

AZ INFORMATIKA ELMÉLETE

Buttyán Levente

Vajda István

KRIPTOGRÁFIA ÉS ALKALMAZÁSAI

TYPOTEX

AZ INFORMATIKA ELMÉLETE

Napjainkra az információs technológiák mélyen behatoltak a gazdaságba és a társadalomba. Az információ elektronikus tárolásával, továbbításával és feldolgozásával kapcsolatos feladatok a minden napjai életet is átszövik. Az információs technológiák biztonságossága az alkalmazhatóság alapvető kritériumává vált. Az információ biztonságát algoritmikus, fizikai illetve rendszabályi technikák kombinálásával érhetjük el. A kriptográfia az algoritmikus biztonsági módszerek tudománya.

A tankönyv célja a kriptográfia legfontosabb módszereinek elméleti megalapozása és azok tipikus alkalmazásainak bemutatása. A tananyag gerincét a kriptográfiai építőelemek (primitívek és alap-protokollok) elméleti kifejtése képezi. Az elmélet megértését nagyszámú példa, valamint a tankönyv végén található feladatok és megoldások is segítik. Az elméletet néhány kriptográfiai alkalmazás is demonstrálja az Internet, a vezeték nélküli (mobil) hálózatok valamint az elektronikus kereskedeleml világából.

Buttyán Levente és Vajda István több mint 10 éves oktatói és kutatói tapasztalattal rendelkeznek a kriptográfia területén. E tankönyv anyagát a Budapesti Műszaki és Gazdaságtudományi Egyetemen oktatják műszaki informatikusoknak.



TYPOTEX

ISBN 963-9548-13-8



9 789639 548138

4300 Ft

Tartalom

<i>Előszó</i>	9
I. rész Kriptográfiai primitívek	13
<i>1. Alapfogalmak</i>	15
1.1. Feltétel nélkül biztonságos rejtjelezés: a One-Time-Pad	19
1.2. Feltétel nélkül biztonságos hitelesítés	21
1.3. Feladatok	22
<i>2. Szimmetrikus kulcsú blokkrejtjelezők</i>	25
2.1. Helyettesítéses-permutációs rejtjelezők és a DES	27
2.2. Helyettesítéses-permutációs rejtjelezők tervezése	32
2.3. Differenciális és lineáris kriptanalízis	52
2.4. Algebrai zártsgág és többszörös rejtjelezés	68
2.5. Az AES blokkrejtjelező	72
2.6. Feladatok	75
<i>3. Nyilvános (aszimmetrikus) kulcsú rejtjelezők</i>	79
3.1. Az RSA algoritmus	80
3.2. Az RSA biztonsága	83
3.3. Prímszámok keresése	86
3.4. Elliptikus görbe kriptográfia	89
3.5. Feladatok	94
<i>4. Kriptográfiai hash függvények</i>	99
4.1. Hash függvény fajták és biztonsági kritériumok	99
4.2. A születésnapi paradoxon	105

6 Tartalom

4.3. Bizonyítható biztonságú konstrukciók	106
4.4. Feladatok	110
II. rész Kriptográfiai alapprotokollok	115
<i>5. Blokkrejtjelezési módok</i>	<i>117</i>
5.1. Az ECB mód	118
5.2. A CBC mód	120
5.3. A CFB mód	133
5.4. Az OFB mód	136
5.5. A CTR mód	139
5.6. Összefoglalás	140
5.7. Feladatok	142
<i>6. Üzenethitelesítés</i>	<i>145</i>
6.1. A CBC MAC	147
6.2. Hash függvényre épülő MAC függvények	149
6.3. Feladatok	154
<i>7. Digitális aláírás</i>	<i>157</i>
7.1. Támadások osztályozása	159
7.2. Lenyomat aláírása (a „hash-and-sign” paradigma)	161
7.3. Példák digitális aláírássémákra	163
7.4. Feladatok	165
<i>8. Kulcscsere protokollok</i>	<i>167</i>
8.1. Kulcscsere protokollok osztályozásának szempontjai	168
8.2. Támadó modell	172
8.3. Példák kulcsszállító protokollokra	174
8.4. Példák kulcsmeggyezés protokollokra	189
8.5. Nyilvános kulcs infrastruktúra alapjai	191
8.6. Informális protokoll-tervezési elvek	199
8.7. Kulcscsere protokollok formális ellenőrzése és a BAN-logika	205
8.8. Feladatok	219
<i>9. Partner-hitelesítés</i>	<i>221</i>
9.1. Jelszó alapú partner-hitelesítés	222
9.2. Kihívás-válasz protokollok	226
9.3. Zero-knowledge-protokollok partner-hitelesítésre	228
9.4. Feladatok	232

III. rész Alkalmazások	235
<i>10. Internet biztonsági protokollok</i>	237
10.1. SSL (Secure Socket Layer)	237
10.2. IPSec	255
10.3. PGP (Pretty Good Privacy)	261
<i>11. Mobil hálózatok biztonsága</i>	265
11.1. GSM biztonság	266
11.2. UMTS biztonság	271
<i>12. Elektronikus fizetési protokollok</i>	275
12.1. Elektronikus fizetési rendszerek (EPS) csoportosítása	276
12.2. Hitelkártyás fizetés az interneten: SET	278
12.3. Digitális készpénz: DigiCash	284
12.4. Mikrofizetési protokollok: PayWord	288
IV. rész Fejezetek a bizonyítható biztonság elméletéből	293
<i>13. Alapfogalmak</i>	295
13.1. Bonyolultságosztályok, orákulum, redukció	297
13.2. Egyirányú függvény (One Way Function – OWF)	303
13.3. Csapda egyirányú permutáció	306
13.4. Keménybit	312
13.5. Feladatok	318
<i>14. Véletlen és algoritmikus megkülönböztethetőség</i>	321
14.1. Valószínűség-eloszlások algoritmikus megkülönböztethetősége	322
14.2. Polinomiális időben megkülönböztethetőség	325
14.3. Feladatok	328
<i>15. Álvéletlen-generátor</i>	331
15.1. Álvéletlen-generátor és az egyirányú függvény	332
15.2. Álvéletlen-generátor konstrukció	334
15.3. Feladatok	335
<i>16. Álvéletlen függvény, álvéletlen permutáció</i>	339
16.1. Véletlen függvény, álvéletlen függvény	340
16.2. Álvéletlen függvény konstrukció	341

