feldolgozást elérni, amilyen sebességgel a bitek beérkeznek. Röviden szólva, ha megduplázzuk a processzor sebességét, gyakran közel kétszeresére növelhetjük az átbocsátóképességet is. A hálózat kapacitásának megduplázása gyakran teljesen hatástalan, mivel a szűk keresztmetszetet általában a hosztok jelentik.

Második szabály: a szoftver overhead csökkentéséhez csökkentsük a csomagok számát

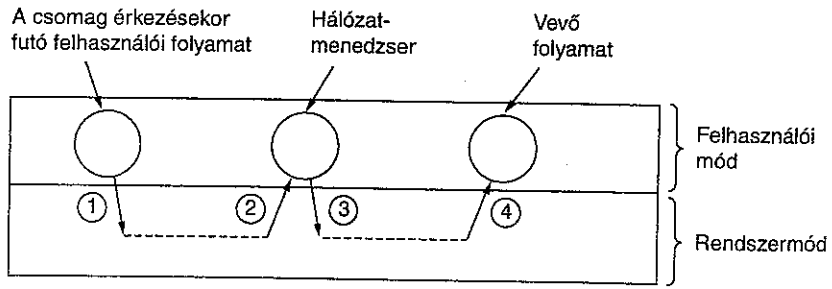
Egy TPDU feldolgozása TPDU-nként (például a fejrész értelmezése) és bájtonként (például ellenőrző összeg számítása) bizonyos mennyiségű overheaddel jár. Ha egymillió bájtot küldünk el, a bájtonkénti overhead a TPDU méretétől függetlenül ugyanaz. 128 bájtos TPDU használata azonban 32-szer akkora TPDU-nkénti overheadet jelent, mint 4 kilobájtos TPDU-k esetében. Ez a fajta overhead gyorsan jelentőssé válik.

A TPDU overheaden kívül az alsó rétegek overheadjét is számításba kell venni. Minden beérkező csomag megszakítást okoz. Egy modern RISC processzor esetén minden megszakítás kiüríti a processzor csővezetékét, megváltoztatja a gyorsítótárat, környezetváltást eredményez és rengeteg CPU regiszter elmentését kényszeríti ki. Egy n-szeres csökkenés az elküldött TPDU-k számában a megszakítás- és a csomag-overheadet is n-ed részére csökkenti.

Ez a megjegyzés azt sugallja, hogy a távoli oldal megszakításainak csökkentése érdekében érdemes sok adatot átvitel előtt összegyűjteni. A Nagle-féle algoritmus és Clark megoldása a buta ablak jelenségre ennek precíz elvégzésére tett próbálkozások.

Harmadik szabály: minimalizáljuk le a környezetváltások számát

A környezetváltások (például rendszer módból felhasználói módba) életveszélyesek. Ugyanazok a rossz tulajdonságaik vannak, mint a megszakításoknak, melyek közül a legrosszabb az, hogy kezdetben sokáig nem lesz cache találat. A környezetváltásokat megritkíthatjuk, ha az adatokat küldő könyvtári eljárást addig kényszerítjük adatok gyűjtésére, amíg jelentős mennyiségű össze nem gyűlt. Hasonlóképpen a vevőoldalon



6.48. ábra. *Egy csomag felhasználói módban futó hálózatmenedzserrel történő feldolgozásához szükséges négy környezetváltás*

a kicsi beérkező TPDU-kat össze kell gyűjteni és egyenként történő átadás helyett egy mozdulattal átdobni a felhasználónak, hogy minimalizálni lehessen a környezetváltások számát.

A legjobb esetben egy beérkező csomag környezetváltást okoz az aktuális felhasználóról a magra, majd a vevő folyamatra, hogy az beolvashassa a frissen érkezett adatokat. Sajnos, sok operációs rendszer esetében még további környezetváltások is történnek. Például, ha a hálózatmenedzser speciális folyamatként felhasználói módban fut, egy csomag érkezése valószínűleg egy környezetváltást okoz az aktuális felhasználóról a magra, majd onnan a hálózatmenedzserre, ezt követően egy újabb környezetváltás történik vissza a magra, és végül egy utolsó vissza a vevő folyamatra. Ezt a sorozatot láthatjuk a 6.48. ábrán. Ezek a csomag érkezésekor bekövetkező környezetváltások nagyon sok processzoridőt elpazarolnak és lerontják a hálózat teljesítőképességét.

Negyedik szabály: minimalizáljuk a másolást

Többszörös másolatok készítése még a környezetváltásoknál is rosszabb. Nem szokatlan, hogy egy beérkező csomagot háromszor vagy négyszer átmásolnak a benne foglalt TPDU kézbesítése előtt. Miután a csomag megérkezett a hálózat illesztő egy speciális, a kártyán található pufferébe, tipikusan egy magpufferbe másolják. Onnan a hálózati réteg pufferébe kerül, majd a szállítási réteg pufferébe, végül a fogadó alkalmazási folyamathoz.

Egy okos operációs rendszer egyszerre egyetlen szót másol, de nem szokatlan, hogy öt utasítás szükséges szavanként (egy olvasás, egy tárolás, egy indexregiszter növelése, a másolás végének ellenőrzése és egy feltételes elágazás). Egy 50 MIPS-es gépen minden csomagról három másolat készítése (32 bites) szavanként öt utasítás végrehajtása esetén beérkező bájtonként 75 ns-ot vesz igénybe. Egy ilyen gép ezért legfeljebb 107 Mb/s sebességgel érkező adatokat tud fogadni. Ha a fejrészfeldolgozásból, megszakításkezelésből és környezetváltásból eredő overheaddel is számolunk, jó, ha 50 Mb/s elérhető, és az adatok tényleges feldolgozását még mindig nem vettük számításba. Világos, hogy egy 1 Gb/s sebességű vonal kezelésére nem is gondolhatunk.

Valójában valószínűleg még az 50 Mb/s sebességű vonal sem kezelhető. A fönti számításban feltételeztük, hogy az 50 MIPS-es gép bármilyen 50 millió utasítást végre tud hajtani másodpercenként. A valóságban a számítógépek csak akkor működhetnek ilyen sebességgel, ha nem hivatkoznak a memóriára. A memóriaműveletek gyakran háromszor lassabbak is lehetnek a regiszter-regiszter utasításoknál, ezért különösen jónak számít, ha az 1 Gb/s vonalból 16 Mb/s-ot ki lehet hozni. Jegyezzük meg, hogy a hardvertámogatás itt nem segít. A probléma leginkább az operációs rendszer által végzett másolás.

Ötödik szabály: nagyobb sávszélességet lehet vásárolni, de kisebb késleltetést nem

A következő három szabály inkább a kommunikációra vonatkozik, mint a protokollal végzett feldolgozásra. Az első szabály szerint, ha nagyobb sávszélességet akarunk, azt csak megvásárolni lehet. Egy második üvegszálat helyezve az első mellé a sávszélesség a kétszeresére nő, viszont a késleltetés semennyit sem csökken. A késleltetés csökkentése a protokoll szoftver, az operációs rendszer vagy a hálózati interfész fejlesztését teszi szükségessé. Még ha ezek közül mindegyiket el is végeztük, a késleltetés nem fog csökkenni, ha a szűk keresztmetszet az átvitel ideje.

Hatodik szabály: jobb elkerülni a torlódást, mint utána talpra állni

A régi mondás, miszerint egy szem megelőzés felér egy véka gyógyítással, biztosan igaz a hálózati torlódásokra is. Amikor torlódás van egy hálózaton, csomagok vesznek el, sávszélesség megy kárba, haszontalan késleltetések lépnek föl és így tovább. Utána a talpra állás időt és türelmet igényel. Legjobb, ha nem hagyjuk hogy előforduljon. A torlódás elkerülése olyan, mint a védőoltás: egy kicsit fáj amikor beadják, de megelőz valamit, ami sokkal fájdalmasabb lenne.

Hetedik szabály: az időtúllépések elkerülése

Az időzítők szükségesek a hálózatban, de módjával kell használni őket és az időtúllépések számát minimalizálni kell. Amikor egy időzítő lejár, általában egy tevékenység megismétlődik. Ha tényleg szükség van az ismétlésre, ám legyen, de a fölösleges ismétlés pazarlás.

A pluszmunka elkerülésének az a módja, hogy az időzítőket gondosan, inkább egy kicsit hosszabbra állítjuk. Egy időzítő, ami hosszabb ideig működik, egy kis extra késleltetést csinál az összeköttetésen abban a (valószínűtlen) esetben, ha egy TPDU elvész. Az az időzítő, ami lejár, amikor még nem kéne, értékes processzoridőt használ, sávszélességet pazarol és fölösleges többletterhelést okoz akár több tucat routernek is.

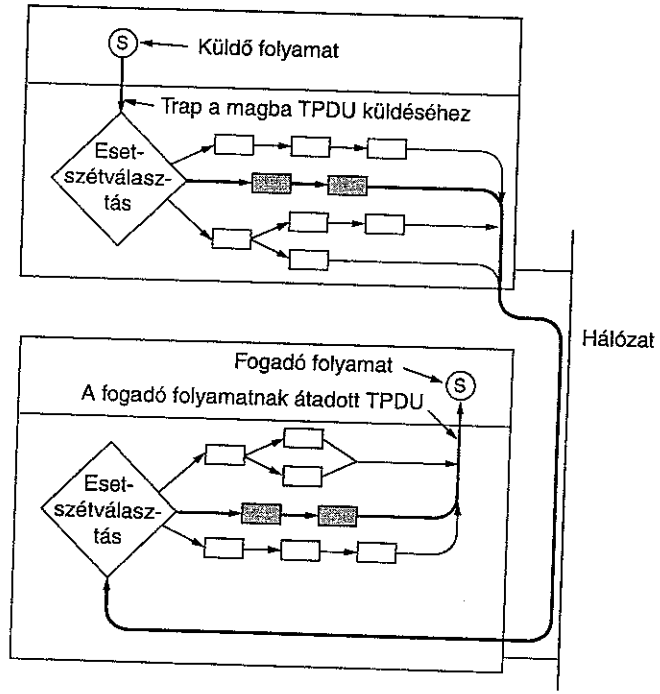
6.6.4. Gyors TPDU feldolgozás

A fönti történet tanulsága az, hogy a gyors hálózat legfőbb kerékkötője a protokoll szoftver. Ebben a részben ennek a szoftvernek felgyorsítására adunk néhány módszert. További információt Clark és munkatársai (1989); Edwards és Muir (1995); Chandranmenon és Varghese (1995) műveiben találunk.

A TPDU feldolgozási overheadjének két összetevője van: a TPDU-nkénti és a bájtonkénti overhead. Mindkettő ellen harcolni kell. A gyors TPDU feldolgozás kulcsa az, hogy válasszuk külön a normális esetet (egyirányú adatátvitel), és kezeljük különleges módon. Bár egy sor speciális TPDU szükséges ahhoz, hogy eljussunk a *ESTABLISHED* állapotba, ha már ott vagyunk, a TPDU-k feldolgozása nyilvánvaló, amíg egyik nem kezdeményezi az összeköttetés bontását.

Kezdjük a tanulmányozást az *ESTABLISHED* állapotban levő küldő oldalán, amikor van átküldésre váró adat. Az egyértelműség kedvéért feltételezzük, hogy a szállítási entitás a mag része, bár ugyan ezeket az ötleteket lehet alkalmazni akkor is, ha ez egy felhasználói folyamat vagy ha egy könyvtári eljárás a küldő folyamatban. A 6.49. ábrán a küldő folyamat SEND mag hívást ad ki. Az első dolog, amit a szállítási entitás csinál, hogy megvizsgálja, normális esetről van-e szó: az állapot *ESTABLISHED*, egyik oldal se próbálja bontani az összeköttetést, egy szabályos (azaz nem sávon kívüli) teljes TPDU-t küldenek és a vevőnél elég nagy ablak áll rendelkezésre. Ha minden feltétel teljesül, nincs szükség további tesztekre, és a gyors utat lehet követni a szállítási entitáson belül.

Normális esetben az egymást követő adat TPDU-k fejrészei majdnem azonosak. Hogy ezt a tényt kiaknázhassa, a szállítási entitás egy fejrész prototípust tárol. A gyors út elején, amilyen gyorsan lehet, ezt szavanként egy átmeneti pufferbe másolja. Azokat a mezőket, amelyek TPDU-ról TPDU-ra változnak, a pufferben fölülírja. Ezek a mezők – mint például a sorszám – gyakran egyszerűen származtathatók az állapotvál-



6.49. ábra. *A küldőtől a vevőig vezető gyors utat vastag vonal jelzi. Ezen út feldolgozási lépéseit besatíroztuk*

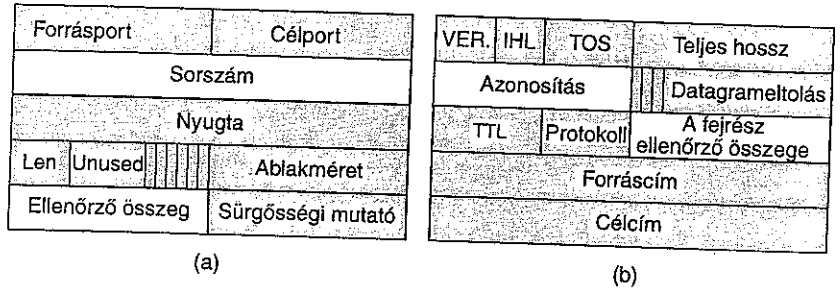
tozókból. Ezután átad a hálózati rétegnek két, a teljes hálózati fejrészre, illetve a felhasználói adatra mutató mutatót. Itt ugyanezt a stratégiát lehet követni (a 6.49. ábrán nem tüntettük fel). Végül a hálózati réteg átadja a csomagot az adatkapcsolati rétegnek továbbításra.

Hogy lássuk az elmélet gyakorlati működését, vegyük például a TCP/IP-t. A 6.50.(a) ábrán a TCP fejrészt láthatjuk. Azokat a mezőket, amelyek az egyirányú folyamban egymás után következő TPDU-kban azonosak, besatíroztuk. A küldő szállítási entitás tennivalója mindössze öt szót átmásolni a fejrész prototípusból a kimeneti pufferbe, kitölteni a *sorszám* mezőt (egy szó bemásolása a memóriából), kiszámítani az ellenőrző összeget és növelni a memóriában a sorszám értékét. Ezután átadhatja a fejrészt és az adatot egy speciális IP eljárásnak, hogy továbbítson egy szabályos, teljes méretű TPDU-t. Az IP ezután átmásolja a saját ötszavas fejrész prototípusát [lásd 6.50.(b) ábra] a pufferbe, kitölti az azonosítás mezőt és kiszámítja a saját ellenőrző összegét. A csomag most átvitelre kész.

Vegyük most szemügyre a gyors feldolgozást a 6.49. ábrán látható vevő szemszögéből. Első lépése a beérkező TPDU összeköttetés rekord meghatározása. ATM esetén a rekord könnyen megtalálható: a *VPI* mező indexként használható az útvonaltáblában, hogy kijelölje az útvonalhoz tartozó virtuális áramkör táblázatot, amelyben a *VCI* mező már az összeköttetés rekordot azonosítja. TCP esetében az összeköttetés rekord hash táblázatban tárolható, amelynek kulcsát a két IP cím és a két portszám valamely egyszerű függvénye adja. Amikor megvan az összeköttetés-rekord, mindkét címet és portot össze kell hasonlítania, hogy ellenőrizhesse, vajon a helyes rekordot találta-e meg.

Egy, a legutóbb használt rekordra mutató mutató gyakran tovább gyorsíthatja az összeköttetés-rekord előkeresését, ha először mindig azzal próbálkozik a TCP. Clark és munkatársai (1989) kipróbálták ezt, és 90 százalékosnál magasabb találati arányt tapasztaltak. Más keresési heurisztikákkal McKenney és Dove (1992) foglalkozott.

Ezután az entitás megvizsgálja, hogy normális TPDU érkezett-e: az állapot *ESTABLISHED*, egyik oldal se próbálja bontani az összeköttetést, a TPDU maximális méretű, nincs speciális jelzése, és a sorszám egyezik a várt értékkel. Ezek a vizsgálatok csak néhány utasítást igényelnek. Ha minden feltétel teljesül, a vezérlés egy speciális gyors út eljárásra kerül a TCP-n belül.



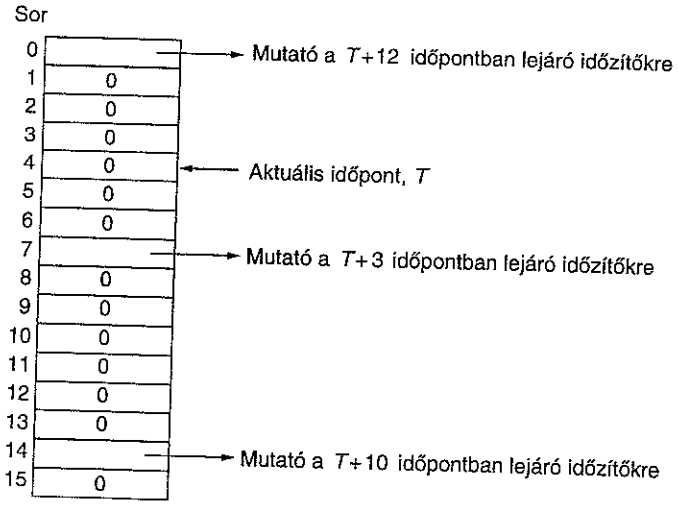
6.50. ábra. *(a) TCP fejrész. (b) IP fejrész. Mindkét esetben a satírozott mezőket változtatás nélkül veszik a prototípusból*

A gyors út rutin frissíti az összeköttetés rekordot és átmásolja az adatot a felhasználóhoz. Másolás közben ellenőrző összeget is számít, így elkerülhető egy további adatfeldolgozási menet. Ha az ellenőrző összeg rendben van, frissíti az összeköttetés-rekordot, és visszaküld egy nyugtát. Azt az általános módszert, aminek során először gyors ellenőrzéssel megbizonyosodik arról, hogy a várt fejrész érkezett-e meg, majd az eset kezelését speciális eljárás végzi, ezt **fejrész-előrejelzésnek (header prediction)** nevezik. Sok TCP implementáció alkalmazza. Ha ezt, és az itt leírt többi optimalizációt együtt használják, elérhető, hogy a TCP a helyi memória-memória másolás sebességének 90 százalékával fusson, feltéve hogy maga a hálózat elég gyors.

Két másik terület, ahol jelentős teljesítőképesség-javulás érhető el, a pufferkezelés és az időzítőkezelés. A pufferkezelés trükkje – mint fönt említettük – a fölösleges másolás elkerülése. Az időzítőkezelés fontos, mert szinte egyetlen elindított időzítő sem jár le. A TPDU-k elvesztését figyelik, de a legtöbb TPDU és visszaküldött nyugta rendben megérkezik. Ezért fontos arra optimalizálni az időzítők kezelését, hogy ritkán járnak le.

Egy általánosan használt módszer az, hogy lejárat szerint sorba rendezve láncolt listában tároljuk az időzítőeseményeket. A lista feje egy számlálót tartalmaz, ami azt mutatja, hogy mostantól számítva hány óraütés múlva jár le a hozzá rendelt időzítő. Az egymásra következő bejegyzések azt tartalmazzák, hogy az egymás után következő időzítők az előzőtől számítva mikor járnak le. Így ha az időzítők 3, 10 és 12 ütem múlva járnak le, a három számláló értéke rendre 3, 7 és 12 lesz.

Minden óraütéskor a lista fejének számlálóját csökkenti az entitás. Amikor eléri a 0-t, a hozzá tartozó eseményt feldolgozza és a következő elem lesz a lista feje. Ennek nem kell módosítani a számlálóját. Ebben a megoldásban időzítők beszúrása és törlése drága művelet, aminek végrehajtási ideje arányos a lista hosszával.



6.51. ábra. *Az időkerék*

Egy hatékonyabb megközelítés alkalmazható, ha a maximális időtartam értéke korlátos és előre ismert. Itt egy **időkeréknek (timing wheel)** nevezett tömböt használunk, amit a 6.51. ábrán láthatunk. Minden sor egy óraütésnek felel meg. Az ábrán a $T = 4$ időpillanatot ábrázoltuk. Az időzítők mostantól számítva 3, 10 és 12 óraütés múlva járnak le. Ha hirtelen egy hét ütem múlva lejáró új időzítőt kell beszúrni, csak egy új bejegyzést nyitunk a 11. sorban. Hasonlóan ha a $T + 10$ időpontra beállított időzítőt törölni akarjuk, a 14. sorban kezdődő listát kell megkeresni és a kívánt elemét törölni. Vegyük észre, hogy a 6.51. ábrán látható tömb nem tud $T + 15$-nél későbbre állított időzítőket kezelni.

Amikor üt az óra, az aktuális idő mutatóját egy sorral előremozdítjuk (cirkulárisan). Ha a bejegyzés, amire most mutat, nullától különbözik, az összes időzítőt feldolgozzuk. Az alapötlet több variációját tárgyalja Varghese és Lauck (1987).

6.6.5. Gigabites hálózatok protokolljai

A kilencvenes évek elején kezdtek föltűnni a gigabites hálózatok. Az emberek első reakciója az volt, hogy alkalmazzuk rajtuk a régi protokollokat, de hamar különböző problémák jelentkeztek. Ebben a részben megbeszélünk néhány ilyen problémát és a megoldásukra kifejlesztett protokollok új irányzatait. További információt találhatunk Baransel és munkatársai (1995); Partridge (1994) műveiben.

Az első probléma az, hogy sok protokoll 16 vagy 32 bites sorszámokat használ. A régi időkben a 2^{32} elég jó közelítése volt a végtelennek, de többé nem az. 1 Gb/s adatsebességnél nagyjából 32 másodpercig tart 2^{32} bájtot elküldeni. Ha a sorszámok – mint a TCP esetében – bájtokra vonatkoznak, a küldő elkezdheti a 0. bájtnál, elviharzik, és 32 másodperccel később ismét a 0. bájtnál tart. Még ha azt feltételezzük, hogy minden bájtot nyugtáztak, a küldő nem adhat biztonságosan új adatot 0 címkétől kezdve, mert a régi csomagok esetleg még mindig bolyonganak valamerre. Az Internetben például 120 másodpercig élhetnek a csomagok. Ha bájtok helyett a csomagokat számozzuk, a probléma kevésbé súlyos, hacsak nem 16 bitesek a sorszámok, amikor még rosszabb a helyzet.

Az a probléma, hogy sok protokolltervező anélkül, hogy rögzítette volna, egyszerűen azt feltételezte, hogy az összes sorszám elhasználásához szükséges idő jóval nagyobb a maximális csomagélettartamnál. Következésképpen még gondolni sem kellett arra a problémára, hogy régi kettőzések még mindig létezhetnek, amikor a sorszámok körbefordulnak. Gigabites sebességnél ezek a ki nem mondott feltételezések kudarcot vallanak.

Egy másik probléma az, hogy a kommunikációs sebességek sokkal gyorsabban növekedtek, mint a számítási sebességek. (Üzenet a számítástechnikai mérnököknek: Menjetek és verjétek meg azokat a távközlési szakembereket! Számítunk rátok.) A hetvenes években az ARPANET 56 kb/s sebességgel működött, a számítógépek körülbelül 1 MIPS-esek voltak. A csomagok hossza 1008 bit volt, ezért az ARPANET körülbelül 56 csomagot tudott másodpercenként továbbítani. A csomagonként rendelkezésre álló 18 ezredmásodperc alatt egy hoszt 18 000 utasítást szánhatott egy csomag feldolgozására. Természetesen ez fölemésztette volna az egész processzort, ezért, ha

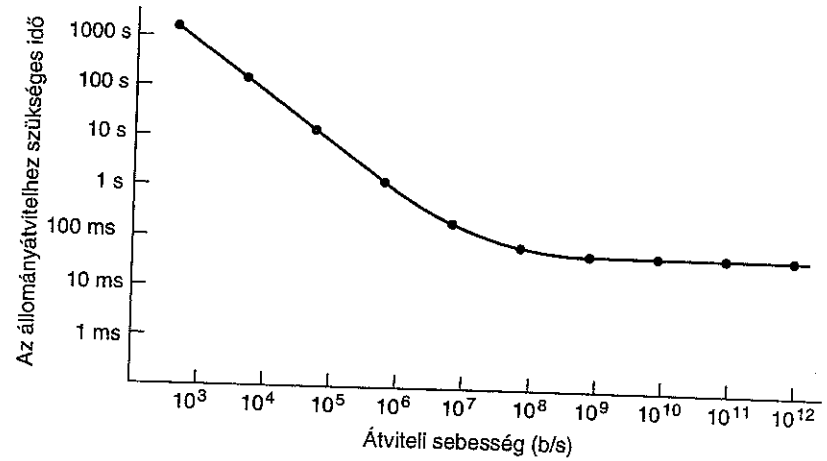
9000 utasítást szentelt egy csomagra, még mindig megmaradt a fél processzor a valódi munka elvégzésére.

Hasonlítsuk össze ezeket a számokat a modern 100 MIPS-es számítógépek esetével, melyek 4 kilobájtos csomagokat cserélnek egy gigabites vonalon. Másodpercenként 30 000 csomag folyhat be, ezért egy csomag feldolgozását meg kell oldani 15 μs alatt, ha a fél processzort alkalmazások számára akarjuk fenntartani. 15 μs alatt egy 100 MIPS-es számítógép 1500 utasítást tud végrehajtani, ez csak egy hatoda annak, ami egy ARPANET hoszt rendelkezésére állt. Ezenkívül a modern RISC utasítások kevesebb dolgot végeznek el, mint amennyit régi CISC utasítások tettek meg, ezért a probléma még rosszabb, mint amilyennek tűnik. A tanulság az, hogy kevesebb idő van protokollfeldolgozásra, mint annak idején volt, ezért a protokolloknak egyszerűbbeknek kell lenniük.

Egy harmadik probléma az, hogy az n visszalépéses protokoll gyengén teljesít nagy sávszélesség-késleltetés szorzattal rendelkező vonalakon. Vegyünk például egy 1 Gb/s sebességgel működő 4000 km-es vonalat. A körülfordulási idő 40 ms, mialatt a küldő 5 megabájtot tud elküldeni. Ha a vevő hibát detektál, ez 40 ms alatt jut az adó tudomására. N visszalépéses protokoll alkalmazása esetén a küldőnek nemcsak a rosszat, hanem esetleg az egész utána következő 5 megabájtnyi csomagot is újra kell küldenie. Világos, hogy ez óriási erőforrás-pazarlás.

A negyedik probléma a gigabites vonalakkal az, hogy ezek alapvetően mások, mint a megabites vonalak: a hosszú vonalak inkább késleltetés-, semmint sávszélesség-korlátosak. A 6.52. ábrán egy 1 megabites állomány 4000 km-es távolságra történő elküldésének időszükségletei láthatók különböző átviteli sebességek esetén. 1 Mb/s sebességig az átviteli időben az adási sebessége meghatározó. 1 Gb/s-nél a 40 ms-es körülfordulási idő dominál az 1 ms fölött, ami a bitek üvegszálba töltéséhez szükséges. A sávszélesség további növelése aligha járhat eredménnyel.

A 6.52. ábrán a hálózati protokollok szerencsétlen hatásait láthatjuk. Ezek szerint



6.52. ábra. *1 megabites állomány átviteléhez és nyugtázásához szükséges idő*

az RPC-hez hasonló megáll-és-vár protokollok teljesítőképességének veleszületett felső korlátja van. Ezt a határt a fénysebesség szabja meg. Az optika területén semmilyen mértékű technikai előrelépés nem fejlesztheti a használt anyagokat (habár új fizikai törvények segítenének).

Egy ötödik említésre érdemes probléma a többitől eltérően nem technológiai, vagy protokollokkal kapcsolatos, hanem az új alkalmazások következménye. Ma már természetes, hogy sok gigabites alkalmazásnak, mint például a multimédia, a csomagbeérkezési idők szórása éppolyan fontos, mint maga a késleltetések átlaga. A kicsi, de egyenletes kézbesítés gyakran előnyösebb egy nagy sebességű, viszont ugráló átvitelnél.

Foglalkozzunk most a problémák helyett kezelési módjaikkal. Először néhány általános megjegyzést teszünk, aztán áttekintjük a protokollmechanizmusokat, csomagok felépítését és a protokoll-szoftvert.

Az alapvető elv, amit minden gigabites hálózat tervezőjének kívülről kéne tudni:

Tervezni sebesség-, és nem sávszélesség-optimumra kell.

A régi protokollokat gyakran úgy tervezték, hogy minimalizálják a vonalon tartózkodó bitek számát, gyakran oly módon, hogy kicsi mezőket használtak és bájtokba vagy szavakba zsúfolták azokat. Mostanában már bőven van sávszélesség. A protokollfeldolgozás jelenti a problémát, ezért a protokollokat úgy kell tervezni, hogy a feldolgozást minimalizálják.

Erős lehet a kísértés a gyors működést a hálózati illesztők hardver megvalósításával elérni. Ez a stratégia azért nehézkes, mert hacsak nem végtelenül egyszerű a protokoll, a hardver csak egy bedugható kártyát jelent egy másik processzorral és a saját programjával. Hogy elkerüljék a hálózati koprocesszor használatát, amely annyira drága, mint a központi processzor, gyakran egy lassúbb IC-t használnak. Ennek a tervezésnek az a következménye, hogy a fő (gyors) processzor idejének nagy részét tétlenül tölti arra várakozva, hogy a második (lassú) processzor elvégezze a kritikus munkát. Tévhit, hogy a fő processzornak várakozás közben van más tennivalója. Ezenkívül, amikor két általános célú processzor kommunikál egymással, versenyhelyzetek alakulhatnak ki, ezért kifinomult protokollok szükségesek helyes szinkronizálásukhoz. Általában az a legjobb megközelítés, hogy egyszerű protokollokat készítünk és a központi processzorral végeztetjük el a munkát.

Vegyük most szemügyre a visszacsatolás kérdését a nagy sebességű protokollokban. A (viszonylag) hosszú késleltetésű hurok miatt a visszacsatolást el kell kerülni: túl sokáig tart, amíg a vevő jelez a küldőnek. A visszacsatolás egy példája az átviteli sebesség szabályozása csúszóablakos protokollal. Hogy elkerüljük a (hosszú) késleltetéseket, melyek abból adódnak, hogy a vevő ablakfrissítéseket küld az adónak, előnyösebb egy sebesség alapú protokoll használata. Ebben a megoldásban a küldő mindent elküldhet, amit akar, feltéve, hogy nem küld gyorsabban egy bizonyos sebességnél, amiben a küldő és a vevő előre megállapodott.

Egy másik példa a visszacsatolásra a Jacobson-féle lassú kezdet algoritmus. Ez az algoritmus több próbálkozást végez, hogy megállapítsa a hálózat terhelhetőségét. Nagy sebességű hálózatoknál a hálózat viselkedését mérő fél tucat ilyen kis próba hatalmas méretű sávszélességet pazarol. Egy hatékonyabb módszer az, ha az összeköttetés felépítésekor a küldő, a vevő és a hálózat mind lefoglalja a szükséges erőforrásokat. Az erőforrások előre történő lefoglalása azzal az előnnyel is jár, hogy így egyszerűbben csökkenthető a dzsitter. Röviden szólva nagyobb sebességek világában a tervezés feltartóztathatatlanul az összeköttetés alapú működés felé tart, vagy legalábbis valami ahhoz nagyon közelihez.

A csomagok felépítése fontos tényező a gigabites hálózatokban. A fejrésznek a feldolgozási idő csökkentése érdekében olyan kevés bitet kell tartalmaznia, amennyit csak lehetséges. Ezeknek a mezőknek ugyanakkor elég nagynak kell lenni ahhoz, hogy feladatukat elláthassák, és a feldolgozás megkönnyítéséhez szóhatáron kell kezdődniük. Ebben a környezetben „elég nagy" azt jelenti, hogy olyan problémák ne fordulhassanak elő, mint a sorszámok körbefordulása miközben a régi csomagok még élnek, vagy a vevők nem tudnak elég nagy ablakméretet bejelenteni, mert túl kicsi az ablak mező és így tovább.

A fejrészt és az adatot külön ellenőrző összeggel kell ellátni két okból kifolyólag is. Egyrészt, hogy lehetőség legyen csak a fejrészre számítani és az adatot kihagyni. Másrészt azért, hogy ellenőrizni lehessen a fejrészt az adat felhasználói területre való másolása előtt. Célszerű akkor végezni az adatok ellenőrző összegének számítását, miközben az adat másolása folyik a felhasználói területre, mert ha a fejrész sérült, a másolás rossz folyamathoz történhet. Hogy elkerülhető legyen a rossz helyre történő másolás, de az ellenőrző összeget másolás közben lehessen kiszámítani, lényeges, hogy a két ellenőrző összeg egymástól független legyen.

A maximális elküldhető adatmennyiségnek nagynak kell lenni, hogy hatékony lehessen a működés még hosszú késleltetések esetén is. Emellett minél nagyobb az adatblokk, a sávszélességből annál kisebb részt foglalnak el a fejrészek.

Egy másik értékes képesség egy normális mennyiségű adat elküldésének lehetősége az összeköttetés-kérés során. Ily módon egy oda-vissza út ideje megtakarítható.

Végül érdemes néhány szót szólni a protokoll-szoftverről. A kulcsgondolat az, hogy a sikeres esetre kell összpontosítani. Sok régi protokoll azt próbálta kihangsúlyozni, hogy mi a teendő hiba esetén (pl. amikor elvész egy csomag). Gyorsan futó protokollok készítéséhez a tervezőnek meg kell próbálni csökkenteni a feldolgozási időt abban az esetben, ha minden jól megy. Másodlagos fontosságú a hibás esetre minimalizálni.

Egy másik szoftverkérdés a másolás idejének minimalizálása. Mint korábban láttuk, az adatok másolása sokszor az overhead fő forrása. Ideális esetben a hardvernek minden beérkező csomagot folytonos adatblokk formájában kéne a memóriába töltenie. A szoftvernek ezután egyetlen blokkmozgatással a pufferbe kéne ezt másolnia. A gyorsítótár működésétől függően még egy másoló ciklus használata is kerülendő lehet. Másképpen, 1024 szó másolásának leggyorsabb módja 1024 MOVE utasítás (vagy 1024 betöltés-tárolás pár) kiadása. A másolórutin annyira kritikus, hogy gondosan meg kell formálni assembly nyelven, hacsak nem lehet a fordítót ügyesen arra kényszeríteni, hogy pontosan az optimális kódot generálja.

A nyolcvanas évek végén heves, de rövid érdeklődéshullám támadt a gyors különleges célú protokollok iránt, mint amilyen a NETBLT (Clark és munkatársai, 1987),

VTMP (Cheriton és Williamson, 1989) és XTP (Chesson, 1989). Egy tanulmányt közöltek Doeringer és munkatársai (1990). Ma már viszont az a tendencia, hogy egyszerűsíteni kell az általános célú protokollokat, hogy gyorsak legyenek. Az ATM és az IPv6 sokat magán hordoz a fönt említett tulajdonságok közül.

6.7. Összefoglalás

A szállítási réteg a rétegezett protokollok megértésének kulcsa. Különböző szolgálatokat nyújt, melyek közül a legfontosabb a küldő és a vevő között végtől végig terjedő megbízható összeköttetés alapú bájtfolyam. A szállítási réteg szolgálati primitíveken keresztül érhető el, amelyek összeköttetések létesítését, használatát és bontását teszik lehetővé.

A szállítási protokolloknak képesnek kell lenniük megbízhatatlan hálózatok fölött történő összeköttetés-kezelésre. Az összeköttetések létesítését késleltetett kettőzött csomagok létezése nehezíti, melyek kellemetlen időpontban is felbukkanhatnak. Kezelésükhöz az összeköttetések létesítésekor háromutas kézfogás protokoll használata szükséges. Az összeköttetések bontása egyszerűbb a felépítésnél, de a két-hadsereg probléma miatt messze nem triviális.

Még ha teljesen megbízható is a hálózati réteg, a szállítási rétegnek bőven van dolga, amint azt a példánkban is láthattuk. Meg kell valósítania az összes szolgálati primitívet, összeköttetéseket és időzítőket kell kezelnie, hiteleket kell foglalnia és érvényesítenie.

Az Internet fő szállítási protokollja a TCP. 20 bájtos fejrészt illeszt minden szegmensre. A szegmenseket az Internetben található routerek szétdarabolhatják, ezért a hosztoknak fel kell készülniük a rekonstrukció elvégzésére. Hatalmas munka fekszik a TCP teljesítőképességének optimalizálásában, amihez Nagle, Clark, Jacobson, Karn és mások algoritmusait is felhasználták.

Az ATM-nek négy protokollja van az AAL rétegben. Mindegyik cellákra tördeli az üzeneteket, és a rendeltetési helyen állítja vissza őket a cellákból. A konvergencia és SAR alrétegek különbözőképpen bővítik az üzeneteket saját fej- és farokrészeikkel. Ennek során 44–48 bájtot hagynak szabadon a cella adat mezejéből.

A hálózat teljesítőképességét tipikusan a protokoll- és TPDU feldolgozási overhead korlátozza, és ez a helyzet nagyobb sebességnél tovább romlik. A protokollokat úgy kell tervezni, hogy minimalizálják a TPDU-k és környezetváltások számát és a TPDU-k mozgatását. Gigabites hálózatokban inkább sebesség, mint hitel alapú forgalomszabályozást megvalósító egyszerű protokollokra van szükség.

Feladatok

1. A 6.3. ábrán látható szállítási primitív példánkban a LISTEN blokkoló hívás. Feltétlenül szükséges ez? Ha nem, magyarázzuk el, hogyan lehetne egy nem blokkoló primitívet használni! Milyen előnnyel rendelkezne ez a szövegben leírtakkal szemben?

2. A 6.5. ábra modelljében azt feltételeztük, hogy a hálózati réteg csomagokat veszíthet, ezért azokat egyenként nyugtázni kell. Tegyük fel, hogy a hálózati réteg 100 százalékosan megbízható, és sohasem veszít csomagokat. Milyen változtatásokat kell – ha egyáltalán szükséges – a 6.5. ábrán végezni?

3. Képzeljünk el egy általánosított n-hadsereg problémát, amelyben bármely két hadsereg megegyezése elegendő a győzelemhez. Van olyan protokoll, ami győzelemre viszi a kékeket?

4. Tegyük fel, hogy a kezdeti sorszámok előállítására az óravezérelt módszert használjuk 15 bites számlálóval, mint órával. Az óra 100 ezredmásodpercenként üt egyet, a maximális csomagélettartam 60 s. Milyen gyakran történik újraszinkronizáció

   (a) a legrosszabb esetben?

   (b) amikor az adatok percenként 240 sorszámot használnak el?

5. Miért kell a T maximális csomagélettartamnak olyan nagynak lenni, hogy biztosítsa nemcsak a csomagok, hanem a nyugtáik kihalását is?

6. Képzeljük el, hogy összeköttetések létesítéséhez nem háromutas, hanem kétutas kézfogás protokollt használunk. Másképpen fogalmazva a harmadik üzenet nem szükséges. Kialakulhatnak ekkor holtpontok? Adjunk egy példát, vagy bizonyítsuk be, hogy nem alakulhatnak ki!

7. Tekintsük a hosztösszeomlásokból történő talpra állás problémáját (lásd a 6.18. ábrát). Ha az írás és nyugta küldése (vagy a fordított sorozat) közötti időintervallum viszonylag kicsivé tehető, mi a küldő illetve a vevő legjobb stratégiája a protokoll hibázási esélyének minimalizálására?

8. Lehetséges-e holtpont kialakulása a szövegben leírt szállítási entitás esetében?

9. A 6.20. ábra szállítási entitásának írója kíváncsiságból úgy döntött, hogy számlálókat épít a *sleep* eljárásba, hogy statisztikát készítsen a *conn* tömbről. Többek között a hét állapot közül az egyes állapotokban levő összeköttetések n_i (i = 1, ..., 7)

számát is vizsgálta. Miután az adatok analizálására írt egy komoly FORTRAN programot, programozónk észrevette, hogy a $\Sigma n_i = MAX\_ÖSSZ$ egyenlőség mindig teljesülni látszik. Van ezen kívül valamilyen állandó mennyiség, ami csak ezt a hét változót tartalmazza?

10. Mi történik, ha a 6.20. ábra szállítási entitásának felhasználója 0 hosszúságú üzenetet küld? Vitassuk meg a válasz jelentőségét!

11. Minden, a 6.20. ábra szállítási entitásában potenciálisan előforduló eseményre döntsük el, hogy legális-e vagy sem amikor a felhasználó *sending* állapotban alszik!

12. Soroljuk fel a hitel alapú protokoll előnyeit és hátrányait a csúszóablakos protokollal szemben!

13. A datagramok feldarabolását és összeállítását az IP végzi a TCP számára láthatatlan módon. Ez azt is jelenti, hogy a TCP-nek nem kell aggódnia az adatok rossz sorrendben érkezése miatt?

14. Egy folyamat az 1-es hoszton p portot használ, egy másik a 2-es hoszton q portot. Lehetséges az, hogy egyidejűleg két vagy több TCP összeköttetés van a két port között?

15. Egy TCP szegmens maximális adat mezeje 65 495 bájt. Miért választottak ilyen furcsa számot?

16. Mutassunk két példát arra, hogy hogyan kerülhet a 6.28. ábra véges automatája a *SYN RCVD* állapotba!

17. Mondjunk példát a Nagle-féle algoritmus egy lehetséges hátrányára, amikor egy erős torlódástól szenvedő hálózatban ezt az algoritmust használjuk.

18. Tekintsük a lassú kezdet protokoll hatását egy 10 ms körülfordulási idővel rendelkező torlódásmentes vonalon. A vevő ablaka 24 kilobájt, a maximális szegmensméret 2 kilobájt. Mennyi idő telik el, amíg az első tele ablak elküldhető?

19. Tegyük fel, hogy a TCP torlódási ablak 18 kilobájtra van állítva és időtúllépés következik be. Mekkora lesz az ablak, ha a következő négy löket mind sikeresen átjut? A maximális szegmensméretet vegyük 1 kilobájtnak.

20. Ha a TCP *RTT* körülfordulási ideje jelenleg 30 ms, és a következő nyugták rendre 26, 32 és 24 ezredmásodperc elteltével érkeznek, mi az új becsült RTT érték? α-t vegyük 0,9-nek!

