

$$x_1 = \frac{\int_{-\infty}^{0} x f(x)\,\mathrm{d}x}{\int_{-\infty}^{0} f(x)\,\mathrm{d}x} = 2\int_{-\infty}^{0} x \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{x^2}{2\sigma^2}}\,\mathrm{d}x = 2\left[-\frac{\sigma}{\sqrt{2\pi}}\mathrm{e}^{-\frac{x^2}{2\sigma^2}}\right]_{-\infty}^{0} = -\sqrt{\frac{2}{\pi}}\sigma,$$

és szimmetria okokból

$$x_2 = \sqrt{\frac{2}{\pi}}\sigma,$$

vagyis

$$Q_2^2(x) = \begin{cases} \sqrt{\frac{2}{\pi}}\sigma, \text{ ha } x \geq 0, \\ -\sqrt{\frac{2}{\pi}}\sigma, \text{ ha } x < 0. \end{cases}$$

Ez már a lépések ismétlésével sem változik, tehát $Q_2^2(x)$ egy Lloyd–Max-kvantáló, melynek torzítása

$$\begin{aligned}
D(Q_2^2) &= \int_{-\infty}^{\infty} (x - Q_2^2(x))^2 f(x)\,\mathrm{d}x = \\
&= \int_{-\infty}^{0} \left(x + \sqrt{\tfrac{2}{\pi}}\sigma\right)^2 \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{x^2}{2\sigma^2}}\,\mathrm{d}x + \int_{0}^{\infty} \left(x - \sqrt{\tfrac{2}{\pi}}\sigma\right)^2 \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{x^2}{2\sigma^2}}\,\mathrm{d}x = \\
&= \int_{-\infty}^{\infty} x^2 \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{x^2}{2\sigma^2}}\,\mathrm{d}x + 2\sqrt{\frac{2}{\pi}}\sigma \int_{-\infty}^{0} x \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{x^2}{2\sigma^2}}\,\mathrm{d}x \\
&\quad -2\sqrt{\frac{2}{\pi}}\sigma \int_{0}^{\infty} x \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{x^2}{2\sigma^2}}\,\mathrm{d}x + \frac{2}{\pi}\sigma^2 \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{x^2}{2\sigma^2}}\,\mathrm{d}x = \\
&= \sigma^2 - 2\sqrt{\frac{2}{\pi}}\sigma \frac{1}{\sqrt{2\pi}}\sigma - 2\sqrt{\frac{2}{\pi}}\sigma \frac{1}{\sqrt{2\pi}}\sigma + \frac{2}{\pi}\sigma^2 = \\
&= \left(1 - \frac{2}{\pi}\right)\sigma^2.
\end{aligned}$$

5.3. megoldás. A 2 bites, egyenletes kvantáló lépésköze $\Delta_4 = 0.5$, és 4 kvantálási tartománya van, ezek: $\mathcal{B}_1 = [-1,-0.5)$, $\mathcal{B}_2 = [-0.5,0)$, $\mathcal{B}_3 = [0,0.5)$, $\mathcal{B}_4 = [0.5,1]$.

a) A kvantálási szintek az intervallumok közepén helyezkednek el, így $x_1 = -0.75$, $x_2 = -0.25$, $x_3 = 0.25$, $x_4 = 0.75$.

b) A négyzetes torzítás definíciója szerint

$$\begin{aligned} D(Q_4) &= \int_{-1}^{1} (x - Q_4(x))^2 f(x)\,\mathrm{d}x = \\ &= \int_{-1}^{-0.5} (x+0.75)^2 (x+1)\,\mathrm{d}x + \int_{-0.5}^{0} (x+0.25)^2 (x+1)\,\mathrm{d}x \\ &\quad + \int_{0}^{0.5} (x-0.25)^2 (1-x)\,\mathrm{d}x + \int_{0.5}^{1} (x-0.75)^2 (1-x)\,\mathrm{d}x = \\ &= \frac{1}{384} + \frac{1}{128} + \frac{1}{128} + \frac{1}{384} = \\ &= \frac{1}{48}. \end{aligned}$$