16.3. Álvéletlen permutáció konstrukció	344
16.4. PRF alkalmazás példák	349
16.5. Feladatok	349
17. Szimmetrikus kulcsú rejtjelező leképezés modelljei	353
17.1. Véletlen függvénytől megkülönböztetés	354
17.2. Kulcsfejtés elleni biztonság	359
17.3. Nyílt szöveg visszafejtő támadás	362
17.4. Üzenet-megkülönböztető támadás	363
18. Biztonságos nyilvános kulcsú rejtjelezés	371
18.1. Szemantikai biztonság	372
18.2. Üzenet-megkülönböztethetetlenség biztonság	374
18.3. Rejtjeles szöveg módosíthatatlanság biztonság	388
18.4. Feladatok	392
19. A véletlen orákulum bizonyítástechnika	393
19.1. <i>ind – cpa</i> -biztonság véletlen orákulum modellben	395
20. Biztonságos digitális aláírás	399
20.1. Biztonságos one-time aláírás	400
20.2. Aláírás az RSA algoritmus felhasználásával	401
21. Üzenethitelesítés (MAC) biztonsága	405
<i>A kitűzött feladatok megoldása</i>	409
Függelékek	439
<i>A – Kapcsolódó szabványok</i>	439
<i>Irodalom</i>	443

Előszó

Az elmúlt 20 év alatt a kriptográfia terén végzett nyilvános kutatás robbanásszerű fejlődésen ment keresztül. Ennek a folyamatnak a hátterében alapvetően az áll, hogy az információs technológiák egyre mélyebben behatoltak a gazdaságokba, azok motorjaivá lettek, s ezeken keresztül az egyes egyén minden napjai életét is átszövik az információ elektronikus tárolásával, továbbításával és feldolgozásával kapcsolatos feladatok. Az információ gyakran érzékeny abban az értelemben, hogy annak illetéktelen megismerése, csalárd célú módosítása anyagi, erkölcsi kár okozására, jogosulatlan előnyszerzésre ad módot. A biztonság az információs technológiák alkalmazhatóságának alapvető kritériumává vált. Az információ biztonságát algoritmikus, fizikai, illetve rendszabályi technikák kombinálásával érhetjük el. A kriptográfia az algoritmikus biztonsági módszerek tudománya.

Tankönyvünk célja a kriptográfia legfontosabb módszereinek elméleti taglalása, s azok tipikus alkalmazásainak bemutatása olyan mélységben, amely – a tankönyv keretei és az adott feladatok önmagukban vett nehézsége mellett – eljuttatja az olvasót a tervezés és minősítés gyakran igen nehéz technikához is. Mindezek alapos elsajátítását számos kidolgozott példa, s feladatok tárba is támogatja. A cél – természetesen – nem kriptográfusok képzése, ugyanakkor a megoldások elméleti hátterének, feltételrendszerének alapos ismerete nélkül információs biztonsági területen bárminemű alkalmazásba fogni vagy létező alkalmazás biztonságát megítélni megalapozatlan, s nem tanácsos. Tankönyvünk törzsanyagának megcélzott olvasóközönsége első sorban a felsőfokú informatikai oktatás hallgatói. A könyv anyaga azonban ezen túlmutat, s néhány speciális elméleti terület (pl. a bizonyítható biz-

tonság elmélete vagy a kriptográfiai protokollok formális analízise), illetve a feladattár nehezebb problémái szakszeminariumok anyagát is alkothatják. Készséggel elismerjük, hogy nem volt – s nem is lehetett – célunk valamennyi élő, izgalmas területet lefedni a kriptográfia elmélete és alkalmazásai teljes spektrumában, s ígykezünk lehetőleg olyan területeken maradni, amelyek valamennyire kötődnek oktatási és kutatási preferenciáinkhoz. A tankönyv alapvetően elméleti indítottatású, s így a biztonságos implementálás kérdéskört nem érinti, bár tudatában vagyunk annak a ténynek, hogy a napi biztonsági problémák nagyrészt implementációs hibából származnak, s nem az elméleti építőelemek hibából. Ugyanakkor meg vagyunk győződve arról, hogy az elméleti háttér alapos megértése hosszabb távra érvényes, biztosabb fogódzót jelent az alkalmazónak az efféle hibák elkerülésére.

A tananyag gerincét a kriptográfiai algoritmusok elméleti taglalása képezi. A kriptográfiai algoritmusok építőelemei a kriptográfiai primitívek és protokollok. A tankönyvben nagy súllyal szerepel ezen építőelemek elméleti megalapozása, azok reprezentánsainak bemutatása, valamint a különböző kapcsolatos algoritmikus támadási módszerek taglalása. A támadások kérdésköre rendkívül fontos a kriptografiában, hiszen a biztonságot a támadásokkal szembeni ellenállóképesség mértéke adja. A bizonyítható biztonság modern elmélete önálló részként szerepel a könyvben. Az elméleti részek megértését nagyszámú példával kívántuk segíteni. Ezen elméleti alapokra építve mélyebben megérthetők az alkalmazási példák, melyeket az internetes és a mobil hálózatokban használt biztonsági protokollok, illetve az elektronikus fizetési protokollok köréből vettük.

A könyv négy nagy részre tagolódik. Az I. rész a kriptográfiai primitíveket taglalja klasszikus felfogásban. Az alapfogalmak bevezetése után a szimmetrikus valamint az aszimmetrikus kulcsú rejtjelezők tervezési módszereit, legfontosabb képviselőit tárgyaljuk. Bemutatjuk a differenciális és a lineáris kriptanalízis alapjait, melyek a ma ismert legerősebb általános támadási módszerek szimmetrikus kulcsú rejtjelezők ellen. A leggyakrabban alkalmazott aszimmetrikus kulcsú rejtjelező, az RSA számos támadását is elemezzük. Az elliptikus görbüken alapuló tervezés elméleti hátterébe is betekintést adunk. Szintén az I. részben taglaljuk még a kriptográfiai tömörítő függvények tervezésének alapjait.

A II. részben a korábban bevezetett kriptográfiai primitívekre épülő legfontosabb kriptográfiai alapprotokollokat mutatjuk be, amelyek már önmagukban is használhatók bizonyos biztonsági szolgáltatások megvalósítására, de sokszor más, komplexebb feladatot ellátó protokollok építőelemeiként ke-

rülnek alkalmazásra. Részletesen tárgyaljuk a blokkrejtjelezők gyakorlatban használt működési módjait, elemezve az egyes módok hátrányait és előnyeit. Bemutatjuk a kriptográfiai tömörítő függvényekre épülő üzenethitelesítési technikákat, az üzenethitelesítő kódok konstrukcióját. Ebben a részben tárgyaljuk a digitális aláírás protokollokat is, részletesen bemutatva azok biztonsági követelményeit és a gyakorlatban használt „tömörít és aláír” parádigma elvét. Végül, de nem utolsó sorban, a partner-hitelesítést és a kriptográfiai kulcsok gondozását támogató protokollok kerülnek bemutatásra, elsősorban a szimmetrikus kulcsok biztonságos cseréjére koncentrálva. Ezek már többszereplős, általában többlépéses üzenetcserét alkalmazó protokollok, s bár szerkezetük könnyen áttekinthető, tervezésük meglepően sok hiba-lehetőséget rejt magában. Az egyes protokollok elleni támadások részletes elemzésén keresztül így betekintést nyerhetünk ezen protokollok tervezésének elméleti hátterébe.