21. Egy TCP-t futtató gép 65 535 bájtos ablakokat küld egy 1 Gb/s sebességű csatornán, melynek egyik irányban mért késleltetése 10 ms. Legfeljebb mekkora átbocsátóképesség érhető el? Mekkora a vonal hatékonysága?

22. Mekkora a maximális adatsebesség összeköttetésenként abban a hálózatban, melyben a maximális TPDU méret 128 bájt, a maximális TPDU élettartam 30 másodperc és 8 bites sorszámokat használnak?

23. Miért van az UDP? Nem volna elegendő a felhasználói folyamatokra hagyni a nyers IP csomagok küldését?

24. N felhasználó dolgozik egy épületben, mindegyikük ugyanazt a távoli gépet használja ATM hálózaton keresztül. Az átlagos felhasználó óránként L sornyi forgalmat (kimenet + bemenet) generál (az ATM fejrész nélkül) átlagosan P bájt hosszú sorokkal. A szolgáltató C centet számol fel minden továbbított felhasználói bájtért, és óránként további X centet minden nyitva tartott ATM virtuális áramkörért. Milyen feltételek mellett éri meg mind az N szállítási összeköttetést egyazon ATM virtuális áramkörre nyalábolni, ha a nyalábolás a csomagok hosszát 2 bájttal megnöveli? Feltételezhetjük, hogy egyetlen ATM virtuális áramkör az összes felhasználó számára elegendő sávszélességgel rendelkezik.

25. Képes az AAL 1 40 bájtnál rövidebb üzeneteket kezelni a *Pointer* mezőt használó módszer segítségével? Magyarázzuk meg válaszunkat!

26. Próbáljuk meg kitalálni, hogy mekkora mezőkkel rendelkezett az AAL 2, mielőtt a méreteket kitörölték volna a szabványból!

27. Az AAL 3/4 lehetővé teszi több viszony nyalábolását egyetlen virtuális áramkörre. Adjunk példát olyan helyzetre, amikor ennek nincs haszna! Feltételezhetjük, hogy egy virtuális áramkör az egész forgalomnak elegendő sávszélességgel rendelkezik. *Segítség:* gondoljunk a virtuális utakra!

28. Mekkora lehet az egyetlen AAL 3/4 cellában elférő maximális hosszúságú üzenet rakománya?

29. Milyen hatékonysággal lehet egy 1024 bájtos üzenetet AAL 3/4 felhasználásával továbbítani? Másképpen fogalmazva az átvitt bitek hányadrésze hasznos adatbit? Oldjuk meg a feladatot AAL 5 esetére is!

30. Egy ATM készülék egycellás üzeneteket küld 600 Mb/s sebességgel. 100 cellából egyet teljesen tönkretesz a véletlen zaj. Hetente hány detektálatlan hiba várható, ha 32 bites AAL 5 ellenőrző összeget használunk?

31. Egy kliens 128 bájtos kérést küld egy tőle 100 km távolságra levő szervernek egy 1 gigabites üvegszálon. Mi a vonal hatékonysága a távoli eljáráshívás alatt?

32. Tekintsük ismét az előző feladatban vázolt helyzetet! Számítsuk ki az elképzelhető legkisebb válaszidőt az adott 1 Gb/s sebességű vonalra és egy 1Mb/s sebességűre is! Milyen következtetést vonhatunk le?

33. Tegyük fel, hogy egy TPDU vételéhez szükséges időt mérjük. Amikor megszakítás történik, leolvassuk az ezredmásodperces felbontású rendszerórát. Amikor a TPDU-t teljesen feldolgoztuk, ismét leolvassuk az óra állását. 270 000 alkalommal kapunk 0 ms eredményt és 730 000 alkalommal 1 ms-et. Mennyi ideig tart egy TPDU vétele?

34. Egy processzor 100 MIPS sebességgel hajtja végre az utasításokat. Egyszerre 64 bit adatot tud átmásolni, és minden szó másolása hat utasításba kerül. Ha egy beérkező csomagot kétszer kell átmásolni, képes ez a rendszer 1 Gb/s sebességű vonal kezelésére? Az egyszerűség kedvéért tegyük fel, hogy minden utasítás, beleértve a memóriaírásokat és -olvasásokat, a teljes 100 MIPS-es sebességgel hajtható végre.

35. Hogy elkerüljük azt a problémát, hogy a sorszámok körbefordulásakor még élnek régi csomagok, 64 bites sorszámokat használhatunk. Elméletileg azonban egy üvegszál 75 Tb/s sebességre képes. Mekkora legyen a maximális csomagélettartam ahhoz, hogy a jövő 75 Tb/s sebességű hálózataiban se fordulhasson elő körbefordulási probléma még a 64 bites sorszámok használata esetén sem? Tegyük fel, hogy a TCP-hez hasonlóan minden bájtnak saját sorszáma van.

36. A fejezetben kiszámítottuk, hogy egy gigabites vonal 30 000 csomagot zúdít másodpercenként a hosztra, aminek csomagonként csak 1500 utasítása marad, ha a processzoridő felét alkalmazások részére tartja fenn. A számítás során 4 kilobájtos csomagméretet feltételeztünk. Ismételjük meg a számítást ARPANET méretű (128 bájtos) csomagokra!

37. Egy 4000 km hosszú 1 Gb/s sebességű hálózaton nem a sávszélesség, hanem a késleltetés a korlátozó tényező. Tekintsünk egy MAN-t átlagosan 20 km-es forrás–cél távolsággal. Milyen adatsebességnél lesz a fény sebességéből eredő körülfordulási késleltetés az 1 kilobájtos csomag elküldési idejével egyenlő?

38. Módosítsuk a 6.20. ábra programját úgy, hogy hiba esetén képes legyen hibaelhárításra! Vezessünk be egy új *reset* csomagtípust, ami akkor érkezhet, ha egy összeköttetést mindkét fél felépített, viszont egyik sem bontott le! Ez az esemény egyszerre történik az összeköttetés mindkét végén, és azt jelenti, hogy az összes elküldött csomag vagy célba ért, vagy megsemmisült, de egyik esetben sincs már az alhálózatban.

39. Írjunk programot, amely a 6.20. ábra hitel alapú rendszere helyett csúszóablakos protokollal megvalósított forgalomszabályozással rendelkező szállítási entitás pufferkezelését szimulálja! Tegyük lehetővé, hogy magasabb rétegbeli folyamatok véletlenszerűen létesíthessenek összeköttetést, küldhessenek adatokat és bonthassák az összeköttetést! Hogy a program egyszerű maradjon, folyjon az összes adat *A* gépről *B* gépre, visszafelé ne legyen forgalom. Kísérletezzünk a *B* gépnél különböző pufferfoglalási algoritmusokkal, mint például az egyes összeköttetésekhez külön pufferek rendelése, közös pufferterület alkalmazása, és mérjük meg az egyes esetekben elért teljes átbocsátóképességet!

7. Az alkalmazási réteg

A bevezető jellegű részek után elérkeztünk az alkalmazási réteghez, ahol a számunkra érdekes felhasználói programok működnek. Az alkalmazási réteg alatt található egyéb rétegek a megbízható szállítási szolgálatot biztosítják, de közvetlenül a felhasználó számára nem végeznek munkát. Ebben a fejezetben valós alkalmazásokkal fogunk megismerkedni.

Még az alkalmazási rétegben is szükség van azonban olyan kiegészítő protokollokra, melyek a felhasználói programok működését biztosítják. Így az alkalmazások tárgyalása előtt ezek közül hármat nézünk meg részletesebben. Az első terület a biztonság lesz, mely nem egy konkrét protokoll, hanem számos fogalom és módszer összessége, melyek segítségével szükség esetén garantálható az adatok titkossága. A második a DNS lesz, mely a nevek használatára ad lehetőséget az Interneten. A harmadik kiegészítő protokoll a hálózatmenedzsmentet célozza meg. Ezek után négy gyakorlati alkalmazást vizsgálunk meg: az elektronikus levelezést, a USENET-et (hálózati hírcsoportokat), a World Wide Web-et és végül a multimédiát.

7.1. Hálózati biztonság

Létezésének első pár évtizedében a számítógép-hálózatokat elsődlegesen egyetemi kutatók elektronikus levelek küldésére, illetve hivatali alkalmazottak nyomtatók megosztására használták. Ilyen feltételek mellett a biztonság kérdésének nem sok figyelmet szenteltek. Napjainkban azonban, amikor hétköznapi emberek milliói használják a hálózatokat banki műveletek közben, vásárláshoz és adóbevallásuk elkészítéséhez, a hálózati biztonság kérdése komoly problémaként dereng fel a láthatáron. A most következő részekben a hálózati biztonságot több nézőpontból fogjuk megvizsgálni, megmutatva számos buktatóját, és ismertetve azokat az algoritmusokat és protokollokat, melyek alkalmazásával a hálózatok biztonságosabbá tehetők.

A biztonság egy igen tág fogalom, és a támadási módok széles skálája elleni védekezést tartalmazza. Legegyszerűbb formája, amikor biztosítani szeretnénk, hogy gonosz emberek ne olvashassák el, és ami még rosszabb, ne módosíthassák különböző címzettekhez küldött üzeneteinket. Tartalmazza azokat az eseteket, amikor felhasználók olyan távoli szolgáltatásokat szeretnének elérni, melyekhez nincs joguk. Foglalkozik azzal a kérdéssel, hogy hogyan döntsük el a következő, látszólag az adóhatóságtól érkezett üzenetről, hogy valóban az adóhatóságtól és nem a maffiától jött: „Fizess péntekig, különben...". A biztonság azokra a problémákra is választ keres, hogy hogyan lehet azonosítani az elfogott és újra lejátszott üzeneteket, valamint azokat a személyeket, akik tagadják, hogy bizonyos üzeneteket elküldtek.

A legtöbb biztonsági problémát rosszindulatú emberek szándékosan okozzák, hogy előnyhöz jussanak vagy másoknak kárt okozzanak. A leggyakoribb elkövetők közül párat mutat be a 7.1. ábra. Az ábra alapján világos kell hogy legyen: egy hálózat biztonságossá tételéhez sokkal több kell, mint pusztán a programhibák kijavítása. Gyakran intelligens, specializálódott, megfelelő alapokkal rendelkező támadók eszén kell túljárni. Az is nyilvánvaló, hogy az egyszerű alkalmi támadókat megfékező intézkedések nem jelentenek számottevő akadályt a komoly betolakodóknak.

A hálózati biztonsággal kapcsolatos problémák nagyjából négy egymást átfedő területre oszthatók: titkosság, hitelesség, letagadhatatlanság, sértetlenség. A titkosság feladata az információ védelme az illetéktelen felhasználóktól. Gyakran ez az, ami az emberek eszébe jut a hálózati biztonságról. A hitelesség abban segít, hogy meggyőződjünk arról, kivel állunk kapcsolatban, mielőtt érzékeny adatokat fednénk fel vagy üzleti megállapodást kötnénk. A letagadhatatlanság aláírásokkal foglalkozik: hogyan bizonyítható, hogy ügyfeled elektronikus úton 10 millió darab bal kezes boxkesztyűt rendelt 89 centért, amikor később azt állítja, hogy az ár 69 cent volt? Végül, hogyan bizonyosodhatunk meg arról, hogy a kapott üzenet valóban az, amelyiket elküldték, nem pedig egy a fondorlatos ellenség által továbbítás alatt módosított vagy kitalált küldemény.

Támadó	Cél
Diák	Örömét leli mások elektronikus levelének elolvasásában
Hacker	Különböző biztonsági rendszereket tesz próbára, adatokat tulajdonít el
Kereskedelmi képviselő	Elhiteti, hogy egész Európát és nemcsak Andorrát képviseli
Üzletember	A konkurencia üzleti stratégiáját szeretné felderíteni
Elbocsátott alkalmazott	Bosszút áll, amiért kirúgták
Könyvelő	A vállalat pénzét sikkasztja el
Tőzsdei bróker	Egy vevőnek elektronikus levélben tett ígéretét szeretné letagadni
Szélhámos	Bankkártyaszámokat szerez meg, hogy azokat eladja
Kém	Az ellenség katonai erejét szeretné megismerni
Terrorista	Biológiai fegyverekkel kapcsolatos titkokat próbál megszerezni

7.1. ábra. *Néhány embertípus, akik biztonsági problémákat okoznak, és motivációik*

Mindezek a kérdések (a titkosság, a hitelesség, a letagadhatatlanság és a sértetlenség biztosítása) a hagyományos rendszerekben is jelentkeznek, azonban némi eltéréssel. A titkosságot és a sértetlenséget ajánlott levéllel és az iratok lezárásával oldják meg. Egy postavonat kirablása ma már jóval nehezebb, mint Jesse James idején volt.

Ugyanígy, az emberek többnyire meg tudják különböztetni az eredeti dokumentumot a fénymásolattól, ami gyakran lényeges. Próbaképpen készíts másolatot egy eredeti csekkről. Próbáld meg beváltani a csekket hétfőn bankodban. Aztán kedden próbálkozz a fénymásolattal. Vizsgáld meg a bank viselkedésében beállt változást. Az elektronikus csekkeknél az eredeti és a másolat megkülönböztethetetlenek. Időbe telhet, míg a bankok használni kezdik ezeket.

Másokat az arcuk, hanguk és kézírásuk alapján azonosítunk. Aláírásunk bizonyítéka lehet a levélpapírra tett kézjegyünk, domború bélyegző vagy más egyéb. A hamisítványt többnyire leleplezik a kézírás-, papír- vagy festékszakértők. Ezen lehetőségek közül egyik sem áll rendelkezésünkre az elektronikus világban. Nyilvánvalóan más megoldásokat kell találni.

Mielőtt maguk a módszerek tárgyalását megkezdenénk, érdemes egy kis időt szánni arra, hogy meghatározzuk: mindegyik protokoll réteghez hozzátartozik a hálózati biztonság. Könnyen megeshet, hogy nem helyezhető kizárólag egyetlen helyre. Minden réteg hozzájárulhat valamivel. A fizikai rétegben a vezeték megcsapolását meghiúsíthatjuk a vezetéknek lepecsételt és argon gázzal feltöltött csőbe helyezésével. Minden kísérlet, amely során a csőbe szeretnének hatolni a gáz elszökésével és így nyomáscsökkenéssel jár, ami riasztást válthat ki. Néhány katonai rendszerben használják ezt a módszert.

Az adatkapcsolati rétegben két végpont között haladó csomagokat elkódolhatjuk, amikor elhagyja a küldőt és visszakódolhatjuk, amikor a másikra megérkezik. Minden lépést elvégezhetünk az adatkapcsolati rétegben anélkül, hogy a felsőbb rétegek tudnának róla. Ennek a megoldásnak akkor jelentkeznek a korlátai, amikor a csomagoknak több routeren kell keresztül jutniuk, mivel ekkor mindegyik eszközön vissza kell kódolni a csomagot sebezhetővé téve a routeren belüli támadásokkal szemben. Emellett nem teszi lehetővé, hogy csak egyes kapcsolatokat védjünk (pl. on-line vásárlás bankkártyával), másokat pedig nem. Mindezek ellenére az **adatkapcsolati titkosítás (link encryption),** ahogy a fenti módszert nevezik, könnyen használható a meglevő hálózatokon, és gyakran hatásos.

A hálózati rétegben tűzfalakat telepíthetünk, hogy egyes csomagokat a hálózatunkon belül vagy azon kívül tartsunk. Ezekkel az 5. fejezetben foglalkoztunk. A szállítási rétegben teljes kapcsolatokat titkosíthatunk, végponttól végpontig, vagyis alkalmazási folyamattól alkalmazási folyamatig. Bár ezek a módszerek a biztonság számos területén segítséget nyújtanak, és sokan dolgoznak ezek tökéletesítésén, egyik sem oldja meg a hitelesség és a letagadhatatlanság kérdését hatékonyan és általánosnak tekinthető módon. Ezeket a problémákat az alkalmazási rétegben tudjuk valóban megoldani, ezért tanulmányozzuk őket ebben a fejezetben.

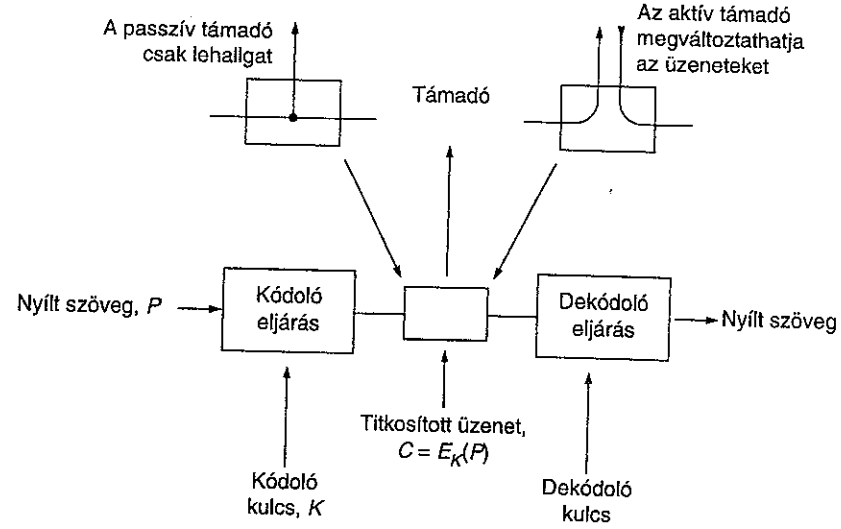
7.1.1. Hagyományos kriptográfia

A kriptográfia hosszú és színes múltra tekint vissza. Ebben a részben csak vázlatosan tekintjük át legfontosabb mozzanatait, a következő részek hátteréül. A teljes történeti áttekintéshez Kahn (1967) könyve még mindig ajánlott olvasmány. A korszerű módszerek átfogó és részletes tárgyalása megtalálható a következő szerzők műveiben (Kaufman és mások, 1995; Schneier, 1996; Stinson, 1995).

Eredetileg négy csoport alkalmazta és vitte tovább a kriptográfia mesterségét: a hadsereg, diplomáciai testületek, naplóírók és a szerelmesek. Ezek közül a hadseregnek volt a legfontosabb szerepe, és ez alakította ki leginkább ezt a területet. A katonai szervezetekben a titkosítandó üzeneteket régebben alacsonyan fizetett hivatalnokok kezébe adták, hogy azokat kódolják és küldjék tovább. Csupán az üzenetek óriási mennyisége is megakadályozta, hogy ezt a munkát pár elit specialista végezze el.

A számítógépek eljöveteléig a kriptográfia egyik legnagyobb korlátját a kódolást végző tisztségviselők azon képessége jelentette, hogy hogyan képesek egyszerű eszközökkel, gyakran háborús körülmények között elvégezni a szükséges transzformációkat. További nehézséget okozott az egyik titkosító módszerről egy másikra való gyors áttérés, mivel ez sok ember átképzését vonta maga után. Mindezek ellenére a veszély, amit egy titkosító ügynök elfogása jelentett, szükségessé tette, hogy ilyen esetekben azonnal megtörténhessen a módszerbeni váltás. Ezek az egymásnak ellentmondó követelmények hozták létre az 7.2. ábrán látható modellt.

A kódolandó üzeneteket, melyeket nyílt **szövegnek (plaintext)** hívunk, egy olyan függvénnyel transzformálunk, melynek paramétere egy **kulcs (key).** A titkosító eljárás



7.2. ábra. *A titkosítási modell*

kimenetét, melyet **titkosított szövegnek (ciphertext)** hívunk, továbbítjuk gyakran rádió vagy futár segítségével. Feltételezzük, hogy az ellenség vagy a támadó hallja és pontosan lemásolhatja a teljes titkosított szöveget. Ennek ellenére, az eredeti címzettel ellentétben, a **támadó (intruder)** nem ismeri a visszakódoláshoz szükséges kulcsot, és így nem képes az üzenet egyszerű visszaalakítására. Néha a támadó nemcsak hallhatja a kommunikációs csatornát (passzív támadó), hanem a felvett üzeneteket később visszajátszhatja, saját üzeneteit indíthatja útnak, vagy egyébként valódi üzeneteket változtathat meg, mielőtt azok a címzetthez megérkeznének (aktív támadó). A titkosírás megfejtésének mesterségét **kriptoanalízisnek (cryptoanalysis)** hívjuk. A titkosító eljárások kifejlesztésének tudománya (kriptográfia) és azok feltörése (kriptoanalízis) együttesen a **kriptológia (cryptology)** témakörét alkotják.

Gyakran hasznos lesz a továbbiakban, ha egységes jelölést vezetünk be a nyílt és titkosított üzenet, valamint a kulcs jelölésére. A $C=E_K(P)$ kifejezés azt fogja jelenteni, hogy a P nyílt szöveget a K kulcsot használva kódoljuk, és így a C titkosított üzenetet kapjuk. Hasonlóan a $P=D_K(C)$ a C üzenet nyílt szöveggé történő visszakódolását fogja reprezentálni. Ezek után nyilvánvaló, hogy:

$$D_K(E_K(P)) = P$$

A fenti jelölés azt sejteti, hogy az E, valamint a D műveletek tulajdonképpen matematikai függvények, ami így is van. Az egyedüli trükk, hogy mindketten két paraméteres függvények, és az egyik paramétert (a kulcsot) alsó indexként tüntettük fel normális paraméter helyett, hogy élesen megkülönböztessük az üzenettől.

A kriptográfia alaptörvénye szerint feltételezzük a kriptoanalitikusról, hogy ismeri a kódoláshoz használt módszer algoritmusát. Más szavakkal, a kódfejtő pontosan tudja, hogy a titkosító eljárás (E a 7.2. ábrán) hogyan működik. Egy új módszer kifejlesztéséhez, teszteléséhez és telepítéséhez szükséges erőfeszítések, melyeket akkor kell tennünk, ha módszerünket kitalálták, vagy úgy gondoljuk, hogy kitalálták, minden esetben célszerűtlenné tette, hogy a módszert titokban tartsuk, és feltételezzük azt, hogy valóban titok.

Itt vesszük hasznát a kulcsoknak. A kulcs egy (aránylag) rövid karaktersorozatból áll, mely egyet választ ki a számos potenciális kódolás közül. Az általános eljárással ellentétben, melyet csak nagyjából éves gyakorisággal változtathatunk, a kulcsot olyan gyakran cserélhetjünk, amilyen gyakran szükséges. Így az alapmodellünk egy stabil és mindenki által ismert általános eljárásból áll, melyet egy titkos és könnyen változtatható kulccsal paraméterezünk.

Az algoritmus nyilvánosságát nem győzzük hangoztatni. Az eljárás publikálása lehetővé teszi, hogy a kriptográfus több elméleti szakemberrel ismertesse módszerét, akik aztán megpróbálják feltörni azt, hogy publikációkat írhassanak ravaszságuk demonstrálására. Ha azonban számos szakértőnek sem sikerül 5 év próbálkozás után sem az algoritmus feltörése, az már egészen megbízhatónak tekinthető.

Az igazi titkosság a kulcsban rejlik, és így annak hossza kulcsfontosságú tervezésbeli kérdés. Vegyünk egy egyszerű kombinációs zárat. Az általános alapelv az, hogy sorrendben számjegyeket adunk meg. Ezt mindenki tudja, de a kulcs titok. Két számjegy hosszúságú kulcs 100 lehetőséget rejt magában. Egy három számjegy hosszúságú

esetén már 1000 lehetőség adódik, és hat számjegy alkalmazásakor elérjük az egy milliót. Minél hosszabb a kulcs, annál nagyobb **munkaténvezővel (work factor)** kell számolnia a kódfejtőnek. Egy rendszer feltörésének munkaigénye a kulcstér elemeinek végigpróbálásával a kulcs hosszával exponenciálisan növekszik. A titkosság alapja egy erős (de nyilvános) algoritmus és egy hosszú kulcs. Ha meg szeretnéd akadályozni, hogy gyermeked elolvassa e-leveleidet, egy 64 bites kulcs megteszi. A kormányzati hivatalok sakkban tartásához azonban legalább 256 bites kulcs szükséges.

A kódfejtő nézőpontjából a kriptoanalízis problémája három területre osztható. Amikor számos titkos szöveggel, de egyetlen nyílt szöveggel sem rendelkezik, a **csak titkosított szöveg alapú (chipertext only)** problémával áll szemben. Az újságok rejtvény oldalain megjelenő kriptogrammák ebbe a problémakörbe tartoznak. Amikor néhány nyílt szöveggel és azok titkosított párjaival rendelkezik a támadó, az **ismert nyílt szöveg alapú (known plaintext)** problémát kell leküzdenie. Végül, amikor a kódtörőnek lehetősége van saját maga által választott szövegrészletek titkosított párjainak előállítására, a **választott nyílt szöveg típusú (chosen plaintext)** támadással találkozunk. Az újságok kriptogrammái triviális módon megfejthetőek lennének, ha a megengednénk az olyan típusú kérdéseket, hogy „Mi az ABCDE titkosított párja?".

A kriptográfia területén kezdőknek számító emberek gyakran azt gondolják, hogy egy kód biztonságos, ha ellenáll a csak titkosított szöveg típusú támadásoknak. Ez a feltételezés nagyon naiv. Sok esetben a kódfejtő jó becslést tud adni a nyílt szöveg bizonyos részeire. Például az időosztásos rendszerek első üzenete, amit kapcsolatfelvételkor küld, valahogy így néz ki: „PLEASE LOGIN.". Néhány nyílt szöveg – kódolt szöveg párral felvértezve a kódtörő munkája jelentősen leegyszerűsödik. A biztonság eléréséhez a kriptográfusnak óvatosnak kell lennie, és biztosítania kell, hogy a rendszer abban az esetben is feltörhetetlen, ha az ellenfél tetszőleges számú és típusú szöveget kódolhat a rendszer kulcsával.

A titkosítási módszereket történelmileg két csoportba soroljuk: a helyettesítő, illetve a keverő típusú eljárások közé. Most ezeket tekintjük át dióhéjban a modern kriptográfiai módszerek alapjaként.

Helyettesítő kódolók

Egy **helyettesítő kódolóban (substitution cipher)** minden betű vagy betűcsoport egy másik betűvel vagy betűcsoporttal helyettesítődik a titkosság elérése érdekében. Az egyik legrégebbi ismert módszer a **Caesar-titkosító (Caesar cipher),** mely nevét Julius Caesarról kapta. A módszert alkalmazva az *a* betűből *D* lesz, a *b*-ből *E*, a *c F*-fé alakul ... és a *z C*-vé válik. Például az *attack* szó titkosított párja a *DWWDFN* szó lesz. Példáinkban a nyílt üzeneteket kisbetűvel, míg a kódolt változatokat csupa nagybetűvel fogjuk megadni.

Egy kissé általánosabb változata a Caesar-kódolónak megengedi, hogy a titkosított szöveg ábécéje k karakterrel legyen eltolva a korábbi fix 3 helyett. Ebben az esetben a k szám a ciklikus eltolást alkalmazó általános módszer mellett kulcsként viselkedik. A Caesar-kódoló talán becsaphatta a karthágóiakat, de azóta már senkit nem vezet félre.

Egy kicsit fejlettebb módszer, amikor a nyílt szöveg minden szimbólumához – le-

gyen ez most az egyszerűség kedvéért pusztán 26 karakter – egy másik karaktert rendelünk. Vegyük a következő hozzárendelést:

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

A fenti általános módszert **egybetű-helyettesítéses titkosításnak (monoalphabetic substitution)** nevezzük, ahol a kulcs az a 26 karakterből álló karaktersorozat, mely a nyílt szöveg teljes ábécéjének felel meg. A fenti kulcsot alkalmazva az *attack* nyílt üzenetet a kódoló a *QZZQEA* titkosított üzenetté transzformálja.

Első látásra ez a rendszer biztonságosnak tűnik, mivel a kódtörő – bár ismeri az általános módszert (betűhelyettesítés) – nem ismeri, hogy a szóba jöhető 26! $\cong 4 \times 10^{26}$ lehetséges kulcs közül melyiket használtuk. Ellentétben a Caesar-kódolóval, minden egyes kulcs kipróbálása nem ígéretes megközelítés. Még abban az esetben is, ha egy kombináció kipróbálása 1 μs időt vesz igénybe, egy számítógépnek 10^{13} évig tartana minden kulcs végigpróbálása.

Ennek ellenére meglepően kevés titkosított üzenet alapján a kód könnyűszerrel megfejthető. A támadás alapvetően a természetes nyelvek statisztikai tulajdonságait használja ki. Az angolban például az *e* a leggyakoribb karakter, sorrendben őt követik a *t,o,a,n,i* stb. A leggyakoribb betűpárok vagy más néven **betűkettősök (digrams)** a *th, in, er, re* és az *an*. A leggyakrabban előforduló három egymás mellett álló betű: **betűhármasok (trigrams)** a *the, ing, and* és az *ion*.

Az a kódfejtő, aki egybetű-helyettesítéses titkosító törésébe fog, a kódolt üzenetben szereplő betűk relatív gyakoriságának meghatározásával kezdi a munkáját. Ezek után próbálkozhat a leggyakoribb karakter *e*-vel való helyettesítésével, és a második leggyakoribbhoz a *t*-t rendelni. Ennek alapján megvizsgálhatja a betűhármasokat a *tXe* alakú mintákat keresve, mivel itt erős a gyanú, hogy az *X* helyén a *h* betű áll. Ehhez hasonlóan, ha a *thYt* minta gyakran felbukkan, az *Y* helyére célszerű az *a* betűt próbálni. Az így nyert információval nekiláthat az *aZW* betűhármasok felkutatásának, ami valószínűleg az *and* szót fogja lefedni. Gyakori karakterek, betűkettősök és betűhármasok megsejtésével és magánhangzó-mássalhangzó minták lehetséges kombinációinak ismeretével a kódtörő némi kísérletezéssel felépítheti a nyílt szöveget, betűről betűre.

Egy másik megközelítés szerint teljes szavakat, illetve kifejezéseket érdemes keresni. Példaképpen vizsgáljuk meg a következő titkosított üzenetet, amit egy könyvelőség küldött (ötös karaktercsoportokba szedve):

CTBMN BYCTC BTJDS QXBNS GSTJC BTSWX CTQTZ CQVUJ
QJSGS TJQZZ MNQJS VLNSX VSZJU JDSTS JQUUS JUBXJ
DSKSU JSNTK BGAQJ ZBGYQ TLCTZ BNYBN QJSW

Egy ilyen jellegű cégtől érkező levélben egy gyakori szó lehet a financial. Felhasználva e szó azon tulajdonságát, hogy benne az *i* betű kétszer szerepel, közrefogva négy másikat, a titkos üzenetben ilyen távolságban levő karakterismétlődéseket fogunk keresni. A fenti szövegben 12 találatunk van ennek alapján: a 6., 15., 27., 31., 42., 48., 56., 66., 70., 71., 76. és 82. pozícióban. Ezek közül azonban csak kettőnél – a 31-nél és a 42-nél ismétlődik a következő karakter (ami az *n*-nek felelne meg) megfelelően. E kettő közül is pedig csak a 31. pozícióban levőnél megfelelő a hipotetikus *a* karakterek ismétlődése. Ennek alapján erős a gyanúnk, hogy a 30. helyen a *financial* szó kezdődik. Ezt kiindulási alapként elfogadva, jóval könnyebb dolgunk van a kulcs megfejtésével, amihez használhatjuk a betűk előfordulásának relatív gyakoriságát, illetve az angol nyelv ezzel kapcsolatos statisztikai jellemzőit.

Keverő kódolók

A helyettesítő kódolók megtartották a nyílt szöveg karaktereinek sorrendjét, csak azokat más alakkal ruházták fel. Ezzel ellentétben a **keverő kódolók (transposition ciphers)** nem keresnek másik betűalakot, viszont az eredeti sorrendet átalakítják. A 7.3. ábra egy egyszerű keverő kódolót mutat be, az oszlop alapú keverőt. A titkosító kulcsként egy olyan szót vagy kifejezést használ, mely nem tartalmaz ismétlődő betűket. Példánk kulcsa a MEGABUCK lesz. A kulcs szerepe az oszlopok megszámozása lesz oly módon, hogy az első oszlopot az a kulcskarakter fogja kijelölni, amelyik az ábécében legelőször szerepel, a többi oszlop sorrendje is hasonlóképpen alakul. A nyílt üzenetet vízszintes sorokba írjuk, a titkosított üzenetet pedig függőlegesen olvassuk ki az oszlopok előbb megállapított sorrendjének megfelelően.

M	E	G	A	B	U	C	K	
7	4	5	1	2	8	3	6	
p	l	e	a	s	e	t	r	Nyílt üzenet
a	n	s	f	e	r	o	n	pleasetransferonemilliondollarsto
e	m	i	l	l	i	o	n	myswissbankaccountsixtwotwo
d	o	l	l	a	r	s	t	Titkosított üzenet
o	m	y	s	w	i	s	s	AFLLSKSOSELAWAIATOOSSCTCLNMOMANT
b	a	n	k	a	c	c	o	ESILYNTWRNNTSOWDPAEDOBUOERIRICXB
u	n	t	s	i	x	t	w	
o	t	w	o	a	b	c	d	

7.3. ábra. *A keverő típusú titkosító*

Egy keverő kódoló feltöréséhez a kódfejtőnek először rá kell jönnie, hogy egy ilyen típusú kóddal áll szemben. Az *E, T, A, O, I* karakterek gyakoriságát könnyen összevetheti a nem kódolt üzenetekben levő gyakoriságukkal. Ha ez nagy hasonlóságokat mutat, akkor egy keverő kódolóval van dolga, mivel itt a karakterek formáját nem változtattuk.

A következő lépés az oszlopok számának meghatározása. Sok esetben egy – az üzenetben szereplő – szó vagy kifejezés kitalálható az üzenet környezetéből. Például elképzelhető, hogy a kódtörő gyanítja: a nyílt üzenetben a *milliondollars* kifejezés

szerepel valahol. Az *MO, IL, LL, LA, IR* és *OS* betűkettősök előfordulásait keresve a titkos üzenetben ennek a kifejezésnek az eldeformált darabjait találhatja meg. A kódolt üzenetben szereplő *O* betű az *M*-et követi (fenti példánkban egymás alatt szerepelnek a 4. oszlopban), mivel a vizsgált kifejezésben a feltételezett kulcstávolságra vannak egymástól. Ha hét-hosszúságú kulcsot használtunk volna, akkor az *MD, IO, LL, LL, IA, OR* és *NS* betűkettősök után kellene keresnünk. Valójában minden egyes kulcshosszakhoz különböző betűkettősök tartoznak a titkosított üzenetben. A különböző lehetőségeket megvizsgálva a kódfejtő gyakran könnyen meghatározhatja a kulcshosszt.

A hátralevő feladat az oszlopok sorrendjének meghatározása. Ha az oszlopok száma (k) kicsi, az összes $k(k-1)$ db oszloppár megvizsgálható a betűkettősök előfordulási gyakorisága szempontjából. Annak a párnak a sorrendjét fogadjuk el, mely legjobban kielégíti az angol nyelv törvényszerűségeit. A megtalált pár után következő oszlopot további próbákkal határozhatjuk meg, azt a jelöltet választva, mely a legjobban kielégíti a betűkettősök és betűhármasok szabályszerűségeit. A megelőző oszlopot is hasonlóképpen határozzuk meg. Az eljárást a végleges oszlopsorrend meghatározásáig folytatjuk. Igen valószínű, hogy ezek után az eredeti üzenet felismerhetővé válik (pl. ha a *milloin* szót látjuk, világos, hogy hol a hiba).

Néhány keverő kódoló fix hosszúságú bemenetből ugyancsak kötött hosszúságú kimeneti blokkot készít. Ezeket a kódolókat egyszerűen leírhatjuk azzal a sorrenddel, ahogy a bemeneti karakterek a kimeneten megjelennek. A 7.3. ábrán bemutatott kódoló például egy 64 karakteres blokk-kódoló. Ennek kimenete: 4, 12, 20, 28, 36, 44, 52, 60, 5, 13, ..., 62. Más szavakkal: a negyedik bemeneti karakter (a) lesz a kimenet első karaktere, amit a bemenet 12. szimbóluma követ (f), és így tovább.

Egyszer használatos bitminta

Feltörhetetlen kódolót készíteni egyébként kifejezetten egyszerű; a módszer évtizedek óta ismert. Válasszunk kulcsnak egy véletlen bitsorozatot. Ezek után a kódolandó üzenetet szintén alakítsuk bitsorozattá, mondjuk a karakterek ASCII kódját felhasználva. Végül számoljuk ki a két sorozat KIZÁRÓ VAGY művelettel adott eredményét bitről bitre. Az így kapott üzenet feltörhetetlen, mivel bármilyen adott hosszúságú szöveg ugyanolyan eséllyel jelölt a nyílt üzenet szerepre. A titkos üzenet semmiféle kiindulási információt nem ad a kódtörőnek. Kellően hosszú üzenetmintában minden egyes karakter előfordulási valószínűsége azonos lesz, akárcsak a betűkettősöké és betűhármasoké.

A fenti módszer, amit **egyszer használatos bitmintának (one-time pad)** hívnak, sajnos számos kedvezőtlen tulajdonsággal rendelkezik. Először is, a kulcsminta nehezen megjegyezhető, így mind a küldő mind a fogadó egy írott másolatot kell hogy magánál tartson. Ha bármelyik példányt valaki megszerzi, a kulcsok használhatatlanná válnak. Másodszor, az elküldhető üzenet hosszát korlátozza a rendelkezésre álló kulcs hossza. Ha egy titkosügynök megüti a főnyereményt és nagy mennyiségű adat birtokába jut, könnyen találhatja magát abban a helyzetben, hogy képtelen azt a központ felé továbbítani, mivel elfogyott a rendelkezésére álló kulcs. Egy következő probléma

ezzel a módszerrel, hogy rendkívül érzékeny az elveszett vagy beékelődött karakterekre. Ha egy ponton a küldő és a fogadó kiesnek a szinkronból, onnantól kezdve minden üzenet zagyvaságnak tűnik számukra.

A számítógépek megjelenésével az egyszer használatos bitminták néhány alkalmazási területen potenciális megoldásként jöhetnek szóba. A kulcs forrása lehet egy speciális CD, mely több gigabitnyi információt tárol. Ha ezt a CD-t egy közönséges zenei CD-tokba rejtjük, és még néhány slágert is rögzítünk rajta, senkinek nem lesz gyanús. Természetesen gigabit nagyságrendű hálózatok esetén kissé fárasztó lehet 5 másodpercenként CD-t cserélni. Többek között ezért ismerkedünk most meg a modern titkosító eljárásokkal, melyek lehetővé teszik tetszőleges méretű nyílt szöveg kódolását.

7.1.2. Két alapvető kriptográfiai elv

Bár a következőkben számos különböző kriptográfiai rendszert fogunk megismerni, ezek mégis megegyeznek abban, hogy két fontos elv szolgál alapjukként, amiket fontos, hogy megértsünk. Az első alapelv szerint a titkosított üzeneteknek némi redundanciát kell hordozniuk, vagyis olyan információt, ami nem szükséges az üzenet megértéséhez. Egy példán keresztül beláthatjuk ennek fontosságát. Vegyünk egy olyan céget, amelytől levélben rendelhetünk valamit. Legyen ez a The Couch Potato (TCP) cég mintegy 60 000 termékkel. Képzeljük el, hogy igen hatékonyan működnek, és a TCP programozói úgy döntenek, hogy minden megrendelő levél alakja a következőképpen nézzen ki: 16 bájt hosszú ügyfélazonosító, majd egy 3 bájtos adat mező (1 bájt a mennyiségnek, 2 bájt pedig a termékkód számára). Az utolsó 3 bájtot egy nagyon hosszú kulccsal titkosítják, amit csak a megrendelő és a TCP programozói ismernek.

Első látásra biztonságosnak tűnik a fenti módszer, és bizonyos értelemben az is, mivel passzív támadók nem tudják megfejteni az üzeneteket. Sajnálatos módon egy súlyos hiányossággal rendelkezik, ami használhatatlanná teszi. Tegyük fel, hogy egy frissen elbocsátott alkalmazott bosszút szeretne állni a cégen. Mielőtt távozna a vállalattól, magával viszi az ügyfelek listáját, vagy annak egy részét. A következő éjszaka elkészít egy programot, mely fiktív rendeléseket generál valódi ügyfélazonosítókkal. Mivel nem ismeri a titkos kulcsokat, az utolsó három bájtot véletlenszerűen tölti fel, majd rendelések százait juttatja el a TCP-hez.

Amikor az üzenetek megérkeznek, a TCP számítógépei az ügyfelekhez rendelt kulcsok segítségével visszafejtik az üzenetet. A TCP szerencsétlenségére, mivel majdnem minden 3 bájtos kombináció értelmes, a számítógépek elkezdik a rendelések teljesítését. Bár furcsának tűnhet, hogy a vásárló 137 gyermekhintát és 240 homokozóládát rendelt, a programok csak annyit „gondolhatnak", hogy minden bizonnyal játszótér hálózatot szeretne nyitni az ügyfél. Ily módon egy aktív támadó (az elbocsátott alkalmazott) nagy kárt okozhat, még úgy is, hogy fogalma sincs a programja által generált üzenetek tartalmáról.

A veszély jelentősen csökkenthető, ha az üzenetekbe redundanciát csempészünk. Ha például a rendeléssel kapcsolatos részt 12 bájt hosszú mezőn tároljuk, az első 9 karakter helyére nullákat írva, a fenti támadás már nem jár sikerrel, mivel a volt alkalmazott nem képes valódinak tűnő üzenetek nagyszámú előállítására. A történet mon-

danivalója, hogy minden üzenet számottevő redundanciát kell hogy hordozzon ahhoz, hogy aktív támadók ne áraszthassanak el bennünket valódinak tűni hamis üzenettel.

A redundancia növelésével azonban a kódfejtőnek egyre könnyebb dolga lesz az üzenet feltörésekor. Tegyük fel, hogy a levélen keresztüli rendelés versenyképes üzletággá válik, és a TCP fő riválisa, a The Sofe Tuber, mindent megadna azért, hogy megtudja, hány homokozóládát ad el a TCP. Így aztán megcsapolják a TCP telefonvonalát. Az eredeti 3 bájtos sémát használva a kriptoanalízis reménytelen, mivel egy kulcs megsejtése után a kódtörő nem ellenőrizheti kellőképpen, hogy valóban kulcsot talált-e, mivel minden üzenet gyakorlatilag érvényes. A módosított 12 bájtos módszert alkalmazva azonban a kriptoanalízist végző támadónak könnyű dolga van egy kulcs ellenőrzésekor.