A kvantáló entrópiájának kiszámításához szükségünk van az eloszlására:

$$p_1 = \mathbf{P}\{Q_4(X) = -0.75\} = \int_{-1}^{-0.5} f(x)\,\mathrm{d}x = \int_{-1}^{-0.5} (x+1)\,\mathrm{d}x = \frac{1}{8},$$

$$p_2 = \mathbf{P}\{Q_4(X) = -0.25\} = \int_{-0.5}^{0} f(x)\,\mathrm{d}x = \int_{-0.5}^{0} (x+1)\,\mathrm{d}x = \frac{3}{8},$$

$$p_3 = \mathbf{P}\{Q_4(X) = 0.25\} = \int_{0}^{0.5} f(x)\,\mathrm{d}x = \int_{0}^{0.5} (1-x)\,\mathrm{d}x = \frac{3}{8},$$

$$p_4 = \mathbf{P}\{Q_4(X) = 0.75\} = \int_{0.5}^{1} f(x)\,\mathrm{d}x = \int_{0.5}^{1} (1-x)\,\mathrm{d}x = \frac{1}{8},$$

ahonnan

$$H(Q_4(X)) = 2\left(-\frac{1}{8}\log\frac{1}{8}\right) + 2\left(-\frac{3}{8}\log\frac{3}{8}\right) = 3 - \frac{3}{4}\log 3 \approx 1.81.$$

c) A négyzetes torzítás közelítésére ismert képlet szerint

$$D(Q_4) \approx \frac{\Delta_4^2}{12} = \frac{1}{48},$$

tehát a közelítés pontos. Az entrópiára a közelítés

$$H(Q_4(X)) \approx H(f) - \log \Delta_4.$$

Ehhez ki kell számítanunk az eloszlás differenciális entrópiáját:

$$\begin{aligned}
H(f) &= -\int_{-1}^{1} f(x)\log f(x)\,\mathrm{d}x = \\
&= -\int_{-1}^{0}(x+1)\log(x+1)\,\mathrm{d}x - \int_{0}^{1}(1-x)\log(1-x)\,\mathrm{d}x = \\
&= -\frac{1}{\ln 2}\int_{0}^{1} t\ln t\,\mathrm{d}t - \frac{1}{\ln 2}\int_{0}^{1} t\ln t\,\mathrm{d}x = \\
&= -\frac{2}{\ln 2}1^2\left(\frac{\ln 1}{2} - \frac{1}{4}\right) = \\
&= \frac{1}{2\ln 2},
\end{aligned}$$

és ebből

$$H(Q_4(X)) \approx \frac{1}{2\ln 2} - \log 0.5 = \frac{1}{2\ln 2} + 1 \approx 1.72,$$

vagyis a hiba 0.1 alatti.

5.5. megoldás. Legyen $\varphi(x)$ az m várható értékű, σ szórású normális eloszlás sűrűségfüggvénye. Ekkor