Könyünk III. része kriptográfiai alkalmazásokkal foglalkozik. Természetesen igényes alapossággal a lehetséges alkalmazások teljes spektrumát nem lehet lefedni egy részben, de talán még egy teljes könyvben sem. Ugyanakkor úgy érezzük, hogy a valós alkalmazások tervezése, bevezetése és használata során nyert tapasztalatok igen hasznosak lehetnek a tankönyv által megcéltolt olvasóközönség, az informatikus hallgatók számára, hiszen nagy valószínűséggel valamilyen szinten maguk is hamarosan kapcsolatba kerülnek kriptográfiai alkalmazásokkal. Ezért a III. rész a lehetséges alkalmazásoknak egy általunk érdekesnek és fontosnak tartott részét mutatja be. A bemutatott alkalmazások három témaorbához csoportosítatók: internetes biztonsági protokollok, vezetéknélküli (mobil) távközlő hálózatokban alkalmazott biztonsági protokollok és az elektronikus kereskedelemben használt vagy tervezett kriptográfiai fizetési protokollok.

A IV. rész egyik fő célja a modern kriptográfia bizonyítható biztonság fogalmainak, főbb elméleti eredményeinek, konstrukcióinak megismertetése. Bonyolultság-elméleti megközelítésben kerül bevezetésre az egyirányú függvény, majd erre építve az álvéletlen bitgenerátor, az álvéletlen függvény és álvéletlen permutáció fogalma. Itt kerülnek bevezetésre azon biztonsági fogalmak és támadási modellek, amelyek mellett a bizonyítható biztonság elmélete és technikái bizonyítást adnak algoritmusok támadhatatlanságára. Bemutatjuk a véletlen orákulum modellben történő bizonyítási technikákat is.

A fejezetek túlnyomó többségének végén, a fejezet anyagához kapcsolódó feladatokat tüztünk ki. Igazából akkor sajátította el az olvasó az anyagot, ha megbirkózik a kitűzött feladatokkal is. A legtöbb feladat megoldását a könyv

végén, külön fejezetben közöljük. Azon feladatokat, melyek megoldása nem került bele a könyvbe, *-gal jelöltük meg.

Végül, szeretnénk megköszönni Ádám Zsolt, Árendás Csaba, Bacsárdi László, Debrei Gábor, Erős Tamás, Frajka Tamás, Holczer Tamás, Maschek Ádám, és Patakfalvi Tamás segítségét, akik drága idejüket feláldozva átvasták a könyv kéziratának korábbi változatait, és megjegyzéseikkel, észrevételeikkel segítették a könyv színvonalának emelését. Külön köszönet jár Frajka Tamásnak, aki ezen túlmenően hasznos tanácsokkal segítette munkánkat, és odaadó segítséget nyújtott a kézirat szerkesztésében is. Hálásak vagyunk továbbá Szabó Istvánnak, a könyv lektorának hasznos észrevételeiért, melyeket igyekeztünk figyelembe venni a végső változat elkészítésénél.

Budapest, 2004. február 9.

Buttyán Levente és Vajda István

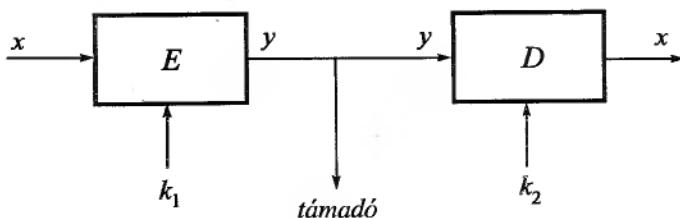
I.

Kriptográfiai primitívek

1.

Alapfogalmak

A kriptológia klasszikusan a titkos kommunikáció tudománya, amelynek két fő ága a kriptográfia és a kriptoanalízis. A kriptográfia azon algoritmikus módszerekkel foglalkozik, amelyek biztosítják az üzenetek (tárolt információk) titkosságát vagy hitelességét. A kriptoanalízis a titok megfejtésére szolgáló módszerek tudománya. Az információk védelmi rendszerének az algoritmikus módszerek általában csak az egyik pillérét jelentik, amelyek megfelelő fizikai védelemmel valamint ügyviteli-rendszabályi eljárások alkalmazásával együtt biztosítják a valóságos rendszerekben az információvédelmet. A kriptográfia az algoritmikus módszerek alapja.



1.1. ábra. A rejtjelezés modellje

A rejtjelezés modelljét az 1.1. ábrán vázoltuk. Tegyük fel, hogy egy üzenetküldő szertné átküldeni üzenetét egy olyan kommunikációs csatornán, amelyről feltételezhető, hogy nem biztonságos, abban az értelemben, hogy egy, a csatornán „hallgatózó” illetéktelen személy képes a csatornán áthaladó biteket megfigyelni. Az üzenetet küldő szertné megakadályozni, hogy

ez a személy illetéktelenül elolvassa az üzenetét, azaz szeretné az üzenetet rejtjelezni.

A rejtjelezéshez az információforrás kimenetét rögzített hosszúságú szeltekre vágjuk, és ezekre sorban alkalmazzuk a rejtjelező kódolást. Amikor a rejtjelező transzformáció az egyes szeltekre azonosan, a megelőző működésétől függetlenül, memóriamentesen történik, akkor blokkrejtjelezésről beszélünk. Blokk rejtjelezés esetén a rejtjelezendő szelletet üzenetblokknak, a rejtjelezettjét rejtett (üzenet) blokknak hívjuk. A teljes üzenet tradicionális neve a nyílt szöveg, a rejtett üzenet a rejtett szöveg.

Az úgynevezett kulcsfolyamatos rejtjelezésnél a rejtjelező kódoló működése időfüggő, a kódoló memóriájában tárolt állapot által is meghatározott. Egy lépésben egy bit vagy néhány bit által meghatározott karakter kerül kódolásra. A továbbiakban alapvetően blokkrejtjelezőkkel foglalkozunk, de találkozni fogunk a kulcsfolyamatos rejtjelezők példáival is, mint az alábbiakban ismertetésre kerülő one-time-pad, vagy kimenet-visszacsatolásos módban (OFB mód) alkalmazott blokk rejtjelező.