Így a kriptográfia első alapelve azt követeli meg minden üzenettől, hogy elég redundanciát hordozzon ahhoz, ami lehetetlenné teszi az aktív támadó számára, hogy hamis üzenetekkel félrevezesse a címzettet. Mivel azonban a minél nagyobb redundancia a passzív támadóknak kedvez, a két véglet között kell a megoldást keresnünk. Továbbá, a redundaciát nem célszerű az üzenet elején csupa nulla karakter beiktatásával elérni, mivel néhány kriptográfiai algoritmus az ilyen sorozatokból előre megjósolható kimenetet gyárt, ami a kódfejtő munkáját leegyszerűsíti. Angol szavak véletlen sorozata egy sokkal jobb megoldás a redundancia biztosítására.

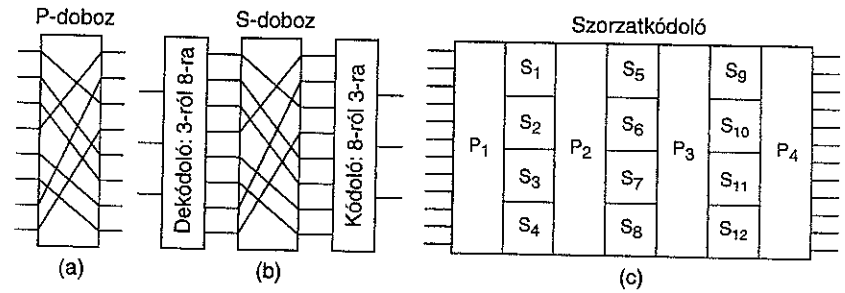
A másik alapelv szerint gondoskodnunk kell arról, hogy egy aktív támadó ne játszhasson vissza régi üzeneteket. Ha ezeket nem tesszük meg, az elbocsátott alkalmazottunk lehallgatva a TCP telefonvonalát folyamatosan visszajátszhatja a korábbi – egyébként valódi – üzeneteket. Egy lehetséges óvintézkedés, hogy minden üzenetet egy időbélyeggel látunk el, ami öt percig érvényes. A fogadó ezek után csak az utolsó öt percben érkezett üzeneteket kell, hogy eltárolja, hogy kiszűrhesse az ismétlődéseket. Az öt percnél régebbi üzenetek eldobhatók, mivel minden olyan beérkező üzenetet, ami öt percnél régebbi bélyeggel van ellátva, visszautasítunk. Az időbélyeg mellett más lehetőségeink is vannak, ezeket a későbbiekben tekintjük át.

7.1.3. Titkos kulcsú algoritmusok

A mai kriptográfia ugyanazokat alapötleteket használja, mint a hagyományos titkosítás: helyettesítést és keverést, de ma már máson van a hangsúly. A tradicionális kódkészítők egyszerű algoritmusokat használtak, és a biztonságot a hosszú kulcsokra bízták. Ma ennek az ellenkezője igaz: a cél olyan bonyolult és szövevényes algoritmusok konstruálása, melyet akkor sem ismerhet ki a kódtörő, ha hatalmas mennyiségű általa választott üzenet kódolt megfelelőjét halmozza is fel.

A helyettesítést és a keverést egyszerű áramkörökkel valósíthatjuk meg. A 7.4.(a) ábra egy **P-doboznak** (a P a permutáció szóból származik) nevezett eszközt mutat, mellyel hatékonyan lehet elvégezni a keverést 8 bites bemeneti adatokon. Ha a nyolc bitet fentről lefelé sorszámmal látjuk el (01234567), akkor ennek a P-doboznak a kimenete a következőképpen alakul: 36071245. A megfelelő belső huzalozással a P-doboz tetszőleges keverést elvégezhet, közel a fény sebességével.

A helyettesítést ún. **S-dobozok** (az S a substitution szóból származik) végzik,



7.4. ábra. *A szorzattitkosítók alapvető elemei. (a) P-doboz. (b) S-doboz. (c) Szorzat*

ahogy azt a 7.4.(b) ábra mutatja. A fenti eszköz 3 bites nyílt üzenetekből ugyancsak 3 bites titkosított szöveget gyárt. A 3 bitnyi bemenet egy vonalat választ ki az első fokozat nyolc kimenetéből, amin 1-es értéket állít be, a többin 0-t. A második lépcsőben egy P-doboz található. A harmadik fokozat a kiválasztott aktív bemeneti vonala alapján újra bináris alakú kódot generál. A bemutatott huzalozással, ha a bemenetre a 01234567 sorozatot adjuk, a kimeneten a 24506713 szekvencia fog megjelenni. Más szavakkal a 0-t 2-vel helyettesíti, az 1-t a 4-gyel stb. Megint igaz, hogy az S-dobozban található P-doboz megfelelő kialakításával bármilyen helyettesítést elvégezhetünk.

Ezeknek az építőköveknek akkor mutatkozik meg az igazi erejük, amikor sorba kapcsoljuk őket, így létrehozva a 7.4.(c) ábrán látható **szorzat típusú titkosítót (product cipher).** Ennél a példánál első lépésben 12 bemeneti vonalat cserélünk fel. Elméletileg a második fokozatban elhelyezhetnénk egy olyan S-dobozt, amely egy 12 bites bemenethez egy 12 bites kimenetet rendelne. Ehhez azonban $2^{12} = 4096$ egymást keresztező vezetékre lenne szükség az eszköz belsejében. Ehelyett a bemenetet 4 db 3 bites csoportra bontjuk, mindegyiket egymástól függetlenül helyettesítünk. Bár ez a megoldás már nem annyira általános, de még mindig hatékony. Kellően nagyszámú fokozatot használva a kódolóban, a kapott kimenet rendkívül bonyolult függvénye lesz a bemenetnek.

DES

1977 januárjában az amerikai kormányzat egy az IBM által kifejlesztett szorzat típusú kódolót fogadott el szabványként a nem bizalmas információk számára. A kódoló, amit **DES (Data Encryption Standard)** névre kereszteltek, az iparban is széles körben elterjedt a biztonsági termékek piacán. Eredeti formájában ma már nem tekinthető biztonságosnak (Wayner, 1995), de némi kiegészítéssel ma is használható. A DES működését tekintjük át a következőkben.

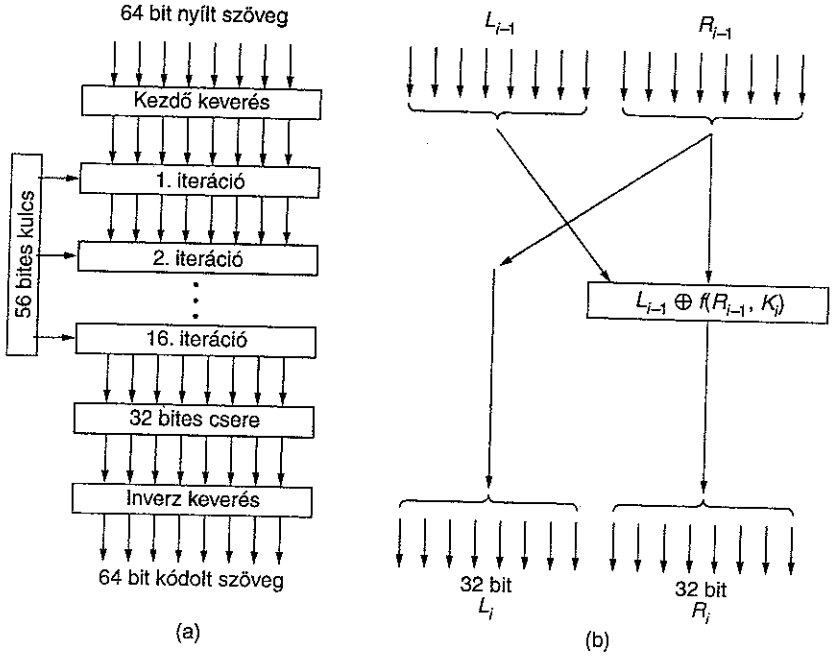
A DES vázlatos működése a 7.5.(a) ábrán követhető nyomon. A nyílt szöveget 64 bites blokkonként kódoljuk, ami során szintén 64 bites titkos üzeneteket kapunk. Az algoritmus, melynek paraméteréül egy 56 bites kulcs szolgál 19 különálló fokozatból épül fel. Az első lépés egy kulcsfüggetlen keverés a 64 bites bemeneten. Az utolsó lépés ennek pontosan az inverz művelete. Az utolsót megelőző lépésben az első 32 bites

részt felcseréljük a hátsó 32 bites résszel. A maradék 16 lépés működése ehhez hasonló, de mindegyik paraméteréül szolgál a kulcs különböző függvényekkel képzett értéke. Az algoritmus lehetővé teszi, hogy a dekódolást ugyanazzal a kulccsal végezhessük, mint a kódolást. Egyedül a lépések sorrendjét kell megfordítanunk.

A közbenső lépések egyikét mutatja részletesebben a 7.5.(b) ábra. Mindegyik ilyen fokozat két 32 bites bemenetből ugyancsak kettő 32 bites kimenetet produkál. A bal oldali kimenet egyszerűen a jobb oldali bemenet másolata. A jobb oldali kimenetet KIZÁRÓ VAGY művelettel kapjuk, amit egyrészt a bal oldali bemenet másrészt a jobb oldali bemenet, valamint a fokozathoz tartozó kulcsérték (K_i) alapján egy adott függvénnyel képzett érték között végzünk el. Az algoritmus szövevényessége ebben az adott függvényben rejlik.

A függvény négy lépésből áll, melyeket egymás után végzünk el. Első lépésben egy 48 bites számot képzünk (E) a 32 bites jobb oldal (R_{i-1}) kiterjesztésével, amit egy rögzített keverés és másolás segítségével kapunk. A második lépésben az E, illetve a K_i értékek között KIZÁRÓ VAGY műveletet hajtunk végre. Az így kapott eredményt 8 db 6 bites csoportra osztjuk, amiket aztán különböző S-dobozokba pumpálunk. Egy ilyen 64 lehetőséget magában hordozó inputból az egyes S-dobozok 4 bites kimenetet generálnak. Végül az így nyert 8 × 4 bitet egy P-dobozon engedjük keresztül.

Mind a 16 iterációs lépésben különböző kulcsokat használunk. Az algoritmus kezdetekor egy 56 bites keverést végzünk a kulcson. Mindegyik lépés megkezdése előtt a



7.5. ábra. *Az adattitkosítási szabvány (DES) működése. (a) Általános áttekintés. (b) Egy részlet*

kulcsot két 28 bites részre particionáljuk, mindegyiket az iteráció sorszámának megfelelő számú bittel balra forgatva. A K_i-t ezekből a szegmensekből egy újabb 56 bites keverés során kapjuk meg. Az 56 bit-es kulcs egy 48 bites részét minden fokozatban még külön permutáljuk.

DES láncolása

A fenti módszer bonyolultságától eltekintve a DES alapvetően egy egybetű-helyettesítéses kódoló, ahol a betűméret 64 bit. Ugyanabból a 64 bites bemeneti blokkból mindig ugyanazt a kimenetet szolgáltatja. A kódfejtő kihasználhatja a DES ezen tulajdonságát.

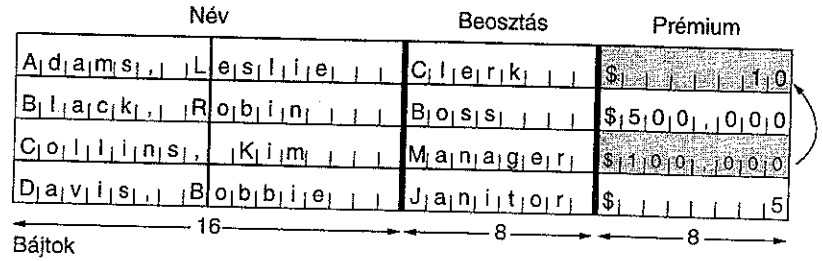
Annak megvizsgálására, hogy az egybetű-helyettesítéses kódolók e tulajdonsága hogyan használható fel a DES feltörésére, vegyük azt az egyszerű esetet, amikor egy üzenetet egymást követő 8 bájtos (64 bites) blokkokra szeletelünk, és azokat egymás után ugyanazzal a kulccsal titkosítunk. Az utolsó blokkot szükség esetén 64 bitessé egészítjük ki. Ezt a technikát **elektronikus kódkönyv módnak (electronic code book mode)** nevezik.

A 7.6. ábrán egy adatállomány elejét találjuk, melyben egy cég az egyes dolgozóinak juttatandó éves prémiumokat tárolta. Az állomány 32 bájtos rekordok sorozatából áll, ahol minden rekord egy-egy alkalmazotthoz tartozik és a következő formátumú: 16 bájt név, 8 bájt a beosztás és 8 bájt a prémium mértéke. Mind a 16 db 8 bájt hosszú blokkot (0-tól 15-ig) DES segítségével kódoltunk.

Leslie éppen összekülönbözött a felettesével, így nem sok jutalékra számíthat. Ezzel ellentétben Kim a főnök kedvence, amit mindenki tud. Az állomány Leslie kezébe kerül titkosítás után, de még mielőtt a bankba küldenék. Vajon kiegyenlítheti-e Leslie a prémiumok közötti különbséget csupán a titkosított állomány birtokában.

Minden különösebb gond nélkül. Csak annyit kell tennie, hogy a titkosított állomány 11. blokkját (ami Kim jutalékát tartalmazza) a 3. blokkba másolja (Leslie prémiumának a helyére). Anélkül, hogy ismerné a 11. blokk tartalmát, Leslie egy vidámabb karácsonynak nézhet elébe. (Igazából a 7. blokkot is lemásolhatná, de ezt sokkal gyorsabban észrevennék, emellett Leslie sem annyira mohó.)

Az ilyen típusú támadásokat kivédhetjük a DES-ben (és az összes blokk-kódoló-



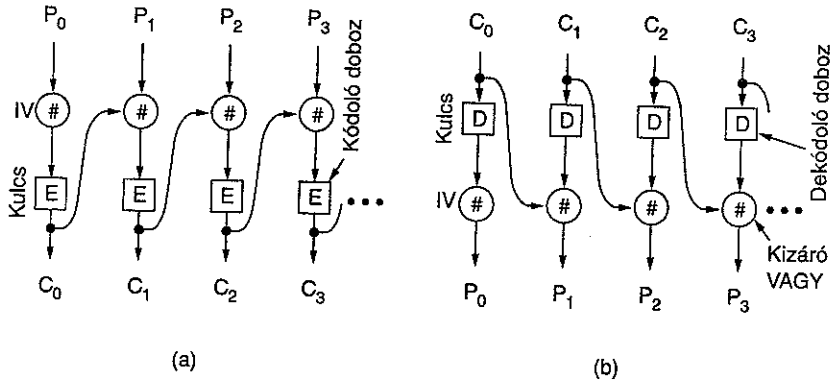
7.6. ábra. *Egy titkosítandó állomány nyílt szövege, mint a 16 blokkos DES*

ban) a kódolók különböző módon történő láncolásával, így a Leslie által is alkalmazott blokkcsere a visszakódolás után szemetet fog eredményezni a megváltoztatott blokk pozíciójától kezdődően. A láncolás egyik módja a **titkosított blokkok láncolása (cipher block chaining).** Ahogy az a 7.7. ábrán nyomon követhetjük, ennél a módszernél mindegyik nyílt szöveg blokk és az azt megelőző titkosított blokk között KIZÁRÓ VAGY műveletet hajtunk végre, mielőtt az adott blokkot kódolnánk. Ennek eredményeképpen ugyanaz a nyílt szövegrész már nem fog ugyanarra a titkosított blokkra transzformálódni, és az algoritmusunk a továbbiakban már nem tekinthető csupán egy robusztus egybetű-helyettesítéses titkosítónak. Az első blokkhoz egy véletlenszerűen választott **inicializáló vektort (initialization vector – IV)** használunk párként, amit aztán az üzenettel együtt továbbítunk.

Nézzük meg lépésről lépésre, hogy hogyan működik a 7.7. ábrán bemutatott blokkkódoló. Legelőször kiszámoljuk a $C_0 = E(P_0 \text{ XOR } IV)$ értéket. Ezután a $C_i = E(P_i \text{ XOR } C_0)$ meghatározásával folytatjuk. Dekódoláskor a $P_0 = IV \text{ XOR } D(C_0)$ lépéssel kezdünk. Vegyük észre, hogy az *i.* szegmens kódolt megfelelője függ mindegyik 0-tól *i*–1-ig terjedő nyílt szövegblokktól, így ugyanazt a szövegrészt más-más alakra kódolja a titkosító attól függően, hogy hányadik blokkban szerepel. A Leslie által véghezvitt transzformáció a hozzá tartozó blokkoktól kezdődően badarságot fog eredményezni két blokkban a kimeneten. Egy agyafúrt biztonsági tisztviselő számára a rendellenesség helye nyilvánvalóvá teszi, hogy hol kezdje a vizsgálatot.

A titkosított blokkok láncolásának azon kedvező tulajdonsága, hogy ugyanaz a szegmens más-más szegmensekre képződik le, a kriptoanalízist is megnehezíti. Ez a legfőbb ok, ami miatt alkalmazzák.

A blokk-kódolók láncba fűzésének hátránya azonban az, hogy meg kell várnunk egy teljes 64 bites blokk megérkezését a dekódolás megkezdése előtt. Interaktív terminálóknál, ahol a felhasználók 8 karakternél rövidebb sorozat bevitele után is válaszra várhatnak, ez a módszer használhatatlanná válik. A bájtonként történő titkosítást az ún. **visszacsatolásos kódolóval (cipher feedback mode)** oldjuk meg, amit a 7.8. ábrán mutatunk be. Az ábrán azt az állapotot látjuk, amikor a kódoló a 0-tól 9-ig terjedő



7.7. ábra. *Titkosított blokkok láncolása*

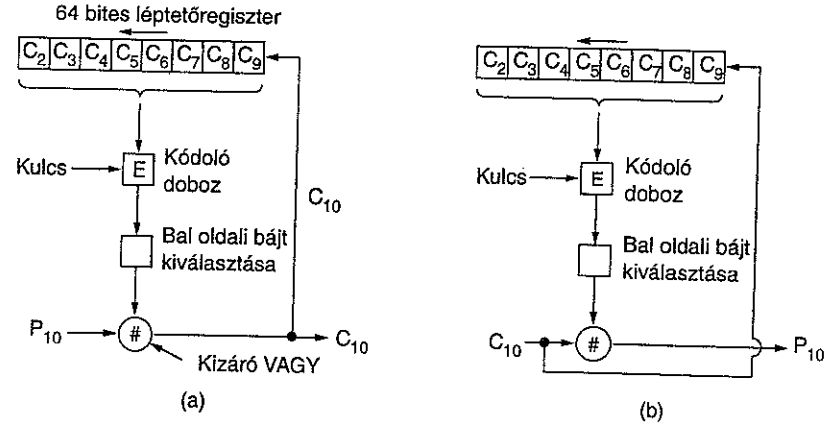
bájtokat már kódolta és elküldte. Amikor az ábrán látható módon megérkezik a 10. bájt, a DES algoritmus a 64 bites shift regiszter tartalmából készíti el a 64 bites kódolt üzenetet. A kódolt szövegrész legbaloldalibb karaktere és a P_{10} között ezután egy KIZÁRÓ VAGY műveletet hajtunk végre, amit aztán továbbítunk. A shift regiszter tartalmát 8 bittel balra mozgatjuk, melynek eredményeképpen a C_2 kipottyan a bal oldalon, a C_{10} pedig beül arra a helyre, amit ekkor hagy el a C_9. Vegyük észre, hogy a shift regiszter tartalma az eddig kódolásra került teljes nyílt szövegtől függ, így egy a kódolandó szövegben többször előforduló minta más-más kódolt párt kap, attól függően, hogy hol szerepel. Akárcsak a láncolt kódolóknál, itt is szükség van egy inicializáló vektorra, hogy elinduljon a játék.

A visszacsatolásos módban a dekódolás ugyanúgy működik, mint a kódolás. Tulajdonképpen a shift regiszter tartalmát inkább kódoljuk, mint dekódoljuk, mivel az a bájt, melyből a P_{10}-et állítjuk elő a C_{10}-zel való KIZÁRÓ VAGY művelet során, ugyanaz, mint amit a C_{10} előállításához használunk a küldő oldalán, amikor a P_{10}-zel hajtjuk végre a KIZÁRÓ VAGY műveletet. A visszakódolás így tökéletesen működik, feltéve, ha a két shift regiszter azonos.

Egy sajátságos mellékhatása a módszernek, hogy a titkosított szöveg egyetlen bitjének véletlenszerű megsérülése nyolc visszakódolt bájt eldeformálódását okozza, melyekkel együtt szerepelt a shift regiszterben. Amint a hibás bit kikerül a dekódoló shift regiszterből, ismét hibátlan szöveget kódolhatunk vissza. Így egyetlen bit véletlen átbillenése lokálisan jelentkezik, és nem teszi tönkre a teljes hátralevő üzenetrészt.

Vannak azonban olyan alkalmazások, ahol egyetlen bit által okozott 64 bitnyi kommunikációs hiba elfogadhatatlan. Az ilyen alkalmazások számára egy negyedik módszer jelentheti a megoldást, a **kimenet visszacsatolási mód (output feedback mode).** Nagyon hasonlít a visszacsatolásos módszerhez azzal a különbséggel, hogy a shift regiszter jobb szélére kerülő elemet a KIZÁRÓ VAGY művelet elvégzése előtt tesszük a regiszterbe, nem pedig a művelet elvégzése után.

A kimenet visszacsatolási mód tulajdonsága, hogy a titkos üzenet egybitnyi hibája



7.8. ábra. *Visszacsatolásos titkosító*

a visszakódolt üzenetben is csak egy bit hibát eredményez. Másrészről azonban ez a módszer kevésbé biztonságos a korábban bemutatottaknál, így általános felhasználásra nem ajánlott. Az elektronikus kódkönyv módot is – egy-két speciális esettől eltekintve (pl. egy viszony kulcs titkosítása) – célszerű elkerülni. Normális esetben a láncolt blokk-kódolót célszerű használni, ha az üzenet 8 bájtos blokkokban továbbítható (pl. lemezes állományok), illetve a visszacsatolásos módot akkor, ha a továbbítandó adatfolyam speciálisabb (pl. billentyűzet input).

A DES feltörése

A DES megszületése óta viták forrása. Története egy az IBM által készített és szabadalmaztatott kódolóval kezdődik, mely a Lucifer névre hallgatott. Az IBM által kifejlesztett sifrírozó azonban nem 56 bites, hanem 128 bites kulcsokkal dolgozott. Amikor az amerikai kormányzat kódoló szabványt szeretett volna elfogadtatni a nem bizalmas információk titkosítására, meghívta az IBM-et, hogy megvitassa a problémát az NSA-val, a kormányzat kódfejtő szervezetével, mely a világ legnagyobb, matematikusokat és kriptográfiai szakembereket foglalkoztató cége. Az NSA körül annyi titok lappang, hogy az iparban közszájon terjed a következő tréfa:

Kérdés: Mit jelent az NSA rövidítés?
Válasz: Nincs Semmiféle Alakulat.

A viccet félretéve, az NSA a National Security Agency (Nemzeti Biztonsági Ügynökség) rövidítése.

A tárgyalások után az IBM a 128 bitről 56 bitesre csökkentette a kulcs hosszát, és úgy döntött, hogy titokban tartja a DES tervezésével kapcsolatos információkat. Sokan feltételezik, hogy a kulcshossz ilyetén való csökkentése azért történt, hogy az NSA még feltörhesse a DES-t, de bármely kisebb költségvetéssel rendelkező szervezet erre képtelen legyen. A tervezéssel kapcsolatos titkok pedig némelyekben az a gyanút keltik, hogy kiskapu van elrejtve az algoritmusban, aminek segítségével az NSA jóval könnyebben töri fel a DES kódot. Amikor aztán egy NSA alkalmazott tapintatosan felkérte az IEEE-t, hogy ne tartson meg egy tervezett kriptográfiai konferenciát, a nyugtalanság csak fokozódott.

1977-ben két stanfordi kriptográfiával foglalkozó szakember, Diffie és Hellman (1977), egy, a DES feltörésére alkalmas gép terveit dolgozta ki, mely körülbelül 20 millió dollárból megépíthető lett volna. Egy viszonylag kicsi üzenetdarabka és annak kódolt párja alapján a gép megtalálta volna a titkosításhoz használt kulcsot a kulcstér mind a 2^{56} db kulcsának végigpróbálásával egy napon belül. Manapság egy ilyen gépet valószínűleg egymillió dollárból meg lehetne építeni. Egy hasonló eszköz részletes terve megtalálható (Wiener, 1994) publikációjában. Ez a szerkezet a végigpróbálásos módszer segítségével 4 óra alatt feltör egy DES kulcsot.

De nézzünk egy másik stratégiát! Bár a szoftver alapú kódolás mintegy 1000-szer lassúbb a hardver által végzett titkosításnál, egy csúcsteljesítményű számítógép így is 250 000 kódolás/másodperc teljesítményre képes szoftverből, és feltehetjük, hogy egy

ilyen gép 2 millió másodpercet henyél egy hónapban. Ezt a holtidőt felhasználhatjuk a DES kulcsok keresésére. Egy megfelelő hírcsoportba postázott üzenet segítségével nem nehéz 140 000 felhasználót rábírni arra, hogy havonta 7×10^{16} kulcsot ellenőrizzen.

A DES feltörésére kitalált egyik legmeghökkentőbb ötlet a **Kínai Lottó elmélet (Chinese Lottery)** (Quisquater és Girault, 1991). Az ötlet szerint minden rádió- és televíziókészüléket fel kéne szerelni egy olyan olcsó DES chippel, mely hardver úton 1 millió kódolást tudna elvégezni másodpercenként. Feltételezve, hogy mind az 1,2 milliárd kínai lakos rendelkezik egy ilyen készülékkel, a kínai kormány könnyűszerrel feltörhet egy DES-sel kódolt üzenetet. Mindehhez csak annyit kell tennie, hogy egy – a keresett kulccsal – titkosított szöveget és annak kódolatlan párját eljuttatja a készülékekhez, amik aztán egyszerre állnak neki a kulcstérben való kereséshez, mindegyik a számára kiosztott területen próbálkozva. Így 60 másodpercen belül legalább egy állomás megtalálja a keresett kulcsot. Hogy a kulcsot biztosan bejelentsék, a készüléknek találat esetén a következő üzenetet kéne megjelenítenie:

GRATULÁLUNK! ÖN NYERT A KÍNAI LOTTÓ NYERESÉNYSORSOLÁSÁN. A NYERESÉNY ÁTVÉTELÉHEZ HÍVJA AZ 1-800-SOK-PÉNZ TELEFONSZÁMOT.

A fenti példákból levonhatjuk azt a következtetést, hogy a DES ma már alkalmatlan fontos információk védelmére. Ennek ellenére, bár 2^{56} csupán vacak 7×10^{16}, a 2^{112} már egy jóval tekintélyesebb nagyságrend (5×10^{33}). Ha egymilliárd DES chippel rendelkeznénk, melyek külön-külön másodpercenként egymilliárd műveletet tudnának elvégezni, még akkor is 100 millió évig tartana egy kulcs után keresve a 112 bites kulcstér átkutatása. Így támadt az az ötlet, hogy használjuk a DES-t kétszer egymás után két különböző kulccsal.

Sajnálatos módon Merkle és Hellman (1981) kifejlesztettek egy olyan módszert, mely alapján a kétszeres kódolás biztonságába vetett hit meggyengült. A módszert **középen találkozó típusú feltörési kísérletnek (meet-in-the-middle)** hívják, és a következőképpen működik (Hellman, 1980). Tegyük fel, hogy valaki kétszeresen kódolt egy hosszú blokksorozatot az elektronikus kódkönyv módszert használva. A kódtörő rendelkezik néhány (P_i, C_i) párral, ahol

$$C_i = E_{K2}(E_{K1}(P_i))$$

Ha ezek után a D_{K2} dekódoló függvényt alkalmazzuk az egyenlet mindkét oldalán, a következőt kapjuk:

$$D_{K2}(C_i) = E_{K1}(P_i),$$

mivel x-et ugyanazzal a kulccsal kódolva, majd dekódolva visszakapjuk x-et.

A középen találkozó típusú feltörési kísérlet a fenti összefüggést a következőképpen használja fel a $K1$, illetve a $K2$ kulcsok megfejtéséhez:

1. Kiszámítja az $R_i = E_i(P_1)$ értékeket az összes 2^{56} db i-re, ahol E a DES kódoló függvény. Majd a táblázatot az R_i értékek alapján növekvő sorba rendezi.

2. Kiszámítja az $S_j = D_j(C_1)$ értékeket az összes 2^{56} db j-re, ahol D a DES dekódoló függvény. Majd a táblázatot az S_j értékek alapján növekvő sorba rendezi.

3. Az első táblában olyan R_i értékeket keres, mellyel megegyező S_j érték szerepel valahol a második táblában. Ha ilyet talál, máris megvan egy kulcspár (i, j), melyre $D_j(C_1) = E_i(P_1)$. Nagy valószínűséggel i $K1$-nek, j pedig $K2$-nek fog megfelelni.

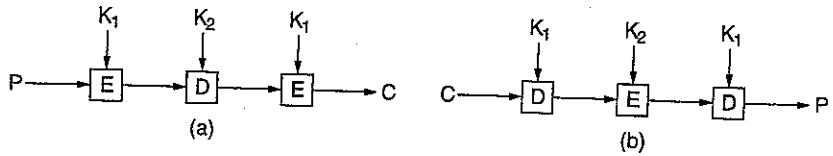
4. Ellenőrzi, hogy $E_j(E_i(P_2))$ megegyezik-e C_2-vel. Ha így van, akkor más nyílt szövegen és kódolt párján is ellenőrzi az eredményt. Egyébként pedig folytatja a táblázatokban való keresést újabb potenciális kulcspárok után.

Számos kulcspár jelölt fog adódni a módszer során, mielőtt a valódiak megkerülnek, de ez előbb-utóbb bekövetkezik. A támadás műveletigénye mindössze 2^{57} kódolás, illetve dekódolás (a két tábla elkészítéséhez), ami jóval kevesebb a 2^{112}-nél. Egyetlen hátulütője, hogy a táblák tárolásához 2^{60} bájt szükségeltetik, így a fenti formájában ma még nem megvalósítható, de Merkle és Hellman számos olyan optimalizálási lehetőséget talált, mellyel csökkenthető a tárigény az elvégzendő műveletszám rovására. Mindent összevetve megállapíthatjuk, hogy a DES kétszeres alkalmazása nem garantál jelentősebben nagyobb biztonságot, mint hagyományos párja.

A háromszoros kódolás már egy másik történet. Már 1979-ben az IBM felismerte, hogy a DES kulcs túlságosan rövid, és a biztonság növelése érdekében kidolgoztak egy háromszoros kódolást használó eljárást (Tuchman, 1979). A módszert, melyet azóta 8732-s Nemzetközi Szabványként elfogadtak, a 7.9. ábrán tüntettük fel. Két kulcsot és három fokozatot használunk hozzá. Az első lépcsőben a nyílt üzenetet a K_1 kulccsal kódoljuk. Második lépésben dekódolást végzünk, melyhez a K_2-t használjuk kulcsként. Az így kapott eredményt ismét az első kulccsal kódoljuk.

Ez a konstrukció rögtön két kérdést vet fel. Először is, miért használtunk csupán két kulcsot három helyett? Másodszor, miért alkalmaztuk az EDE algoritmust, és nem az EEE sorrendet? A két kulcs oka, hogy még a legparanoiásabb kriptográfusok is egyetértenek abban, hogy az elkövetkező időben az üzleti alkalmazások számára a 112 bites kulcshossz bőven elegendő. 168 bit alkalmazása csak felesleges többletköltséget jelentene a tárolás és a kulcscsere során.

Az alkalmazott sorrendre, mely szerint először kódolunk, dekódolunk majd ismét kódolunk, a régi egykulcsos DES rendszerekkel való kompatibilitás miatt volt szük-



7.9. ábra. *DES-t használó háromszoros titkosítás*

ség. Mind a kódolás, mind a dekódolás egy függvénynek tekinthető a 64 bites számok halmazán. Kriptográfiai szempontból mindkét leképzés egyforma erejű. Az EEE helyett az EDE sorrend alkalmazása viszont lehetővé teszi, hogy egy háromlépcsős kódoló egy hagyományos társával működjön együtt a K_1=K_2 azonos kulcsok használatával. A háromszoros DES ezen tulajdonsága alkalmassá teszi fokozatos bevezetését, a konzervatívabb kriptográfus világ bolygatása nélkül, mégis jelentős lépést téve az IBM és ügyfelei számára.

Jelenleg nem ismert az EDE sorrendet használó háromszoros DES-t feltörő módszer. Van Oorschot és Wiener (1988) publikáltak egy olyan eljárást, mellyel az EDE alapú kulcskeresés egy 16 szorzó erejéig felgyorsítható, de a háromszoros DES még e mellett is kifejezetten biztonságosnak tekinthető. Azok számára, akik csak a legjobbal elégednek meg, az EEE módszer három 56 bites (összesen 168 bites) kulccsal történő használata javallott.

Mielőtt befejeznénk a DES-sel való vizsgálódásunkat, érdemes megemlíteni két újkeletű kriptoanalitikai módszert. Az elsőt **differenciális kriptoanalízis (differential cryptoanalysis)** néven emlegetik (Biham és Shamir, 1993). A módszer tetszőleges blokk kódoló elleni támadásra használható. Működésének alapja, hogy minimálisan eltérő szövegblokkok útját követik párhuzamosan nyomon a kódolás során. Könnyen megeshet, hogy lesznek olyan minták, melyek sokkal nagyobb közös viselkedést mutatnak, és ezek felismerése lehetővé tesz valószínűségi alapon történő támadásokat.

A másik figyelemre méltó módszer a **lináris kriptoanalízis (linear cryptoanalysis)** (Matsui, 1994). Segítségével mindössze 2^{43} ismert nyílt szöveg alapján megtalálható a DES kulcs. A nyílt és titkosított szövegek adott bitjei között KIZÁRÓ VAGY műveletet végez, melyet elég gyakran ismételve az eredményben az 1-es és 0-s biteknek nagyjából egyenlő arányban kellene megjelenniük. Gyakran azonban a kódolók hajlamosak vagy az egyik, vagy a másik érték nagyobb arányára, s bár az eltérés nem jelentős, mégis felhasználható a törés munkaigényének jelentősebb csökkentésére. A részletes ismertetése megtalálható Matsui publikációjában.

IDEA

A hagyományos DES gyengeségének ostorozása egy kicsit olyan, mintha egy döglött lovat ütlegelnénk, de az igazság az, hogy az egykulcsos DES-t még mindig széles körben alkalmazzák olyan érzékeny alkalmazásokban, mint a pénzügyi műveleteket végző bankautomaták. Bár egy-két évtizeddel ezelőtt, amikor ezeket megalkották, ez a választás még elfogadható volt, ma már nem jelent kielégítő megoldást.

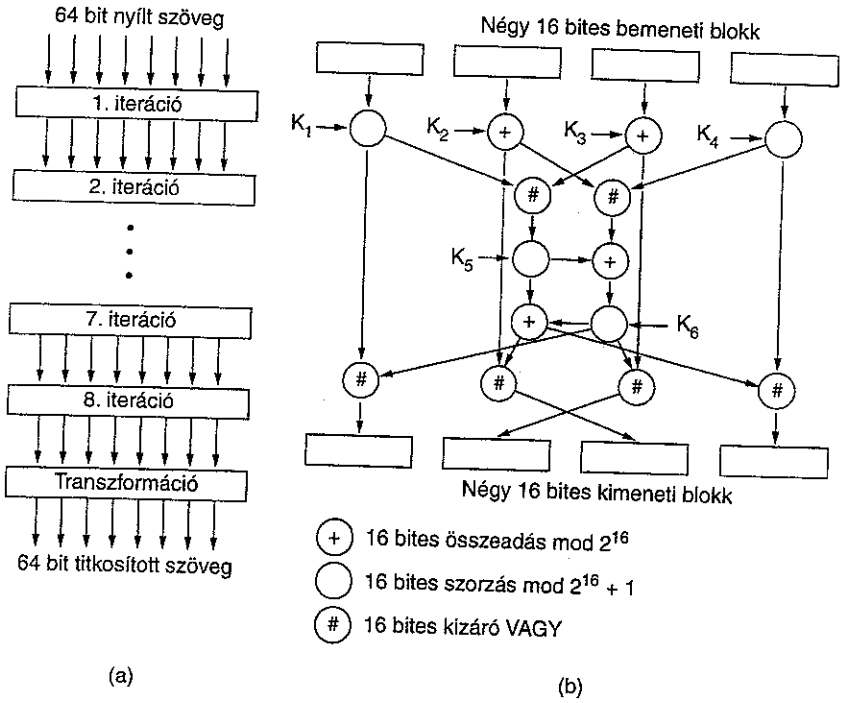
Most felvetődhet az olvasóban a jogos kérdés: „Ha a DES ennyire gyenge, miért nem találtak ki ennél jobb blokk-kódolót?" A helyzet az, hogy valóban kifejlesztettek számos blokk kódolót, mint a BLOWFISH (Schneier, 1994), a Crab (Kaliski és Robshaw, 1994), a FEAL (Shimizu és Miyaguchi, 1988), a KHAFRE (Merkle, 1991), a LOKI91 (Brown és mások, 1991), a NEWDES (Scott, 1985), a REDOC-II (Cusick és Wood, 1991) és a SAFER K64 (Massey, 1994). Schneier (1996) a fentiek mellett számtalan más módszert ismertet könyvében. A DES után kitalált blokk-kódolók között talán az egyik legfigyelemreméltóbb az **IDEA (International Data Encryption**

Algorithm – nemzetközi adat kódoló algoritmus) (Lai és Massey, 1990; Lai, 1992). Ismerkedjünk meg tehát részletesebben az IDEA-val.

Az IDEA-t két svájci kutató fejlesztette ki, így feltehetően mentes az NSA azon törekvéseitől, melyek esetleges kiskapuk beépítését szorgalmaznák. 128 bites kulcsot használ, ami a következő évtizedekben megvédi az olyan faltörő típusú támadásoktól, mint a Kínai Lottó vagy a középen találkozó típusú módszer. Fejlesztésekor ügyeltek arra, hogy ellenálljon a differenciális kriptoanalízisnek. Jelenleg nem ismert olyan módszer vagy számítógép, mely emberi idő alatt feltörné az IDEA-t.

Az algoritmus sok közös vonást mutat a DES-sel, mivel ez is 64 bites nyílt szöveg blokkok sorozatát sifrírozza egy paraméterezett iteráció segítségével, melynek során 64 bites kódolt blokkokat kap, ahogy ezt a 7.10.(a) ábrán feltüntettük. Az alapos keveroeljárásnak köszönhetően (minden iterációs lépésben minden kimeneti bit minden bemeneti bittől függ) nyolc lépcső elegendő. Mint minden más blokk kódoló, az IDEA is használható visszacsatolt módban és a DES-nél bemutatott egyéb változatokban.

A 7.10.(b) ábrán egy iterációs lépés belsejét vázoltuk. Háromfajta műveletet használunk, mindegyiket 16 bites számokon. Ezek a műveletek: a KIZÁRÓ VAGY, a mo-



7.10. ábra. *(a) IDEA. (b) Egy iteráció részletei*

dulo 2^{16} összeadás és a modulo $2^{16} + 1$ alapú szorzás. Mindhárom művelet könnyen elvégezhető egy 16 bites mikorszámítógépen az eredmények felső részének eldobásával. A fenti műveletek kedvező tulajdonsága, hogy bármelyik kettő között áll fenn az asszociativitás, illetve a disztributivitás, ami jelentősen megnehezíti a differenciális kriptoanalízist. A 128 bites kulcs alapján 52 db egyenként 16 bites alkulcsot generálunk, minden iterációs lépés számára 6-ot és a végső transzformációhoz 4-et. A dekódolás ugyanazt az algoritmust használja, mint a kódolás, csak az alkulcsok mások.

Az IDEA-nak elkészítették mind a szoftveres, mind a hardveres implementációját. Az első szoftveres megvalósítás egy 33 Mhz-es 386-os számítógépen futott, és 0,88 Mb/s kódolási sebességre volt képes. Egy mai számítógép ennél legalább 10-szer gyorsabb, így szoftver úton 9 Mb/s sebesség érhető el. A zürichi ETH intézetben egy 25 Mhz-es VLSI chipet építettek, melyből 177 Mb/s sebességet sikerült kicsiholni.

7.1.4. Nyilvános kulcsú algoritmusok

A legtöbb kriptográfiai rendszer gyenge pontja hosszú időn keresztül a kulcskiosztás problémája volt. Függetlenül a titkosító rendszer erejétől, ha a kulcs a támadó kezébe került, a rendszer teljesen védtelenné vált. Mivel a kriptológusok mindig elfogadták azt a nézetet, hogy a kódoláshoz és dekódoláshoz szükséges kulcsoknak azonosnak (vagy egymásból könnyűszerrel generálhatóknak) kell lenniük, és ezeket minden érintett felhasználóhoz el kell juttatni, feloldhatatlan problémának tűnt a kulcsok illetéktelen kezektől való megóvása, mivel azokat nem lehetett páncélszekrénybe zárni a kulcskiosztás szükségessége miatt.

1976-ban két stanfordi kutató, Diffie és Hellman (1976) egy merőben új kriptográfiai rendszert javasolt, amiben a kódoló és dekódoló kulcsok nemcsak különbözőek, de egymásból nem állíthatók elő. A módszerükben szereplő kódoló (E) és dekódoló (D) algoritmussal szemben három követelményt támasztottak. Ezeket a feltételek a következők:

1. $D(E(P)) = P$.

2. D előállítása E alapján rendkívül nehéz feladat legyen.

3. E feltörhetetlen legyen választott nyílt szöveg típusú támadással.

Az első szabály azt mondja, hogyha a D műveletet alkalmazzuk a kódolt szövegre, $E(P)$-re, akkor az eredeti szöveget, P-t kell, hogy visszakapjuk. A második pont önmagáért beszél. A harmadik követelményre azért van szükség, mivel – ahogy ezt mindjárt látni fogjuk – a támadók kényük-kedvük szerint kísérletezhetnek majd a kódoló eljárással.