$$\begin{aligned}
H(\varphi) &= -\int_{-\infty}^{\infty}\varphi(x)\log\varphi(x)\,\mathrm{d}x = \\
&= -\int_{-\infty}^{\infty}\varphi(x)\log\frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-\frac{(x-m)^2}{2\sigma^2}}\,\mathrm{d}x = \\
&= -\int_{-\infty}^{\infty}\varphi(x)\log\frac{1}{\sqrt{2\pi}\sigma}\,\mathrm{d}x + \frac{1}{\ln 2}\int_{-\infty}^{\infty}\varphi(x)\frac{(x-m)^2}{2\sigma^2}\,\mathrm{d}x = \\
&= -\log\frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty}\varphi(x)\,\mathrm{d}x + \frac{1}{\ln 2}\frac{1}{2\sigma^2}\int_{-\infty}^{\infty}x^2\varphi(x)\,\mathrm{d}x \\
&\quad -\frac{1}{\ln 2}\frac{2m}{2\sigma^2}\int_{-\infty}^{\infty}x\varphi(x)\,\mathrm{d}x + \frac{1}{\ln 2}\frac{m^2}{2\sigma^2}\int_{-\infty}^{\infty}\varphi(x)\,\mathrm{d}x = \\
&= -\log\frac{1}{\sqrt{2\pi}\sigma} + \frac{1}{\ln 2}\frac{1}{2\sigma^2}(\sigma^2+m^2) - \frac{1}{\ln 2}\frac{2m}{2\sigma^2}m + \frac{1}{\ln 2}\frac{m^2}{2\sigma^2} = \\
&= \log\sqrt{2\pi\mathrm{e}}\sigma.
\end{aligned}$$

5.6. megoldás. Legyen $\varphi(x)$ az m várható értékű, σ szórású normális eloszlás sűrűségfüggvénye, $f(x)$ pedig egy szintén m várható értékű, σ szórású, de egyébként tetszőleges eloszlás sűrűségfüggvénye. Ekkor

$$
\begin{aligned}
H(\varphi)-H(f) &= -\int_{-\infty}^{\infty}\varphi(x)\log\varphi(x)\,\mathrm{d}x+\int_{-\infty}^{\infty}f(x)\log f(x)\,\mathrm{d}x= \\
&= -\int_{-\infty}^{\infty}\varphi(x)\log\frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-\frac{(x-m)^2}{2\sigma^2}}\mathrm{d}x+\int_{-\infty}^{\infty}f(x)\log f(x)\,\mathrm{d}x= \\
&= -\int_{-\infty}^{\infty}\varphi(x)\log\frac{1}{\sqrt{2\pi}\sigma}\,\mathrm{d}x+\frac{1}{\ln 2}\int_{-\infty}^{\infty}\varphi(x)\frac{(x-m)^2}{2\sigma^2}\,\mathrm{d}x+\int_{-\infty}^{\infty}f(x)\log f(x)\,\mathrm{d}x= \\
&= -\log\frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty}\varphi(x)\,\mathrm{d}x+\frac{1}{\ln 2}\frac{1}{2\sigma^2}\int_{-\infty}^{\infty}x^2\varphi(x)\,\mathrm{d}x-\frac{1}{\ln 2}\frac{2m}{2\sigma^2}\int_{-\infty}^{\infty}x\varphi(x)\,\mathrm{d}x \\
&\quad+\frac{1}{\ln 2}\frac{m^2}{2\sigma^2}\int_{-\infty}^{\infty}\varphi(x)\,\mathrm{d}x+\int_{-\infty}^{\infty}f(x)\log f(x)\,\mathrm{d}x= \\
&= -\log\frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty}f(x)\,\mathrm{d}x+\frac{1}{\ln 2}\frac{1}{2\sigma^2}\int_{-\infty}^{\infty}x^2f(x)\,\mathrm{d}x-\frac{1}{\ln 2}\frac{2m}{2\sigma^2}\int_{-\infty}^{\infty}xf(x)\,\mathrm{d}x \\
&\quad+\frac{1}{\ln 2}\frac{m^2}{2\sigma^2}\int_{-\infty}^{\infty}f(x)\,\mathrm{d}x+\int_{-\infty}^{\infty}f(x)\log f(x)\,\mathrm{d}x= \\
&= -\int_{-\infty}^{\infty}f(x)\log\varphi(x)\,\mathrm{d}x+\int_{-\infty}^{\infty}f(x)\log f(x)\,\mathrm{d}x= \\
&= -\int_{-\infty}^{\infty}f(x)\log\frac{\varphi(x)}{f(x)}\,\mathrm{d}x\geq \\
&\geq -\frac{1}{\ln 2}\int_{-\infty}^{\infty}f(x)\left(\frac{\varphi(x)}{f(x)}-1\right)\mathrm{d}x= \\
&= -\frac{1}{\ln 2}\int_{-\infty}^{\infty}\varphi(x)\,\mathrm{d}x+\frac{1}{\ln 2}\int_{-\infty}^{\infty}f(x)\,\mathrm{d}x= \\
&= 0,
\end{aligned}
$$