Legyen a csatornán átküldendő x üzenet egy blokk méretű, amelyet egy blokk méretű y rejtett üzenetbe kódolunk:

$$y = E_{k_1}(x)$$

ahol E_{k_1} egy k_1 paraméterű rejtjelező (kódoló) transzformáció, amely bijektív a k_1 paraméter tetszőleges rögzített értéke mellett. A k_1 bináris vektor a rejtjelezés kulcsa, az információ, amely meghatározza (kiválasztja) az aktuális rejtjelező transzformációt. A kódoló transzformáció inverze a D_{k_2} dekódoló transzformáció, amely egy y rejtett üzenetet egyértelműen leképez egy x üzenetbe egy k_2 dekódoló kulcs felhasználásával:

$$x = D_{k_2}(y)$$

Ha $k_1 = k_2$, akkor szimmetrikus kulcsú (konvencionális vagy titkos kulcsú) rejtjelezésről beszélünk. A közös titkos kulcsot a kódoló és a dekódoló között a rejtjelezést megelőzően biztonságosan ki kell cserélni.

A $k_1 \neq k_2$ eset az aszimmetrikus kulcsú rejtjelezés. Az 1970-es évek vége től kezdődően rohamosan fejlődik az ide tartozó nyilvános kulcsú rejtjelezők (és általában az úgynevezett nyilvános kulcsú algoritmusok) családja. A nyilvános kulcsú titkosítás alapgondolata, hogy megoldható egy titkos üzenetküldés anélkül, hogy ezt megelőzően a kommunikáló partnerek bármiféle titkos kulcsot cseréltek volna egymással.

Minden rendszerbeli résztvevőhöz tartozik egy kulcspár: a $k_1 = k^P$ nyilvános kódoló kulcs (nyilvános kulcs), valamint a $k_2 = k^S$ titkos dekódoló kulcs (titkos kulcs). Az egyes felhasználók maguk generálhatják a (k^P, k^S) kulcs-pártukat, amelyből a nyilvános részt közzéteszik, a másikat titokban tartják. Egy A, B, C, \dots résztvevőket tartalmazó rendszerben a $k_A^P, k_B^P, k_C^P, \dots$ nyilvános kulcsokat egy nyilvános kulcstárba teszik le, amelyet bárki olvashat. Ha A szeretne egy x üzenetet rejtjelezetten küldeni B -nek, akkor kiolvassa nyilvános kulcsok tárából a k_B^P kódoló kulcsot és az

$$y = E_{k_B^P}(x)$$

rejtett üzenetet küldi B -nek, ahol k_B^P a B résztvevő nyilvános kódoló kulcsa. Ezen y rejtett üzenetből

$$x = D_{k_B^S}(y)$$

dekódolással nyeri ki B az x üzenetet. A kódolás a nyilvános kulcs ismeretében „könnű” feladat, míg a dekódolás a rejtett kulcs ismeretének hiányában gyakorlatilag nem végrehajtható („nehéz feladat”).

Rejtjelezésre (és kriptográfiara) azért van szükség, mivel feltételezhető egy támadó. Alapfeltételezés a támadó ismereteivel kapcsolatosan az, hogy a titkos k kulcs kivételével a kódolás, dekódolás alkalmazott algoritmusát teljes részletességgel ismeri. Ez a feltételezés azt is jelenti, hogy nem függhet a rejtjelezés nyújtotta biztonság attól, hogy eltitkoljuk az algoritmusunkat. Egyszerűen szólva mindenki rendelkezésére állhat, például, ugyanaz a szoftver implementáció. Súlyos történelmi példák vannak ezen feltétel figyelmen kívül hagyására. Ez azt is jelenti, hogy egy rejtjelező algoritmus által nyújtott védeettség nem haladhatja meg a (titkos) kulcsa védeettségének mértékét. A (titkos) kulcs tehát az a relatíve kisméretű titkos információ, amely gyorsan és kívánatosan megfelelő időközönként cserélhető, míg a rejtjelező rendszer többi elemét hosszabb ideig nem szükséges változtatni.

A támadó által alkalmazható módszerek közül csak az algoritmikus típusú támadási módszereket tekintjük a továbbiakban (emellett elképzelhetők további módszerek, pl. betörés, megvesztegetés, szabotázs stb.). Az algoritmikus típusú támadásnak passzív és aktív módját különböztetjük meg, amelyet a nyilvános csatornán hajt végre a támadó.

Passzív módszer a már említett lehallgatás, amikor a támadó a nyilvános csatornán áramló rejtett üzenetek sorozatának birtokába jut. A támadó célja az, hogy egy megfigyelt rejtjelezett kapcsolatból kinyert információk felhasználásával algoritmikus támadást indítson az üzenet vagy az aktuális

titkos kulcs meghatározására. Ezt az eljárást szokás rejtjelfejtésnek nevezni, amelynek eszköztárát a kriptoanalízis adja.

A passzív típusú támadásokat azok növekvő ereje szerint klasszikusan a következő kategóriákba soroljuk:

1. Támadás azonos kulccsal kódolt rejtett üzenetek birtokában, amit rejtett szövegű támadásnak nevezünk (ciphertext only attack).
2. Támadás azonos kulccsal kódolt nyílt-rejtett párok birtokában, amit ismert nyílt szövegű támadásnak hívunk (known plaintext attack).
3. Támadás abban az esetben, ha a támadó maga választhatja meg a nyílt (rejtett) üzeneteket, amelynek rejtett (nyílt) párját látni szeretné, amit választott szövegű támadásnak nevezünk (chosen text attack).

Aktív támadási módszer a rejtett üzeneteknek vagy részeinek csatornában történő törlése, kicserélése, módosítása, amelynek célja a dekódolt üzenet támadó szempontjából kedvező, észrevétilen módosítása (üzenetmódosítás). Egy másik aktív támadási módszer az, amikor a támadó megpróbálja egy legális felhasználó szerepét eljátszani azért, hogy valamely másik legális résztvevőtől információt csaljon ki (megszemélyesítés). Még számos aktív támadási módszer ismeretes a rejtjelkulcsok, illetve a nyílt üzenet illetéklen megismerésére, melyek az adott alkalmazásoktól függnek (pl. program-beépülés, a kulcs hordozó olvasója és a felhasználói processz közötti kommunikáció lehallgatása, titkos kulcsokra vonatkozó következtetés áramfelvétel méréséből vagy futási idők analizálásából, stb.).

Egy üzenet titkossága azt jelenti, hogy csak a kívánt partner számára rekonstruálható annak nyílt tartalma. Egy dekódolt üzenet hitelessége azt jelenti, hogy a kódolt üzenet módosítatlanul, eredeti állapotában érkezett meg a dekódolóhoz, s a tartalmából kiderülő partner küldte. Szemléletes példával illusztrálva: ha a postás felbontja a levelet, akkor annak tartalma már nem titok, de ha a levél tartalmát nem módosítja, s úgy kézbesíti, az üzenet még hiteles marad.