A módszer a következő módon működik. Tegyük fel, hogy Alice titkos üzeneteket szeretne fogadni, ehhez először kifejleszt két olyan algoritmust és a hozzájuk tartozó kulcsokat, E_A-t és D_A-t, melyek eleget tesznek a fenti követelményeknek. Ezek után a kódoló algoritmust, illetve a titkosító kulcsot nyilvánosságra hozza, innen ered a **nyil-**

vános kulcsú titkosítás (public-key cryptography) elnevezés is (ellentétben a hagyományos titkos kulcsú módszerekkel). Ezt a legegyszerűbben egy mindenki által olvasható fájl segítségével oldja meg, mely tartalmazza a nyilvános információkat. Alice a dekódoló eljárást is elárulja, egyedül a dekódoló kulcsot tartja titokban. Így tehát az E_A nyilvános, míg a D_A személyes marad.

Most nézzük meg, hogyan használható a módszer Alice és Bob között létrehozandó titkosított csatorna létrehozására, ha előtte még sohasem találkoztak. Mindenképpen szükséges, hogy Alice titkosító kulcsa (E_A) és Bob titkosító kulcsa (E_B) egy mindenki által olvasható területen legyenek. (A legjobb megoldás az lenne, ha minden hálózathasználó első belépésekor publikálná kódoló kulcsát.) Alice az első elküldendő üzenetet, P-t titkosítja Bob kulcsával, így kapja E_B (P)-t, amit aztán elküld Bob-nak. Bob ezt könnyűszerrel visszakódolhatja a titkos kulcsa (D_B) seítségével, mivel $D_B(E_B(P)) = P$. Rajta kívül senki más nem képes az üzenet visszafejtésére, mivel feltételeztük, hogy a kódolás erős védelmet nyújt, és a D_B előállítása E_B alapján szinte lehetetlen. Ezek után Alice és Bob biztonságosan kommunikálhatnak.

Érdemes talán egy kis megjegyzést tenni a terminológiával kapcsolatban. A nyilvános kulcsú módszerek minden felhasználótól két kulcsot követelnek meg: a nyilvános kulcsot, mely segítségével bárki kódolhat számukra információt, és egy egyéni kulcsot, melyet tulajdonosa az üzenetek dekódolására használ. Ebben a könyvben következetesen nyilvános és egyéni kulcsként fogunk rájuk hivatkozni, hogy élesen megkülönböztessük őket azoktól a titkos kulcsoktól, melyeket egyszerre használnak kódolásra és dekódolásra a hagyományos (vagy más néven **szimmetrikus**) kriptográfia terén.

Az RSA algoritmus

Az egyetlen buktató, hogy a fenti három követelménynek eleget tevő eljárásokat kéne találnunk. A nyilvános kulcsú titkosítás nyilvánvaló előnyei miatt számos szakember komoly munkába kezdett, és sikerült mára néhány ilyen módszert kifejlesztenie. Az egyiket ezek közül egy, az M.I.T-n dolgozó csoport (Rivest és mások, 1978) alkotta meg. Az algoritmus azóta a felfedezők (Rivest, Shamir, Adleman) kezdőbetűi alapján **RSA** néven vonult be a köztudatba. Módszerük a számelmélet tételein alapszik. Most röviden foglaljuk csak össze az algoritmus lépéseit, a részletek megismeréséhez az eredeti művet ajánljuk.

1. Válasszunk két nagy prímszámot, p-t és q-t (lehetőleg 10^{100}-nál nagyobb számok legyenek).
2. Számoljuk ki az $n = p \times q$ és a $z = (p - 1) \times (q - 1)$ számokat.
3. Válasszunk egy z-hez relatív prímet, jelöljük d-vel.
4. Keressünk egy olyan e számot, melyre $e \times d = 1 \bmod z$.

A fenti paraméterek előzetes meghatározása után megkezdhetjük a titkosítást. A nyílt szöveget, mint egyszerű bitsorozatot blokkokra osztjuk oly módon, hogy egy kódolandó szegmens (P) a $0 \le P < n$ intervallumba essen. Ezt legkönnyebben úgy érhetjük el, ha az eredeti üzenet bitjeit k bit hosszú csoportokba szedjük, ahol k az a legnagyobb egész, melyre még fennáll a $2^k < n$ összefüggés.

A titkosítandó üzenetdarab (P) alapján kiszámítjuk $C = P^e \pmod n$ értéket. C visszakódolásához a $P = C^d \pmod n$ összefüggést használjuk. Bebizonyítható, hogy minden olyan P-re, mely a fenti tartományba esik, a megadott kódoló és dekódoló függvények egymás inverzei. A kódoláshoz az e és az n számokra van szükség, míg a visszafejtéshez a d-re és az n-re. Ennek alapján a nyilvános kulcs az (e, n) párból, míg az egyéni kulcs a (d, n) párból fog állni.

A módszer biztonsága a nagy számok faktorizálásának nehézségén alapszik. Ha a támadó képes lenne szorzatalakra hozni a mindenki által ismert n számot, megkapná a p-t és a q-t, ami alapján a z-t kiszámolhatná. A z érték ismeretében pedig az e és a d meghatározása az euklideszi algoritmussal elvégezhető. Szerencsére azonban az utóbbi 300 év próbálkozásai alapján a matematikusok többsége azon a véleményen van, hogy a nagy számok szorzatra bontása rendkívül nehéz feladat.

Rivest és kollégái szerint egy 200 bites szám számítógépes faktorizálása 4 milliárd évet venne igénybe, egy 500 bitesé pedig 10^{25} évig tartana. Mindkét esetben az általuk leghatékonyabb algoritmussal és egy olyan számítógéppel számoltak, mely egy utasítást 1 μs alatt hajt végre. Még ha folytatódna is a számítógépek sebességének évtizedenkénti egy nagyságrenddel való növekedése, akkor is évszázadok múlva kerül az 500 bites számok gyors faktorizálása elérhető közelségbe. Utódainknak ekkor sem kell mást tenniük, mint a p és a q számokat hosszabbra választani.

Az RSA algoritmus egy klasszikus iskolapéldáját mutatjuk be a 7.11. ábrán. A példa kedvéért kis számokat választottunk: $p = 3$ és $q = 11$, amiből $n = 33$ és $z = 20$ adódik. A $d = 7$ választás megfelelőnek tűnik, mivel 7-nek és 20-nak nincs közös osztója. A fenti értékekkel e könnyen kiszámolható a $7e = 1 \pmod{20}$ összefüggés alapján. A P nyílt szöveg sifrírozott párja, C, a $C = P^3 \pmod{33}$ kifejezésből adódik. Az üzenet fogadója az eredeti szöveget a $P = G^7 \pmod{33}$ képlet segítségével kapja vissza. Az ábra a „SUZANNE" tartalmú üzenet kódolási lépéseit mutatja.

Nyílt szöveg (P)		Titkos szöveg (C)			Dekódolás után	
Szimbólum	Számérték	P^3	$P^3 \pmod{33}$	C^7	$C^7 \pmod{33}$	Szimbólum
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	1	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	5	E
A küldő számítása			A fogadó számítása			

7.11. ábra. *Egy példa az RSA algoritmusra*

Mivel a példában szereplő prímeket készakarva kicsire választottuk, a P értékének is alacsonynak (33-nál kisebbnek) kell lennie, ezért egy kódolandó szövegblokk csak egy karaktert tartalmazhat. Ennek eredményeképpen egy egyszerű egybetű-helyettesítéses kódolót kapunk, ami nem túl hatásos. Ha ehelyett a p-t és a q-t a 10^{100} nagyságrendben választottuk volna meg, akkor az n-re 10^{200} körüli érték adódott volna, így egy blokk mintegy 664 bit, vagy ha úgy jobban tetszik 83 db 8 bites karakter lehetett volna, ellentétben a DES 8 karakterével.

Érdemes megjegyeznünk, hogy a fenti módszerrel használt RSA hasonlóan viselkedik a DES ECB módjához abban a tekintetben, hogy ennél is ugyanaz a bemeneti blokk mindig azonos kimeneti blokkot fog eredményezni. Így célszerű lehet valamilyen láncolást bevezetni nagyobb adatmennyiség titkosítására. A gyakorlatban azonban a legtöbb RSA alapú rendszer a nyilvános kulcsokat csak egy egyszer használatos viszonykulcs biztonságos szétosztására használja. A viszonykulcsot ezután klasszikus titkos kulcsként használja DES, IDEA vagy más hasonló jellegű algoritmusokhoz. Az RSA jelenleg túl lassú nagyobb adathalmaz kódolására.

Más nyilvános kulcsú eljárások

Bár az RSA igen széles körben alkalmazott, korántsem ez az egyetlen ismert nyilvános kulcsú algoritmus. Az első nyilvános kulcsú eljárás a hátizsák (knapsack) módszer volt (Merkle és Hellman, 1978). Tegyük fel, hogy az üzenetet küldeni szándékozó fél nagyszámú különböző súlyú tárggyal rendelkezik. Az üzenet kódolásához a tárgyak közül titokban kiválaszt párat, és azokat a zsákba teszi. A zsák összsúlyát, valamint a szóba jöhető tárgyak listáját ezek után nyilvánosságra hozza. A zsákban levő tárgyak listáját titokban tartja. Néhány magától értetődő korlátozás mellett azon tárgyak listájának előállítása, mely adott össztömeggel rendelkezik, algoritmikusan nehéz feladat, és ez képezi a módszer alapját.

Az algoritmus kifejlesztője, Ralph Merkle, olyannyira biztos volt a módszer feltörhetetlenségében, hogy 100 dolláros díjat ajánlott fel annak, akinek mégis sikerül. Adi Shamir (az RSA-ban szereplő „S" betű tulajdonosa) hamarosan jelentkezett a megoldással a 100 dollárért. Merkle nem riadt vissza, így nemsokára 1000 dolláros díjat tűzött ki feljavított algoritmusának feltöréséért. Ron Rivest (az RSA „R" betűje) nem késlekedett a megoldással, és bezsebelte a díjat. Merkle ezek után nem mert a következő változatra 10 000 dolláros tétet tenni, így az „A" betű (Leonard Adleman) hoppon maradt. Annak ellenére, hogy újra javítottak rajta, a hátizsák algoritmust nem tekintik biztonságosnak és nem igazán alkalmazzák.

Más nyilvános kulcsú módszerek a diszkrét logaritmus számításának számítási igényére alapoznak (Rabin, 1979). El Gamal (1985) és Schnorr (1991) erre az elvre épülő algoritmusokat fejlesztett ki.

Néhány egzotikusabb módszer is létezik, mint amilyen az ellipszisíveken alapuló módszer (Menezes és Vanstone, 1993), de alapvetően három kategóriába sorolhatók az eljárások: a nagy számok faktorizálásán alapuló, a diszkrét logaritmust számoló, illetve a hátizsák tartalmát összsúly alapján meghatározó algoritmusok.

7.1.5. Hitelességvizsgáló protokollok

A **hitelességvizsgálat (authentication)** olyan módszer, amivel egy folyamat ellenőrizheti, hogy kommunikációs partnere valóban az-e, akinek lennie kell, nem pedig egy csaló. Egy távoli folyamat azonosságának ellenőrzése meglehetősen nehéz, egy rosszakaratú, aktív támadó jelenlétében és titkosításon alapuló komplex protokollokat igényel. Ebben a részben megvizsgálunk néhány olyan hitelességvizsgáló protokollt, amelyek a nem-megbízható számítógépes hálózatokban használatosak.

Félreértésből néhányan keverik a hitelesség- és jogosultságvizsgálat fogalmát. A hitelességvizsgálat azzal foglalkozik, hogy valóban a megfelelő folyamattal történik-e a kommunikáció. A jogosultságvizsgálat pedig a folyamat hatáskörének eldöntésére vonatkozik. Például egy fájlszerverhez fordul egy ügyfél folyamat és azt kéri: „Én Scott folyamata vagyok, és le akarom törölni a *cookbook.old* fájlt.". A fájlszerver szempontjából két kérdés merül fel:

1. Ez tényleg Scott folyamata-e (hitelességvizsgálat)?

2. Scott letörölheti-e a *cookbook.old* fájlt (jogosultságvizsgálat)?

A kérést csak azután lehet teljesíteni, miután mindkét kérdésre egyértelműen igenlő a válasz. A hangsúly az első kérdésen van. Miután a szolgáltató tudja kivel beszél, a jogosultságvizsgálathoz már csak meg kell nézni egy helyi táblázat megfelelő bejegyzését. Éppen ezért ebben a részben a hitelességvizsgálatra koncentrálunk.

A hitelességvizsgáló protokollok általában a következő modellt használják. Egy kezdeményező felhasználó (valójában folyamat), mondjuk Aliz, szeretne létrehozni egy biztonságos kapcsolatot egy másik felhasználóval, Bobbal. Aliz és Bob történetünk fontos résztvevői, ezért néha **főszereplőknek (principals)** is nevezik őket. Bob egy bankár, akivel Aliz üzletelni szeretne. Aliz azzal kezdi, hogy küld egy üzenetet Bobnak, vagy egy megbízható **kulcsszétosztó központnak (key distribution center – KDC),** ami mindig becsületes. Ezután számos üzenetváltás történik mindkét irányban. Miközben az üzenetek haladnak egy kellemetlen betolakodó, Trudy, lehallgathatja, módosíthatja vagy Bob és Aliz megtévesztésére, esetleg munkájuk ellehetetlenítésére újraküldi azokat.

Mindazonáltal, amikor a protokoll tökéletes, Aliz biztos lehet, hogy Bobbal beszél, és Bob is biztos lehet, hogy Alizzal beszél. Ezenkívül, a protokollok többségében, létrejön egy kettejük kapcsolatára jellemző titkos **viszony kulcs (session key),** amit ezután a párbeszédhez használnak. A gyakorlatban, hatékonysági okokból, minden adatforgalmat titkos kulcsú titkosítással kódolnak annak ellenére, hogy a nyilvános-kulcsú titkosítás széles körben használatos magában a hitelességvizsgáló protokollban, és a viszony kulcs létrehozásánál.

A véletlenszerűen választott friss viszony kulcs azért hasznos, mert használatával minimalizálni lehet a felhasználó saját titkos vagy nyilvános kulcsával kódolt adatforgalmat, ezáltal csökkentve a támadó által megkaparintható titkosított szöveget és, mert minimális lehet a kár, ha egy folyamat összeomlik, és a memóriamentés rossz kezekbe kerül. A viszony felépítése után minden visszamaradó kulcsot gondosan ki kell nullázni.

Osztott titkos kulcson alapuló hitelességvizsgálat

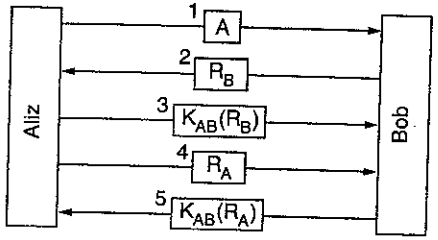
Első protokollunkhoz feltételezzük, hogy Aliznak és Bobnak már van egy osztott titkos kulcsa, K_{AB}. (A protokollok formális leírásában Alizt *A* Bobot pedig *B* helyettesíti.) A felek a megosztott kulcsot megbeszélhetik telefonon vagy személyesen, de semmiképpen sem a (nem megbízható) hálózaton.

Ez a protokoll is azt az alapötletet használja, amit számos más hitelességvizsgáló protokoll: az egyik résztvevő egy véletlen számot küld a másiknak, aki azt speciális módon transzformálja, majd az eredményt visszaküldi. Az ilyen típusú protokollokat **kihívás-válasz (challenge-response)** protokolloknak nevezzük. Ebben és az ezt követő hitelességvizsgáló protokollokban a következő jelölések érvényesek:

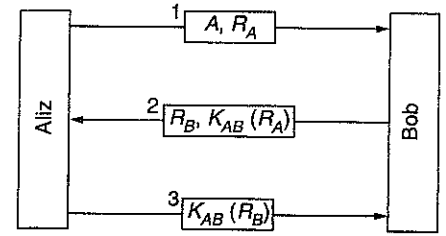
A, *B* Aliz és Bob azonosítója
R_i-k a kihívások, ahol az i index a kihívót azonosítja
K_i-k kulcsok, ahol i a tulajdonosra vonatkozik; K_S a viszony kulcs

Az első protokollunkhoz tartozó üzenetváltási sorrend a 7.12. ábrán látható. Az első üzenetben Aliz elküldi azonosítóját, *A*-t, Bobnak, mégpedig Bob által érthető formában. Bob, természetesen, nem tudhatja, hogy az üzenet Aliztól jött-e vagy Trudytól, ezért kiválaszt egy nagy véletlen számot R_B-t, majd kihívásként visszaküldi ezt Aliznak a 2-es számú kódolatlan, nyílt szövegű üzenetben. Ezután Aliz titkosítja a Bobbal megosztott titkos kulccsal az üzenetet, majd a titkosított szöveget visszaküldi Bobnak a 3-as számú üzenetben. Amint Bob megkapja ezt az üzenetet, már biztos benne, hogy Aliztól jött, hiszen Trudy nem tudja K_{AB}-t, és így ő nem generálhatta azt. Továbbá, mivel R_B-t véletlenszerűen generálta egy nagy halmazból (mondjuk 128-bites számok közül) nem valószínű, hogy Trudy egy korábbi viszony során megfigyelhette R_B-t, és az arra küldött választ.

Ekkor Bob már biztos benne, hogy Alizzal beszél, de Aliz még nem biztos semmiben. Amit Aliz tud, az még biztosítja arról, hogy Trudy nem hallgatta le az 1-es üzenetet, és nem ő küldte vissza R_B-t. Lehet, hogy Bob az éjszaka meghalt. Ahhoz, hogy megtudja kivel beszél, Aliz választ egy nagy véletlen számot, R_A-t, és a kódolatlan 4-es üzenetben elküldi Bobnak. Ha Bob $K_{AB}(R_A)$-val válaszol, akkor Aliz már tudja, hogy Bobbal beszél. Ha ezek után szeretnének megbeszélni egy viszonykulcsot, akkor Aliz kitalál egyet, és elküldi Bobnak K_{AB}-vel titkosítva.



7.12. ábra. *Kétirányú hitelességvizsgálat kihívás-válasz protokollal*



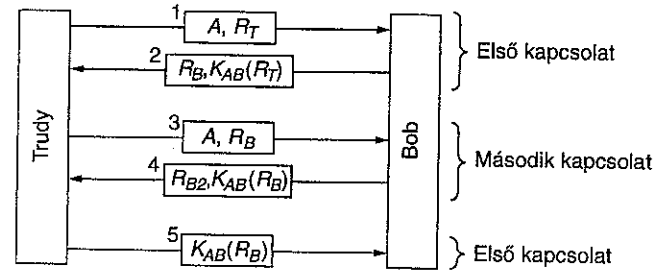
7.13. ábra. *Rövidített kétirányú hitelességvizsgálat protokoll*

A 7.12. ábrán látható protokoll működik, azonban tartalmaz néhány felesleges üzenetet. Ezeket kiküszöbölhetjük az információk kombinálásával, ahogyan azt a 7.13. ábra mutatja. Itt Aliz nem várakozik Bobra, hanem rögtön kezdeményezi a kihívás-válasz protokollt. Bob is rögtön elküldi kihívását az Aliznak küldött válaszában. Így az egész protokollt 5 helyett 3 üzenetre lehet rövidíteni.

Vajon ez az új protokoll jobb-e, mint az eredeti. Egy szempontból igen: rövidebb. Sajnos azonban rosszabb is. Bizonyos körülmények között Trudy legyőzheti ezt a protokollt, a **visszatükrözéses támadás (reflection attack)** segítségével. Gyakorlatilag, Trudy betörhet, ha lehetséges egyszerre több viszonyt létrehozni Bobbal. Ez lehet a helyzet, például, ha Bob egy bank, amely kész több szimultán, a pénzkiadó automatáktól jövő hívás fogadására.

Trudy visszatükrözéses támadása a 7.14. ábrán látható. Először is Trudy Aliznak adja ki magát, és elküldi R_T-t. Bob szokás szerint válaszol saját kihívásával, R_B-vel. Most Trudy tehetetlen. Mit csinálhatna? Nem tudja $K_{AB}(R_B)$-t.

Kezdeményezhet egy másik kapcsolatot a 3-as üzenettel, amiben elküldheti azt az R_B-t, mint kihívást, amit a 2-es üzenetben kapott. Bob gyanútlanul titkosít, és visszaküldi $K_{AB}(R_B)$-t a 4-es üzenetben. Ezzel Trudy megkapta a hiányzó információt, és be tudja fejezi az első kapcsolat felépítését, a másodikat pedig megszakítja. Bob meg van győződve róla, hogy Trudy Aliz, és mikor a bankszámla egyenlegét lekérdezi, szó nélkül megadja a választ. Nem kételkedik akkor sem, mikor arra kéri, hogy mindent utaljon át egy titkos svájci bankszámlára.



7.14. ábra. *Visszatükrözéses támadás*

A történet tanulsága:

Egy korrekt hitelességvizsgáló protokoll készítése nehezebb, mint amilyennek látszik.

Sok esetben segít a következő három általános szabály:

1. A kezdeményező igazolja magát előbb, mint a fogadó. Ebben a példában Bob fontos információt ad, mielőtt Trudy egyáltalán igazolná kilétét.

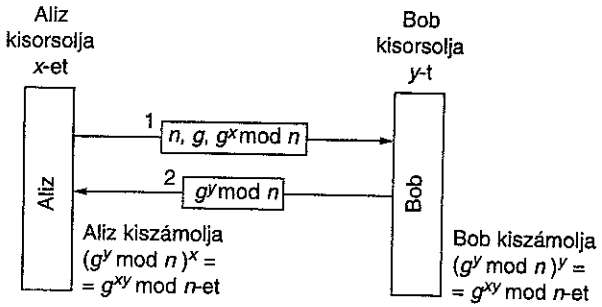
2. A kezdeményező és a fogadó különböző kulcsokat használjon a hitelesítéshez, még akkor is, ha ehhez két megosztott kulcs K_{AB} és K'_{AB} kell.

3. A kezdeményező és a fogadó különböző halmazokból válassza a kihívást. Például a kezdeményezőnek páros, a fogadónak pedig páratlan számokat kelljen használnia.

Mindhárom szabály megsértése esetünkben katasztrofális eredményre vezetett. Vegyük észre, hogy az első (az 5 üzenetes) hitelességvizsgáló protokoll megköveteli Aliztól, hogy elsőként igazolja magát, tehát az ellen nem lehet a visszatükrözéses támadást alkalmazni.

Osztott kulcs létesítése: A Diffie–Hellman-féle kulcscsere

Eddig feltételeztük, hogy Aliznak és Bobnak van egy osztott titkos kulcsa. Mi van, ha mégsincs? Hogyan beszélhetnek meg egyet? Egy lehetséges módszer az lenne, ha Aliz felhívná Bobot telefonon, és megmondaná neki a sajátját, de ő valószínűleg azt mondaná: „Honnan tudjam, hogy tényleg Aliz vagy és nem Trudy?" Vagy megbeszélhetnének egy találkozót, ahova mindketten elviszik az útlevelüket, a jogosítványukat, és három közismert hitelkártyát, de mivel elfoglalt emberek, valószínűleg hónapokba telne, mire mindkettejüknek megfelelő időpontot találnának. Hihetetlenül hangzik, de szerencsére létezik egy megoldás, mely segítségével vadidegenek is megbeszélhetnek



7.15. ábra. *Diffie–Hellman-féle kulcscsere*

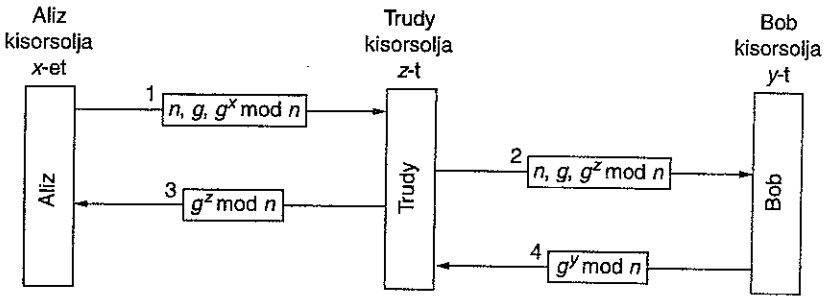
egy osztott titkos kulcsot fényes nappal, annak ellenére, hogy Trudy minden üzenetet gondosan rögzít.

A protokollnak, amely lehetővé teszi, hogy idegenek is megbeszélhessenek egy osztott titkos kulcsot **Diffie–Hellman-féle kulcscsere (Diffie–Hellman key exchange)** a neve (Diffie–Hellman, 1976), és a következőképpen működik. Aliznak és Bobnak meg kell egyeznie két nagy prímszámban, n-ben és g-ben, ahol $(n-1)/2$ is prím, és néhány követelmény teljesül g-re is. Ezek a számok nyilvánosak lehetnek, azaz bármelyikük választhat egy n-et és g-t, majd nyíltan elküldheti a másiknak. Ezután Aliz kisorsol egy nagy (mondjuk 512-bites) számot, x-et, és ezt titokban tartja. Hasonlóképpen Bob is kisorsol egy nagy titkos számot, y-t.

Aliz kezdeményezi a kulcscsere protokollt azzal, hogy elküldi Bobnak egy üzenetben $(n, g, g^x \bmod n)$-et (l. 7.15. ábra). Bob válaszol Aliznak, és elküldi $g^y \bmod n$-et. Aliz a kapott számot az x-edik hatványra emeli, így megkapja $(g^y \bmod n)^x$-t. Bob hasonló módon előállítja $(g^x \bmod n)^y$-t. A modulusos aritmetika szabályai szerint mindkét kifejezés megegyezik $(g^{xy} \bmod n)$-el. Íme Aliz és Bob ezzel megbeszélték megosztott kulcsukat, $g^{xy} \bmod n$-et.

Természetesen Trudy mindkét üzenetet látta. Ismeri tehát g-t és n-et az 1. üzenetből. Ha ki tudná számolni x-et és y-t, akkor meghatározhatná a titkos kulcsot. A probléma az, hogy $g^x \bmod$ n-ből nem tudja meghatározni x-et. Nem ismert olyan használható algoritmus, amely nagy prímszám modulos diszkrét logaritmusát állítja elő. A fenti példa konkretizálása érdekében az n=47 és g=3 (gyakorlatban használhatatlan) számokat használjuk. Aliz x=8-at sorsol, míg Bob y=10-et. Ezeket mindketten titokban tartják. Aliz üzenete Bobnak (47, 3, 28), mert $3^8 \bmod 47=28$. Bob üzenete Aliznak (17). Aliz kiszámolja $17^8 \bmod 47=4$-et. Bob kiszámolja $28^{10} \bmod 47=4$-et. Aliz és Bob függetlenül megállapította, hogy a titkos kulcs most 4. Trudynak meg kell oldani a $3^x \bmod 47=28$ egyenletet, ami kimerítő kereséssel valósítható meg kis számokra, de nem megy, ha a számok mind százbites nagyságrendűek. Minden eddig ismert algoritmus egyszerűen túl sok időt venne igénybe, még egy nagy teljesítményű párhuzamos szuperszámítógéppel is.

A Diffie–Hellman-féle kulcscsere algoritmus eleganciájának ellenére, felmerül egy probléma: honnan tudja Bob, mikor megkapja a (47, 3, 28) számhármast, hogy ez tényleg Aliztól származik-e, és nem Trudytól. Nincs rá mód, hogy megbizonyosodjon



7.16. ábra. *Élő-lánc támadás*

efelől. Sajnos Trudy kihasználhatja ezt a tényt, hogy megtévessze mind Alizt, mind Bobot, amint azt a 7.16. ábra mutatja. Miközben Aliz és Bob külön-külön kisorsolja *x*-et és *y*-t, addig Trudy is kiválasztja saját véletlen számát, *z*-t. Aliz elküldi az 1-es üzenetet, amit Bobnak szán. Trudy ezt elfogja, és egy 2-es üzenetet küld Bobnak, a helyes *g* és *n* számokkal (amelyek amúgy is nyilvánosak), de *x* helyett saját véletlen számával, *z*-vel. Ezenkívül a 3-as üzenetben visszaüzen Aliznak. Később Bob elküldi a 4-es üzenetet Aliznak, amit Trudy ismét elfog.

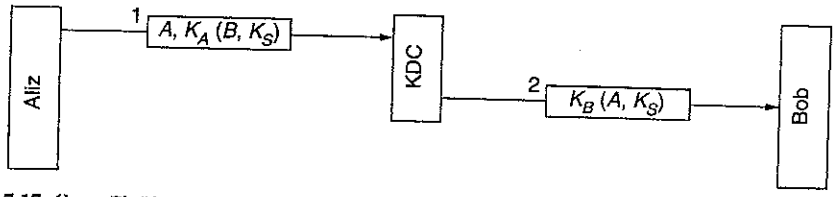
Ekkor mindenki alkalmazza a modulos aritmetikát. Aliz titkos kulcsként g^{xz} mod n-et számol, és Trudy is ezt kapja (Alizzal történő üzenetváltásokhoz). Bob eredménye g^{yz} mod n, és Trudy is ezt kapja (Bobbal való üzenetváltásokhoz). Aliz azt hiszi, hogy Bobbal beszél, és létrehoz egy viszonykulcsot (Trudyval). Bob is így tesz. Trudy elfog és tárol, vagy tetszés szerint módosít minden üzenet, amit Aliz a titkosított viszonyon küld, majd (ha akarja) továbbküldi Bobnak. A másik irányban is hasonló a helyzet. Trudy minden üzenetet lát, és tetszés szerint módosíthat, miközben Aliz és Bob mindketten abban az illúzióban vannak, hogy egy biztonságos csatorna van köztük. Ez a fajta támadás **élő-lánc támadás (bucket brigade attack)** néven ismert, mert némileg emlékeztet a hajdani önkéntes tűzoltókra, akik élő láncot alkottak a tűzoltókocsitól a tűzig, és úgy adogatták egymásnak a vödröket. Másik ismert neve a **közbeékelődéses támadás ((wo)man-in-the-middle),** ami nem tévesztendő össze a blokk-kódolók ellen alkalmazott középen találkozás támadással. Szerencsére ennél összetettebb algoritmusok ki tudják védeni ezt a támadást.

Hitelességvizsgálat kulcsszétosztó központ alkalmazásával

Egy idegennel megosztott titok megbeszélése kis híján sikerült már, de tulajdonképpen mégsem. Ha jól belegondolunk, nem is biztos, hogy érdemes. Ezzel a módszerrel, ha *n* partnerrel tartjuk a kapcsolatot *n* kulcsot kell tárolnunk. Egy népszerű embernek a kulcsok karbantartása komoly terhet jelenthet, főleg, ha minden kulcsot különálló műanyag chip-kártyán kell tárolnia.

Egy másik megközelítés, ha bevezetünk egy megbízható kulcsszétosztó központot (Key Distribution Center – KDC). Ebben a modellben minden felhasználónak csak egy a KDC-vel megosztott kulcsa van. A hitelességvizsgálat, és a kulcskarbantartás így a KDC-n keresztül történik. A legegyszerűbb két felhasználót és egy megbízható KDC-t tartalmazó hitelességvizsgáló protokoll a 7.17. ábrán látható.

A protokoll alapötlete egyszerű: Aliz kisorsol egy viszonykulcsot, K_S-t, és megüzeni a KDC-nek, hogy K_S-t használva Bobbal szeretne beszélni. Ez az üzenet titkosítva van azzal a K_A titkos kulccsal, amit Aliz (csak) a KDC-vel oszt meg. A KDC visszakódolja az üzenetet, és kiveszi belőle Bob azonosítóját, és a viszonykulcsot. Ezután létrehoz egy új üzenetet, ami tartalmazza Aliz azonosítóját és a kapcsolat kulcsot, majd elküldi Bobnak. A titkosítás a Bob és a KDC között megosztott K_B kulccsal történik. Bob az üzenetet visszakódolva megtudja a viszonykulcsot, és azt, hogy Aliz szeretne beszélni vele.



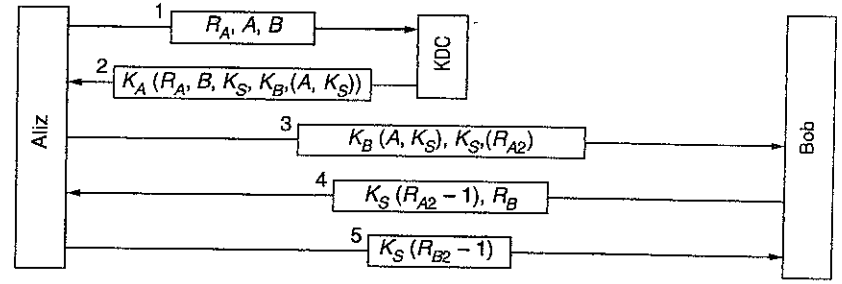
7.17. ábra. *Első kísérlet KDC-t használó hitelességvizsgáló protokollra*

A hitelességvizsgálat itt ingyen történik. A KDC tudja, hogy az 1-es üzenet biztosan Aliztól származik, hiszen azt más nem tudta volna titkosítani Aliz titkos kulcsával. Hasonlóan Bob tudja, hogy a 2-es üzenet biztosan a KDC-től jött, amiben megbízik, hiszen más nem tudja az ő titkos kulcsát.

Sajnos ennek a protokollnak van egy súlyos hibája. Trudy pénzszűkében van, így keres valami legális Aliz által igényelt szolgáltatást, kecsegtető ajánlatot tesz, és megkapja a munkát. A munka végeztével Trudy udvariasan megkéri Alizt, hogy banki átutalással fizessen. Aliz tehát megbeszél egy kapcsolat kulcsot bankárával, Bobbal. Ezután egy üzenetben arra kéri Bobot, hogy utaljon át pénzt Trudy számlájára. Eközben Trudy visszatér régi énjéhez, és a hálózatot kémleli. Felveszi mind a 7.17. ábrán látható 2-es üzenetet, mind az azt követő pénz átutalási kérelmet. Később visszajátssza mindkettőt Bobnak. Bob megkapja őket, és azt hiszi: „Aliz biztosan megint felbérelte Trudyt. Ezek szerint érti a dolgát." Bob ismét átutalja a pénzösszeget Aliz számlájáról Trudyéra. Valamikor az 50-edik üzenetpáros vétele környékén Bob kirohan irodájából, hogy megkeresse Trudyt, és felajánljon neki egy nagyobb hitelt, hogy fejleszthesse a nyilvánvalóan sikeres vállalkozását. Ezt a problémát nevezik **visszajátszásos támadásnak (replay attack).**

Jó pár megoldás van a visszajátszásos támadás kivédésére. Egyik lehetőség, hogy minden üzenetet időbélyeggel látunk el. Ha valaki egy idejemúlt üzenetet kap, eldobhatja azt. Ezzel a megközelítéssel az gond, hogy az órák sohasem teljesen szinkronizáltak egy hálózatban, ezért az időbélyeg egy bizonyos időtartamig érvényes kell legyen. Trudy ebben az időtartamban még sikeresen visszajátszhatja az üzenetet.

A második megoldás az egyszer használatos, egyedi üzenetszám, beillesztése az üzenetekbe, amit általában **egyszer-elküldött, egyedi üzenetszámnak (nonce)** neveznek. Így minden résztvevőnek tárolnia kell az elhasznált üzenetszámokat, és kidobni a régi üzenetszámmal érkező leveleket. Az üzenetszámokat azonban örökre meg kell



7.18. ábra. *Needham–Schroeder-hitelességvizsgáló protokoll*

őrizni, nehogy Trudy egy 5 éves üzenetet próbáljon visszajátszani. Továbbá, ha egy gép összeomlik, és elveszti az üzenetszám listáját ismét sebezhetővé válik a visszajátszásos támadással szemben. Az időbélyegek és az üzenetszámok kombinálhatók úgy, hogy csak korlátos ideig kelljen az üzenetszámokat tárolni, természetesen azonban így sokkal komplikáltabb lesz a protokoll.

Egy kifinomultabb megközelítése a hitelességvizsgálatnak a többutas kihívás-válasz protokoll. Melynek egy közismert változata a Needham–Schroeder-hitelességvizsgáló protokoll (Needham–Schroeder, 1978), ennek egy változatát mutatja a 7.18. ábra.

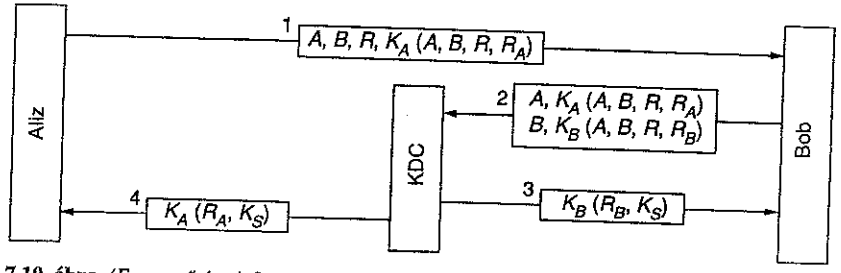
A protokoll első lépésében Aliz megüzeni a KDC-nek, hogy Bobbal szeretne beszélni. Ez az üzenet tartalmaz egy nagy véletlen számot R_A-t, mint egyszer-elküldött, egyedi üzenetszámot. A KDC visszaküldi a 2-es üzenetben Aliz véletlen számát R_A-t, egy viszonykulcsot és egy jegyet, amit elküldhet Bobnak. R_A azért szükséges, hogy Aliz biztosítva legyen afelől, hogy az üzenet friss, nem pedig egy visszajátszás. Bob azonosítója szintén mellékelve van, arra az esetre, ha Trudynak az az ötlete támadna, hogy kicseréli az 1-es üzenetben B-t a sajátjára azért, hogy a KDC a 2-es üzenetben a jegyet K_T-vel titkosítsa és nem K_B-vel. A K_B-vel titkosított jegyet a titkosított üzenet tartalmazza, nehogy Trudy az Alizhoz vezető visszaúton kicserélhesse valamivel.

Aliz ezután elküldi a jegyet Bobnak, egy új véletlen szám, R_{A2}, kíséretében, amit K_S-sel titkosít. A 4-es üzenetben Bob visszaküldi $K_S(R_{A2}-1)$-et, hogy biztosítsa Aliz-t arról, hogy az igazi Bobbal beszél. $K_S(R_{A2})$ visszaküldése nem lenne jó, hiszen azt Trudy egyszerűen kilophatja a 3-as üzenetből.

A 4-es üzenet kézhezvétele után Aliz biztos benne, hogy Bobbal beszél, valamint, hogy eddig nem történhetett visszajátszás. Hiszen pár ezredmásodperccel ezelőtt generálta R_{A2}-t. Az 5-ös üzenet célja, hogy Bob is meggyőződjön, róla, hogy tényleg Alizzal beszél, és itt sem történt visszajátszás. Azzal, hogy mindkét fél generált egy kihívást és válaszolt is egyre, a visszajátszásos támadás lehetősége kizárható.

Annak ellenére, hogy ez a protokoll elég megbízhatónak látszik, mégis van egy gyenge pontja. Ha Trudy egyszer megszerez egy régi viszonykulcsot kódolatlan formában, akkor a 3-as üzenet visszajátszásával kezdeményezhet egy új viszonyt Bobbal, és elhitetheti vele, hogy ő Aliz (Denning és Sacco 1981). Ezúttal kifoszthatja Aliz bankszámláját, anélkül, hogy törvényesen valaha is dolgozott volna neki.

Needham és Schroeder később publikáltak egy protokollt, amely megoldja ezt a problémát (Needham és Schroeder 1987). Ugyanannak a folyóiratnak ugyanabban a



7.19. ábra. *(Egyszerűsített) Otway–Rees-hitelességvizsgáló protokoll*

számában Otway és Rees (1987) szintén ismertetett egy rövidebb protokollt, amely megoldja a problémát. A 7.19. ábrán egy kis mértékben módosított változata látható az Otway–Rees protokollnak.

Az Otway–Rees-protokollban Aliz először előállít egy véletlen számpárost, R-et, ami egy általános azonosító, és R_A-t, amit Aliz majd kihívásként használ Bob felé. Mikor Bob megkapja ezt az üzenetet létrehoz egy új üzenetet Aliz üzeneténck titkosított részéből, és egy hasonlót saját magától. A K_A-val és K_B-vel titkosított mindkét rész azonosítja Alizt, és Bobot, tartalmazza az általános azonosítót, és tartalmaz egy kihívást.

A KDC leellenőrzi, hogy R mindkét részben ugyanaz-e. Lehet, hogy nem, hiszen Trudy megváltoztathatta R-et az 1-es üzenetben, vagy kicserélhette a 2-es üzenet egy részét. Amennyiben az R-ek megegyeznek, a KDC elhiszi, hogy a Bob által üzent kérés érvényes. Ezután generál egy viszonykulcsot, és két példányban titkosítja azt, egyszer Aliznak, egyszer pedig Bobnak. Mindegyik üzenet tartalmazza a fogadó véletlen számát biztosítékul arra nézve, hogy nem Trudy hozta létre azokat. Ezek után mind Aliz, mind Bob birtokában van ugyanannak a viszonykulcsnak, és elkezdhetnek kommunikálni. Mikor először cserélnek adatot tartalmazó üzenetet mindketten láthatják, hogy a másiknak is birtokában van K_S tökéletesen azonos másolata, ezzel a hitelességvizsgálat befejeződött.

Hitelességvizsgálat Kerberos alkalmazásával

Sok valódi rendszerben a **Kerberos** hitelességvizsgáló protokollt használják, ami a Needham–Schoeder egyik variációján alapul. Egy sokfejű kutyaszörnyetegről kapta nevét, amely a görög mitológiában Hadész bejáratát őrizte (feltételezhetően a nem kívánatosak távoltartása végett). A Kerberost az M.I.T.-n fejlesztették ki azért, hogy a munkaállomások felhasználói biztonságosan hozzáférhessenek a hálózati erőforrásokhoz. Legfontosabb különbsége a Needham–Schroeder-hez képest az a feltételezés, hogy az órák meglehetősen jól szinkronizáltak. A protokoll sok változtatáson ment keresztül. Az iparban a V4 változat a legelterjedtebb, ezért ezt ismertetjük. Utána néhány szót ejtünk majd utódjáról, a V5-ről. További információk tekintetében lásd (Neumann és Ts'o, 1994; valamint Steiner és mások, 1988).