ahol kihasználtuk a $\log x\leq\frac{1}{\ln 2}(x-1)$ egyenlőtlenséget.

5.7. megoldás. Legyen az X változó sűrűségfüggvénye a $[-A,A]$ intervallumban $f(x)$, azon kívül pedig 0. Ekkor az cX változó sűrűségfüggvénye a $[-cA,cA]$ inter-

vallumban $\frac{1}{c}f\left(\frac{x}{c}\right)$, azon kívül 0, differenciális entrópiája pedig

$$\begin{aligned}
H(cX) &= -\int_{-cA}^{cA} \tfrac{1}{c}f\left(\tfrac{x}{c}\right)\log\left(\tfrac{1}{c}f\left(\tfrac{x}{c}\right)\right)\mathrm{d}x = \\
&= -\int_{-A}^{A} f(x)\log\left(\tfrac{1}{c}f(x)\right)\mathrm{d}x = \\
&= -\int_{-A}^{A} f(x)\log f(x)\,\mathrm{d}x + \log c\int_{-A}^{A} f(x)\,\mathrm{d}x = \\
&= H(X)+\log c.
\end{aligned}$$

5.8. megoldás. Az N-szintű egyenletes kvantáló négyzetes torzítására ismert közelítés szerint

$$D(Q_N) \approx \frac{\Delta_N^2}{12},$$

amelyből $\Delta_N \approx 0.6$. A kvantáló kimenetének entrópiája

$$H(Q_N(X)) \approx H(f) - \log\Delta_N,$$

amelyhez szükségünk van a $[0,60]$ intervallumon egyenletes eloszlás differenciális entrópiájára:

$$\begin{aligned}
H(f) &= -\int_{0}^{60} f(x)\log f(x)\,\mathrm{d}x = \\
&= -\int_{0}^{60} \frac{1}{60}\log\frac{1}{60}\,\mathrm{d}x = \\
&= \log 60,
\end{aligned}$$

vagyis

$$H(Q_N(X)) \approx \log 60 - \log 0.6 = \log\frac{60}{0.6} = \log 100 \approx 6.64.$$

5.9. megoldás. A kvantáló négyzetes torzítása:

$$\begin{aligned}
D(Q) &= \int_{-\infty}^{\infty} (x-Q(x))^2\varphi(x)\,\mathrm{d}x = \\
&= \int_{-\infty}^{0} (x+a)^2\varphi(x)\,\mathrm{d}x + \int_{0}^{\infty} (x-a)^2\varphi(x)\,\mathrm{d}x = \\
&= \int_{-\infty}^{0} x^2\varphi(x)\,\mathrm{d}x + \int_{-\infty}^{0} 2xa\varphi(x)\,\mathrm{d}x + \int_{-\infty}^{0} a^2\varphi(x)\,\mathrm{d}x
\end{aligned}$$

$$
\begin{aligned}
&+\int_0^\infty x^2\varphi(x)\,\mathrm{d}x-\int_0^\infty 2xa\varphi(x)\,\mathrm{d}x+\int_0^\infty a^2\varphi(x)\,\mathrm{d}x=\\
&=\int_{-\infty}^\infty (x^2+a^2)\varphi(x)\,\mathrm{d}x-4\int_0^\infty ax\varphi(x)\,\mathrm{d}x=\\
&=1+a^2-\frac{2\sqrt{2}a}{\sqrt{\pi}}=\\
&=\left(a-\sqrt{\frac{2}{\pi}}\right)^2+1-\frac{2}{\pi},
\end{aligned}
$$