Azt mondjuk, hogy a támadó feltörte a titkosító algoritmust, ha „gyorsan” meg tudja állapítani egy lehallgatott üzenet nyílt tartalmát, függetlenül attól, hogy éppen melyik kulcsot alkalmazzák. A gyorsaság olyan időintervallumot jelent, amelyen belül a támadó sikeresen használhatja céljaira a megszerzett információt.

A rejtjelezés célja alapvetően a passzív támadások megakadályozása. Az aktív típusú támadásokat algoritmikus eszközökkel nem tudjuk megakadályozni, de megfelelő kriptoprotokollok alkalmazásával a támadást észreve-

hetővé tehetjük. A protokollok általában véve egy előre meghatározott üzenetcsere folyamatot jelentenek, amelyet kettő vagy több partner bonyolít le kooperatívan valamely feladat végrehajtására. A kriptográfiai protokollok építő elemként használják (többek között) a rejtjelező kódolót, s biztosítják a kapcsolat védett felépülését, a kommunikáció alatti aktív támadások észlelhetőségét, összességében garantálják a partnerek és üzenetfolyamataik hitelességének ellenőrizhetőségét.

Azt mondjuk, hogy egy biztonsági algoritmus feltétel nélkül biztonságos, ha a támadó tetszőleges számítási kapacitás (erőforrás) mellett sem képes feltörni azt. Ennek ellenére a feltételes (algoritmikus) biztonság, amikor az algoritmus mindaddig biztonságos, amíg a támadó erőforrása egy megadott korlát alá esik. minden gyakorlati biztonságot nyújtó rejtjelezés törhető ki-merítő kulcskereséssel. Az egyetlen ismert feltétel nélkül biztonságos rejtjelező algoritmus az egyszer használatos kulcsú rejtjelező, a *one time pad* (*OTP*). Ez tehát akkor sem törhető, ha végig tudnánk próbálni a kulcstér összes elemét.

1.1. Feltétel nélkül biztonságos rejtjelezés: a One-Time-Pad

Jelölje X és K az üzenet és a kulcs valószínűségi változókat, amelyek egy realizációja x és k az aktuális üzenet és kulcs. Természetes feltevés, hogy X és K független valószínűségi változók. Az Y rejtett üzenet valószínűségi változót a kódoló transzformáció definiálja, mely legyen a következő:

$$Y = (X + K) \bmod 2$$

ahol X, Y, K bináris N bites vektorok, s az összeadás koordinátánkénti mod 2 művelettel történik. K egyenletes eloszlású az N bites vektorok halmazán. Ez a kódolás a One-Time Pad.

1.1. Definíció (Tökéletes titkosítás). *Akkor beszélünk tökéletes titkosításról, ha az X és Y valószínűségi változók statisztikailag függetlenek, azaz kölcsönös információjuk zérus, $I(X, Y) = 0$.*

Lehallgató támadót tételezünk fel, azaz csak rejtett üzenetek állnak a támadó rendelkezésére. A rejtjelezés tökéletessége azt jelenti, hogy ezen támadó a megfigyelt rejtett szövegek alapján, erőforrásától függetlenül nem képes a nyílt szöveg megfejtésére.

1.2. Tétel. *A One-Time Pad tökéletesen titkosító algoritmus.*

Bizonyítás: Azt kell belátnunk, hogy a One-Time Pad esetén X és Y függetlenek:

$$\begin{aligned}\Pr\{Y = y|X = x\} &= \Pr\{X + K = y|X = x\} \\ &= \Pr\{K = y - x|X = x\} \\ &= \Pr\{K = y - x\} \\ &= 2^{-N},\end{aligned}$$

ahol felhasználtuk X és Z függetlenségét. Innen

$$\begin{aligned}\Pr\{Y = y\} &= \sum_x \Pr\{X + K = y|X = x\} \Pr\{X = x\} \\ &= 2^{-N} \\ &= \Pr\{Y = y|X = x\}\end{aligned}$$

adódik, következésképpen X és Y függetlenek. \square

Vegyük észre, hogy X és Y függetlenségéhez nem kellett semmit sem feltételezni X eloszlásáról, továbbá azt, hogy Y eloszlása egyenletes lesz. Azonban ahhoz, hogy a szóban forgó kódolás realizálható legyen, minden egyes üzenet kódolásához véletlenül sorsolt új kulcs szükséges. Ezt az állítást pontosítja a következő téTEL, amely szerint a tökéletes titkosítás szükséges feltétele az, hogy a K kulcs entrópiája ne legyen kisebb, mint az X üzenet entrópiája:

1.3. Tétel. Tökéletes titkosító algoritmus esetén

$$H(K) \geq H(X)$$

Bizonyítás: Az entrópia tulajdonságainak felhasználásával

$$H(X) = H(X|Y) + I(X;Y) = H(X|Y) + 0 = H(X|Y)$$

továbbá

$$H(X|Y) \leq H(X, K|Y) = H(K|Y) + H(X|Y, K) = H(K|Y) \leq H(K),$$

ahol felhasználtuk az $X = D_k(Y)$ determinisztikus leképezés alapján fennálló $H(X|Y, K) = 0$ egyenlőséget. \square

A K kulcs valószínűségi változó egyenletes eloszlását figyelembe véve

$$H(X) \leq |K|,$$

ahol $|K|$ a bináris kulcs hossza. Gyakorlati következmény, hogy érdemes tömöríteni az információforrás kimenetét az OTP kódolást megelőzően, hogy minimalizáljuk a felhasznált kulcsbitek számát.

1.2. Feltétel nélküli biztonságos hitelesítés

Egyszer használatos kulcsokkal a feltétel nélküli biztonságos hitelesítés feladat is megoldható. Tegyük fel, hogy A és B résztvevők közti párkommunikációban az A által B -nek küldött x üzenet csatornabeli módosítását szeretnékn detektálhatóvá tenni B oldalon. Ehhez egy $F(x, k)$ kriptográfiai ellenőrzőösszeggel kiegészítjük az üzenetünket az A oldalon, ahol F az ellenőrzőösszeg képző függvény, amelynek bemenete az x üzenet és a k kulcs. A B oldal a kulcs és az üzenet ismeretében újragenerálja az ellenőrzőösszeget, s egybeveti azt a vett ellenőrzőösszeggel. A két ellenőrzőösszeg különbözőssége jelenti a támadás detektálását. Tekintsük az alábbi F leképezést:

$$F(x, [a, b]) = ax + b \pmod{p}, \quad (1.1)$$

ahol $0 \leq x, a, b < p$, a és b az egyszer használatos k kulcs részei, továbbá p egy nyilvános nagy prímszám. A támadó a csatornában megfigyeli az $[x, F(x, k)]$, hitelesítő kódolással ellátott üzenetet, s megpróbálja úgy módosítani, hogy az észrevétlen maradjon a vételi oldalon. Ennek elérésére szerezné egy csaló x' üzenethez a helyes

$$W = ax' + b \pmod{p} \quad (1.2)$$

ellenőrzőösszeget előállítani. Nem nehéz látni, hogy W különböző lehetséges értékeihez ($0 \leq W < p$) egyértelműen létezik a, b pár (kulcs), amely kielégíti a (1.1), (1.2) egyenletek rendszerét. Mivel a kulcs választása egyenletes eloszlású, a támadó legjobb stratégiája is csak az lehet, hogy $1/p$ sikervalószínűséggel találhat a helyes W ellenőrzőösszegre.