A Kerberos három további szervert is igénybe vesz az Aliz (egy kliens munkaállomás) mellett:

Hitelességvizsgáló szerver (Authentication Server – AS): ellenőrzi a felhasználót belépéskor
Jegyadó szerver (Ticket-Granting Server – TGS): „személyazonosság igazoló jegyeket" ad
Bob, a szerver: elvégzi az Aliz által kért munkát

A hitelességvizsgáló szerver (AS) a KDC-hez hasonlóan minden felhasználóhoz, fenntart egy titkos jelszót. A jegyadó (TGS) feladata, hogy olyan jegyeket adjon, amelyek biztosítják a valódi szervereket, hogy a jegytulajdonos tényleg az, akinek mondja magát.

Egy viszony kezdeményezéséhez Aliz begépeli nevét egy tetszőleges nyilvános munkaállomáson. A munkaállomás nyílt szövegként kódolatlanul elküldi nevét az AS-hez (l. 7.20. ábra). Válaszul kap egy viszonykulcsot és egy jegyet, $K_{TGS}(A, K_S)$, a TGS számára. Ezek az adatok egy közös, Aliz titkos kulcsával titkosított csomagban vannak, hogy csak Aliz tudja őket visszakódolni. A munkaállomás csak azután kérdezi meg Aliz jelszavát, amikor a 2-es üzenet megérkezett. A jelszóból ezután generálja K_A-t, hogy visszakódolhassa a 2-es üzenetet, és így megkapja belőle a kapcsolat kulcsot, valamint a TGS jegyet. Ezután a munkaállomás felülírja Aliz jelszavát, hogy az legfeljebb néhány ezredmásodpercig legyen a munkaállomásban. Ha Trudy megpróbál belépni Aliz helyett, az általa begépelt jelszó nem lesz jó, és ezt a munkaállomás észreveszi, hiszen a 2-es üzenet szerkezetileg helytelen lesz.

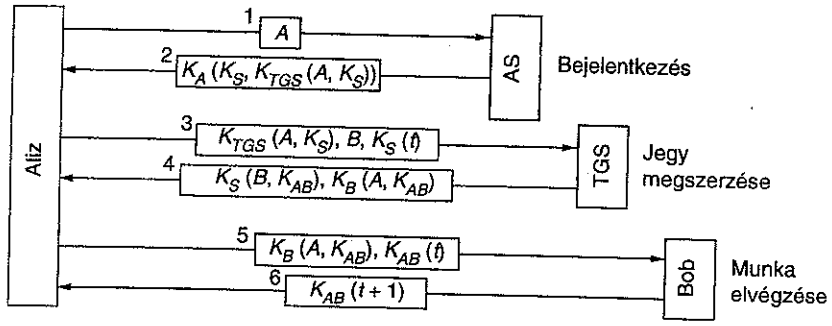
Aliz, miután belépett, megmondhatja a munkaállomásnak, hogy Bobbal, a fájlszerverrel szeretne kapcsolatot teremteni. A munkaállomás ezután a TGS-nek küldött 3-as üzenetben kér egy Bobbal való viszonyában használható jegyet. A kulcselem itt a $K_{TGS}(A, K_S)$, ami a TGS titkos kulcsával van titkosítva, és arra szolgál, hogy bizonyítsa, hogy a küldő tényleg Aliz.

A TGS azzal válaszol, hogy létrehoz Aliznak egy Bobbal használható viszonykulcsot, K_{AB}-t. Ennek kétféle változata megy vissza. Az első csak K_S-sel van titkosítva, hogy Aliz el tudja olvasni. A második Bob kulcsával, K_B-vel, van titkosítva, hogy Bob el tudja olvasni.

Trudy lemásolhatja a 3-as üzenetet, és megpróbálhatja újra használni, de ezt meghiúsítja az üzenethez kapcsolódó titkosított t időbélyeg. Trudy nem tudja lecserélni az időbélyeget egy frissebbel, mert nem ismeri K_S-t, a viszonykulcsot, amelyet Aliz a TGS-el való beszédhez használ. Még, ha Trudy gyorsan visszajátssza is a 3-as üzenetet, akkor is csak egy újabb másolatát szerezhetné meg a 4-es üzenetnek, amit már elsőre sem volt képes visszakódolni, és másodszorra sem fog tudni.

Most már Aliz elküldheti K_{AB}-t Bobnak, hogy létrehozzon egy viszonyt. Ez az üzenetváltás is időbélyeget tartalmaz. A válasz biztosítja Alizt afelől, hogy tényleg Bobbal és nem Trudyval beszél.

Az üzenetváltások sorozatának lezajlása után Aliz K_{AB} védelme alatt kommunikálhat Bobbal. Ha később úgy dönt, hogy egy másik szerverrel, Carollal is beszélnie kell, egyszerűen megismétli a 3-as üzenetet, azzal a különbséggel, hogy ezúttal B helyett



7.20. ábra. *A Kerberos V4 működése*

C-t küld. A TGS azonnal válaszol, megküldve a K_C-vel titkosított jegyet, amit Aliz elküldhet Carolnak, akit ez biztosít arról, hogy az Aliztól jött.

Ennek a sok munkának az az értelme, hogy most már Aliz minden hálózaton elérhető szerverhez biztonságosan hozzákapcsolódhat, és a jelszavát sohasem kell a hálózaton keresztül küldenie. Sőt, annak csak néhány ezredmásodpercre kell a saját munkaállomásában lennie. Vegyük azonban észre, hogy minden szerver saját maga végzi a jogosultságvizsgálatot. Mikor Aliz megmutatja a jegyét Bobnak, ez Bobot csupán arról győzi meg, hogy azt ki küldte. Pontosabban szólva, hogy Aliz mit tehet, azt Bob dönti el.

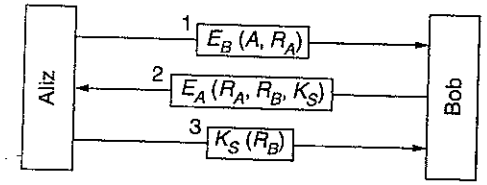
Minthogy a Kerberos fejlesztői nem várták, hogy az egész világ egyetlen hitelességvizsgáló szerverben bízzon meg, ezért gondoskodtak róla, hogy több **birodalom (realm)** létezhessen külön-külön saját AS-sel, és TGS-sel. Egy távoli szerverhez úgy szerezhet jegyet Aliz, ha kér egy jegyet saját TGS-étől, amit a távoli TGS is elfogad. Ha a távoli TGS-t számontartja a helyi TGS (hasonlóan a helyi szerverekhez), akkor a helyi TGS ad Aliznak egy jegyet, amely a távoli TGS-re érvényes. Ezután elvégezheti dolgát a távoli birodalomban is, például szerezhet jegyet ottani szerverekhez. Vegyük észre azonban, hogy ahhoz, hogy két különböző birodalombeli fél együttműködhessen, mindkettőnek meg kell bíznia a másik TGS-ében.

A Kerberos V5 díszesebb a V4-nél, és több benne a kommunikációból adódó többletfogalom. Használja az OSI ASN.1-et (Abstract Syntax Notation 1) is az adat típusok leírására, valamint apró változtatások történtek a protokollban is. Továbbá a jegyek élettartama hosszabb lett, lehetőség van benne a jegyek megújítására, és képes későbbre keltezett jegyek kiadására is. Ezek mellett, legalábbis elméletben, nem DES függő, mint a V4 volt, és a több-birodalmúságot is támogatja.

Hitelességvizsgálat nyilvános kulcsú titkosítással

A kölcsönös hitelességvizsgálat megoldható nyilvános kulcsú titkosítás használatával is. Kezdetnek feltételezzük, hogy Aliz és Bob már ismerik egymás nyilvános kulcsát (nem triviálisan előálló helyzet). Viszonyt szeretnének létesíteni, majd a titkos kulcsú titkosítást szeretnék használni, mivel ez általában 100-szor vagy 1000-szer gyorsabb, mint a nyilvános kulcsú titkosítás. A kezdeti üzenetváltás feladata ezek után annyi, hogy kölcsönösen elvégezzék a hitelességvizsgálatot, és megegyezzenek egy osztott titkos viszonykulcsban.

Az ilyen típusú indítás sokféleképpen megoldható. Egy tipikus változata a 7.21. áb-



7.21. ábra. *Kölcsönös hitelességvizsgálat nyilvános kulcsú titkosítással*

rán látható. Itt Aliz először titkosítja azonosítóját, és egy R_A véletlen számot. Bob nyilvános (vagy más néven titkosító) kulcsával, E_B-vel. Mikor Bob megkapja ezt az üzenetet, még nem tudja, hogy Aliztól vagy Trudy-tól jött-e az üzenet, de együttműködik, és visszaküld Aliznak egy üzenetet, ami tartalmazza az Aliz-féle R_A-t, saját véletlen számát, R_B-t, és egy javasolt viszonykulcsot, K_S-t.

Mikor Aliz megkapja a 2-es üzenetet, visszakódolja azt saját egyéni kulcsának segítségével. Megtalálja benne R_A-t, ami örömmel tölti el. Az üzenet csakis Bobtól jöhet, hiszen Trudynak nem áll módjában megállapítani R_A-t. Továbbá biztosan friss, hiszen épp az előbb küldte el R_A-t Bobnak. Aliz a 3-as üzenet elküldésével beleegyezik a viszonyba. Bob azonnal tudja, hogy Aliz megkapta a 2-es üzenetet, és leellenőrizte R_A-t, amint meglátja R_B-t, az imént generált viszonykulccsal, K_S-sel titkosítva.

Mit tehet Trudy, hogy aláaknázza ezt a protokollt? Fabrikálhat egy 1-es üzenetet, és beugrathatja Bobot, hogy próbára tegye Alizt, de Aliz látja R_A-t, amit nem ő küldött, és nem folytatja tovább. Trudy nem tudja meggyőzőre kovácsolni a 3-as üzenetet, mert nem ismeri R_B-t vagy K_S-t és nem tudja megállapítani azokat, Aliz egyéni kulcsa nélkül. Nincs szerencséje.

Ennek a protokollnak azonban mégis van egy gyengéje: feltételezi, hogy Aliz és Bob már ismerik egymás nyilvános kulcsát. Tegyük fel, hogy mégsem így van. Aliz egyszerűen elküldhetné Bobnak a nyilvános kulcsát az 1-es üzenetben, és megkérhetné Bobot, hogy küldje vissza sajátját a következő üzenetben. Ezzel a megközelítéssel az a baj, hogy egy élő-lánc támadás áldozatává válhat. Trudy elfoghatja Aliz Bobnak szóló üzenetét, és visszaküldheti saját nyilvános kulcsát Aliznak. Aliz azt fogja hinni, hogy van egy kulcsa a Bobbal való beszélgetéshez, de valójában a Trudyval való beszélgetéshez van kulcsa. Ezek után Trudy minden üzenetet elolvashat, amit Aliz a Bobnak tulajdonított nyilvános kulccsal titkosít.

A kezdeti nyilvános kulcs csere elkerülhető, ha az összes nyilvános kulcsot egy nyilvános adatbázisban tárolják. Így Aliz és Bob beszerezhetik egymás nyilvános kulcsát az adatbázisból. Sajnos azonban Trudy még mindig keresztülviheti az élő-lánc támadást, azzal, hogy elfogja az adatbázishoz érkező kéréseket, és szimulálja a válaszokat, melyekben saját nyilvános kulcsát küldi el. Hiszen honnan tudhatná Aliz és Bob, hogy a válaszok a valódi adatbázistól jönnek, és nem Trudytól?

Rhivest és Shamir (1984) kiterveltek egy protokollt, ami meghiúsítja Trudy élő-lánc támadását. **Keresztbezáró protokolljukban (interlock protocol)** a nyilvános kulcsok cseréje után Aliz csak az üzenetének felét küldi el Bobnak, mondjuk csak a (titkosítás után kapott) páros biteket. Bob ezután válaszol saját páros bitjeivel. Miután Aliz megkapta Bob páros bitjeit, elküldi saját páratlan bitjeit, ezt követően Bob is így tesz.

A trükk az, hogy amikor Trudy megkapja Aliz páros bitjeit, nem tudja visszakódolni az üzenetet, annak ellenére, hogy ismeri az egyéni kulcsot. Következésképpen nem is tudja újratitkosítani a páros biteket Bob nyilvános kulcsával. Amennyiben zagyvaságot küld Bobnak, a protokoll folytatódik, de Bob hamarosan észreveszi, hogy a teljesen összeállított üzenet nem értelmes, és rájön, hogy átverték.

7.1.6. Digitális aláírások (Digital Signatures)

Sok hivatalos, pénzügyi és más dokumentum hitelességvizsgálatát eldönti egy kézzel írott aláírás jelenléte vagy hiánya. A fénymásolatok ilyen szempontból szóba sem jönnek. A papírból és tintából álló dokumentumok fizikailag vándorolnak, a számítógépesített levelező rendszerekben azonban megoldást kell találni erre a problémára.

Nehéz feladat a kézi aláírások helyettesítésének problémájára megoldást találni. Alapvetően azt kell elérni, hogy az egyik fél elküldhessen a másiknak egy „aláírt" üzenetet úgy, hogy:

1. A fogadó ellenőrizhesse a feladó valódiságát.

2. A küldő később ne tagadhassa le az üzenet tartalmát.

3. A fogadó saját maga ne rakhassa össze az üzenetet.

Az első követelmény például egy pénzügyi rendszerben szükséges. Mikor egy ügyfél számítógépe utasítja a bank számítógépét, hogy vegyen egy tonna aranyat, annak meg kell tudnia állapítani, hogy a számítógép, amely az utasítást adta, tényleg annak a cégnek a tulajdona, amelyiknek a bankszámláját terhelni kell.

A második követelmény azért szükséges, nehogy a bankot kijátsszák. Tegyük fel, hogy a bank megveszi az egy tonna aranyat, és közvetlenül utána az arany ára drasztikusan leesik. Egy nem becsületes ügyfél esetleg beperelheti a bankot, mondván, hogy sohasem adott utasítást arany vásárlására. Amikor a bank a bíróságon bemutatja az üzenetet az ügyfél letagadja, hogy ő küldte volna.

A harmadik követelmény abban az esetben fontos, ha az arany ára megugrik, és a bank megpróbál egy olyan üzenetet konstruálni, amelyben az ügyfél egy tonna arany helyett egy rudat rendelt.

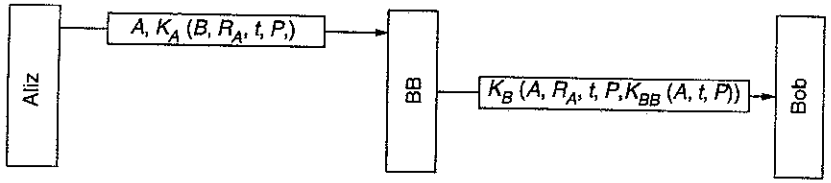
Titkos kulcsú aláírások

A digitális aláírás egyik megközelítésében adva van egy központi hitelességvizsgáló szerv, amely mindent tud, és amelyben mindenki megbízik, nevezzük mondjuk Nagy Testvérnek (Big Brother – *BB*). Ezután minden felhasználó választ egy titkos kulcsot, és sajátkezűleg elviszi *BB* irodájába. Ily módon csak Aliz és *BB* ismeri Aliz titkos kulcsát, K_A-t és így tovább.

Ha Aliz egy aláírt, kódolatlan üzenetet, P-t, szeretne küldeni bankárjának Bobnak, akkor generálja $K_A(B, R_A, t, P)$-t, és ahogy az a 7.22. ábrán látható elküldi *BB*-nek. *BB* látja, hogy az üzenet Aliztól jött, visszakódolja, és az ábrán látható módon küld egy üzenetet Bobnak. A Bob fele menő üzenet tartalmazza Aliz kódolatlan üzenetét és az aláírt üzenetet $K_{BB}(A, t, P)$-t, ahol t egy időbélyeg. Bob ezután végrehajtja Aliz kérését.

Mi történik, ha Aliz utóbb letagadja az üzenetet? Először is mindenki beperel mindenkit (legalábbis az Egyesült Államokban). Amikor az ügy végül is bíróságra kerül, és Aliz nyomatékosan tagadja, hogy a szóban forgó levelet ő küldte volna Bobnak, a

bíró meg fogja kérdezni Bobot, hogy mivel tudja bizonyítani, hogy az üzenet Aliztól, és nem Trudytól jött. Bob először is rámutat, hogy *BB* nem fogad el üzenetet Aliztól, ha az nem K_A-val van titkosítva, tehát Trudynak nincs lehetősége egy hamis levelet küldeni *BB*-nek Aliz nevében.



7.22. ábra. *Digitális aláírás a Nagy Testvér közreműködésével*

Bob ezek után drámai módon felmutatja az A bizonyítékot, $K_{BB}(A, t, P)$-t. Bob azt állítja, hogy ez egy üzenet *BB* aláírásával, ami bizonyítja, hogy Aliz elküldte *P*-t Bobnak. A bíró tehát megkéri *BB*-t (akiben mindenki megbízik), hogy kódolja vissza az A bizonyítékot. Mikor *BB* alátámasztja, hogy Bobnak igaza van, a bíró a pert Bob javára dönti el. Az ügy ezzel véget ért.

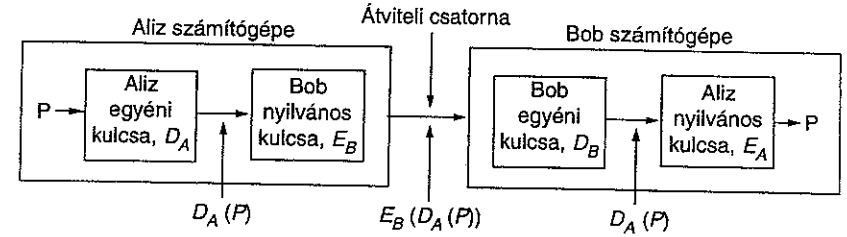
Egy lehetséges probléma a 7.22. ábrán látható aláírás protokollal az, hogy Trudy mindkét üzenetet képes visszajátszani. Ennek a problémának a minimalizálására minden időbélyegekkel látnak el. Ezenkívül Bob megvizsgálhatja az összes nemrég érkezett üzenetet, hogy R_A szerepelt-e bennük. Ha igen, akkor az üzenet, mint visszajátszás eldobható. Vegyük észre, hogy Bob visszautasít minden elég régi üzenetet az időbélyegek miatt. A gyors visszajátszás ellen Bob úgy védekezik, hogy ellenőrzi minden beérkező üzenetben R_A-t, hogy lássa, nem érkezett-e az utóbbi egy órában hasonló üzenet. Ha nem, akkor nyugodtan feltételezheti, hogy a kérés vadonatúj.

Nyilvános kulcsú aláírások

A digitális aláírásoknál egy strukturális probléma, hogy titkos kulcsú titkosítás használata mellett mindenkinek egységesen meg kell bíznia a Nagy Testvérben. Továbbá, hogy a Nagy Testvér minden aláírt levelet elolvashat. A leglogikusabb választás, ha egy állami szerv vagy bankok vagy ügyvédek üzemeltetik a Nagy Testvér szervert. Ezek a szervezetek nem keltenek minden állampolgárban bizalmat. Ezenkívül jó lenne, ha a dokumentumok aláírásához nem kellene egy hitelességvizsgálóban megbízni.

Szerencsére ezen a téren jelentős eredményt kínálhat a nyilvános kulcsú titkosítás. Tegyük fel, hogy a nyilvános kulcsú titkosítási algoritmusok a szokásos $D(E(P))=P$ tulajdonság mellett szintén rendelkeznek a $E(D(P))=P$ tulajdonsággal. (A feltételezés nem alaptalan, hiszen az RSA rendelkezik ezzel a tulajdonsággal.) Ezt feltételezve Aliz elküldhet egy aláírt kódolatlan üzenetet Bobnak, ha $E_B(D_A(P))$-t küldi el. Fontos észrevenni, hogy Aliz ismeri mind saját (egyéni) visszakódoló kulcsát D_A-t, mind Bob nyilvános kulcsát E_B-t, tehát képes egy ilyen üzenet megkonstruálására.

Amikor Bob megkapja ezt az üzenetet, átalakítja azt saját titkos kulcsával, és eredményül megkapja $D_A(P)$-t, ahogy azt a 7.23. ábra is mutatja. Ezt a szöveget egy biztonságos helyen tárolja, majd visszakódolja E_A segítségével, hogy megkapja az eredeti kódolatlan üzenetet.



7.23. ábra. *Digitális aláírások nyilvános kulcsú titkosítás használatával*

Annak érdekében, hogy lássuk, hogyan is működik az aláírás, tételezzük fel, hogy Aliz ismételten tagadja, hogy *P*-t ő küldte volna Bobnak. Mikor az ügy a bíróság elé kerül Bob bemutathatja mind *P*-t, mind $D_A(P)$-t. A bíró egyszerűen ellenőrizheti, hogy a Bob-féle, D_A-val titkosított üzenet tényleg érvényes azzal, hogy visszakódolja azt E_A-val. Mivel Bob nem ismeri Aliz egyéni kulcsát, ezért csakis úgy tehetett szert egy, azzal titkosított üzenetre, hogy azt Aliz küldte neki. Amíg Aliz börtönben ül hamis eskü és csalás vádjával, rengeteg ideje lesz érdekes, új nyilvános kulcsú algoritmusok kigondolására.

Annak ellenére, hogy a nyilvános kulcsú titkosítás használata digitális aláírásokhoz egy elegáns módszer, azért vannak problémák, amelyek nem az algoritmus alapjaiból, hanem a futtatási környezetből adódnak. Csak egy a sok közül, hogy Bob csak addig tudja bizonyítani, hogy a levelet Aliz küldte, ameddig D_A titokban van. Ha Aliz felfedi titkos kulcsát, akkor az állítás már nem helytálló, hiszen bárki küldhette az üzenetet, akár maga Bob is.

A probléma akkor jelentkezhet, ha például Bob Aliz tőzsdeügynöke. Aliz utasítja Bobot, hogy vegyen meg egy bizonyos részvényt vagy kötvényt. Közvetlenül vásárlás után az árak drasztikusan leesnek. Aliz, hogy letagadhassa a Bobnak küldött üzenetet, elszalad a rendőrségre azzal, hogy betörtek otthonába, és lemásolták a kulcsát. Attól függően, hogy melyik országban él, lehet, hogy Aliz törvényesen felelősségre vonható, de az is lehet, hogy nem, főleg akkor, ha azt állítja, hogy csak órákkal később, a munkából hazaérkezvén fedezte fel a betörést.

Szintén probléma, az aláírási rendszerrel kapcsolatban, ha Aliz úgy dönt, hogy lecseréli egyéni kulcsát, ami teljesen törvényes, és időről időre ajánlott is. Előfordulhat, később egy a fent leírt bírósági ügyben, hogy a bíró visszakódolja $D_A(P)$-t az aktuális E_A-val, majd megállapítja, hogy nem *P* az eredmény. Bob ekkor igen kínosan érezné magát. Következésképpen valami hiteles szerv mégiscsak kell, hogy feljegyezzék a kulcscseréket és azok időpontjait.

Gyakorlatilag bármely nyilvános kulcsú titkosítási algoritmus használható a digitális aláírásokhoz. A de facto hivatalos szabvány az RSA algoritmus. Sok titkosítást használó termék alapul ezen. Mindazonáltal 1991-ben a NIST (National Institute of Standards and Technology) az El Gamal nyilvános kulcsú algoritmus egy variációjá-

nak használatát javasolta új **digitális aláírás szabványukban (Digital Signature Standard – DSS).** Az El Gamal biztonságát nem a nagy számok prímtényezős felbontása, hanem a diszkrét logaritmus számolásának kivitelezhetetlenségéből nyeri.

Ahogyan az lenni szokott, ha kormányzat szeretné a digitális titkosítási szabványokat diktálni, most is nagy felhördülés volt. A DSS-t a következők miatt kritizálták:

1. Túlságosan titkos (az NSA az El Gamalra alapozva fejlesztette ki).

2. Túlságosan új (az El Gamal kimerítő vizsgálata még nem fejeződött be).

3. Túlságosan lassú (10-tól 40-szer lassúbb a digitális aláírások ellenőrzése, mint az RSA-ban).

4. Nem elég biztonságos (fix 512 bit hosszúságú kulcs).

Egy következő átdolgozott kiadásban a 4-es pontot vita tárgyává tették azzal, hogy a kulcsok akár 1024 bitesek is lehetnek. Még nem dőlt el, hogy a DSS beválik-e vagy sem. További részletek tekintetében lásd (Kaufman és mások, 1995; Schneier, 1996; Stinson, 1995).

Üzenet pecsétek (Message digests)

Az aláírások egyik kritikája, hogy gyakran párosítanak két eltérő funkciót: a hitelesítést és a titkosítást. Gyakran csak a hitelességvizsgálatra van szükség, és a titkosításra nem. Mivel a titkosítás lassú, ezért gyakran van igény aláírt kódolatlan dokumentumok küldésére. Az alábbiakban egy olyan hitelesítési módszert mutatunk be, amelyhez nem kell az egész üzenetet titkosítani (De Jonge és Chaum, 1987).

Ez a módszer egy egyirányú hash függvényen alapszik, amely egy tetszőlegesen hosszú szövegből fix hosszúságú bitfüzért generál. Ez a hash függvény, amelyet gyakran üzenet pecsétnek neveznek, három fontos tulajdonsággal bír:

1. Adott P-hez könnyen számolható $MD(P)$.

2. Adott $MD(P)$-hez gyakorlatilag lehetetlen P-t megtalálni.

3. Senki sem képes két különböző üzenetet generálni, amelyekhez ugyanaz az üzenet pecsét tartozik.

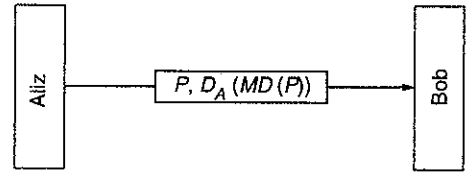
A 3. pont teljesítéséhez a pecsét legalább 128 vagy lehetőleg ennél több-bites kell legyen.

Az üzenet pecsétek kiszámolása egy szöveghez sokkal gyorsabban elvégezhető, mint ugyanazon szöveg kódolása egy nyilvános kulcsú algoritmussal, tehát az üzenet pecsétek használhatók a digitális aláírás algoritmusok gyorsítására. Ahhoz, hogy lássuk ez hogyan is működik, tekintsük ismét a 7.22. ábrát. Ahelyett, hogy *BB* P-t $K_{BB}(A, t, P)$-vel

írná alá, most kiszámolja az üzenet pecsétet, azaz MD-t kiszámolja P-re, aminek eredménye $MD(P)$. *BB* ezek után $K_{BB}(A, t, P)$ helyett becsomagolja $K_{BB}(A, t, MD(P))$-t, mint ötödik elemet a K_B-vel titkosított listába, amelyet Bobnak küld.

Amennyiben vita támad, Bob előveheti mind P-t, mind $K_{BB}(A, t, MD(P))$-t. Miután ezt a Nagy Testvér visszakódolta a bíróságnak, Bob birtokában van a garantáltan eredeti $MD(P)$-nek és az állítólagos P-nek. Bárhogy is van, mivel gyakorlatilag lehetetlen, hogy Bob egy másik üzenetet találjon, aminek szintén ez a pecsétje, a bíró könnyen meggyőzhető afelől, hogy Bob igazat mond. Az üzenet pecsétek ilyen módon történő használata mind a kódolási idő, mind az üzenet átvitel és tárolási költség szempontjából megtakarítást jelent.

Az üzenet pecsétek a nyilvános kulcsú titkosítási rendszerekben is használhatók, amint az a 7.24. ábrán látható. Itt Aliz először kiszámolja az üzenet pecsétet saját szövegéhez. Ezt követően aláírja az üzenet pecsétet, és elküldi Bobnak mind az aláírt pecsétet mind a kódolatlan szöveget. Ha Trudy útközben kicseréli P-t, Bob észreveszi ezt, amint maga is kiszámolja $MD(P)$-t.



7.24. ábra. *Digitális aláírások üzenet pecsétek használatával*

Számos üzenet pecsét függvény látott napvilágot. A legelterjedtebb az MD5 (Rivest, 1992) és az SHA (NIST, 1993). Az **MD5** az ötödik egy hash függvény sorozatban, amelyet Ron Rivest tervezett. Úgy működik, hogy a biteket megfelelően komplikált módon tördeli úgy, hogy a kimeneti bitek mindegyike függ minden bemeneti bittől. Egészen vázlatosan azzal kezdi, hogy az üzenetet kitölti 448 bit hosszúságra (mod 512). Majd hozzáfűzi az eredeti üzenet hosszát, mint egy 64 bites egészet, így a bemenet hossza 512 bit többszöröse lesz. Az utolsó előkészület egy 128 bites puffer feltöltése egy megadott értékkel.

Ezután megkezdődik a számolás. Minden körben egy 512 bites szeletet vesz a bemenetből és alaposan összekeveri a 128 bites pufferrel. Ráadásul belekevernek egy, a szinusz függvény értékeiből konstruált táblázatot is. Nem azért használja a szinusz, egy ismert függvény értékeit, mert ez véletlenszerűbb, mint egy véletlenszám-generátor, hanem hogy elkerülje a gyanút, hogy a tervező egy ügyes kiskaput épített be, amelyen csak ő nyithat be. Az, hogy az IBM nem hozta nyilvánosságra a DES-beli S-dobozok alapelvét, rengeteg kiskapukkal foglalkozó spekulációkhoz vezetett. Minden bemeneti blokkon négyszer hajtja végre a kört. Ez a folyamat ismétlődik addig, amíg minden bemeneti blokk sorra nem került. Végül a 128 bites puffer alkotja az üzenet pecsétet. Ez az algoritmus a 32 bites gépekre írható szoftverekre optimalizált. Ennek eredményeképpen nem biztos, hogy a jövőbeli nagy sebességű hálózatokhoz is elég gyors lesz (Touch, 1995).

A másik jelentős üzenet pecsét függvény az **SHA (Secure Hash Algorithm – biztonságos hash algoritmus)**, amit az NSA fejlesztett ki, és a NIST áldását adta rá. Csakúgy, mint az MD5, ez is 512 bites blokkokban dolgozza fel a bemenetet, de az MD5-tel ellentétben 160 bitből álló pecsétet generál. Azzal kezdi, hogy kitölti az üzenetet, és hozzáadja a 64 bites hosszat, hogy a hossznak 512 bit többszörösét kapja. Ezután beállítja a 160 bites kimeneti puffer kezdeti értékét.

Minden bemeneti blokknál a kimeneti blokk frissítődik az 512 bites blokk függvényében. Nem használ véletlen számokat, (vagy szinusz függvény értékeket), de minden blokkra 80-szor hajtja végre a kört, ezzel egy alapos keverést eredményez. Minden 20 körönként változik a keverési függvény.

Mivel az SHA hash kódja 32 bittel hosszabb, mint az MD5-é és minden másban megegyeznek, ezért 2^{32}-szer biztonságosabb, mint az MD5. Mindazonáltal lassúbb is, mint az MD5 és bizonyos esetben hátrányt jelenthet, hogy a hash kód nem kettő hatvány hosszúságú. Egyébként nagyjából megegyező technikájúak. Politikai szempontból nézve, az MD5 egy RFC-ben található, és főleg az Interneten használják. Az SHA egy kormányzati szabvány, és azok a cégek használják, amelyeket a kormányzat erre utasít, vagy azok, akiknek szükségük van a különleges biztonságra. Egy átdolgozott változatát, az SHA–1-et szabványként jóváhagyta a NIST.

A születésnap támadás

A titkosítás világában semmi sem az, aminek tűnik. Az ember azt hiheti, hogy egy m bites üzenet pecsét kicselezése nagyságrendileg 2^m műveletet igényel. Gyakorlatilag azonban sokszor $2^{m/2}$ művelet is elég, amennyiben a születésnap támadást használjuk, amelyet Yuval (1979) publikált a ma már klasszikusnak számító cikkében a „Hogyan játsszuk ki Rabint"-ban.

Ennek a támadásnak az alapja egy, a matematikaprofesszorok által, valószínűségszámítás előadásokon, gyakran használt módszer. A kérdés: Hány embernek kell egy osztályban lenni ahhoz, hogy annak a valószínűsége, hogy két ember ugyanazon a napon született, meghaladja az ½-et. A legtöbb hallgató messze több mint 100-ra tippel megoldásként. Valójában azonban a valószínűség számítás szerint csak 23. Anélkül, hogy precíz bizonyítást adnánk, szemléletesen 23 emberre $(23\times22)/2 = 253$ különböző párt alkothatunk, és minden pár találati valószínűsége 1/365. Ebben a megközelítésben nem is olyan meglepő a tény.

Általánosabban, ha a bemenet és a kimenet között létezik egy megfeleltetés, n bemenet (ember, üzenet stb.) és k lehetséges kimenet (születésnap, üzenet pecsét stb.) esetén, akkor $n(n-1)/2$ bemeneti pár van. Ha $n(n-1)/2 > k$, akkor annak az esélye, hogy legalább egy megfelelő párt találunk elég nagy. Így megközelítőleg $n > \sqrt{k}$ esetén számíthatunk találatra. Ez az eredmény azt jelenti, hogy egy 64 bites üzenet pecsét nagy valószínűséggel feltörhető, ha generálunk 2^{32} db üzenetet, és keresünk kettőt, aminek ugyanaz az üzenet pecsétje.

Nézzünk meg egy gyakorlati példát. Az Állami Egyetem Informatika karán egy végleges kari oktatói állásra két jelentkező van Tom és Dick. Tomot két évvel korábban vették fel, mint Dicket, először tehát őt bírálják el. Ha megkapja az állást, akkor Dick pórul jár. Tom tudja, hogy a kar elnöke, Marilyn, jó véleménnyel van a munkájáról, tehát megkéri, hogy írjon róla egy ajánlást a dékánnak, aki majd Tom ügyét el fogja bírálni. Minden levél bizalmassá válik, miután elküldték.

Marilyn utasítja titkárnőjét, Ellent, hogy írjon egy levelet a dékánnak, melyben kiemeli véleményét. Amikor elkészül, Marilyn átnézi, és aláírja a 64 bites pecséttel, majd elküldi a dékánnak. Ellen elküldheti a levelet később emailben. Tom számára a helyzet szerencsétlen, mert Ellen romantikus kapcsolatban van Dick-kel, és át akarja őt verni, tehát az alább látható levelet írja, 32 szögletes zárójellel ellátott választási lehetőséggel.

Kedves Smith Dékán Úr,

Ez a [*levél* | *üzenet*] [*őszinte* | *nyílt*] véleményemet tolmácsolja Tom Wilson professzorról, aki [*éppen* | *ebben az évben*] véglegesítésre [*jelölt* | *van jelölve*]. [*Ismerjük egymást* | *Együtt dolgoztam*] Wilson professzorral [*már* | *majdnem*] hat éve. [*Kiváló* | *Kiemelkedő*] [*tehetséggel* | *képességekkel*] megáldott kutató, aki [*világszerte* | *nemzetközileg*] ismert [*briliáns* | *kreatív*] meglátásaiért [*sok* | *a legkülönbözőbb*] [*nehéz* | *bonyolult*] problémakörben.

Emellett [*nagyon* | *kiemelkedően*] [*tisztelt* | *csodált*] [*tanár* | *oktató*]. Hallgatói [*jó* | *kitűnő*] véleménnyel vannak [*óráiról* | *előadásairól*]. [*Tanszékünkön* | *Nálunk*] a [*legnépszerűbb* | *legkedveltebb*] [*tanár* | *előadó*].

[*Sőt mi több* | *Ezenfelül*] Wilson professzor [*tehetséges* | *hatékony*] alapítványi pályázó. [*Szerződései* | *alapítványi pénzei*] [*sok* | *számot tevő*] pénzt hoztak [*nekünk* | *a tanszékünknek*]. Ez a [*pénz* | *támogatás*] [*lehetővé tette* | *engedte meg*] számunkra, hogy sok [*speciális* | *fontos*] programokkal [*foglalkozhassunk* | *dolgozhassunk*], [*mint például* | *többek között*] az ön State 2000 programjával. Ezen pénzek nélkül nem [*lennénk képesek* | *tudnánk*] folytatni tevékenységünket, ami pedig mindkettőnknek [*fontos* | *lényeges*]. Azt tanácsolom, hogy véglegesítse őt.

Tom pechére, amint Ellen befejezte a levél fogalmazását és begépelését, rögtön ír egy másikat is.

Kedves Smith Dékán Úr,

Ez a [*levél* | *üzenet*] [*őszinte* | *nyílt*] véleményemet tolmácsolja Tom Wilson professzorról, aki [*éppen* | *ebben az évben*] véglegesítésre [*jelölt* | *van jelölve*]. [*Ismerjük egymást* | *Együtt dolgoztam*] Professzor Wilsonnal [*már* | *majdnem*] hat éve. [*Gyenge* | *szegényes*] képességekkel bíró kutató, és nem nagyon ismert a [*szakterületén* | *tevékenységi körében*]. Kutatásai [*ritkán* | *csak elvétve*] tartalmaznak jó [*meglátásokat* | *észrevételeket*] [*napjaink* | *az aktuális*] problémáinak [*fő* | *kulcs*] gondolatával kapcsolatban.

Továbbá nem kifejezetten [*kedvelt* | *tisztelt*] [*előadó* | *tanár*]. [*Előadásairól* | *Óráiról*] [*rossz* | *pocsék*] véleménnyel vannak hallgatói. [*Tanszékünkön* | *Nálunk*] a legnépszerűtlenebb [*előadó* | *tanár*], aki [*főleg* | *elsősorban*] arról azon [*tulajdonságáról* | *hajlamáról*] ismert [*nálunk* | *tanszékünkön*], hogy [*zavarba* | *kellemetlen helyzetbe*] hozza azokat a diákokat, akik kérdezni [*mernek* | *merészelnek*] óráin.

[*Sőt, mi több* | *Ezenfelül*] Tom [*gyenge* | *átlagon aluli*] pénzszerző. [*Ösztöndíjai* | *Szerződései*] csak [*kevés* | *elenyésző*] pénzt hoznak [*nekünk* | *tanszékünknek*]. Ha nem találunk gyorsan új [*pénzbevételi* | *anyagi*] forrást elképzelhető, hogy le kell állítanunk néhány fontos programunkat, mint például az ön State 2000 programját. Sajnos ezen [*körülmények* | *helyzet*] mellett nem tudom [*becsülettel* | *tiszta lelkiismerettel*] ajánlani [*véglegesítésre* | *végleges pozícióra*].

Ezek után Ellen beállítja számítógépét, hogy az éjszaka számolja ki mind a 2^{32} db pecsétet mindkét üzenethez. Jó esély van rá, hogy egy az első levél pecsétjei közül egy megegyezik a második levél egyik pecsétjével. Ha nem, akkor hozzáadhat még egy-két lehetőséget, és újra próbálkozhat a hétvégén. Tegyük fel, hogy talál két egyezőt. Nevezzük a jó levelet *A*-nak, a rosszat pedig *B*-nek.

Ellen ezután jóváhagyás végett elküldi az *A* levelet Marilynnek. Marilyn természetesen jóváhagyja azt, és kiszámolja a 64 bites üzenet pecsétet, aláírja azt, és feladja Smith dékán úrnak. Függetlenül ettől, Ellen elküldi a *B* levelet a dékánnak.

A dékán, miután megkapta a levelet és az aláírt pecsétet, lefuttatja az üzenet pecsét algoritmust a *B* levélen és látja, hogy az megegyezik a Marilyn által küldöttel, és kirúgja Tomot. (Lehetséges végkifejlet: Ellen elmondja Dicknek, hogy mit tett. Dick megdöbben, és szakít vele. Ellen dühbe jön és bevallja bűnét Marilynnek. Marilyn felhívja a dékánt. Tomot végül is véglegesítik.) Az MD5-nél a születésnap támadás keresztülvihetetlen, mert még percenként 1 billió üzenet pecsétes sebességgel is 500 évig tartana kiszámolni mind a 2^{64} db pecsétet két levélhez, melyek mindegyikében 64 variáció lehetséges, és még utána sem garantált a találat.

7.1.7. Társadalmi kérdések

A hálózati biztonság magánéletbe és társadalomba való bevonása első hallásra megdöbbentő. Az alábbiakban csak néhányat említünk a kiemelkedő problémák közül.

A hatóság nem szereti, ha az állampolgárok titkolóznak előtte. Néhány országban (pl. Franciaországban) minden nem hatósági titkosítás egyszerűen be van tiltva, hacsak a hatóság meg nem kapja az összes használatban levő kulcsot. Ahogyan Kahn (1980) és Selfridge és Schwartz (1980) kiemelte, a hatóság sokkal szélesebb körben végzett lehallgatásokat, mint azt a legtöbb ember gondolná, és a hatóság többet akar egy halom megfejthetetlen bitnél a fáradozásaiért.

Az USA hatósága javasolt egy titkosítási sémát, a jövőbeli digitális telefonokhoz, amely lehetővé teszi a rendőrség számára, hogy lehallgathassa, és visszakódolhassa az USA-beli hívásokat. A hatóság megígéri, hogy nem használja ezt a lehetőséget bírósági engedély nélkül, de sokan vannak, akik emlékeznek még, hogy a volt FBI-igazgató J. Edgar Hoover illegálisan lehallgattatta ifj. Martin Luther King és mások telefonját azért, hogy megpróbálják semlegesíteni őket. A rendőrség azt állítja, hogy kell nekik ez az erő a bűnözők elfogásához. A viták enyhén szólva viharosak mindkét fél részéről. Az alkalmazott módszerről (Clipper) szóló értekezés megtalálható (Kaufman és mások, 1995). Egy, a módszert kijátszó megoldás ismertetése, ami segítségével olyan

üzeneteket lehet küldeni, amit a hatóság nem tud elolvasni (Blaze, 1994; Schneider, 1996). Az összes fél állásfoglalása olvasható (Hoffman, 1995) művében.

Az Egyesült Államokban van egy törvény (22 U.S.C. 2778), amely megtiltja az állampolgároknak, hogy a Hadügyminisztérium engedélye nélkül hadianyagot (háborús eszközt), azaz pl. tankokat vagy lökhajtásos repülőket exportáljon. E törvény tekintetében a titkosítási szoftverek hadianyagnak számítanak. Phil Zimmermann-t, aki a PGP-t (Pretty Good Privacy, egy e-levél-titkosító program), írta, megvádolták ennek a törvénynek a megszegésével, annak ellenére, hogy a hatóság elismeri, hogy nem exportálta (de odaadta egy barátjának, aki felrakta az Internetre, ahonnan bárki lemásolhatja). Sokan úgy vélték, hogy ez a nagy port kavart incidens az emberek biztonságán dolgozó amerikai állampolgár jogainak durva megsértése.