amelyet az $a=\sqrt{\frac{2}{\pi}}$ választás minimalizál. Ez az eredmény megegyezik a súlypont feltétellel, hiszen

$$
x_1=\frac{\int_{-\infty}^0 x\varphi(x)\,\mathrm{d}x}{\int_{-\infty}^0 \varphi(x)\,\mathrm{d}x}=\frac{-\frac{\sqrt{2}}{2\sqrt{\pi}}}{\frac{1}{2}}=-\sqrt{\frac{2}{\pi}}\qquad\text{és}\qquad x_2=\frac{\int_0^\infty x\varphi(x)\,\mathrm{d}x}{\int_0^\infty \varphi(x)\,\mathrm{d}x}=\frac{\frac{\sqrt{2}}{2\sqrt{\pi}}}{\frac{1}{2}}=\sqrt{\frac{2}{\pi}}.
$$

5.10. megoldás. Az X valószínűségi változó sűrűségfüggvénye:

$$
f(x)=\frac{\sqrt{3}}{\sqrt{2\pi}}\,\mathrm{e}^{-\frac{(x-1)^2}{2\frac{1}{3}}}.
$$

Az (5.3) egyenlet szerint az optimális kompresszorfüggvény

$$
\begin{aligned}
G^*(x)&=-\frac{1}{2}+\frac{\int_{-\infty}^x f^{1/3}(z)\,\mathrm{d}z}{\int_{-\infty}^\infty f^{1/3}(z)\,\mathrm{d}z}=\\
&=-\frac{1}{2}+\frac{\int_{-\infty}^x \sqrt[6]{\frac{3}{2\pi}}\,\mathrm{e}^{-\frac{(z-1)^2}{2}}\,\mathrm{d}z}{\int_{-\infty}^\infty \sqrt[6]{\frac{3}{2\pi}}\,\mathrm{e}^{-\frac{(z-1)^2}{2}}\,\mathrm{d}z}=\\
&=-\frac{1}{2}+\frac{\sqrt[6]{\frac{3}{2\pi}}\int_{-\infty}^{x-1}\frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-\frac{y^2}{2}}\,\mathrm{d}y}{\sqrt[6]{\frac{3}{2\pi}}\int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-\frac{y^2}{2}}\,\mathrm{d}y}=\\
&=-\frac{1}{2}+\Phi(x-1).
\end{aligned}
$$

5.11. megoldás. A tényleges tömörítés a kvantálási lépésben történik (a DCT után). Különböző tömörítési arányokat úgy kaphatunk, ha a kvantálási táblát (vagyis a kvantálási lépésközöket) megváltoztatjuk, s így a DCT együtthatók pontosabb vagy kevésbé pontos kvantáltját kapjuk.

5.12. megoldás.

a) Több helyes válasz is elképzelhető. Azt kell biztosítanunk, hogy az I képet küldjük el először, majd ezt követik a P képek. A B képek csak azon P és I képek után következhetnek, amelyeket felhasználtunk a predikció során. Például az $I_0P_2B_1P_5B_3B_4P_8B_6B_7I_9$ és az $I_0P_2P_5P_8B_1B_3B_4B_6B_7I_9$ egyaránt helyes sorrendek.

b) P_5 megsérülése esetén az összes olyan kép dekódolhatatlanná válik, amelyeknek a predikciójához felhasználtuk. B_3, B_7 és P_8 függenek P_5-től, míg B_1 nem.

5.7. Összefoglalás

Ebben a fejezetben a veszteséges tömörítési eljárásokat vettük sorra, amelyeknél nem követelmény a tömörített adatokból az eredeti adatok tökéletes helyreállítása [2, 3, 4, 7, 8, 12, 15]. Az érzékszerveinkkel sokszor szinte észrevehetetlen minőségromlást a tömörítés hatékonyságának növelése érdekében engedjük meg.