Vegyük észre, hogy teljesen analóg ezen aktív támadó helyzete az OTP rejtjelezés passzív támadójáéhoz. Az OTP esetén a támadó a kulcsrétek méreteinek reciproka valószínűséggel találhatott az üzenetre, mivel tetszőleges üzenet egyértelműen meghatározott egy kulcsot, amely utóbbi viszont a támadó számára véletlenül került kiválasztásra.

Az (1.1) ellenőrzőösszeg fenti aktív támadása üzenetmódosító támadás. Mi történne, ha megszemélyesítő támadással próbálkozna a támadó, azaz A nem küld üzenetet B -nek, hanem a támadó próbálna egy csaló x üzenetet a nevében elküldeni? Ekkor hiányzik az (1.1) megkötés, s az (1.2) egyenlet bal

oldalát szeretné helyesen megválasztani. A helyes F leképezés az x üzenetet a véletlenszerűen választott kulccsal véletlenszerű ellenőrzőösszegbe képezi, azaz a támadó szemében a $\{0, 1, \dots, p - 1\}$ halmazból egy elem (a helyes ellenőrzőösszeg) egyenletes eloszlással kerül kisorsolásra. Következésképpen a megszemélyesítő támadás sikervalószínűsége szintén $1/p$.

* * *

Láttuk tehát, hogy létezik ideális megoldás mind a rejtjelezés, mind az üzenenethitelesítés feladatra. Mi indokolja ezek után a probléma további vizsgálatát, azaz mi szükség van a könyünk további fejezeteire? A válasz: ezen megoldások a gyakorlat számára nem kielégítők. Az egyszer használatos kulcs a gyakorlati alkalmazásokban szokásos üzenetforgalom, résztvevők nagy száma (hálózatok) mellett megoldhatatlan, irreálisan nagy költségű lenne. Az OTP kódolás védetlen rejtjeles üzenetet módosító támadás ellen. Az (1.1) szerinti ellenőrzőösszeg bitmérete az üzenetével megegyező, s ez a gyakorlatban nem megengedhető. Ezenfelül számos további biztonsági problémához is szükséges algoritmusok kidolgozása, mint például a letagadhatatlanság problémája.

1.3. Feladatok

1.1. Feladat. Tekintsük az $y = ax + b \pmod{N}$ betűnkénti lineáris rejtjezést, ahol N az ábécé mérete.

1. Hány lineáris transzformáció van, ha $N = 26, 27, 30$?
2. x a transzformáció fixpontja, ha $y = x$ teljesül. Legyen $a \neq 1$. Mutassuk meg, hogy ha N prímszám, akkor pontosan egy fixpont van!
3. Adjunk meg olyan N értéket, amelyre nincs fixpontja a transzformációnak.

1.2. Feladat. Egy egyszerű lineáris rejtjelezőt konstruálunk az

$$y = ax + b \pmod{N}$$

lineáris transzformációval, ahol a és b a kulcs részei, x a nyílt szöveg, y a rejtejt szöveg, továbbá N az ábécé mérete.

1. Adja meg a kulcstér méretét, ha $N = 26$!

2. A kulcsot szeretnénk megfejteni. Tegyük fel, hogy a forrás ideálisan tömörített, s így véletlen forrásként modellezhető. Rejtett szövegű támadásban is gondolkodhatunk?
3. Milyen információt kellene birtokolnunk egy véletlen forrás esetén a sikeres támadáshoz?
4. Mit mondhatunk az esetben, ha a forrás írott szöveg, s ismerjük a karakterek gyakoriságát?

1.3. Feladat. Írott szövegek rejtjelezését betűpárokra alkalmazott lineáris rejtjelezéssel végezzük: $y = ax + b \pmod{N^2}$, ahol a és b a kódoló kulcs elemei, $(a, N) = 1$, továbbá N az ábécé mérete. Az ábécé elemeit a $\{0, 1, \dots, N-1\}$ halmaz elemeinek feleltetjük meg. A nyílt és rejtett szöveg, azaz x illetve y a $\{0, 1, \dots, N^2-1\}$, halmzból veszi értékét, olyan módon, hogy egy (u, v) forrás-betűpárt az $uN + v$ egészbe képezve áll elő egy x nyílt szöveg. Az y rejtett szöveget továbbítás előtt visszaképezzük betűpárba. A betűpárok összefogásától azt várjuk, hogy a rejtett szövegben az egyenletesebb betűpár gyakoriság érvényesüljön a gyakran nagyon inhomogén betűgyakoriság esetén is, ezáltal csökkenve egy statisztikai alapú támadás esélyét. Helyes-e ezen várakozásunk?

1.4. Feladat. Tekintsük a következő kulcsfolyamatos rejtjelezést. A nyílt szöveg (x), a rejtett szöveg (y), továbbá a kulcs (k) karakterei az $N = \{0, 1, 2, \dots, n-1\}$ halmzból származnak. A k kulcs az alábbi rekurzióval definiált sorozat:

$$k_i = \begin{cases} r & \text{ha } i = 1 \\ r \cdot k_{i-1} + x_{i-1} \pmod{n} & \text{ha } i > 1 \end{cases}$$

ahol r a kódolás előtt sorsolt véletlen elem az N halmzból, ami közös titkot képez a kommunikáló felek között. Az $x = x_1, x_2, \dots, x_i, \dots$ nyílt szöveget az $y = y_1, y_2, \dots, y_i, \dots$ rejtett szövegbe az $y_i = x_i + k_i \pmod{n}$, $i > 1$ transzformációval kódoljuk.

1. Ideális-e ez a rejtjelezés?
2. Adjon meg választott nyílt szövegű támadást a rejtjelező ellen.
3. Ha a kommunikációs csatornában az y_i karakter meghibásodik, mi a következménye a dekódoló kimenetén?