Az sem segít, ha nem amerikai valaki. 1986. július 9-én három izraeli kutató, akik a Weizmann Intézetben, Izraelben dolgoztak, dokumentáltak egy USA-beli szabadalmú új digitális aláírásokkal foglalkozó programot, amit ők találtak fel. A következő 6 hónapot azzal töltötték, hogy szerte a világon konferenciákon vitatták meg kutatásuk eredményét. 1987. január 6-án, az USA-beli szabadalmi iroda utasította őket, hogy közöljék minden amerikaival, akik tudnak az eredményekről, hogy a kutatás nyilvánosságra hozása két év börtönbüntetést, vagy 10 000 dollár pénzbírságot, vagy mindkettőt von maga után. A szabadalmi iroda ezenkívül egy listát is akart a külföldiekről, akik szintén tudtak a kutatásról. A történet befejezését lásd (Landau, 1988) művében.

Egy másik forró téma a szabadalmak. Majdnem minden nyilvános kulcsú algoritmus szabadalmaztatva van. A szabadalmi védelem 17 évre szól. Az RSA szabadalma pl. 2000. szeptember 20-án jár le.

A hálózati biztonsággal kiterjedtebben foglalkoznak politikai szemszögből, mint néhány más műszaki kérdéssel, ami érthető is, hiszen a digitális világban ez a demokrácia és rendőrállam közötti különbség kérdésével van kapcsolatban. A *Communications of the ACM* 1993. márciusi és 1994. novemberi számaiban hosszan foglalkoznak a telefon- és a hálózati biztonsággal, külön-külön védve és nyomós érveket hozva a különböző álláspontokra. Schneier biztonsággal foglalkozó könyvének 25. fejezete a titkosítás politikájával foglalkozik (Schneier, 1996). Csakúgy, mint az e-levéllel foglalkozó könyvének 8. fejezete (Schneier, 1995). A titkosság és számítógépek témája szintén előkerül (Adam, 1995) művében. Ezek ajánlott irodalmak azoknak, akik tanulmányaikat ezen a téren szeretnék folytatni.

7.2. DNS – Körzeti névkezelő rendszer

A programok ritkán utalnak a hosztokra, levelesládákra és más erőforrásokra a bináris hálózati címükkel. A bináris számok helyett ASCII karakterláncokat használnak, mint pl. *tana@art.ucsb.edu*. Mindazonáltal a hálózat maga csak a bináris címeket érti meg, kell tehát valami mechanizmus az ASCII karakterláncok hálózati címekké konvertálására. A következő részekben megvizsgáljuk, hogyan is történik ez a megfeleltetés az Interneten.

Még az ARPANET idejében, egyszerűen volt egy fájl, a *hosts.txt*, amiben fel voltak sorolva a hosztok és az IP címeik. Minden éjszaka az összes hoszt kiolvasta ezt a fájlt arról a gépről, ahol azt karbantartották. Ez a megközelítés egészen jól működött néhány száz nagy időosztásos gép esetében.

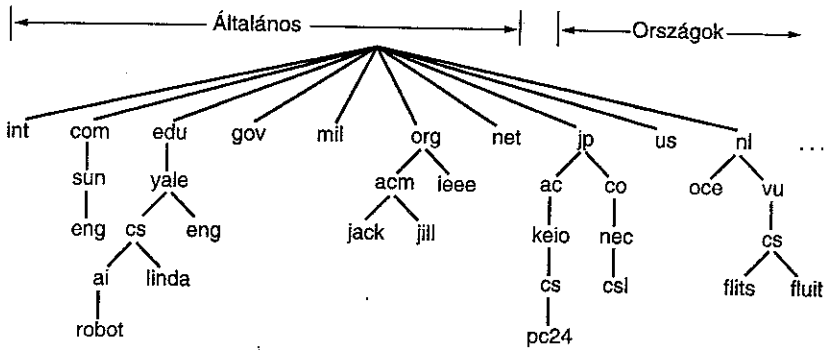
Amikor azonban több ezer munkaállomást kapcsoltak a hálózatra, mindenki rájött, hogy ez a módszer nem működhet örökké. Ha másért nem, hát azért, mert a fájl mérete túl nagy lenne. Ami még fontosabb azonban az, hogy hoszt név konfliktusok is állandóan előfordulnának, amennyiben nem központilag kezelnék a neveket, ami pedig elképzelhetetlen egy kiterjedt nemzetközi hálózatnál. Ezen problémák megoldására találták ki a **DNS-t (Domain Name System – körzeti névkezelő rendszer).**

A DNS lényege egy hierarchikus körzetalapú névkiosztási séma, és az azt megvalósító osztott adatbázisrendszer kitalálásában rejlik. Elsősorban arra szolgál, hogy hoszt neveket, és e-levél címeket feleltessen meg IP címeknek, de más célokra is használható. A DNS-t az RFC 1034-ben és 1035-ben definiálja.

Vázlatosan a következőképpen zajlik a DNS használata. Egy felhasználói program a névről IP címre való leképzéséhez meghívja a névvel, mint paraméterrel a **címfeloldó (resolver)** nevű könyvtári eljárást. A címfeloldó elküld egy UDP csomagot a helyi DNS szervernek. A szerver megkeresi és visszaküldi az IP címet a címfeloldónak, ami visszaadja azt a hívónak. Az IP cím birtokában a program már felépítheti a TCP kapcsolatot a célgéppel, vagy küldhet neki UDP csomagokat.

7.2.1. A DNS név tér

Nagy mennyiségű, állandóan változó nevek halmazát nem triviális probléma kezelni. A postai rendszerben a névkezelés úgy történik, hogy a címzett eléréséhez a levélen (implicit vagy explicit) módon fel kell tüntetni az országot, az államot vagy megyét, a várost és az utcát. Ezen hierarchikus címzés mellett nincs kavarodás a Main St.-en, Bostonban, Massachusettsben lakó Marvin Anderson, és a Main St.-en, Austinban, Texasban lakó Marvin Anderson között. A DNS is így működik.



7.25. ábra. *Az Internet DNS név tér egy darabja*

Az Internet egy koncepció szerint néhány száz elsődleges **körzetre (domain)** van osztva, ahol minden körzethez sok hoszt tartozik. Minden körzet alkörzetekre van osztva, amelyek tovább vannak osztva és így tovább. Ezeket a körzeteket egy fával lehet ábrázolni (l. 7.25. ábra). A fa levelei olyan körzeteket reprezentálnak, amelyek nincsenek alkörzetekre bontva (de természetesen tartoznak hozzájuk gépek). Egy levélben levő körzet tartalmazhat egyetlen hosztot is, vagy képviselhet egy egész vállalatot, és tartalmazhat több ezer hosztot.

A legfelső szinten levő, ún. elsődleges körzetek kétfélék lehetnek: általánosak és országra vonatkozó körzetek. Az általános körzetek a következők lehetnek, *com* (kereskedelem), *edu* (oktatási intézmények), *gov* (az USA szövetségi kormánya), *int* (néhány nemzetközi szervezet), *mil* (az USA fegyveres erői), *net* (hálózati szolgáltatók) és *org* (profit nélküli szervezetek). Az országra vonatkozó körzetek esetén minden országhoz tartozik egy ország körzet, az ISO 3166-nak megfelelően.

Minden körzet neve a (névtelen) gyökérhez felfele vezető út. A komponenseket ponttal választják el. A Sun Microsystems műszaki részlegének neve pl. *eng.sun.com.*, szemben egy UNIX-stílusú névvel, pl. */com/sun/eng*. Vegyük észre, hogy a hierarchikus felépítésből adódóan nincs konfliktus az *eng.sun.com.*-ban és az esetleg az *eng.yale.edu.*-ban használt eng között, ami a Yale angol tanszékének neve lehetne.

A körzetnevek lehetnek mind abszolútak, mind relatívak. Az abszolút körzetnevek ponttal végződnek (pl. *eng.sun.com.*), míg a relatívak nem. A relatív neveket egy adott környezetben kell értelmezni, hogy a valódi jelentésüket megállapíthassuk. Mindkét esetben egy körzetnév a fa egyik csomópontjára, és az alatta levő részfára vonatkozik.

A körzetnevekben mindegy, hogy kis- vagy nagybetűket használunk-e, tehát az *edu* és az *EDU* ugyanazt jelenti. A névkomponensek maximum 63 karakter hosszúak lehetnek, és az egész útvonalnév nem haladhatja meg a 255 karaktert.

Elvileg a körzetneveket kétféleképpen illeszthetjük be a fába. Például, a *cs.yale.edu* egyenértékű megfelelője lehet az *us* ország körzet alá illeszkedő *cs.yale.ct.us* névnek. Gyakorlatilag azonban majdnem minden egyesült államokbeli szervezet az általános körzetekhez tartozik, és majdnem minden Egyesült Államokon kívüli szervezet a saját ország körzetéhez tartozik. Nincs szabály az ellen, hogy két elsődleges körzet alá is tartozzon valami, de ez zavaró lehet, így csak kevés szervezetnél fordul elő.

Minden körzet maga ellenőrzi az alatta levő körzetek kiosztását. Például Japánnak külön körzetei vannak, *ac.jp*, *co.jp*, a *com* és *edu* megfeleltetésére. Hollandiában nincs ilyen szétosztás, hanem minden szervezet egyenesen az *nl*-hez tartozik. Ily módon mindhárom alábbi cím egyetemek informatikai tanszékének címei:

1. *cs.yale.edu* (Yale Egyetem, az Egyesült Államokban).

2. *cs.vu.nl* (Vrije Universiteit, Hollandiában).

3. *cs.keio.ac.jp* (Keio Egyetem, Japánban).

Egy új körzet létrehozásához engedély kell attól a körzettől, amelyhez tartozni fog. Például, ha létrejön egy VLSI csoport a Yale Egyetemen belül, és *vlsi.cs.yale.edu* néven akar futni, akkor attól kell engedélyt kérnie attól, aki a *cs.yale.edu*-t karbantartja.

Hasonlóképpen, ha egy új egyetem nyílna, mondjuk a Dél-Dakotai Egyetem, akkor az *edu* körzet karbantartójától kellene engedély kérni, hogy rendelje hozzá a *unsd.edu* címet. Ily módon nincsenek névkonfliktusok, és minden körzet számon tarthatja a hozzá tartozó alkörzeteket. Miután egy új körzet létrejött már szabadon létrehozhat hozzá tartozó alkörzeteket anélkül, hogy a fán egy fentebb elhelyezkedőtől engedélyt kérne (pl. *cs.unsd.edu*).

Az elnevezések nem a hálózat fizikai elrendezését, hanem a szervezetek határait követik. Például annak ellenére, hogy az informatikai és a villamosmérnöki tanszék ugyanabban az épületben van, és ugyanazt a hálózatot használja, különböző körzetekhez tartozhatnak. Hasonlóan, még ha az informatika tanszék a Babbage Hallban és Turing Hallban megosztva helyezkedik el, akkor mindkét épületben a hosztok normális esetben ugyanahhoz a körzethez tartoznak.

7.2.2. Erőforrás-nyilvántartás

Minden körzethez tartozhat egy **erőforrás-bejegyzés (resource record)** halmaz, attól függetlenül, hogy a körzet csak egyetlen hosztból áll, vagy egy elsődleges körzet. Egy egyedüli hoszt esetén általában ez az erőforrás-bejegyzés csak az IP cím, de ezenkívül még sok másféle erőforrás-bejegyzés létezik. A címfeloldó a DNS-nek küldött névhez tartozó erőforrás-bejegyzéseket kapja vissza. Ezek szerint a DNS igazi feladata az, hogy megfeleltesse a nevet az erőforrás-bejegyzéseknek.

Az erőforrás-bejegyzés egy adatötösből áll. Annak ellenére, hogy az erőforrás-bejegyzéseket a hatékonyság miatt binárisan tárolják, a legtöbb ismertetőben az erőforrás-bejegyzések ASCII formában szerepelnek, bejegyzésenként egy sorban. Az általunk használt formátum a következő:

```
Körzet_név  Élettartam  Osztály  Típus  Érték
```

A *Körzet_név* jelenti azt a körzetet, amelyhez a rekord tartozik. Normális esetben minden körzethez sok bejegyzés tartozik, és az adatbázis minden másolata több körzettel kapcsolatos információt hordoz. Ez a mező az elsődleges kulcs a kereséshez. A bejegyzések sorrendje nem érdekes az adatbázisban. Ha egy lekérdezés történt valamely körzethez, az eredmény tartalmazni fogja az összes kérésnek megfelelő osztályú bejegyzést.

Az *Élettartam* mező jelzést ad arról, hogy a bejegyzés mennyire stabil. A nagyon stabil információkhoz magas értékek tartoznak, mint a 86 400 (1 nap másodpercekben). Azokhoz az információkhoz, amelyek erősen ingatagok, kis értékek tartoznak, mint a 60 (1 perc). Ehhez a témához visszatérünk még, ha majd a gyorsítótárak használatát tárgyaljuk.

Minden erőforrás-bejegyzés harmadik mezője az *Osztály*. Az Internethez tartozó információknál ez mindig *IN*. A nem Internetes információkhoz más kódokat lehet rendelni, de a gyakorlatban ilyet ritkán lehet látni.

A *Típus* mező a bejegyzés értékének típusára vonatkozik. A legfontosabb típusok a 7.26. ábrán láthatók.

Típus	Jelentés	Érték
SOA	Lista kezdete	Ehhez a zónához tartozó paraméterek
A	Egy hoszt IP címe	32 bites egész
MX	Levél csere	Prioritás, a levelet váró körzet
NS	Név szerver	Egy ehhez a körzethez tartozó szerver neve
CNAME	Kanonikus Név	Körzet név
PRT	Mutató	Álnév egy IP címhez
HINFO	Hoszt leírás	CPU és az operációs rendszer ASCII formában
TXT	Szöveg	Tetszőleges ASCII szöveg

7.26. ábra. *A legfontosabb DNS erőforrás-bejegyzés típusok*

Az *SOA* bejegyzés megadja az elsődleges információforrás nevét a zónához tartozó név szerverről (lásd alább), az adminisztrátor e-levél címét, egy egyedi sorozatszámot, valamint különböző jelzőket és időzítőket.

A legfontosabb bejegyzés típus az *A (cím)* bejegyzés. Egy 32 bites IP címet tartalmaz valamely hoszthoz. Minden Internet hosztnak legalább egy IP címmel kell rendelkeznie, hogy más gépek kommunikálhassanak vele. Egyes hosztok kettő vagy több hálózati csatlakozással is rendelkeznek, ebben az esetben minden hálózati csatlakozáshoz pontosan egy *A* típusú rekord tartozik (és ily módon minden IP címhez is).

A fontossági sorrendben következő bejegyzés típus az *MX* bejegyzés. Ez tartalmazza annak a hosztnak a nevét, amely kész a körzethez tartozó levelek fogadására. Gyakran használják ezt a bejegyzést arra, hogy lehetővé tegyék egy, az Internetre nem csatlakozó gép számára, hogy leveleket fogadhasson az Internetről. A kézbesítés úgy történik, hogy a nem Internetes gép megállapodik egy Internetre kapcsolt géppel, hogy fogadja a hozzá érkező leveleket, és továbbítja őket valamilyen mindkettejük által elfogadott protokollal.

Tegyük fel példaképpen, hogy Cathy végzős informatikus hallgató az UCLA-n. Miután megszerzi a diplomáját a AI szakon, alapít egy céget, az Electrobrain Corporationt, hogy forgalomba hozhassa ötleteit. Egyenlőre nem telik neki Internet-csatlakozásra, tehát megegyezik az UCLA-val, hogy oda küldhessenek neki e-levelet. Majd párszor naponta betelefonál, és elhozza őket.

Ezt követően regisztráltatja cégét a *com* körzetben, és megkapja az *electrobrain.com* címet. Ezután megkérheti a *com* körzet adminisztrátorát, hogy a következő sort adja hozzá az adatbázishoz.

```
Electrobrain.com 86400 IN MX 1 mailserver.cs.ucla.edu
```

Ily módon a levelek az UCLA-ra továbbítódnak, ahonnan, ha bejelentkezik, elhozhatja őket. Esetleg az UCLA felhívhatja őt, és elküldheti neki az e-leveleket, egy kölcsönösen elfogadott protokollal.

Az *NS* bejegyzések a név szervert adják meg. Például, normális esetben minden DNS adatbázis tartalmaz az összes elsődleges körzethez egy rekordot, hogy név-fa messzi részeibe is lehessen e-levelet küldeni. Ehhez a témához később még visszatérünk.

```
; A cs.vu.nl-hez tartozó irányadó adatok
cs.vu.nl      86400   IN   SOA      star boss (952771,7200,7200,2419200,86400)
cs.vu.nl      86400   IN   TXT      "Faculteit Wiskunde en Informatica."
cs.vu.nl      86400   IN   TXT      "Vrije Universiteit Amsterdam."
cs.vu.nl      86400   IN   MX       1 zephyr.cs.vu.nl.
cs.vu.nl      86400   IN   MX       2 top.cs.vu.nl.

flits.vu.nl   86400   IN   HINFO    Sun Unix
flits.vu.nl   86400   IN   A        1310.37.16.112
flits.vu.nl   86400   IN   A        192.31.231.165
flits.vu.nl   86400   IN   MX       1 flits.cs.vu.nl.
flits.vu.nl   86400   IN   MX       2 zephyr.cs.vu.nl.
flits.vu.nl   86400   IN   MX       3 top.cs.vu.nl.
www.vu.nl     86400   IN   CNAME    star.cs.vu.nl
flits.vu.nl   86400   IN   CNAME    zephyr.cs.vu.nl

rowboat               IN   A        130.37.56.201
                      IN   MX       1 rowboat
                      IN   MX       2 zephyr
                      IN   HINFO    Sun Unix

little-sister         IN   A        130.37.62.23
                      IN   HINFO    Mac MacOS

laserjet              IN   A        192.31.231.216
                      IN   HINFO    "HP Laserjet IIISi" Proprietary
```

7.27. ábra. *A cs.vu.nl-hoz tartozó lehetséges DNS adatbázis egy része*

A *CNAME* bejegyzések segítségével álneveket lehet létrehozni. Például, ha valaki, aki ismeri az Internetes névkonvenciókat, egy levelet szeretne küldeni valakinek, akinek a login neve *paul* az M.I.T. Informatika tanszékén, akkor úgy gondolhatja, hogy a *paul@cs.mit.edu* cím valószínűleg megfelelő. Valójában azonban ez a cím nem jó, mert az M.I.T Informatika tanszékének körzete *lcs.mit.edu*. Az M.I.T azonban azok részére, akik ezt nem tudják, segítségképpen létrehozhat egy *CNAME* bejegyzést, ami átirányítja az embereket és programokat a helyes útra. Ezt megteszi pl. a következő bejegyzés:

```
cs.mit.edu 86400 IN CNAME lcs.mit.edu
```

A *CNAME*-hez hasonlóan a *PTR* is egy másik névre mutat. Azonban, a *CNAME*-mel ellentétben, ami tulajdonképpen csak egy makró, a *PTR* egy valódi DNS adattípus, melynek értelmezése az adott környezettől függ. A gyakorlatban majdnem mindig arra használják, hogy egy nevet megfeleltessenek egy IP címnek, hogy lehetővé váljon az IP címek szerinti keresés, ahol a keresett gép neve az eredmény.

A *HINFO* bejegyzések lehetővé teszik bárkinek, hogy megállapíthassa, hogy a kérdéses körzet milyen gépet és operációs rendszert használ. Végül a *TXT* bejegyzések arra szolgálnak, hogy a körzetek tetszés szerinti módon is azonosíthassák magukat. Ez

utóbbi két bejegyzés a felhasználók kényelmét szolgálja. Nem is kötelező jellegűek, azaz a programok nem számíthatnak rá, hogy megkapják őket (és ha mégis megkapják, valószínűleg nem tudnak mit kezdeni velük).

Utolsóként nézzük az *Érték* mezőt. Ez a mező tartalmazhat egy számot, egy körzetnevet vagy egy ASCII karakterláncot. A szemantika a bejegyzés típusától függ. A legfontosabb bejegyzés-típusokhoz tartozó érték mezők leírása a 7.26. ábrán látható. Arra nézve, hogy milyen típusú információk találhatók egy körzethez a DNS adatbázisban a 7.27. ábra ad útmutatást. Ez az ábra a 7.25. ábrán látható *cs.vu.nl* körzet (feltételezett) adatbázisának egy részét mutatja. Az adatbázis hét különböző típusú erőforrás-bejegyzést tartalmaz.

Az első nem megjegyzés sor a 7.27. ábrán a körzetről ad némi alapinformációt, de ezzel tovább nem foglalkozunk. A következő két sor egy szöveges leírást ad arról, hogy hol található a körzet. A következő két bejegyzés az elsődleges és másodlagos levélfogadó helyet adja meg, a *person@cs.vu.nl* címre érkező e-levelek számára. Először a *zephyr*-rel (egy sepcifikus géppel) kell próbálkozni. Ha nem sikerül, akkor a *top* következik.

Az üres sort, ami az olvashatóságot segíti, követő sorok megadják, hogy a *flits* egy Sun munkaállomás, amelyen UNIX fut, valamint megadják ennek mindkét IP címét is. Ezután három lehetőség szerepel a *flits.cs.vu.nl*-re küldött e-levelek kezelésére. Az első választási lehetőség természetesen maga a *flits*, de amennyiben éppen nem üzemel, akkor a *zephyr* és a *top* jöhet szóba, mint második és harmadik lehetőség. A következő egy álnév, a *www.cs.vu.nl*, így ezt a címet anélkül lehet használni, hogy néven kellene nevezni egy adott szervert. Ezen álnév létrehozása lehetővé teszi a *cs.vu.nl* számára, hogy anélkül cserélje le Világháló (World Wide Web) szerverét, hogy érvénytelenné válna a cím, amit az emberek az eléréshez használnak. Az *ftp.cs.vu.nl* is ugyanez vonatkozik.

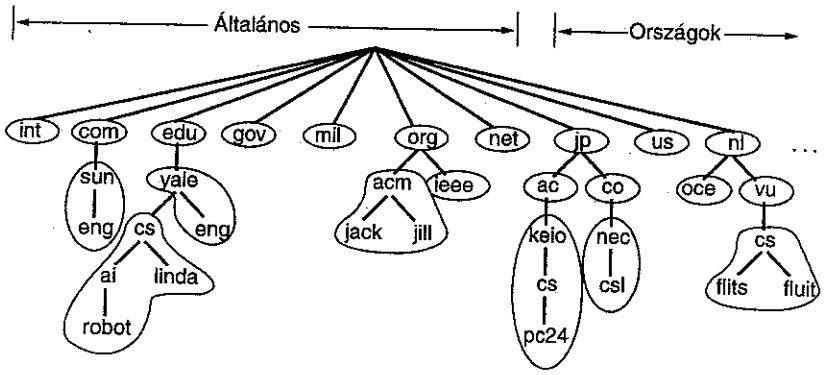
A következő négy sor egy tipikus munkaállomás-leírást tartalmaz, esetünkben a *rowboat.cs.vu.nl*-ét. A megadott információ tartalmazza az IP címet, az elsődleges és másodlagos levélfogadót, és a gépről szóló információkat. Ezután egy nem UNIX-os rendszer leírása jön, amely magától nem képes leveleket fogadni, valamint ezt követően egy lézernyomtatóhoz tartozó bejegyzés.

Amik nem láthatóak (és nincsenek is a fájlban), az elsődleges körzetek kereséséhez szükséges IP címek. Ezek a távoli körzetek eléréséhez szükségesek, de mivel azok nem részei a *cs.vu.nl* körzetnek, ezért nem szerepelnek ebben a fájlban. Ezeket a rootszerverek szolgáltatják, amelyek IP címei egy rendszer konfigurációs fájlban vannak tárolva, ami betöltődik a DNS gyorsítótárba, mikor a DNS szervert elindítják.

7.2.3. Név szerverek

Elvileg egyetlen szerver elegendő lenne a DNS adatbázis tárolására, és a kérések megválaszolására. Gyakorlatilag ez a szerver annyira túl lenne terhelve, hogy használhatatlan lenne. Továbbá, ha egyszer csak felmondaná a szolgálatot, az egész Internet lebénulna.

Az egyetlen szerver miatt adódó problémák elkerülése végett a DNS név tér egy-



7.28. ábra. *A DNS név tér egy része zónákra osztással*

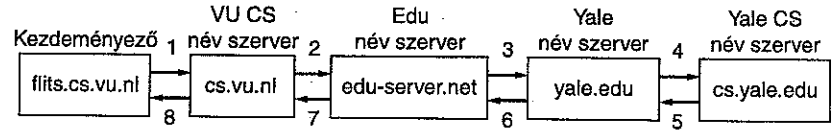
mást nem fedő **zónákra (zones)** van osztva. A 7.25. ábra név terének egy lehetséges felosztása a 7.28. ábrán látható. Minden zóna tartalmazza a fa egy részét, valamint tartalmaz név szervereket, amelyek a zóna hiteles információit tárolják. Normális esetben zónánként egy elsődleges név szerver van, ami egy lemezen levő fájlból nyeri információit, valamint egy vagy több másodlagos név szerver, amelyek az elsődleges név szervertől nyerik információikat. A megbízhatóság növelése érdekében a zónáért felelős szerverek egy része lehet a zónán kívül is.

A zónán belüli zónahatárok meghatározása a szóban forgó zóna adminisztrátorától függ. A döntés meghozatalában nagy szerepet játszik, hogy hol és mennyi név szerver kell. Például a 7.28. ábrán a Yale-nek van egy szervere a *yale.edu*-hoz, ami kiszolgálja az *eng.yale.edu*-t, de a *cs.yale.edu*-t nem, ami egy különálló zóna saját név szerverrel. Egy ilyen döntés akkor születik, ha egy tanszék, mint pl. az Angol nyelvi tanszék nem akar saját név szervert üzemeltetni, de pl. az Informatika tanszék igen. Ennek megfelelően a *cs.yale.edu* egy különálló zóna, míg az *eng.yale.edu* nem.

Ha egy címfeloldó meg szeretne tudni valamit egy körzetnévről, akkor elküldi a lekérdezést egy helyi név szervernek. Ha a keresett körzet a név szerver hatáskörébe tartozik, mint ahogy pl. az *ai.cs.yale.edu* a *cs.yale.edu* alá tartozik, akkor az visszaküldi a hiteles erőforrás-bejegyzéseket. A **hiteles bejegyzés (authoritative record)** azt jelenti, hogy a bejegyzés attól szervtől származik, amelyik azt a bejegyzést kezeli, tehát mindig helyes. A hiteles bejegyzések tehát ellentétben vannak a gyorsítótárban levő bejegyzésekkel, amelyek esetleg idejétmúltak lehetnek.

Ha azonban egy távoli körzetről van szó, amelyről nincsen információ a helyi adatok közt, akkor a név szerver elküld egy lekérdező üzenetet a szóban forgó körzetet tartalmazó elsődleges körzetnek. A művelet megértéséhez vizsgáljuk meg a 7.29. ábrát. Itt egy címfeloldó a *flits.cs.vu.nl*-en meg szeretné tudni, hogy mi az IP címe a *linda.cs.yale.edu*-nak. Az 1. lépésben elküldi kérését a helyi név szervernek, a *cs.vu.nl*-nek. Ez a lekérdezés tartalmazza a keresett körzet nevét, a típust (*A*) és az osztályt (*IN*).

Tegyük fel, hogy a helyi név szerverhez még sohasem érkezett ezzel a körzettel kapcsolatban lekérdezés, és nem is tud róla semmit. Megkérdezhet néhány közeli név



7.29. ábra. *Hogyan keres meg a címfeloldó egy távoli nevet 8 lépésben*

szervert, de amennyiben egyikük sem tud segíteni, elküld egy UDP csomagot az adatbázisában levő *edu*-hoz tartozó szervernek (lásd 7.29. ábra), az *edu-server.net*-nek. Nem valószínű, hogy ez a szerver tudja a *linda.cs.yale.edu* címét, és valószínűleg a *cs.yale.edu*-ét sem, de ismernie kell minden gyermekét, tehát továbbítja a kérést a *yale.edu* név szervernek (3. lépés). Ez szintén továbbküldi a kérést a *cs.yale.edu*-nak (4. lépés), ahol már biztos megtalálhatók a hiteles bejegyzések. Mivel minden kérés egy klienstől egy szerverig haladt, így az erőforrás-bejegyzés az 5.-től 8.-ig terjedő lépéseken keresztül jut vissza.

Amikor aztán ezek a bejegyzések eljutnak a *cs.vu.nl* név szerverhez, bekerülnek majd egy gyorsítótárba, arra az esetre, ha esetleg később szükség lenne rájuk. Ez az információ azonban nem hiteles, hiszen a *cs.yale.edu*-nál történt változások nem kerülnek frissítésre a világ összes gyorsítótárában, ahol számon vannak tartva. Ezért kell, hogy a gyorsítótárbeli bejegyzések rövid élettartamúak legyenek. Emiatt került be az *Élettartam* mező minden erőforrás-bejegyzésbe. Ez jelzi a távoli név szerverek számára, hogy meddig tárolják a gyorsítótárban a bejegyzést. Ha egy gépnek már évek óta ugyanaz az IP címe, akkor lehet, hogy biztonságos ezt az információt 1 napig tárolni. A gyakrabban változó információk esetében biztonságosabb lehet, ha a bejegyzéseket néhány másodperc, vagy egy perc elteltével kitörlik.

Érdemes megemlíteni, hogy az itt leírt lekérdezési eljárást **rekurzív lekérdezésnek (recursive query)** nevezik, mert minden szerver, amely nem rendelkezik a keresett információval tovább keresi azt máshol, majd beszámol az eredményről. Egy másik megoldás is lehetséges. Ebben a megoldásban, ha egy lekérdezést nem lehet kielégíteni lokálisan, akkor a keresés sikertelen, de annak a szervernek a neve lesz az eredmény, amellyel a sorban következőként próbálkozni lehet. Ennél a módszernél a kliens jobban irányíthatja a keresést. Vannak olyan szerverek, ahol a rekurzív keresés nincs megvalósítva és mindig annak a szervernek a nevét adják meg, amelyikkel következőleg próbálkozni lehet.

Érdemes még arra rámutatni, hogy amennyiben egy DNS kliens nem kap választ, mielőtt a várakozási ideje lejárna, akkor normális esetben legközelebb egy másik szerverrel fog próbálkozni. A feltételezés ebben az esetben inkább az, hogy a szerver valószínűleg nem üzemel, mint az, hogy a kérés vagy üzenet elveszett.

7.3. SNMP – egyszerű hálózatfelügyelő protokoll

Az ARPANET kezdeti korszakában, ha egy hoszt irányába nem várt módon megnőtt a késleltetés, akkor a problémát észlelő személy egyszerűen lefuttatta a Ping programot, hogy visszapattintson egy csomagot a célállomástól. A visszaküldött csomag fejlécében levő időbélyegeket megvizsgálva a probléma helye általában hajszálpontosan meghatározható és megfelelő intézkedéssel orvosolható volt. Ezenfelül a nem nagy számú router miatt mindegyikre le lehetett futtatni a Pinget, és ellenőrizni lehetett, hogy vajon meghibásodtak-e.

Amikor az ARPANET-ből sok gerinchálózattal és rengeteg operátorral üzemelő világméretű Internet lett, ez a megoldás már nem volt alkalmazható, és jobb hálózatfelügyelő eszközöket kellett találni. Két korai kísérletet az 1028-as és 1067-es RFC-kben definiáltak, de ezek nem voltak hosszú életűek. 1990 májusában definiálták és hozták nyilvánosságra az 1157-es RFC-ben az **SNMP (Simple Network Management Protokol – egyszerű hálózatfelügyelő protokoll)** 1-es verzióját. A felügyelettel kapcsolatos információkat tartalmazó társ dokumentummal (RFC 1155) együtt az SNMP egy szisztematikus módszert adott a számítógépes hálózatok figyelésére és felügyeletére. Ezt a keretet és protokollt széles körben implementálták a különböző kereskedelmi termékekben, és a hálózatfelügyelet de facto szabványaivá váltak.

Amint egyre több tapasztalat gyűlt össze, fény derült a SNMP gyenge pontjaira, így (az 1441-es és 1452-es RFC-kben) egy továbbfejlesztett változatot, az SNMPv2-t definiáltak, ami aztán elindult az Internetes szabvánnyá válás útján. A következő részekben röviden ismertetjük az SNMP modellt és protokollt (ami ezután az SNMPv2-t fogja jelenteni).

Annak ellenére, hogy az SNMP-t abban a szellemben tervezték, hogy egyszerű legyen, legalább egy szerzőnek sikerült 600 oldalas könyvet írni róla (Stallings, 1993a). Ennél tömörebb leírás (450–550. oldal) tekintetében lásd Rose (1994) valamint Rose és McCloghrie (1995) könyveit, akik mind részt vettek a SNMP tervezésében. További referenciák (Feit, 1995; valamint Hein és Griffiths, 1995).

7.3.1. Az SNMP modell

Az SNMP hálózatfelügyeleti modellje négy összetevőből áll.

1. Felügyelt csomópontok.
2. Felügyeleti állomások.
3. Felügyeleti információ.
4. Felügyeleti protokoll.

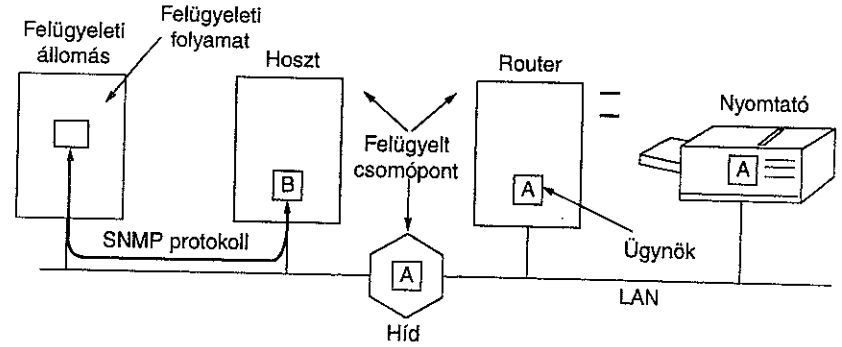
Ezek a részek a 7.30. ábrán láthatók, és alább kerülnek ismertetésre.

A felügyelt csomópontok lehetnek hosztok, routerek, hidak, nyomtatók vagy bármely más készülékek, amelyek képesek a külvilág számára állapot információt küldeni. Ahhoz, hogy egy csomópontot közvetlenül az SNMP felügyelhessen, képesnek kell lennie egy **SNMP ügynöknek (SNMP agent)** nevezett SNMP felügyeleti folyamat futtatására. Minden számítógép képes erre, és egyre nagyobb számban képesek a hidak, routerek és a hálózati használatra tervezett perifériák is. Minden ügynök egy változókból álló helyi adatbázist tart fent, amely leírja az állapotot, adatokat tárol múltbeli eseményekről, és befolyásolja a működést.

A hálózat felügyelete a **felügyeleti állomásokon (management stations)** keresztül történik, amelyek gyakorlatilag speciális szoftvert futtató általános célú számítógépek. A felügyeleti állomáson egy vagy két folyamat fut, amelyek a hálózaton keresztül kommunikálnak az ügynökökkel, parancsokat osztogatnak és válaszokat fogadnak. Ebben a változatban minden intelligencia a felügyeleti állomásokban koncentrálódik, annak érdekében, hogy az ügynökök a lehető legegyszerűbbek legyenek, és minél kevésbé befolyásolják az őket futtató készülékeket. Sok felügyeleti állomásnak van grafikus felhasználói felülete, hogy a hálózat felügyelőjének lehetősége nyíljon a hálózat állapotának figyelésére és szükség esetén beavatkozásra.

A legtöbb valóságos hálózat több kézben van, egy vagy több gyártótól származó hosztokkal, más cégektől származó hidakkal és routerekkel, és megint más cégektől származó nyomtatókkal. Ahhoz, hogy a felügyeleti állomás (ami potenciálisan szintén egy újabb gyártótól származik) beszélni tudjon az összes különböző összetevővel, a készülékek által tárolt információ természetét szigorúan meg kell határozni. A felügyeleti állomás hiába kérdezi meg a forgalomirányítót, hogy mennyi a csomagvesztési arány, ha a forgalomirányító nem tartja számon a csomagvesztési arányt. Éppen ezért az SNMP minden egyes ügynök számára előírja (a legnagyobb részletességgel), hogy pontosan milyen információt kell tárolniuk, és milyen formátumban kell azt rendelkezésre bocsátaniuk. Az SNMP modell legnagyobb része azzal foglalkozik, hogy definiálja: kinek, mit kell számon tartania, és hogy ez az információ hogyan továbbítódik.

Vázlatosan minden készülék egy vagy több az állapotát leíró változót tart számon.



7.30. ábra. *Az SNMP felügyeleti modell összetevői*

Az SNMP irodalomban ezeket a változókat **objektumoknak (objects)** hívják, azonban ez a terminológia félrevezető lehet, mivel ezek nem olyan értelemben objektumok, mint az objektumorientált rendszerekben előfordulóak, hiszen ezeknek csak állapotuk van, és nem rendelkeznek (az értékolvasás és -íráson kívül más) metódusokkal. Mindazonáltal használata annyira megszokott (pl. a használt specifikációs nyelvben is több fenntartott szóban szerepel), hogy itt is ezt fogjuk használni. Az összes, egy hálózatban lehetséges objektumot az **MIB (Management Information Base – felügyeleti adatbázis)** nevű adatstruktúrában adják meg.

A felügyeleti állomás az SNMP protokoll segítségével tartja a kapcsolatot az ügynökökkel. Ez a protokoll lehetővé teszi a felügyeleti állomás számára, hogy lekérdezhesse egy ügynök helyi objektumainak értékét, és szükség szerint megváltoztathassa azokat. Az SNMP többnyire ebből a lekérdezés-válasz típusú kommunikációból áll.

Történnek azonban néha előre el nem tervezett események. A felügyelt csomópontok felmondhatják a szolgálatot és újraindulhatnak, vonalak megszakadhatnak és helyreállhatnak, torlódások lehetnek és így tovább. Minden lényeges eseményt definiálnak az MIB egy moduljában. Ha egy ügynök észleli, hogy egy lényeges esemény történt, azonnal jelenti azt a konfigurációs listájában szereplő összes felügyeleti állomásnak. Ezt a fajta jelentést (történelmi okok miatt) SNMP **csapdának (trap)** nevezik. A jelentés általában csak azt mondja meg, hogy valami esemény történt. Ezután a felügyeleti állomás feladata, hogy lekérdezéseket küldjön, és kiderítse a részleteket. A felügyelt csomópontoktól a felügyeleti állomásig ugyanis nem megbízható a kommunikáció (pl. nem nyugtázott), és különben is ajánlott a felügyeleti állomás számára, hogy a biztonság kedvéért alkalmanként megszólítsa a felügyelt csomópontokat és ellenőrizze a szokásos eseményeket. A gyorsított csapdafogadással történő hosszú időközönkénti lekérdezést **csapda irányított lekérdezésnek (trap directed polling)** nevezik.

Ez a modell feltételezi, hogy minden felügyelt csomópont képes magától futtatni az SNMP ügynököt. A régebbi készülékek, valamint azok a készülékek, amelyeket eredetileg nem hálózati használatra terveztek nem feltétlenül képesek erre. Ezek kezelésére az SNMP definiált egy **helyettesítő ügynököt (proxy agent),** azaz egy olyan ügynököt, amely egy vagy több nem SNMP készülékre felügyel, és a nevükben kommunikál a felügyeleti állomással, miközben magukkal a készülékekkel valószínűleg valami nem szabványos protokoll segítségével kommunikál.

Végül, a biztonság és a hitelességvizsgálat nagy szerepet játszik az SNMP-ben. A felügyeleti állomásnak lehetősége nyílik rá, hogy sok mindent megtudjon a hatáskörébe tartozó csomópontokról, és le is képes állítani őket. Ezért aztán nagyon fontos, hogy az ügynökök megbizonyosodjanak arról, hogy az állítólag a felügyeleti állomástól érkező lekérdezések tényleg a felügyeleti állomástól származnak. Az SNMPv1-ben a felügyeleti állomás azzal bizonyította kilétét, hogy minden üzenetbe (kódolatlan szövegként) behelyezett egy jelszót. Az SNMPv2-t biztonsági szempontból jelentékeny mértékben feljavították a korábban tárgyalt típusú modern titkosítási technikák segítségével. Ez azonban a már amúgy is bonyolult protokollt még bonyolultabbá tette, így ezt később elvetették.

7.3.2. ASN.1 – Absztrakt szintaxis jelölés 1

Az SNMP lelke az ügynökök által kezelt objektumok halmaza, amelyeket a felügyeleti állomások írhatnak és olvashatnak. A többtulajdonosú kommunikáció lehetővé tételéhez létfontosságú, hogy ezeket az objektumokat egy tulajdonostól független szabványos módon definiálják. Ezenkívül szükség van egy szabványos kódolási módra, a hálózaton történő továbbításhoz. A C-beli definíciók kielégítik ugyan az első kritériumot, azonban az ilyen definíciók nem definiálnak bitkódolást a vezetékek szintjén úgy, hogy egy 32 bites kettes komplemensű alsóvég-felügyeleti állomás egyértelműen információt tudjon cserélni egy 16 bites egyes komplemensű felsővég CPU-n futó ügynökkel.

Emiatt szükség van egy szabványos objektum definíciós nyelvre, és a hozzá tartozó kódolási szabályokra. Az SNMP-ben az **ASN.1-et (Abstract Syntax Notation One – absztrakt szintaxis jelölés)** használják, amit az OSI-tól vettek át. Csakúgy, mint az OSI-nak sok más része ez is nagy, komplex és nem kifejezetten hatékony. (A szerző ezt akarta kifejezni azzal, hogy ASN.1-nek nevezte el ahelyett, hogy egyszerűen ASN-nek nevezte volna – a tervezők implicit módon beismerték, hogy hamarosan felcserélik majd az ASN.2-vel –, de udvariasan elhallgatta ezt.) Az egyetlen állítólagos erőssége az ASN.1-nek (az egyértelmű bitkódolási szabályok létezése) mostanra már igazi hátránnyá vált, hiszen a bitkódolási szabályokat arra optimalizálták, hogy a mindkét végen kódolással és dekódolással töltött többlet CPU idő használata árán ugyan, de minél kevesebb bit vándoroljon a vezetékeken. Egy egyszerűbb séma, ami 32 bites egészeket osztott volna 4 bájtos részekre, valószínűleg jobb lett volna. Ennek ellenére, lesz ami lesz, az SNMP lépten nyomon az ASN.1-et (jóllehet csak egy egyszerűsített részét) használja, tehát mindenkinek, aki igazán meg szeretné érteni az SNMP-t, annak el kell mélyednie az ASN.1-ben. Ennélfogva a következő leírásban is.