Megismerkedtünk a kvantálóval, amely folytonos értékkészletű jelek diszkrét értékkészletűvé alakítását végzi. Bemutattuk a Lloyd–Max kvantálótervező algoritmust és kiszámítottuk az egyenletes kvantáló torzítását [2, 7, 10]. Megvizsgáltuk a kompanderes és a vektorkvantálás műveletét, amely technikák nem egyenletes eloszlású jelek esetén kedvezőbb torzítást nyújtanak [2, 9]. Az utóbbi egy speciális esetére, a transzformációs kódolásra is vettünk néhány példát. A térben vagy időben közeli jelek és minták közötti összefüggéseket kihasználó eljárás, a prediktív (memóriával rendelkező) kódolás került ezután sorra, amely családból a DPCM-et, a delta modulációt és a Jayant-kvantálót vizsgáltuk, jellemző hibáikra is utalva [17].

A veszteséges tömörítési módszerek alkalmazásaként először beszéd- és hangtömörítőket tekintettünk [6, 10, 11, 17]. Kitértünk érzékszerveink (fülünk és szemünk) sajátosságaira és hiányosságaira, amelyeket kihasználva hatékony tömörítési módszereket dolgozhatunk ki. A képtömörítők közül a GIF és a JPEG formátumot vizsgáltuk, majd az MPEG videotömörítő került terítékre [6, 10, 16, 17].

Irodalomjegyzék

[1] Ash, R.B. *Information Theory*. Interscience Publishers, 1965.

[2] Cover, T., Thomas, J. *Elements of Information Theory, Second Edition*. John Wiley and Sons, New Jersey, 2006.

[3] Csibi S. (szerk.) *Információ közlése és feldolgozása*. Tankönyvkiadó, Budapest, 1986.

[4] Csiszár I., Fritz J. *Információelmélet*. ELTE TTK jegyzet, Budapest, 1986.

[5] Csiszár I., Körner J. *Information Theory: Coding Theorems for Discrete Memoryless Systems*. Akadémiai Kiadó, Budapest, 1981.

[6] Ferenczy P. *Video- és hangrendszerek*. Műszaki Könyvkiadó, Budapest, 1986.

[7] Gallager, R.G. *Information Theory and Reliable Communication*. Wiley, 1968.

[8] Géher K. (szerk.) *Híradástechnika*. Műszaki Könyvkiadó, Budapest, 1993.

[9] Gersho, A., Gray, R.M. *Vector Quantization and Signal Compression*. Kluwer, 1992.

[10] Gibson, J.D., Berger, T., Lookabaugh, T., Lindbergh, D., Baker, R.L. *Digital Compression for Multimedia (Principles and Standards)*. Morgan Kaufmann Publishers, San Francisco, 1998.

[11] Gordos G., Takács Gy. *Digitális beszédfeldolgozás*. Műszaki Könyvkiadó, Budapest, 1983.

[12] Györfi L., Győri S., Vajda I. *Információ- és kódelmélet*. TypoTEX Kiadó, Budapest, 2002.

[13] Linder T., Lugosi G. *Bevezetés az információelméletbe*. Műegyetemi Kiadó, Budapest, 1993.

[14] Massey, J.L. *Applied Digital Information Theory*. Course Notes, ETH Zürich, 1984.

[15] McEliece, R.J. *The Theory of Information and Coding*. Addison-Wesley, 1977.

[16] Rao, K.R., Hwang, J.J. *Techniques and Standards for Image, Video and Audio Coding*. Prentice Hall, 1996.

[17] Sayood, K. *Introduction to Data Compression*. Morgan Kaufmann Publishers, San Francisco, 1996.

[18] Shannon, C.E. A mathematical theory of communication. *Bell System Technical Journal*, 1948.

Tárgymutató