1.5. Feladat. Kriptográfiai alkalmazásunkban egy véletlen bitsorozatra van szükségünk, de csak hamis pénz áll a rendelkezésünkre, amely nagyobb valószínűsséggel esik egyik oldalára. Hogyan oldaná meg a problémát?

1.6. Feladat*. Rendeljünk páros gráfot egy rejtjelező kódoló leképezéshez a következőképp: a gráf pontjainak egyik színhalmaza feleljen meg az üzenet blokkoknak, a másik színhalmaz a rejtjelezett blokkoknak, ahol két pont között akkor vezet él, ha létezik olyan kulcs a kulcstérben, amely a kapcsolatos üzenet-rejtjeles üzenet párokat köti össze.

1. Igazolja, hogy egy rejtjelezés akkor és csak akkor tökéletes, ha páros gráf-jában ezen pontpárok között azonos számú él vezet.
2. Mi a kapcsolat a tökéletes titkosítás és a latin négyzetek között?

1.7. Feladat. Egy F_r bináris véletlen forrás kimenetén 0,25 valószínűsséggel jelenik meg 0 bit. Ennek felhasználásával szeretnénk rejtjelkulcsot generálni. Tudjuk, hogy a rejtjelezendő F_x forrásunk felére tömöríthető. Ideális forrás-kódolást tudunk végezni. Lehetőséges-e tökéletes rejtjelezés, ha F_r sebessége 5 kbit/sec, F_x sebessége 8 kbit/sec?

1.8. Feladat. Tekintsük a következő rejtjelező kódolást: az üzenetek tere $X = \{a, b\}$, a kulcsok tere $K = \{k_1, k_2, k_3, k_4, k_5\}$, a rejtett üzenetek tere $Y = \{1, 2, 3, 4, 5\}$. A kódolás az 1.1. táblázat szerinti.

k	a	b
k_1	1	2
k_2	2	4
k_3	3	1
k_4	5	3
k_5	4	5

1.1. táblázat.

Például $E_{k_3}(a) = 3$. A kulcs valószínűségeloszlása $P_K = \{2/5, 1/5, 1/5, 1/10, 1/10\}$, az üzenet valószínűségeloszlása $P_X = \{1/3, 2/3\}$. Tökéletes rejtjelezést valósítunk-e meg?

2.

Szimmetrikus kulcsú blokkrejtjelezők

A blokkrejtjelező egy $E : \{0, 1\}^n \times \{0, 1\}^k \rightarrow \{0, 1\}^n$ transzformáció, ami az n bites nyílt blokkot és a k bites kulcsot az n bites rejtett blokkba transzformálja. Rögzített kulcs mellett, az E transzformációnak természetesen invertálhatónak kell lennie, különben a rejtjeles blokkból nem tudnánk egyértelműen visszaállítani a nyílt blokkot. A fent definiált blokkrejtjelezőt tehát úgy is tekinthetjük, mint az $E_K : \{0, 1\}^n \rightarrow \{0, 1\}^n$ invertálható leképezések egy családját, ahol a család egyes elemeit a $K \in \{0, 1\}^k$ kulcs választja ki. Az E_K transzformáció invertálhatóságából következik, hogy E_K egy permutációt határoz meg az n bites bináris vektorok halmazán.

Mivel a K kulcs csak véges sok értéket vehet fel, ezért a blokkrejtjelező ellen mindenig megkísérhető az ún. *kimerítő kulcskeresés* támadás. Tegyük fel, hogy a támadó rendelkezik néhány nyílt blokk – rejtett blokk párral. Az i -edik ilyen párt jelöljük (X_i, Y_i) -vel, ahol tehát $Y_i = E_K(X_i)$. A támadó K -t szeretné kitalálni. Ehhez az összes lehetséges kulcsot kipróbálja. minden lehetséges K^* kulcsra ellenőrzi, hogy $E_{K^*}(X_1)$ megegyezik-e Y_1 -gyel. Ha nem, akkor K^* -ot eldobja és újabb kulccsal próbálkozik. Ha viszont $E_{K^*}(X_1) = Y_1$, akkor a támadó a többi rendelkezésre álló páron is teszteli K^* -ot. Ha K^* minden párra működik, akkor a támadó úgy tekinti, hogy $K = K^*$. Bár lehetséges, hogy a támadó téved, ennek valószínűsége már néhány rendelkezésre álló pár esetén is elenyészően kicsi. Egy jól megtervezett blokkrejtjelező esetén, az $E_{K^*}(X_1) = Y_1$ egyenlőség 2^{-n} valószínűsséggel áll fenn, ha $K^* \neq K$. t rendelkezésre álló pár esetén tehát a tévedés valószínűsége 2^{-nt} . Mivel az összes kulcsok száma 2^k , ezért a hibásan elfogadott rossz kulcsok várható

száma 2^{k-n^t} . Ez k és n tipikus értékeire ($k \approx n$) már $t = 2$ esetén is nagyon kicsi lehet.

A kimerítő kulcskeresés akkor is működik, ha csak megfigyelt Y_1, Y_2, \dots rejtett blokkok állnak a támadó rendelkezésére, de tudja, hogy a nyílt blokkok redundánsak. Ilyenkor azonban tipikusan több megfigyelt rejtett szövegetre van szükség a hibásan elfogadott kulcsok várható számának csökkenése érdekében.

A kimerítő kulcskereséssel a támadó átlagosan a kulcstér felének átvizsgálása után találja meg a kulcsot. Ezért a kimerítő kulcskeresés ellen úgy védekezhetünk, hogy a rejtjelező kulcsméretét nagyra választjuk. A túl nagy kulcs sem jó azonban, mert akkor feleslegesen sok bitet kell titokban tartani. A megfelelő kulcsméret ezért az a legkisebb méret, ami a kimerítő kulcskeresést már praktikusan lehetetlenné teszi. A gyakorlatban ma a 128 bites minimális kulcsméret javasolt.