Kezdjük az adatleíró nyelvvel, amelyet az ISO 8824-es nemzetközi szabvány definiál. Ezután tárgyaljuk majd az ISO 8825-ös nemzetközi szabványban leírt adatkódolási szabályokat. Az ASN.1 absztrakt szintaxis alapjában véve egy primitív adatdeklarációs nyelv. Lehetőséget nyújt a felhasználónak, hogy primitív objektumokat definiáljon, majd azokat összetettebbekké kombinálja. Egy sor ASN.1 deklaráció funkcionálisan megegyezik a C programokhoz hozzárendelt fejléc fájlokban található deklarációkkal.

Az SNMP nyelvi konvencióit fogjuk követni. Ezek azonban nem teljesen egyeznek meg a sima ASN.1 jelöléseivel. A beépített adattípusok nevei nagybetűkkel vannak írva (pl. *INTEGER*). A felhasználó által definiált adattípusok nevei nagybetűvel kell kezdődjenek, de legalább egy karaktert kell tartalmazniuk, ami nem nagybetű. Az azonosítók tartalmazhatnak kis- és nagybetűket, számjegyeket és kötőjeleket, de kisbetűvel kell kezdődjenek (pl. *counter*). A puha szóköz (white space) karakternek nincs jelentősége (pl. tabulátor, kocsi-vissza jel stb.). Végül a megjegyzések kezdetét -- jelzi, és a sor végéig, vagy a következő -- jelzés előfordulásáig tartanak.

Az SNMP-ben engedélyezett ASN.1 alap adattípusok a 7.31. ábrán láthatóak. (Általánosságban figyelmen kívül hagyjuk az SNMP-ben nem engedélyezett ASN.1-beli lehetőségeket, mint a *BOOLEAN,* illetve *REAL* típusok.) A kódok használatára később térünk ki.

Primitív típus	Jelentés	Kód
INTEGER	Tetszőleges hosszúságú egész	2
BIT STRING	0 vagy több bitből álló füzér	3
OCTET STRING	0 vagy több előjel nélküli bájtból álló füzér	4
NULL	Helyfoglaló	5
OBJECT IDENTIFIER	Egy hivatalosan definiált adattípus	6

7.31. ábra. *Az SNMP-ben engedélyezett ASN.1-beli egyszerű adattípusok*

Elvileg egy *INTEGER* típusú változó akármilyen egész értéket tartalmazhat, de más SNMP szabályok korlátozzák a tartományt. A változók használatára példaképpen vizsgáljuk meg, hogy az *INTEGER* típusú *count* változót hogyan deklaráljuk, és (opcionálisan) hogyan állítjuk be kezdeti értékét 100-ra az ASN.1-ben.

```
count INTEGER ::= 100
```

Gyakran szükség van olyan altípusra, amelyhez olyan változók tartoznak, amelyek értékkészletüket egy bizonyos halmazból veszik. Ez a következőképpen deklarálható:

```
Status ::= INTEGER{up(1), down(2), unknown(3)}
PacketSize ::= INTEGER(0..1023)
```

A *BIT STRING* és *OCTET STRING* változó típusok nulla, vagy több bitet, illetve bájtot tartalmaznak a típusnak megfelelően. Egy bit értéke lehet 0 vagy 1. A bájt a 0-tól 255-ig terjedő mindkét oldalról zárt halmazból veheti értékeit. Mindkét típushoz megadható a füzér hossza, valamint egy kezdőérték.

Az *OBJECT IDENTIFIER*-ek az objektumokra való hivatkozást teszik lehetővé. Gyakorlatilag minden hivatalos szabványban definiált minden objektum egyedileg megkülönböztethető. Az ehhez használt mechanizmus egy szabvány-fa definiálásából áll, amelyen az összes szabványban levő minden objektum egy egyedi helyen található. A fának az a része, amelyik az SNMP MIB-t tartalmazza, a 7.32. ábrán látható.

A fa legfelső szintje tartalmazza az összes fontos szabványokkal foglalkozó szervezetet a világon (az ISO szemszögéből nézve), vagyis az ISO-t és a CCITT-t (amit újabban ITU-nak neveznek), plusz a kettő kombinációját. Az *iso* csomópontból négy élet definiáltak, ezek közül az egyik az *identified-organization*, ami az ISO elképzelése arról, hogy esetleg mások is foglalkozgatnak szabványokkal. Az USA-beli Nemzetvédelmi Minisztérium (Dept. of Defense) is ebben a részfában kapott helyet, és az Internethez a DoD az 1-es számot rendelte a hierarchiájában. Az Internet-hierarchiában az SNMP MIB-é az 1-es kód.

A 7.32. ábrán minden élhez tartozik egy címke és egy szám is, így a csomópontokat élek listájával lehet definiálni, a címke(szám) sorozat, vagy számok sorozatának segítségével. Így minden SNMP MIB objektumot a következő formula egy címkéje definiál:


```
{iso identified-organization(3) dod(6) internet(1) mgmt(2) mib-2(1) ...}
```

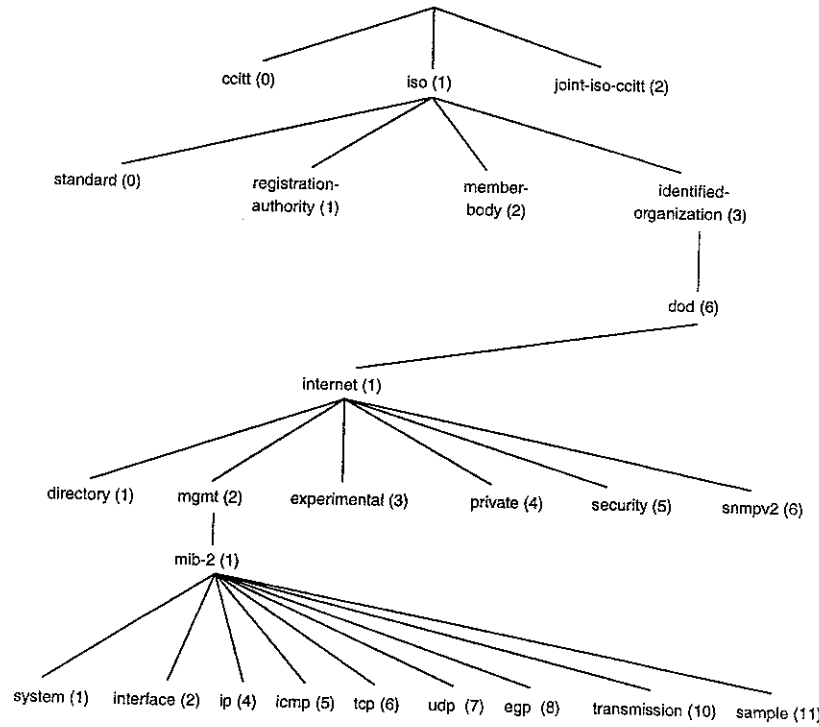
vagy másképpen {1 3 6 1 2 1 ...}. Kevert formulák is megengedettek. Például a fenti azonosító így is írható

```
{internet(1) 2 1 ...}
```

Ezzel a módszerrel bármely szabványban levő bármely objektum reprezentálható egy *OBJECT IDENTIFIER*-el.

Az ASN.1-ben ötféleképpen lehet az alaptípusokból új típust létrehozni. A *SEQUENCE* a típusok egy rendezett sorozata, hasonló a C-beli struktúrához, vagy a Pascal-beli rekordhoz. A *SEQUENCE OF* egy adott típusból álló egydimenziós tömb. A *SET* és a *SET OF* hasonlóak, de nem rendezettek. A *CHOICE* egy uniót képez egy típusokból álló megadott listából. Az SNMP egyik dokumentációjában sem használják a *SET*-ek egyikét sem.

Egy másik mód az új típusok létrehozására már meglevők felcímkézése. A típusok címkézése hasonlít ahhoz, ahogyan a C-ben egy új típus definiálása történik, vegyük



7.32. ábra. *Az ASN.1 objektum elnevezési fájának egy része*

például a *time_t*-t és a *size_t*-t, mindkettő hosszú egész, de különböző környezetben használhatóak. A címkéknek négy kategóriája van: univerzális, alkalmazásra szóló, környezetspecifikus és privát. Minden címke egy azonosítóból és a hozzá rendelt egész számból áll. Például a

```
Counter32 ::= [APPLICATION 1] INTEGER(0..4294967295)
```

és a

```
Gauge32 ::= [APPLICATION 2] INTEGER(0..4294967295)
```

két különböző alkalmazásra szóló típust definiálnak, melyek közül mindkettő 32 bites egész, de más koncepcióban használható. Az előbbi például körbefordulhat, ha eléri a maximumot, míg az utóbbi esetleg egyszerűen mindig a maximum értéket szolgáltatja, amíg nem csökkentik vagy alapállapotba nem állítják.

A felcímkézett típusoknál a bezáró szögletes zárójel után állhat az *IMPLICIT* kulcsszó, ha az azt követő típusra a környezetből egyértelműen következtetni lehet (ez a *CHOICE* esetében például nem igaz). Ennek használata hatékonyabb bitkódolást tesz lehetővé, hiszen a címkét nem kell átküldeni. Ha egy típusban *CHOICE* van, ami két különböző típusra vonatkozik, akkor a címkét mindenképpen át kell küldeni ahhoz, hogy a fogadó tisztában legyen vele, melyikről is van szó éppen.

Az ASN.1 egy összetett makró mechanizmust definiál, amelyet az SNMP-ben sűrűn használnak. Egy makrót, mint prototípust új, külön-külön saját szintaxissal rendelkező változókból, illetve értékekből álló halmaz létrehozására lehet használni. Minden makró (esetlegesen opcionális) kulcsszavakat definiál, amelyek a meghívásnál azonosítják a paramétert (azaz a makró paraméterek azonosítása kulcsszavak segítségével, nem pedig elhelyezkedés szerint történik). Az ASN.1 makrók működésének részletei már túlmennek e könyv határain. Elég annyit megemlíteni, hogy egy makró meghívása a makró nevének megadásával és kulcsszavainak (vagy egy részüknek) felsorolásával, és az ehhez a híváshoz tartozó értékük megadásával történik. A makrók nem futási időben, hanem fordítási időben helyettesítődnek. A későbbiekben majd lesznek még makrókra vonatkozó példák.

ASN.1 Átviteli szintaxis

Az ASN.1 **átviteli szintaxis (transfer syntax)** azt definiálja, hogy miként történik az ASN.1-beli típusok értékeinek egyértelmű bájtsorozattá konvertálása az átvitelhez (és hogyan kell azt a fogadónak egyértelműen visszaállítania). Az ASN.1 által használt átviteli szintaxis neve **BER (Basic Encoding Rules – alap kódolási szabályok).** Az ASN.1-nek vannak más átviteli szintaxisai is, azokat azonban a SNMP nem használja. A szabályok rekurzívak, így egy strukturált objektum kódolása egyszerűen a kódolt komponensek egymás után fűzéséből áll. Így tehát minden objektum kódolása szétbontható kódolt primitív objektumok jól definiált sorozatára. Ezen objektumok kódolását viszont a BER definiálja.

Az alap kódolási szabályok mögött álló vezérelv az, hogy minden átvitelre kerülő érték, legyen az egyszerű vagy összetett, legfeljebb 4 mezőből áll:

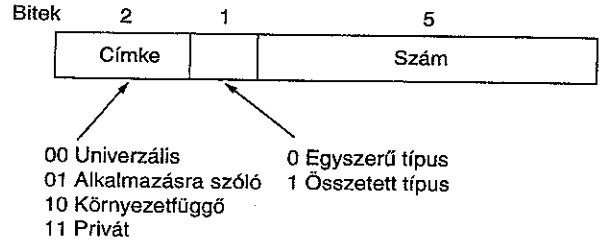
1. Az azonosító (típus vagy címke).
2. Az adat mező hossza bájtokban.
3. Az adat mező.
4. A tartalom-vége jelzés, ha az adat mező hossza ismeretlen.

Az utolsó mező használata ugyan megengedett az ASN.1-ben, de kifejezetten tiltott az SNMP-ben, tehát azzal a feltételezéssel élünk, hogy az adat hossza mindig ismert.

Az első mező a soron következő elemet azonosítja. Három részre van osztva, a 7.33. ábrán látható módon. A legmagasabb két helyi értéken levő bitek a címke típusát adják meg. A következő bit jelzi, hogy egyszerű(0) vagy összetett(1)-e az érték. A címkét jelző bitek értéke rendre 00, 01, 10 és 11 az univerzális, alkalmazásra szóló, környezetfüggő és privát típusnak megfelelően. A maradék 5 bit a címke értékének tárolására használható, ha az érték a 0-tól 30-ig levő tartományba esik. Ha 31 vagy több, akkor az alsó helyi értéken levő 5 bit értéke 11111, és a valódi érték az ezután következő bájtban vagy bájtokban van tárolva.

A címkék kódolására szolgáló szabályokat úgy tervezték, hogy tetszőlegesen nagy számok kezelésére is képesek legyenek. Minden, az elsőt követő azonosító bájtban 7 adatbit található. A legmagasabb helyi értéken levő bit az utolsó kivételével mindegyikben 0. Ennek megfelelően két bájtban maximum 2^7-1 értékű címkét lehet tárolni, három bájtnál ez az érték $2^{14}-1$. Az univerzális típusok kódolása egyszerű. Minden egyszerű típushoz hozzá van rendelve egy kód, amelyek a 7.31. ábra harmadik oszlopában olvashatóak. A *SEQUENCE* és a *SEQUENCE OF* típusok a 16-os kódon osztoznak. A *CHOICE*-nak nincs kódja, mivel minden ténylegesen elküldött értéknek meghatározott típusa van. A többi kódot az SNMP-ben nem használják.

Az 1 bájtos azonosító mező után következő mező azt mondja meg, hogy az adat hány bájtot foglal el. A 128-nál kisebb hosszok egy bájtban vannak kódolva, amelynek a legmagasabb helyi értéken levő bitje 0. Az ennél nagyobbak több bájtot használnak, melyek közül az elsőben a legmagasabb helyi értéken levő bit értéke 1, a többi 7 pedig a hosszat jelzi (maximum 127 bájt). Például, ha az adat hossza 1000 bájt, akkor



7.33. ábra. *Az ASN.1 átviteli szintaxissal küldött adatelem első bájtja*

az első bájt értéke 130, ami azt jelenti, hogy két bájtból álló hossz mező következik. Ezt követi a két bájt a legmagasabb helyi értékűvel kezdve, ezeknek értéke 1000.

Az adat mező kódolása az adat típusától függ. Az egészek 2-es komplemensű kódolással vannak kódolva. Egy 128 alatti pozitív egészhez egy bájt kell, egy 32 768 alatti pozitív egészhez 2 bájt és így tovább. A legmagasabb helyi értékű bájt kerül először átvitelre.

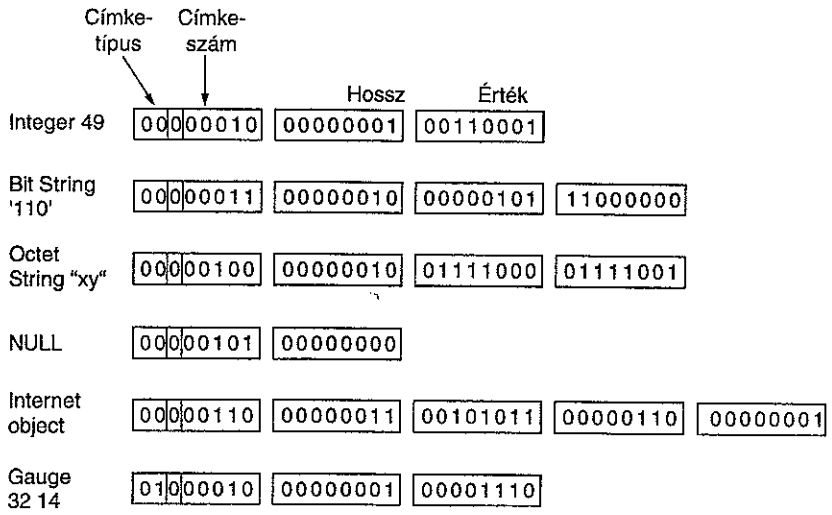
A bit füzérek kódoltjai önmaguk. Az egyetlen probléma a hossz jelzésével van. A hosszat tartalmazó mező azt mondja meg, hogy hány *bájtból* áll az érték, nem azt, hogy hány *bitből*. A megoldásként azt választották, hogy egy bitfüzér előtti bájt megadja az utolsó bájtban nem használt bitek számát (ez 0-tól 7-ig terjedhet). Ily módon a '010011111' 9 bites füzér (hexadecimális) kódoltja a 07, 4F, 80 lenne.

Az oktett füzérek esete egyszerű. A bájtok egyszerűen szabályos felsővég stílusban balról jobbra kerülnek átvitelre.

A null értéket a hossz mezőben levő 0 jelzi. Ekkor gyakorlatilag semmilyen numerikus érték nem kerül átvitelre.

Az *OBJECT IDENTIFIER* az őt reprezentáló egészek sorozataként van kódolva. Például az Internet a {1, 3, 6, 1}. Mindazonáltal, mivel az első szám mindig 0, 1 vagy 2, a második szám pedig kisebb mint 40 (definíció szerint – az ISO egyszerűen nem fogja észrevenni az ajtóján 41-ediknek kopogtató kategóriát), ezért az első két számot *a*-t és *b*-t egy $40a+b$ értékű bájtba kódolják. Az Internet esetében ez a szám 43. Szokás szerint a 127-nél nagyobb számokat több bájton kell kódolni, úgy hogy az első bájt legmagasabb helyi értékén 1-es áll, a maradék 7 pedig a hosszat tárolja.

Mindkét sequence típusnál elsőként a típus vagy a címke kerül átvitelre, ezt követi a kódolt mezők összességének hossza, majd maguk a kódolt mezők. A mezőket sorrendben kell küldeni.



7.34. ábra. *Néhány érték ASN.1-es kódoltja*

Egy *CHOICE* érték kódolása megegyezik az éppen elküldendő adattípus kódolásával.

A 7.34. ábrán látható néhány értékre vonatkozó kódolási példa. A kódolt értékek rendre az *INTEGER* 49, a *BIT STRING* '110', az *OCTET STRING* ''xy'', a *NULL* egyetlen lehetséges értéke, az Internet *OBJECT IDENTIFIER*-e, és a 14 értékű *Gauge32*.

7.3.3. SMI – Felügyeleti adatok struktúrája

Az ezt megelőző részekben megtárgyaltuk az ASN.1 azon részeit, amelyek az SNMP-ben használatosak. Valójában az SNMP dokumentumok máshogyan épülnek fel. Az RFC 1442 először leszögezi, hogy az SNMP adatstruktúrák leírása az ASN.1-es jelölésrendszerrel történik, majd 57 oldalon keresztül részletezi, hogy az ASN.1 szabvány mely részei nem kellenek, és egyúttal egy csomó új (ASN.1-beli) definíciót ad, amelyek viszont kellenek. Gyakorlatilag az 1442-es RFC négy kulcsfontosságú makrót definiál, valamint nyolc új adattípust, amelyek az SNMP-ben sűrűn használatosak. Az ASN.1-nek ez felbővített része az, ami az SNMP adatszerkezetek leírását szolgálja, és amit a suta **SMI (Structure of Management Information – felügyeleti adatok struktúrája)** néven emlegetnek.

Annak ellenére, hogy ez a hozzáállás kissé bürokratikus, azért bizonyos szabályokat nem lehet figyelmen kívül hagyni, ha az az elvárás, hogy százféle gyártó termékei beszéljenek, és tényleg meg is értsék egymást. Ezért tehát essék néhány szó az SMI-ről.

Az SNMP legalsó szintjén a változókat egyéni objektumokként definiálják. Az egymással rokon objektumokat csoportokba szedik, a csoportok pedig modulokba tömörülnek. Léteznek például IP és TCP csoportok. Egy router támogathatja esetleg az IP csoportot, hiszen a felügyelőjét érdekli a csomagvesztések száma. Másrészről viszont egy kisebb router nem feltétlenül támogatja a TCP csoportot, hiszen a forgalomirányításhoz neki nem kell TCP-t használnia. A szándék az, hogyha egy gyártó egy csoportot támogat, akkor minden benne levő objektumot is támogasson. Ha azonban egy gyártó egy modult támogat, akkor nem kell feltétlen minden benne levő csoportot is támogatnia, mivel nem biztos, hogy egy készülékben minden megvalósítható.

Minden MIB modul a *MODULE-IDENTITY* makró meghívásával kezdődik. Paraméterei megadják az implementáló nevét és címét, egy visszatekintést a módosításokról, és más adminisztratív információt. Ezt általában az *OBJECT-IDENTIFIER* meghívása követi, amely megadja a modul nevét a 7.32. ábrán levő elnevezési fában.

Később következik egy vagy több *OBJECT-TYPE* makróhívás, amelyek megnevezik felügyelt változók nevét és tulajdonságait. A változók csoportosítása konvenciókon alapul; az ASN.1-ben, illetve az SMI-ben nincsenek *BEGIN-GROUP* és *END-GROUP* jelzők.

Az *OBJECT-TYPE* makrónak négy kötelező, és négy (esetenként) opcionális paramétere van. Az első kötelező paraméter a *SYNTAX*, amely megadja a változó típusát a 7.35. ábrán felsoroltak közül. A következő megjegyzések segítségével legtöbbjük könnyen megérthető. A 32-es utótag azt jelenti, hogy az implementáló tényleg 32 bites számot szeretne használni, annak ellenére, hogy a legtöbb szóba jövő gép 64

bites CPU-t használ. A mérők abban különböznek a számlálóktól, hogy nem fordulnak körbe, mikor elérik értékük maximumát, hanem megtartják azt. Amennyiben egy router 2^{32} db csomagot vesztett, jobb a 2^{32}–1-et jelentenie, mint a 0. Az SMI szintén támogatja a tömböket, de ebbe most nem megyünk bele. A részletek tekintetében lásd (Rose, 1994) művét.

A deklarált változó által használt adattípus specifikációján kívül az *OBJECT TYPE* makró még három másik paramétert is követel. A *MAX-ACCESS* a változóhoz való hozzáférésről tárol információkat. A leggyakrabban használt értékek az írás-olvasás (read-write) és a csak-olvasás (read-only). Ha egy változó írható-olvasható is, akkor a felügyeleti állomás állíthatja is. Ha csak-olvasható, akkor a felügyeleti állomás olvashatja, de nem állíthatja.

A *STATUS*-nak három lehetséges értéke van. Az aktuális (current) változó összeillik az aktuális SNMP változattal. Az elavult (obsolete) változó nem illeszkedik a jelenlegi változathoz, de volt egy régebbi SNMP változat, amelyikkel összeillett. A kifogásolható (deprecated) változó valahol ezek között helyezkedik el. Valójában elavult, de a bizottság, akik a szabványt készítették, nem merték ezt a nagyközönség előtt kimondani, mert féltek azoknak gyártóknak a reakciójától, akiknek termékei ezt használják. Akárhogy is van, kilóg a lóláb.

A legutolsó kötelező paraméter a *DESCRIPTION*. Ez egy ASCII karakterfüzér, amely leírja, amit a változó csinál. Ha egy hálózatfelügyelő vesz egy gyönyörű csillo-

Név	Típus	Bájt-méret	Jelentés
INTEGER	Numerikus	4	Egész (a jelenlegi implementációkban 32 bites)
Counter32	Numerikus	4	Előjel nélküli 32 bites körbeforgó számláló
Gauge32	Numerikus	4	Előjel nélküli nem körbeforgó érték
Integer32	Numerikus	4	64 bites CPU esetén is csak 32 bites
UInteger32	Numerikus	4	Mint az Integer32, de előjel nélküli
Counter64	Numerikus	8	64 bites számláló
TimeTicks	Numerikus	4	Egy időpillanat óta eltelt idő századmásodpercekben
BIT STRING	Füzér	4	1-től 32 bitig terjedő bit-térkép
OCTET STRING	Füzér	≥0	Változó hosszúságú bájtfüzér
Opaque	Füzér	≥0	Elavult; lefelé kompatibilitás miatt
OBJECT IDENTIFIER	Füzér	>0	Egy a 7.32. ábrán látható egészekből álló lista
IpAddress	Füzér	4	Decimális ponttal elválasztott IP cím
NsapAddress	Füzér	<22	Egy OSI NSAP cím

7.35. ábra. *Az SNMP figyelővváltozókhoz tartozó adattípusok*

gó-villogó új készüléket, majd miután lekérdezte a felügyeleti állomásról, felfedezi, hogy képes a *pktCnt* számontartására, akkor a *DESCRIPTION* lekérdezésével megtudhat valamit arról, hogy tulajdonképpen milyen csomagokat is számol a készülék. Ez a mező kifejezetten arra hivatott, hogy (a számítógéppel szemben) az embernek nyújtson információt.

Az *OBJECT TYPE* deklarációnak egy egyszerű példája a 7.36. ábrán látható. A változó neve *elvesztettCsomagok (lostPackets)*, és bármilyen router vagy más csomagokkal foglalkozó készülékben lehet hasznos. A ::= után következő érték helyezi el e változót a fában.

```
lostPackets OBJECT TYPE
  SYNTAX Counter32              -- egy 32-bites számlálót használ
  MAX-ACCESS read-only          -- a felügyeleti állomás nem változtathatja meg
  STATUS current                -- ez a változó nem elavult (legalábbis egyelőre)
    "Az utolsó indítás óta elveszett csomagok száma"
  ::={experimental 20}
```

7.36. ábra. *Példa egy SNMP változóra*

7.3.4. Az MIB – Felügyeleti adatbázis

Az SNMP által felügyelt változók gyűjteményét az MIB-ben definiálják. A kényelem érdekében ezek az objektumok (jelenleg) tíz kategóriába vannak csoportosítva, a 7.32. ábrán levő *mib-2* alatt levő csomópontoknak megfelelően. (Vegyük észre, hogy az *mib-2* a SNMPv2-nek felel meg, és hogy a 9-es csoport nem létezik többé.) A tíz kategória egy bázist szándékszik nyújtani azoknak a dolgoknak, amelyeket egy felügyeleti állomásnak meg kell értenie. A jövőben biztosan új kategóriákkal és objektumokkal is

Csoport	Objektumok száma	Magyarázat
System	7	A készülék neve, helye és leírása
Interfaces	23	Hálózati interfészek és az általuk mért forgalom
AT	3	Címtranszformáció (kifogásolható – deprecated)
IP	42	IP csomag statisztikák
ICMP	26	Statisztikák a fogadott ICMP üzenetekről
TCP	19	TCP algoritmusok, paraméterek és statisztikák
UDP	6	UDP forgalom statisztikák
EGP	20	EGP statisztikák
Transmission	0	Fenntartva a médiaspecifikus MIB-eknek
SNMP	29	SNMP forgalomstatisztikák

7.37. ábra. *Az Internet MIB-II objektum csoportjai*

bővülni fog, valamint a forgalmazók is szabadon definiálhatnak további objektumokat termékeikhez. A tíz kategóriát a 7.37. ábra foglalja össze.

Annak ellenére, hogy hely hiányában nem tudjuk mind a 175, MIB-II-ben definiált objektumot részletesen tárgyalni, segítségképpen néhány megjegyzés jól jöhet. A system csoport lehetővé teszi a hálózatfelügyelő számára, hogy megállapítsa egy készülékről, hogy mi a neve, ki készítette, milyen hardvert és szoftvert tartalmaz, hol található és mire való. A legutolsó indítás ideje, valamint a gond esetén megkeresendő személy neve és címe szintén megtalálható. Ez az információ azt jelentheti, hogy egy cég szerződtethet egy messzi városban levő másik céget a rendszer felügyeletére úgy, hogy ez utóbbi cég könnyedén kideríthesse, hogyan is néz ki a felügyelendő rendszer, és kit kell megkeresni, ha a különböző készülékkel netán probléma adódik.

Az interfaces csoport a hálózati adapterekkel foglalkozik. Számon tartja a hálózatból elküldött és a hálózatba érkező csomagok és bájtok számát, az eldobott csomagok számát, az adatszórások számát, és a kimeneti sor hosszát.

Az AT csoport az MIB-I-ben volt jelentős, és információt szolgáltatott a címek megfeleltetéséről (pl. Ethernet és IP között). Ez az információ az SNMPv2-ben átkerült a protokollspecifikus MIB-ek közé.

Az IP csoport a csomópontba érkező és onnan kimenő IP forgalommal foglalkozik. Különösen gazdag számlálókban, amelyek a különböző okok miatt (pl. nincs ismert út a célig vagy erőforráshiány) eldobott csomagokat számlálják. A datagram tördeléséről és összeállításról szóló statisztikák is rendelkezése állnak. Ezen elemek mindegyike igen fontos a routerek felügyeletéhez.

Az ICMP csoport az IP hibaüzenetekkel foglalkozik. Alapjában véve minden ICMP üzenethez hozzá van rendelve egy számláló, ami számolja az adott típusból elküldött üzenetek számát.

A TCP csoport a nyitott összeköttetések aktuális és összesített számát, a fogadott és elküldött szegmensek aktuális és kumulatív számát figyeli, valamint számos hibastatisztikát ad.

Az UDP csoport jegyzi az elküldött és fogadott UDP csomagok számát, és az utóbbiak közül mennyi volt ismeretlen port vagy más okok miatt kézbesíthetetlen.

Az EGP csoport azon routerek számára van, amelyek támogatják a külső átjáró protokollt. Számon tartja, hogy hány és milyen csomag távozott, érkezett és helyesen továbbítódott, illetve érkezett és eldobódott.

A transmission csoport a helyet foglalja a médiaspecifikus MIB-ek számára. Például az Ethernet-specifikus statisztikáknak itt lehet a helye. Az MIB-II-beli üres csoport létrehozásának az az értelme, hogy foglalja az {internet 2 1 9} azonosítót ilyesmi célokra.

Az utolsó csoport arra szolgál, hogy magával az SNMP-vel kapcsolatban gyűjtsön statisztikákat. Mennyi üzenet került elküldésre, hogy milyen típusúak voltak és így tovább.

Az MIB-II formailag az 1213-as RFC-ben van definiálva. Az 1213-as RFC nagy részét 175, a 7.36. ábrán olvashatóhoz hasonló makróhívás teszi ki, a tíz kategóriát körvonalazó megjegyzésekkel. Mind a 175 definiált objektumhoz meg van adva az adattípus egy angol nyelvű leírással arról, hogy mire is használható. Az MIB-II-vel kapcsolatos további információk tekintetében ebben az RFC-ben keressen az olvasó.

7.3.5 Az SNMP Protokoll

Láttuk, hogy az SNMP mögött meghúzódó modell egy felügyeleti állomásból áll, amely a most említett 175 változóval, és számos más gyártóspecifikus változóval kapcsolatban kérdezősködik. Utolsó témánk a felügyeleti állomások és ügynökök beszédéhez használt protokoll. Magát a protokollt az 1448-as RFC definiálja.

Az SNMP működése általában abból áll, hogy a felügyeleti állomás kérdést intéz az egyik ügynökhöz, vagy utasítást ad az egyik ügynöknek, amiben bizonyos információkra kíváncsi, vagy az állapot valamilyen formában való frissítését kéri. Ideális esetben az ügynök egyszerűen visszaküldi a kért információt, vagy jelenti, hogy a kérésnek megfelelően frissítette állapotát. Az adatok az ASN.1 átviteli szintaxis használatával vándorolnak. Mindazonáltal, visszajelzésként különböző hibajelzések is előfordulhatnak, mint pl. a Nincs ilyen változó.

Az SNMP hét elküldhető üzenettípust definiál. A kezdeményező által küldhető hat üzenet a 7.38. ábrán látható (a hetedik üzenet a válasz). Az első három üzentre egy változó értéke a válasz. Az első formátum explicit módon megadja a kért változók nevét. A második a következő változót kéri, ezzel lehetővé téve a felügyelőnek, hogy ábécérendben végigkérdezhesse az egész MIB-t (az alapértelmezett az első változó). A harmadik nagy mennyiségű adat átvitelére alkalmas, mint pl. táblázatok átvitelére.

Üzenet	Magyarázat
Get-request	Egy vagy több változó értékét kéri
Get-next-request	Az ezt követő változó értékét kéri
Get-bulk-request	Lekérdez egy nagy táblázatot
Set-request	Egy vagy több változót frissít
Inform-request	Helyi MIB-t leíró felügyelő-felügyelő üzenet
SnmpV2-trap	Ügynök-felügyelő csapda jelentés

7.38. ábra. *SNMP üzenettípusok*

Ezután egy olyan üzenet következik, amely lehetővé teszi a felügyelő számára, hogy frissítse egy ügynök változóit, feltéve persze, hogy az objektum specifikációja ezt megengedi. A következő egy információs üzenet, amely lehetővé teszi egy felügyelő számára, hogy közölje egy másik felügyelővel, hogy melyik változókat tartja számon. Utolsónak következik az az üzenet, amit csapda esetén egy ügynök a felügyelőnek küld.

7.4. Elektronikus levél

Most, hogy megvizsgáltunk néhány segéd protokollt az alkalmazási rétegben, rátérhetünk az igazi alkalmazásokra. Ha valakitől megkérdezzük: „Mit fogsz most csinálni?", ritkán kapjuk a következő választ: „Kikeresek néhány nevet a DNS segítségével". Az emberek többsége azt fogja felelni, hogy elolvassa az e-leveleit vagy a híreket, körülnéz a Hálón, vagy megnéz egy filmet a hálózaton keresztül. A fejezet hátralevő részében részletesen megtárgyaljuk, hogyan is működik ez a négy alkalmazás.

Az elektronikus levél vagy e-levél, ahogyan sok kedvelője nevezi, már két évtizede használatban van. Az első e-levél rendszerek egyszerű fájlátviteli protokollokból álltak, azzal a konvencióval, hogy a levelek (azaz a fájlok) első sora a címzettre utalt. Az idő múlásával egyre világosabbá váltak ezen megközelítés hátrányai. Többek között a következő panaszok merültek fel

1. Kényelmetlen volt több embernek egyszerre levelet küldeni. Az adminisztrátoroknak gyakran kellett ezt tenniük ahhoz, hogy emlékeztetőt küldjenek beosztottjaiknak.

2. Az üzeneteknek nem volt belső szerkezetük, így bonyolult volt a számítógépes feldolgozásuk. Például, ha egy továbbított üzenet be volt ágyazva egy másik üzenetben, akkor nehéz volt azt onnan kihámozni.

3. A küldő sohasem tudta, hogy a levél megérkezett-e vagy sem.

4. Nehéz volt megoldani, hogyha valaki azt szerette volna, hogy míg üzleti célokkal hetekre elutazik, a titkárnője kezelje e-leveleit.

5. A felhasználói felület nem volt egybeintegrálva az átviteli rendszerrel, így a felhasználó először megszerkesztette a fájlt, majd elhagyta a szerkesztőt, és meghívta a fájlátviteli programot.

6. Nem volt lehetséges szöveg, képek, fax és hang keverékéből álló levelek létrehozása és küldése.

Ahogy a tapasztalatok gyűltek egyre kifinomultabb e-levél rendszerek láttak napvilágot. 1982-ben került nyilvánosságra az ARPANET e-levél rendszer javaslata, a 821-es (átviteli protokoll) és 822-es (üzenet formátum) RFC-kben. Ezek lettek később a de facto Internet szabványok. Két évvel ezután a CCITT megfogalmazta az X.400 javaslatát, ami később az OSI féle MOTIS alapját képezte. 1988-ban a CCITT módosította az X.400-at, hogy összehangolja a MOTIS-szal. A MOTIS rendszer jelentette az OSI legfőbb alkalmazását, azt akarták, hogy ez legyen az emberek mindene.

Egy évtizedes versengés során a 822-es RFC-re épülő e-levél rendszerek széles körben elterjedtek, míg az X.400-on alapulóak szépen eltünedeztek. Dávid és Góliát bibliai történetét idézi az, hogy egy rendszer, amit egy maréknyi végzős informatikus

barkácsolt össze, hogyan győzött le egy, a világ minden PTT-je, számos kormánya és a számítógépes ipar nagy része által támogatott, hivatalos nemzetközi szabványt. A 822-es RFC nem azért sikeres, mert jó, hanem azért, mert az X.400 olyan gyengén sikerült, és olyan bonyolult, hogy senki sem tudja rendesen implementálni. Ha választani kell egy a 822-es RFC-n alapuló egyszerű, de működő, illetve egy amúgy csodálatos, de nem működő X.400 levelező rendszer között, a legtöbb szervezet az előbbit választaná. Egy hosszú elmarasztaló jegyzék található az X.400 hibáiról a (Rose, 1993) művének C függelékében. Ennek megfelelően tehát az e-levél tárgyalásánál az Interneten használt 821-es és 822-es RFC-re fogunk koncentrálni.

7.4.1. Architektúra és szolgáltatások

Ebben a részben egy átfogó ismertetést adunk az e-levél rendszerek felépítéséről és arról, hogy mire képesek. Általában két alrendszerből állnak, a **felhasználói ügynökből (user agent),** amely a lehetővé teszi a felhasználók számára az üzenetek olvasását és küldését, valamint az **üzenetkézbesítő ügynökből (message transfer agent),** ami a leveleket eljuttatja a feladótól a címzettig. A felhasználói ügynökök helyi programok, amelyek parancs alapú, menü alapú vagy grafikus módszereket ajánlanak az e-levél rendszerrel való érintkezésre. Az üzenetkézbesítő ügynökök általában rendszer daemonok, amelyek a háttérben dolgoznak és mozgatják a leveleket a rendszerben.

Az e-levél rendszerek általában az alább ismertetésre kerülő öt alapvető funkciót támogatják. A **szerkesztés (composition)** a levelek és válaszok létrehozására utal. Ugyan bármely szövegszerkesztő használható a levél törzsének megírásához, a rendszer azonban segítséget nyújthat a címzésnél, valamint a levélhez csatolt számos, fejlécben található mezővel kapcsolatban. Például egy levél megválaszolásakor az e-levél rendszer kihámozhatja a kapott levélből a feladó címét, és beilleszthetí azt a megfelelő helyre.

A **kézbesítés (transfer)** a levelek feladótól címzettig való eljuttatására utal. Ez nagyrészt a címzettel vagy egy közbülső géppel való kapcsolatfelépítést, az üzenet elküldését és a kapcsolat bontását igényli. Az e-levél rendszernek ezt a felhasználó bevonása nélkül, magától kell elvégeznie.

A **visszajelzés (reporting)** feladata, hogy megmondja a feladónak, hogy mi történt az üzenettel. Sikeres volt-e a kézbesítés? Visszautasították-e a levelet? Esetleg elveszett? Számos alkalmazás létezik, amelyekben a kézbesítés nyugtázása fontos, esetleg hivatalos jelentőséggel is bírhat („Tisztelt Bíróság, sajnos az én e-levél rendszerem nem valami megbízható, így azt hiszem, elveszhetett valahol az elektronikus idézés.").

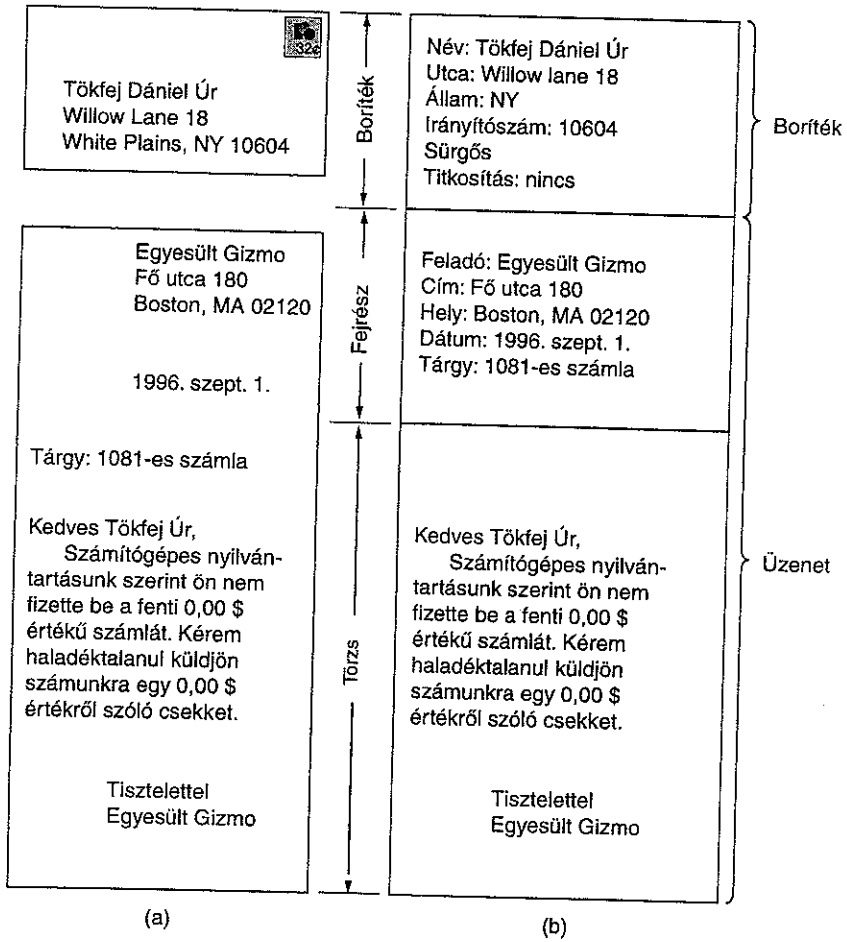
Az érkező levelek **megjelenítése (displaying)** azért szükséges, hogy az emberek el tudják azokat olvasni. Néha konverzióra van szükség, vagy egy speciális megjelenítőt kell meghívni, ha pl. az üzenet egy PostScript fájl vagy digitalizált hang. Egyszerű konverzióval és formázással is próbálkoznak néha.

Az **elrendezés (disposition)** az utolsó lépés, és arra vonatkozik, hogy a címzett mit tesz a kapott üzenettel. A lehetőségek közé tartozik például, hogy eldobhatja olvasás nélkül, vagy elolvasás után, esetleg elmentheti és így tovább. Szintén lehetőség kell

legyen a levelek előhívására, újraolvasására, továbbítására vagy másképp történő feldolgozására.

Ezen alapvető szolgáltatások mellett a legtöbb e-levél rendszer számos más fejlettebb szolgáltatást is kínál. Említsünk meg néhányat ezek közül. Ha valaki elköltözik vagy hosszabb időre távol van, esetleg szeretné átirányítani leveleit, tehát a rendszernek ezt automatikusan el kell tudnia végezni.

A legtöbb rendszer lehetővé teszi a felhasználóknak, hogy postaládákat hozzanak létre a kapott levelek tárolására. Szükség van tehát parancsokra, amelyek segítségével létre lehet hozni vagy megsemmisíteni postaládákat, megnézni tartalmukat, betenni vagy törölni belőlük leveleket és így tovább.



7.39. ábra. *Borítékok és üzenetek. (a) Postai levél. (b) Elektronikus levél*

Nagy iparvállalatok igazgatóinak gyakran szüksége van rá, hogy egyszerre küldjön minden beosztottjának, ügyfelének vagy ellátójának levelet. Ez adott ötletet a **levelezési listákhoz (mailing lists),** amelyek e-levél címekből álló listák. Amennyiben egy üzenet érkezik a levelezési listára, akkor minden listán levő kap egy másolatot róla.

Szintén fontos a tértivevényes e-levél ötlete, hogy a küldő értesítést kapjon a kézbesítésről. Ugyanakkor a kézbesíthetetlen levelekre figyelmeztető automatikus visszajelzésre is szükség lehet. Mindenesetre a küldőnek mindenképpen kell némi visszajelzés az elküldött levelek sorsa felől.

További fejlett szolgáltatás pl. az indigó másolatok, a magas prioritású e-levél, a titkos (titkosított) e-levél, az arra az esetre az alternatív címzettel rendelkező e-levél, mikor a címzett elérhetetlen, és a titkárnők számára a lehetőség, hogy a főnökük e-levelét kezelhessék.

Az e-leveleket manapság már elterjedten használják az iparban a cégen belüli kommunikációban. Lehetővé teszi az akár különböző időzónákban tartózkodó alkalmazottak részére, hogy összetett projekteken dolgozzanak. Azzal, hogy többnyire eltakarják a beosztást, életkort és a nemet az e-levél segítségével lebonyolított eszmecserék egyre inkább az ötletekre és nem a hivatali beosztásra koncentrálódnak. Az e-levél segítségével egy nyári munkára szerződött tanuló briliáns ötletének nagyobb hatása lehet, mint az ügyvezető alelnök egy szegényes ötletének. Egyes cégek úgy tartják, hogy az e-levél 30 százalékkal növelte hatékonyságukat (Perry és Adam, 1992).

A kulcsötlet a modern e-levél rendszerekben a **boríték (envelope)** és a tartalom különválasztása. A boríték magába foglalja az üzenetet. Tartalmazza az üzenet kézbesítéséhez szükséges információkat, mint a címet, a prioritást és biztonsági szintet, amelyek mindegyike az üzenettől teljesen elkülönül. Az üzenetkézbesítő ügynökök a borítékot használják az útvonal meghatározására, a postához hasonlóan.

A borítékon belüli üzenet két részből áll: a **fejrész (header)** és a **szövegrészből vagy törzsből (body).** A fejrész vezérlési információkat tartalmaz a felhasználói ügynökök részére. A szövegrész teljesen az emberi címzettnek szól. A boríték és a szövegrész a különféle levelek esetén a 7.39. ábrán látható.

7.4.2. A felhasználói ügynök

Az e-levél rendszereknek, ahogyan már láttuk is, két alapvető része van: a felhasználói ügynök és az üzenetkézbesítő ügynök. Ebben a részben a felhasználói ügynökről lesz szó. A felhasználói ügynök általában egy program (melyet néha üzenetolvasónak neveznek), ami számtalan az üzenetek létrehozásával, fogadásával, azok megválaszolásával és a postaládák kezelésével kapcsolatos parancsot képes értelmezni. Egyes felhasználói ügynököknek díszes menü- vagy ikonvezérelt felhasználói felülete van, melynek kezeléséhez egér szükséges, mások viszont csak 1 betűs parancsokat várnak a billentyűzetről. Funkcionálisan ezek megegyeznek.

E-levél küldése

Az e-levél küldéséhez a felhasználónak meg kell adnia az üzenetet, a címet és valószínűleg még egy-két más paramétert (pl. prioritási szintet vagy biztonsági szintet) is. Az üzenetet meg lehet írni egy különálló butább vagy okosabb szövegszerkesztővel vagy a felhasználói ügynök beépített szerkesztőjével.

A címet a felhasználói ügynök által érthető formában kell megadni. Számos felhasználói ügynök a *mailbox@location* formában várja a DNS címet. Mivel ezekről már volt szó ebben a fejezetben korábban, ezért itt nem tárgyaljuk ismételten.

Érdemes azonban megjegyezni, hogy más címzési formák is léteznek. A gyakorlatban az X.400 címek teljesen eltérőek a DNS címektől. Ezek *attribútum = érték* párokból tevődnek össze, például,

/C=US/SP=MASSACHUSETTS/L=CAMBRIDGE/PA=360 MEMORIAL DR./CN=KEN SMITH/

Ez a cím megadja az országot, államot, helyet, személyes címet és egy szokványos nevet (Ken Smith). Sok más attribútum is szóba jöhet, tehát annak is lehet levelet küldeni, akinek a neve ugyan ismeretlen, de ismert más elegendő attribútuma (pl. cég és betöltött állás). Sokan úgy érzik, hogy ez a forma nagyságrenddel kényelmetlenebb, mint a DNS nevek használata. Igazság szerint azonban az X.400 tervezői feltételezték, hogy az emberek fedőnevekkel (rövid felhasználó által hozzárendelt karakterláncokkal) fogják azonosítani a címzetteket, így soha nem is fogják látni a teljes címet. Mindazonáltal soha nem volt széles körben elérhető olyan szoftver, amely ezt lehetővé tette, így az X.400-zal levelező embereknek gyakran kellett a fenti címhez hasonló karakterláncokat begépelnie. Ezzel szemben az Internetes e-levelező rendszerek többségében mindig is megvolt a lehetőség fedőneveket tartalmazó fájlok létrehozására.

A legtöbb e-levelező rendszer támogatja a levelezési listákat, tehát levelét a felhasználó egyetlen utasítással elküldheti egy listában felsorolt embereknek egyszerre. Amennyiben a levelezési lista helyben van nyilvántartva, akkor a felhasználói ügynök egyszerűen minden címzettnek elküldhet egy különálló üzenetet. Ha azonban a listát valahol máshol tárolják, akkor a levelek csak ott lesznek szétválasztva. Például, ha egy madárbarát csoportnak van egy *birders* nevű levelezési listája, ami a *meadowlark.arizona.edu*-n üzemel, akkor minden a *birders@meadowlark.arizona.edu* címre küldött levél először eljut az Arizonai Egyetemre, és csak ott válik szét a lista tagjainak megfelelő üzenetekre, bárhol éljenek is a világban. A levelezési lista használói a címből nem tudják megállapítani, hogy ez egy levelezési lista. Lehet akár Gabriel O. Birders professzor személyes postaládája.

E-levelek olvasása

Általában amikor a felhasználói ügynök elindul, mielőtt még bármit is kiírna a képernyőre, megnézi, hogy a felhasználó postaládájában van-e új e-levél. Ezután vagy közli a postaládában levő üzenetek számát, vagy megmutat minden levélből egy rövid, egysoros összefoglalót, miközben parancsra várakozik.

#	Jelzők	Hossz bájtokban	Küldő	Tárgy
1	K	1 030	asw	A MINIX-szel kapcsolatos változások
2	KA	6 348	radia	Megjegyzéseim az általad küldött anyaggal kapcsolatosan
3	K F	4 519	Amy N. Wong	Információkérés
4		1 236	bal	Pályázat határideje
5		103 610	kaashoek	A DCS cikk szövege
6		1 223	emily E.	Mutató egy WWW oldalra
7		3 110	saniya	Bírálatok a cikkhez
8		1 204	dmr	Vá: A hallgatóm látogatása

7.40. ábra. *Példa egy postaláda tartalmára*

A felhasználói ügynök működésének bemutatására példaképpen vegyünk egy tipikus levélkezelő forgatókönyvet. Miután a felhasználói ügynök elindult, a felhasználó egy összefoglalást kér az e-leveleiről. Ekkor egy a 7.40. ábrán láthatóhoz hasonló kép jelenik meg a képernyőn. Minden sor egy levélre vonatkozik. Ebben a példában a postaládában nyolc levél van.

Minden sorban több mező van, amelyek a megfelelő levél borítékjából vagy a fejrészéből származnak. Az egyszerűbb e-levél rendszerekben a programba beépített mezők jelennek meg. A kifinomultabb rendszerekben a felhasználó beállíthatja, hogy mely mezők jelenjenek meg, egy **felhasználói profil (user profile)** megadásával, ami egy a megjelenítési formát tartalmazó fájl. Ebben a példában az első mező az üzenet száma. A második mező, a *Jelzők*, tartalmazhat *K*-t, ami azt jelenti, hogy a levél nem új, hanem már elolvasták és későbbre eltették; *A*-t, ami azt jelenti, hogy a levelet már megválaszolták; és/vagy *F*-et, ami azt jelenti, hogy feladták a levél egy másolatát valakinek. Más jelzők is elképzelhetőek.

A harmadik mező megadja a levél méretét, a negyedik pedig, hogy ki küldte azt. Mivel ez a mező egyszerűen a levélből lett kihámozva, ezért tartalmazhat keresztneveket, teljes neveket, névkezdőbetűket, loginneveket vagy amit a küldő megad. Végül a *Tárgy* mező egy rövid leírást ad a levél tartalmáról. Azok, akik elfelejtenek tárgyat adni leveleiknek, gyakran tapasztalják, hogy a válaszokat nem sietik el.

Miután a fejrészek megjelentek a képernyőn, a felhasználó a rendelkezésre álló parancsok közül bármelyiket végrehajthatja. A 7.41. ábra egy tipikus gyűjteményt mutat. Némely parancs paramétert is igényel. A # jel azt jelenti, hogy a parancshoz az üzenet száma (vagy esetleg több üzenet száma) szükséges. Az *a* betű használata az összes levelet helyettesíti.

Megszámlálhatatlan e-levél program létezik. Az általunk hozott példa a UNIX Mmdf rendszer által használt levelező programot utánozza. A *h* parancs megjelenít egy vagy több levél fejrészt a 7.40. ábrán látható módon. A *c* parancs megjeleníti az aktuális üzenet fejrészét. A *t* parancs kiírja (azaz megjeleníti a képernyőn) a kért üzenetet, vagy üzeneteket. Lehetséges parancsok a *t 3*, ami kiírja a 3-as levelet, *t 4–6*, ami

Parancs	Paraméter	Leírás
h	#	A fejrész(ek) megjelenítése
c		Az aktuális fejrész megjelenítése
t	#	Az üzenet(ek) kiíratása
s	Cím	Üzenet küldése
f	#	Üzenet(ek) továbbítása
a	#	Üzenet(ek) megválaszolása
d	#	Üzenet(ek) törlése
u	#	Korábban törölt üzenet(ek) visszahívása
m	#	Üzenet(ek) átrakása másik postaládába
k	#	Üzenet(ek) megtartása kilépés után
r	Postaláda	Új postaláda elolvasása
n		A következő üzenetre ugrás és kiírása
b		Az előző üzenetre ugrás és kiírása
g	#	Egy adott üzenetre ugrás, kiírás nélkül
e		Kilépés a levelező rendszerből és a postaláda frissítése

7.41. ábra. *Tipikus levélkezelő parancsok*

kiírja a 4.-től a 6.-ig a leveleket, és *t a*, ami kiírja mindet. A következő három parancslevél a küldéssel, és nem vétellel foglalkozik. Az *s* parancs meghív egy szerkesztőt (pl. a felhasználói profilban megadottat), hogy a felhasználó megírhassa a levelet, majd elküldi azt. Helyesírást, nyelvi helyességet és nyelvezetet ellenőrzők ellenőrizhetik, hogy a levél szintaktikusan helyes-e. Sajnos az e-levél programok jelenlegi generációja nincs ellátva olyan ellenőrzőkkel, amelyek ellenőrzik, hogy küldő tisztában van-e vele, hogy mit beszél. Mikor a levélírás befejeződött, a program előkészíti azt az üzenetkézbesítő ügynöknek való átadásra.

Az *f* parancs egy, az elküldéshez használandó cím elkérése után továbbít egy levelet a postaládából. Az *a* parancs kihámozza a levélből a küldő címét, és meghívja a szerkesztőt, hogy a felhasználó megírhassa a választ.

A következő csoport a postaládák kezelésére szolgál. A felhasználóknak általában annyi postaládájuk van, ahány emberrel leveleznek, azokon a postaládákon kívül, amelyek a beérkező levelek tárolására szolgálnak. A *d* parancs kitöröl egy levelet a postaládából, de az *u* parancs ezt visszavonja. (Az üzenet addig nem törlődik ténylegesen, amíg a programból ki nem lép a felhasználó.) Az *m* parancs áttesz egy levelet egy másik postaládába. Ez a szokásos módja a fontos levelek olvasás utáni tárolásának. A *k* parancs a postaládában tartja a levelet még elolvasás után is. Miután a felhasználó elolvasott egy levelet, de nem adta meg explicit módon, hogy tárolódjon is, a levelező program kilépéskor valamiféle alapértelmezett műveletet hajt végre rajta, pl. átteszi egy speciális postaládába. Végül az *r* parancs segítségével el lehet hagyni az aktuális postaládát, és egy másikra lehet váltani.

Az *n*, *b* és *g* parancsok a postaládán belüli mozgásra valók. Általában a felhasználók az 1-es levelet olvassák el először, majd válaszolnak rá, áthelyezik vagy letörlik, és az *n* beírásával következőre ugranak. Ennek a parancsnak az az előnye, hogy a felhasználónak nem kell számon tartania, hogy hol is tart éppen. A *b* használatával lehetőség van a visszafele vagy a *g* használatával egy adott üzenetre való ugrásra.

Végül az *e* parancs kilép az e-levél programból, és elvégzi a szükséges változtatásokat, mint amilyen levelek törlése, vagy más levelek tárolásra való megjelölése. Ez a parancs felülírja a postaládát, kicserélve annak tartalmát.

A kezdők számára készült levelező rendszerekben ezekhez a parancsokhoz a képernyőn megjelenő ikonok tartoznak, így nem kell megjegyezniük, hogy az *a* a levél megválaszolására való. Így inkább arra kell emlékezni, hogy a kis nyitott szájú embert ábrázoló kép nem a megjelenítésre, hanem a válaszolásra való. Ebből a példából világossá kell váljon, hogy az e-levelezés nagyot fejlődött az egyszerű fájlátvitel óta. A kifinomult levelezési rendszerek nagy mennyiségű levél kezelését teszik lehetővé. Az olyan embereknek szemében, mint a szerző, aki (kelletlenül) ezernyi levelet kap és ír évente, ezek a programok felbecsülhetetlen értéket képviselnek.

7.4.3. Üzenet formátumok

Térjünk most át a felhasználói ügynököktől magukhoz a levél formátumokhoz. Először megvizsgáljuk az alap ASCII e-leveleket az RFC 822 használatával. Aztán megvizsgáljuk a 822-es RFC egy multimédia kiterjesztését.

RFC 822

A levelek egy egyszerű borítékból (ami az RFC 821-ben van leírva), néhány fejrész mezőből, egy üres sorból és az üzenetrészből állnak. Minden fejrész mező (logikailag) egyetlen ASCII szövegű sorból áll, amely tartalmazza a mező nevét, egy kettőspontot és a legtöbb mezőnél egy értéket. Az RFC 822 egy régi szabvány, és nem különbözteti tisztán meg a borítékot és a fejrész mezőket, ahogyan azt egy új szabvány tenné. Normális esetben a felhasználói ügynök összerakja a levelet, majd átadja azt az üzenetkézbesítő ügynöknek, amely aztán a fejrész egyes mezőiből összeállítja a tényleges borítékot, ami némileg régimódi keveréke a borítéknak és üzenetnek.

Az üzenetkézbesítéssel kapcsolatos legfontosabb mezők a 7.42. ábrán vannak felsorolva. A *To:* mező a DNS címét tartalmazza az elsődleges címzettnek. Egy üzenetnek több címzettje is lehet. A *Cc:* mező az esetleges másodlagos címzettek címét adja meg. A kézbesítésnél nincs különbség az elsődleges és másodlagos címzettek között. Ez teljesen pszichológiai természetű megkülönböztetés, amely pusztán a levelezőknek lehet fontos, azonban a levelező rendszernek nem. A *Cc:* (Indigó másolat – Carbon copy) elnevezés bevett szokás, azonban kissé elavult, hiszen a számítógépek nem használnak indigót. A *Bcc:* (Vak Indigó másolat – Blind carbon copy) hasonló a *Cc:* mezőhöz, ez a sor azonban kitörlődik minden elsődleges és másodlagos címzettnek küldött másolatból. Ez a sajátosság lehetővé teszi, hogy kívülállóknak is lehessen másolatot küldeni anélkül, hogy azt az elsődleges és másodlagos címzett megtudná.

A következő két mező a *From:* és *Sender:* rendre az üzenet szerzőjét, valamint elküldőjét adja meg. Ezek nem feltétlenül egyeznek meg. Például egy ügyintéző megír

Fejrész mező	Jelentés
To:	Az elsődleges címzett(ek) e-levél címe(i)
Cc:	A másodlagos címzett(ek) e-levél címe(i)
Bcc:	A vak indigó másolatot címzettjének/címzettjeinek e-levél címe(i)
From:	A levél szerzőjének neve
Sender:	A tényleges feladó e-levél címe
Received:	A úton minden üzenetkézbesítő ügynök hozzáad egy ilyen sort a levélhez
Return-Path:	Arra használható, hogy megadjon egy visszafele vezető utat a feladóhoz

7.42. ábra. *Az RFC 822-es üzenetkézbesítéssel kapcsolatos fejrész mezői*

egy levelet, de a titkárnője az, aki ténylegesen elküldi azt. Ebben az esetben az ügyintéző szerepel a *From:* mezőben, míg a *Sender:* mezőben a titkárnő jelenik meg. A *From:* mező kötelező, azonban a *Sender:* mező elhagyható, ha értékük megegyezik. Ezek a mezők abban az esetben kellenek, ha a levél nem kézbesíthető és vissza kell küldeni a feladónak.

A levél útja mentén minden üzenetkézbesítő ügynök hozzáad egy *Received:* mezőt tartalmazó sort a levélhez. Ez a sor tartalmazza az ügynök azonosítóját, a dátumot, az időt és más információkat, amelyek a forgalomirányító rendszerben levő hibák felderítéséhez használhatóak.

A *Return-Path:* mezőt az utolsó üzenetkézbesítő ügynök ragasztja a levélhez, és arra szolgál, hogy megadja a feladóhoz visszavezető utat. Elvileg ez az információ a *Received:* mezőkből is összeállítható (kivéve a feladó postafiókjának nevét), de ezt ritkán teszik, és ez a mező többnyire különben is csak a feladó címét tartalmazza.

A 7.42. ábrán látható mezőkön kívül az RFC 822 levelei tartalmazhatnak még néhány a felhasználói ügynökök vagy az emberek által használt mezőt. A legáltaláno-

Fejrész mező	Jelentés
Date:	Az üzenet küldésének dátuma és ideje
Reply-To:	A választ erre az e-levél címre kell küldeni
Message-Id:	Egyedi üzenetazonosításra használható szám
In-Reply-To:	Annak a levélnek a Message-Id-je, amire ez a válasz
References:	Más kapcsolódó levelek Message-Id-je
Keywords:	A felhasználó által választott kulcsszavak
Subject:	A levél tartalmának rövid összefoglalása az egysoros megjelenítéshez

7.43. ábra. *Néhány, az RFC 822 üzenetek fejrészében előforduló mező*

sabbak ezek közül a 7.43. ábrán látható. Legtöbbjük jelentése könnyen megérthető, ezért nem ismertetjük mindegyiket részletesen.

A *Reply-To:* mezőt akkor használják, ha sem a szerző, sem a feladó nem szeretné látni a választ. Tegyük fel például, hogy egy marketingmenedzser egy e-levelet ír, amely egy új termékről számol be az üzletfeleknek. A levelet a titkárnő küldi el, de a *Reply-To:* mezőben az eladási osztály vezetőjének címe szerepel, aki képes a kérdéseket megválaszolni, és a rendeléseket felvenni.

Az RFC 822 határozottan kijelenti, hogy a felhasználók kitalálhatnak saját használatra új fejrész mezőket, feltéve, hogy azok X-szel kezdődnek. A saját használatú és hivatalos mezők közti konfliktus elkerülése végett garantált, hogy semelyik hivatalos fejrész mező nem fog a jövőben sem X-szel kezdődni. Néhány okostojás egyetemista az *X-Fruit-of-the-Day:* vagy *X-Disease-of-the-Week:*-hez hasonló mezőket ragaszt a levelekhez, amelyek legálisak ugyan, de nem mindig az értelemről tanúskodnak.

A fejrész mezők után következik az üzenetrész. A felhasználók azt írnak ide, amit akarnak. Sokan ASCII rajzokat tartalmazó aláírással, kisebb-nagyobb költőktől származó idézetekkel, politikai megjegyzésekkel, felelősségelhárító nyilatkozatokkal (mint pl. Az ABC részvénytársaság nem felelős azért, amit mondok; nem is érti meg) zárják leveleiket.

MIME – többcélú hálózati levelezéskiterjesztés

Az ARPANET korai szakaszában az e-levelek kizárólag ASCII karakterekből álló angol nyelven írt szöveges üzenetekből álltak. Ehhez tökéletesen megfelelt az RFC 822: megadta a fejrész mezőket, de a tartalmat teljesen a felhasználó tetszésére bízta. Manapság a világméretű Internethez ez a megközelítés már nem elég. Problémák merülnek fel, többek között, a következő üzenetek küldésével:

1. Amelyek ékezetes betűket tartalmazó nyelven íródtak (pl. francia és német).

2. Amelyek nem latin betűkkel íródtak (pl. héber és orosz).

3. Amelyek ábécé nélküli nyelven íródtak (pl. kínai és japán).

4. Amelyek egyáltalán nem is tartalmaznak szöveget (pl. hang és kép).

Az 1341-es RFC-ben javasoltak egy megoldást, ennek frissített változata az 1521-es RFC. Ez a megoldás, amit **MIME-nak (Multipurpose Internet Mail Extensions – többcélú hálózati levelezéskiterjesztés)** neveznek, ma igen elterjedt. A következőkben erről lesz szó. A MIME-vel kapcsolatos további információk tekintetében lásd az 1521-es RFC-t vagy (Rose, 1993).

A MIME alapötlete az, hogy tovább használja az RFC 822 által leírt formát, de a szövegrésznek is struktúrát ad, és kódolási szabályokat definiál a nem ASCII üzenetek számára. Azzal, hogy a MIME üzenetek nem térnek el az RFC 822-től tovább lehet használni a meglevő levelező programokat, valamint protokollokat. Csupán a küldésre

és fogadásra való programokat kell lecserélni, ezt pedig a felhasználók maguk is meg tudják tenni.

A MIME öt új üzenet fejrész mezőt definiál, amelyek a 7.44. ábrán láthatóak. Az első ezek közül egyszerűen megadja az üzenetet fogadó felhasználói ügynöknek, hogy ez egy MIME üzenet, és azt, hogy a MIME melyik verzióját használja. Minden olyan üzenet, amelyben nem szerepel a *MIME-version:* fejrész mező egyszerű angol szöveges üzenetnek számít, és ennek megfelelően kerül feldolgozásra.

A *Content-Description:* fejrész mező egy ASCII karakterlánc, amely megadja az üzenet típusát. Ez a fejrész mező azért kell, hogy a címzett eldönthesse, hogy érdemes-e visszakódolni az üzenetet. Ha a mező azt tartalmazza, hogy: „Fénykép Barbara versenyegeréről", és az illető aki kapja nem kedveli éppen a versenyegereket, akkor valószínűleg eldobja azt, és nem kódolja vissza nagy felbontású színes képpé.

A *Content-Id:* fejrész mező azonosítja a tartalmat. Ugyanazt a formátumot használja, mint a standard *Message-Id:* fejrész mező.

A *Content-Transfer-Encoding:* azt adja meg, hogy a szövegrészt milyen módszerrel csomagolták a hálózati átvitelre, melynek a legtöbb nem betű, szám vagy írásjel karakter áldozatul eshet. Öt séma áll rendelkezésre (plusz egy lehetőség az új sémák számára). A legegyszerűbb séma a sima ASCII szöveg. Az ASCII karakterek 7 biten kódolhatók, és az e-levél protokoll közvetlenült szállítani tudja őket, feltéve, hogy egy sor nem haladja meg az 1000 karaktert.

A következő legegyszerűbb séma ugyanez, csak 8 bites karaktereket használ, azaz minden érték 0-tól 255-ig terjed. Ez a kódolási séma megszegi az (eredeti) Internet protokoll szabályait, azonban használják az Internetnek olyan részein, ahol egy kibővített változata működik az eredeti protokollnak. Azzal, hogy deklarálják ezt a kódolást, még nem lesz legális, de ha külön feltüntetik létezését, az legalább segít megmagyarázni, ha valami gond adódik. A 8 bites üzeneteknek azonban még mindig tartaniuk kell magukat a maximum sorhossz szabályhoz.

Azok az üzenetek, amelyek bináris kódolást használnak még rosszabbak. Ezek tetszőleges bináris fájlok, amelyek nemcsak 8 bitesek, hanem az 1000 karakteres sorhossz határt sem veszik figyelembe. Nincs garancia rá, hogy a bináris üzenetek korrektül érkeznek meg, ennek ellenére sokan küldenek ilyet.

A bináris üzenetek korrekt csomagolási módja a **base64 kódolás (base64 encoding),** vagy ahogyan néha nevezik az **ASCII vértezet (ASCII armor).** Ebben a sémában 24 bites csoportokat 6 bites egységekre tördelnek úgy, hogy minden egység értéke szerint, egy legális ASCII karakter formájában kerül átvitelre. A kód a következő: az „A" 0-nak, a „B" az 1-nek, a „C" a 2-nek stb. felel meg, majd a nagybetűket a 26 kisbetű követi, ezután jön a tíz szám, végül a + jel és a / jel, a 62-nek és a 63-nak megfelelően. Az == és = szekvenciák arra szolgálnak, hogy jelezzék, ha az utolsó csoport csak 16 vagy 8 bitet tartalmaz. A soremelés és kocsi-vissza jelek a kódban nem számítanak, így ezeket tetszés szerint lehet használni a rövid sorok elérése érdekében. Ezzel a sémával biztonságosan lehet tetszőleges bináris szöveget küldeni.

Azoknál az üzeneteknél, amelyek majdnem ASCII formátumúak, és csak néhány nem ASCII karaktert tartalmaznak, a base64 kódolás nem kifejezetten hatékony. Ilyenkor inkább az **idézett nyomtatható karakteres kódolás (quoted-printable encoding)** használható. Ez csak 7 bites, és a 127 feletti értékű karaktereket egy egyenlőségjelet követő két hexadecimális szám kódolja.

Fejrész mező	Jelentés
MIME-Version:	Azonosítja a MIME verziót
Content-Description:	Ember által olvasható leírás az üzenet tartalmáról
Content-Id:	Egyedi azonosító
Content-Transfer-Encoding:	Megadja, hogy a szövegrész milyen módon lett csomagolva az átvitelre
Content-Type:	Megadja az üzenet természetét

7.44. ábra. *A MIME-féle RFC 822 fejrész mezők*

Összefoglalva, a bináris adatot a base64 vagy az idézett nyomtatható karakteres kódolással kell küldeni. Ahol jogos érvek szólnak ezek ellen a kódolások ellen, ott lehetőség van egy felhasználó által definiált kódolást megadására a *Content-Trasfer-Encoding:* fejrész mező segítségével.

A 7.44. ábrán utolsónak feltüntetett fejrész mező tulajdonképpen a legérdekesebb. Az üzenet szövegrészének természetét határozza meg. Az 1521-es RFC hét típust definiál, és mindegyiknek egy vagy több altípusa is van. A típus és altípus perjellel van elválasztva, valahogy így

```
Content-Type: video/mpeg
```

Az altípust is explicit módon meg kell adni a fejrész mezében; nem léteznek alapértelmezettek. A kezdeti típusokat és altípusokat a 7.45. ábra mutatja. Azóta sok új típussal bővült a lista, és folyamatosan tovább bővül, ahogyan az igények diktálják.

Nézzük most végig a típusokat. A *text* típus a rendes szöveget jelenti. A *text/plain* kombináció az általános üzeneteket jelenti, amelyeket további kódolás és alakítás nélkül lehet fogadni és megjeleníteni. Ez teszi lehetővé az általános üzenetek MIME formában való küldését mindössze néhány extra fejrész mező hozzáadásával.

A *text/richtext* altípus lehetővé teszi, hogy a levelet egy egyszerű jelölő nyelvvel lehessen tarkítani. A nyelv egy rendszerfüggetlen módot nyújt a félkövér, a dőlt, a kisebb és nagyobb betűk, a betűtípus, a sorkizárás, az alsó és felső indexelés, és az egyszerű elrendezés jelzésére. A jelzési nyelv az SGML-en, a Szabványos általános jelzési nyelven (Standard Generalized Markup Language) alapul, ami a Világháló HTML formátumának is alapja. Például a

```
Az <bold> idő </bold> elérkezett, mondta a <italic> rozmár</italic>...
```

üzenet a következőképpen nézne ki

Az **idő** elérkezett, mondta a *rozmár*...

A fogadó rendszer feladata, hogy kiválassza a megfelelő megjelenítési módot. Ha a félkövér és dőlt betűk rendelkezésre állnak, akkor használhatók; egyébként valami

szín, villogtatás, aláhúzás, inverz kiemelés használható a hatás elérésére. A különböző rendszerek választhatnak a lehetőségek közül, és ezt meg is teszik.

A következő MIME típus az *image*, ami az állóképek átvitelére alkalmazható. Számos elterjedt, tömörítéssel ellátott vagy anélküli formátum létezik manapság a képek tárolására és átvitelére. Ezek közül kettő, a GIF és a JPEG, hivatalos altípusok, de később kétségkívül mások is azzá válnak.

Az *audio* és *video* rendre a hang és mozgókép átvitelére szolgál. Vegyük észre, hogy a *video* csak a mozgóképet tartalmazza, a hangot nem. Ha hangos filmet szeretnénk átvinni lehet, hogy a hangot és képet a kódolási rendszertől függően külön kell elküldeni. Az egyedüli mozgókép formátum jelenleg a szerény elnevezésű Mozgókép Szakértői Csoport (Moving Picture Experts Group – MPEG) által javasolt formában.

Az *application* egy gyűjtőtípus azon formátumok számára, amelyek a többi típushoz nem hasonlítható feldolgozást igényelnek. Az *octet-stream* egy értelmezhetetlen bájtsorozat. Az ilyen folyamok fogadásakor a felhasználói ügynök valószínűleg azt javasolja a felhasználónak, hogy fájlban tárolja azt, és kér egy fájlnevet. A további feldolgozás ezután a felhasználóra vár.

A másik definiált altípus a *postscript*, ami a PostScript nyelvre utal, amelyet az Adobe Systems cég készített, és amely széles körben elterjedt a nyomtatható dokumentumok leírására. Számos nyomtató rendelkezik PostScript értelmezővel. A felhasználói ügynökök ugyan egyszerűen meghívhatnának egy programot a beérkező PostScript fájlok megmutatására, azonban ez nem veszélytelen. A PostScript egy kész programozási nyelv. Elegendő idő rááldozásával egy kellőképpen mazochista egyén

Típus	Altípus	Leírás
Text	Plain	Formázatlan szöveg
	Richtext	Szöveg egyszerű formázási parancsokkal
Image	Gif	Állókép, GIF formátumban
	Jpeg	Állókép, JPEG formátumban
Audio	Basic	Hallható hang
Video	Mpeg	Film, MPEG formátumban
Application	Octet-stream	Értelmezhetetlen bájtsorozat
	Postscript	Nyomtatható dokumentum, PostScript formátumban
Message	Rfc822	MIME RFC 822-es üzenet
	Partial	Az átvitelhez több részre bontott üzenet
	External-body	Magát az üzenetet a hálózaton keresztül kell elhozni
Multipart	Mixed	Független üzenetdarabok megadott sorrendben
	Alternative	Ugyanaz az üzenet különböző formátumokban
	Parallel	Az üzenetdarabokat egyszerre kell nézni
	Digest	Minden rész egy komplett RFC 822-es üzenet

7.45. ábra. *Az 1521-es RFC-ben definiált MIME típusok és altípusok*

akár C fordító vagy adatbázis-kezelő rendszert is írhat PostScript nyelven. A PostScript üzenetek megjelenítése a bennük levő PostScript program futtatásával történik. Amellett, hogy ez a program megjelenít némi szöveget, elolvashatja, módosíthatja vagy letörölheti a felhasználó fájljait, és más mellékhatásai is lehetnek.

A *message* típus lehetővé teszi, egy üzenet teljes beágyazását egy másikba. Ez a séma például a levelek továbbítására használható. Ha egy teljes 822-es RFC formájú levél beágyazódik egy másikba, akkor az *rfc822* altípus használandó.

A *partial* altípus lehetővé teszi egy beágyazott üzenet több darabra tördelését és külön-külön történő átvitelét (például, ha a beágyazott üzenet túl hosszú). A paraméterek lehetővé teszik a célállomáson a darabok sorrendhelyes összeállítását.

Végül az *external-body* altípus a nagyon hosszú üzenetek küldésénél lehet hasznos (p. mozgóképek). Az MPEG helyett egy FTP cím kerül elküldésre, és a felhasználói ügynök akkor hozhatja el a hálózatról, amikor szükséges. Ez a képesség különösen hasznos, ha egy levelezési listára kell filmet küldeni, ahol kevesen fogják azt ténylegesen megnézni (gondoljunk például az elektronikus kacatlevelekre, amelyek hirdetési filmeket tartalmaznak).

Az utolsó típus a *multipart*, amely lehetővé teszi, hogy egy üzenet ne csak egy részt tartalmazzon, és a részek tisztán elhatárolhatóak legyenek. A *mixed* altípus lehetővé

```
From: elinor@abc.com
To: carolyn@xyz.com
MIME-Version: 1.0
Message-Id:<0704760941.AA00747@abc.com>
Content-Type: multipart/alternative: boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Earth orbits sun integral number of times

Ez a bevezető. A felhasználói ügynök figyelmen kívül hagyja. Minden jót.

--qwertyuiopasdfghjklzxcvbnm
Content.Type: text/richtext

Boldog szülinapot,
Boldog szülinapot,
Boldog szülinapot, kedves <bold> Carolyn </bold>
Boldog szülinapot.

--qwertyuiopasdfghjklzxcvbnm
Content.Type: message/external-body;
  access-type="anon-ftp";
  site="bicycle.abc.com";
  directory="pub";
  name="birthday.snd"

content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--
```

7.46. ábra. *Egy richtext és hang alternatívákat tartalmazó többrészes üzenet*

teszi, hogy minden rész különböző legyen, további előírt struktúra nélkül. Ennek ellentettje az *alternative* altípus, ahol minden résznek különböző formában vagy kódolásban ugyanazt az üzenetet kell tartalmaznia. Például egy üzenet elküldhető egyszerre sima ASCII, richtext, és PostScript formátumban. Egy jól tervezett felhasználói ügynök egy ilyen üzenetet, ha lehet PostScript formában jelenít meg. A második választás a richtext lenne. He ezek közül egyik sem lehetséges, akkor marad a szegényes ASCII szöveg. A részeknek a legegyszerűbbel kell kezdődniük, és egyre bonyolultabbakkal kell folytatódniuk, hogy a MIME előtti felhasználói ügynökkel rendelkező felhasználók is megérthessék (pl. még a MIME előtti felhasználói ügynökkel felszerelt felhasználók is el tudják olvasni a sima ASCII szöveget).

Az *alternative* altípus használható ezenkívül több nyelv esetén is. Ilyen környezetben a Rosetta Kő egy *mutipart/alternative* üzenetnek képzelhető.

A 7.46. ábrán egy multimédiás példa látható. Ahol egy születésnapi köszöntés kerül mind szöveg, mind hang formában átvitelre. Ha a fogadó le tud játszani hangokat, akkor a felhasználói ügynök elhozza a hálózatról a *birthday.snd* hang fájlt és lejátssza azt. Ha nem, akkor csak a dalszöveg látszik majd síri csendben. A részeket két kötőjel, és az azt követő a *boundary* mezőben megadott (felhasználó által definiált) karakterlánc különíti el.

Vegyük észre, hogy a *Content-Type* fejrész mező háromszor fordul elő ebben a példában. Legfelül arra szolgál, hogy jelezze, az üzenet több részből áll, a részekben pedig azok típusát és altípusát adja meg. Végül a második rész szövegrészében meg kell adni az elhozandó fájl nevét. Hogy érzékeltessük ezt a kis különbséget, itt kisbetűket használtunk a fejrész mezőkben, egyébként minden fejrész mező érzéketlen a kis-nagybetűk használatára. A *content-transfer-encoding* fejrész mező ugyanúgy szükséges azon külső szövegtörzs esetén, amely nem 7 bites ASCII kódolású.

Visszatérve a többrészű üzenetek altípusaihoz, még két lehetőség van. A *parallel* altípus azt jelzi, hogy a részeket egyszerre kell „megjeleníteni". Például a filmek gyakran két részből, a mozgóképből és a hangból állnak. A filmek sokkal hatásosabbak, ha ezen részeik egyszerre, nem pedig egymás után kerülnek lejátszásra.

Végül a *digest* altípust akkor használják, ha egy összetett üzenet számos üzenetből álló csomagot tartalmaz. Például az Interneten néhány vitafórum összegyűjti előfizetői leveleit, és egyetlen *multipart/digest* üzenetként küldi őket szét.

7.4.4. Üzenetkézbesítés

Az üzenetkézbesítő rendszer az üzenetek feladótól címzettig való eljuttatásával foglalkozik. Ennek legegyszerűbb módja, egy átviteli kapcsolat létrehozása a forrásgép és a célgép között, amelyen egyszerűen átvándorolhat az üzenet. Miután megvizsgáltuk, hogy ez általában hogyan történik, megvizsgálunk és megoldunk majd olyan helyzeteket, amelyekben ez nem kivitelezhető.

SMTP – Egyszerű levéltovábbító protokoll

Az Interneten belül, az e-levelek kézbesítése úgy történik, hogy a forrásgép kapcsolatot teremt a célgép 25-ös portjával. Ezt a portot egy e-levél daemon figyeli, aki az **SMTP (Simple Mail Transfer Protocol – egyszerű levéltovábbító protokoll)** nyelvet beszéli. Ez a daemon fogadja a beérkező kapcsolatokat, és átmásolja belőlük az üzeneteket a megfelelő postafiókokba. Ha egy üzenetet nem lehet kézbesíteni, akkor a feladó visszakapja a kézbesíthetetlen üzenet első részét.

Az SMTP egy egyszerű ASCII protokoll. A 25-ös porttal való TCP kapcsolatteremtés után a kliens küldő gép megvárja, amíg a szerver fogadó gép meg nem szólal. A szerver azzal kezdi, hogy küld egy sornyi szöveget, amelyben azonosítja magát, és megadja, hogy fel van-e készülve levelek fogadására. Ha nem, a kliens bontja a kapcsolatot és később újra próbálkozik.

Ha a szerver felkészült a levelek fogadására, akkor a kliens megadja, hogy kitől és kinek megy az e-levél. Ha a megadott címzett létezik a célgépen, akkor a szerver szabad utat ad a kliens számára az üzenetküldéshez. Ezután a kliens elküldi a levelet, és a szerver nyugtázza azt. Semmi ellenőrző összeget nem generál, mivel a TCP kapcsolat megbízható bitfolyamot biztosít. Ha van még e-levél, akkor azt is elküldi. Miután mindkét irányban megtörtént a e-levélcsere, a kapcsolat lebomlik. A 7.47. ábrán látható egy, az SMTP által használt számkódokat is feltüntető, mintadialógus a 7.46. ábrán levő levél elküldéséhez. A kliens által küldött sorokat *C:*; a szerver által küldött sorokat *S:* jelöli.

Néhány megjegyzés segíthet a 7.47. ábra megértéséhez. A kliens részéről az első parancs valójában a *HELO*. A *HELLO* két, négybetűs rövidítése közül, ez számos előnnyel rendelkezik versenytársához képest. Azt már senki sem tudja, hogy miért kellett minden parancsnak négybetűsnek lennie.

A 7.47. ábrán levő üzenetnek csak egy címzettje van, tehát csak egy *RCPT* parancs látható. Több ilyen parancsot is ki lehet adni, ha ugyanazon levélnek több címzettje is van. Mindegyikre egyedileg jön nyugta, vagy visszautasítás. Még ha néhány címzettől visszautasítás jön is (mert nem léteznek a célgépen), a többieknek akkor is el lehet küldeni az üzenetet.

Végül, a négybetűs parancsok szintaxisa ugyan szigorúan definiált, a válaszok szintaxisa azonban kevésbé az. Egyedül a számkód számít igazán. Megvalósítástól függően bármilyen karaktersorozat állhat a kód után.

Annak ellenére, hogy az SMTP protokoll jól definiált (az RFC 821-ben), néhány probléma azért előfordulhat. Az egyik probléma az üzenethosszal kapcsolatos. Néhány régebbi program nem tud 64 KB-ot meghaladó üzeneteket kezelni. Egy másik probléma a várakozási időkből (timeout) fakad. Ha a szervernek és a kliensnek más a várakozási ideje, akkor valamelyikük feladhatja a várakozást, míg a másik még dolgozik, ezzel váratlanul megszakítva a kapcsolatot. Végül, néhány ritkán előforduló helyzetben, vég nélküli levélesőt lehet előidézni. Például, ha az 1-es hoszt tárolja az *A* levelezési listát, és a 2-es hoszt tárolja a *B* levelezési listát, valamint mindkettő a másikra vonatkozó utalást tartalmaz, akkor bármelyikre érkezik is levél, az egy véget nem érő e-levél forgalmat fog gerjeszteni.