A kulcs méretének megfelelő megválasztása azonban még nem garantálja a rejtjelező biztonságát. A támadó ugyanis a kimerítő kulcskeresésen túl egyéb algoritmikus támadásokkal is próbálkozhat, melyek a rejtjelező algebrai struktúrájában rejlő gyengeségeket igyekeznek kihasználni. A lehetséges támadások megjósolhatatlansága miatt, a blokkrejtjelező tervezője nehéz feladat előtt áll. Olyan rejtjelezőt kell terveznie, ami a jelenleg még nem ismert támadásoknak is ellenáll. Sajnos ezen probléma megoldására nem ismert semmilyen szisztematikus eljárás. A blokkrejtjelezővel szemben támásztott minimális (magas szintű) tervezési követelmények a következők:

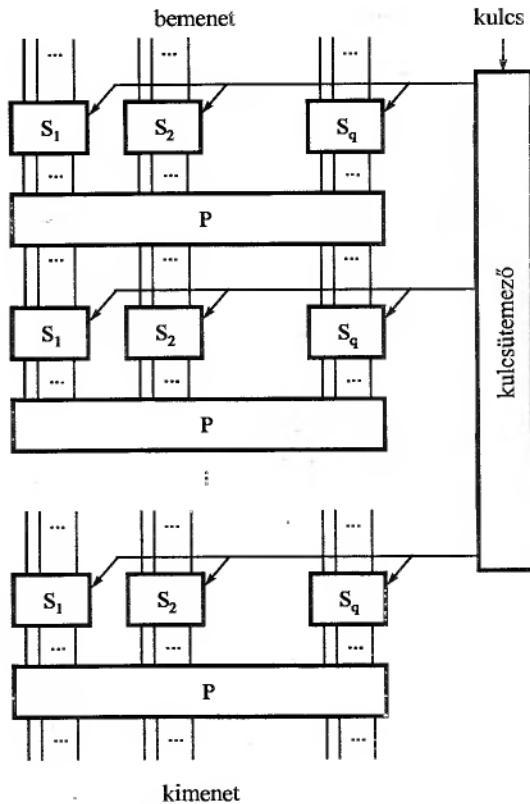
- *Teljesség:* A kimenet minden bitje minden bemeneti bittől és minden kulcsbittől függjön.
- *Statisztikai függetlenség:* Ne legyen statisztikai kapcsolat a nyílt blokkok és a rejtjeles blokkok között.
- *Lavina hatás:* A bemenet egy bitjének megváltoztatására a kimeneten minden bit $\frac{1}{2}$ valószínűséggel változzon meg. Ennek az a jelentősége, hogy lényegében azonos nyílt szövegek teljesen eltérő rejtett szövegekbe menjenek át. (Így például eltérő sorszámmal rendelkező azonos tartalmú üzenetek a rejtett szövegek figyelése alapján nem csoportosíthatók). Ha sonlóképpen, egy kimeneti bit megváltoztatása eredményezzen nem predikálható változásokat a bemeneten.

Az ún. *szorzat rejtjelezők* (product cipher) önmagukban egyszerű transzformációk egymásután többszöri alkalmazásával próbálják meg elérni, hogy az eredményül kapott transzformáció (az egyszerű transzformációk szorzata)

kielégítse a fenti kritériumokat. A ma ismert szimmetrikus kulcsú blokkrejtjelezők erre az elvre épülnek. A következő fejezetben mi is részletesen megvizsgáljuk ezt a tervezési hozzállást, abban az esetben, mikor az egyszerű transzformációk kis méretű helyettesítések és nagy méretű bitpermutációk. Az így nyert szorzat rejtjelezőt helyettesítéses-permutációs rejtjelezőnek nevezzük. Jellegzetes példája a DES, amit szintén ismertetünk.

2.1. Helyettesítéses-permutációs rejtjelezők és a DES

A Shannon által javasolt helyettesítéses-permutációs rejtjelező felépítése a 2.1. ábrán látható. A bemenetre kerülő blokk permutációs és kulcsvezérelt helyettesítő rétegeken keresztül jut a kimenetre. A permutációs réteg egyet-



2.1. ábra. A Shannon által javasolt helyettesítéses-permutációs rejtjelező felépítése

len P bit permutációból (P -dobozból) áll. A helyettesítő réteg több kis méretű S_i helyettesítő transzformációból (S -dobozból) épül fel, melyek mindegyike több helyettesítő táblát tartalmaz. Az S -dobozba bevezetett kulcsbritek határozzák meg, hogy melyik táblát kell használni a kódolás során.

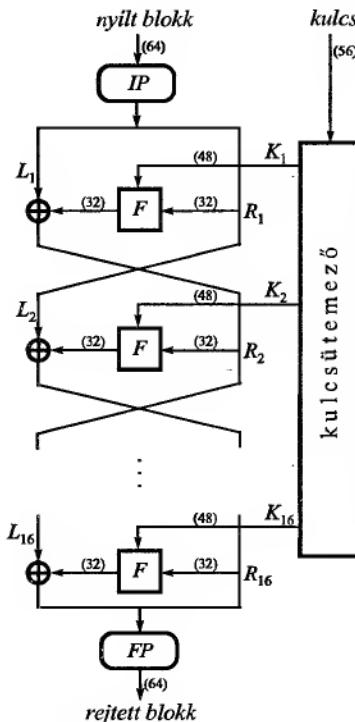
A permutációk és helyettesítések egymás utáni használatának célja a lavinahatás elérése. Ha például az S -dobozok olyanok, hogy tetszőleges bemenet esetén a bemeneten egy bit megváltoztatása a kimeneten legalább két bit megváltozását okozza, akkor az első réteg bemenetén fellépő egy bit változás az első helyettesítő réteg kimenetén legalább 2 bit változását eredményezi, amit a bit permutáció szétszór a teljes blokkon, így azok tipikusan a következő réteg két különböző S -dobozának bemenetére kerülnek. A második réteg kimenetén így már legalább 4 bit változik meg, és így tovább. Az eredmény az lesz, hogy egymáshoz Hamming-távolságban közeli (nagy korrelációt mutató) nyílt szöveg blokkok pszeudovéletlen módon tipikusan távolra kerülnek egymástól.

Fontos továbbá, hogy az S -dobozok nemlineáris leképezést valósítsanak meg, azaz a rejttett szöveg vektor a nyílt szöveg vektorból egy bináris mátrixszal történő szorzással ne legyen előállítható. Átaláatosabban megfogalmazva: az S -dobozok által megvalósított helyettesítés ne legyen egy lineáris transzformáció. Ez a teljes rejttelező lineáris voltát vonná maga után, hiszen a permutációk lineáris leképezések.

2.1.1. A DES blokkrejtjelező

A DES (Data Encryption Standard) a legismertebb helyettesítéses-permutációs blokkrejtjelező, melyet az IBM szakemberei fejlesztettek ki az USA-ban a hetvenes években. Később a világ több országában szabvánnyként fogadták el, és azóta is használják. A DES kiáltta az idők próbáját, ma sem ismert olyan gyakorlatban hatékonyan alkalmazható algoritmikus módszer, amivel fel lehetne törni. Ugyanakkor a DES 56 bites kulcs méretét ma már nem tartjuk elegendően biztonságosnak. Ezért a DES-t ma leginkább 3 kulcsos (168 bites) 3DES konfigurációban használjuk. Ezt a közeljövőben várhatóan felváltja majd a 2001-ben elfogadott AES (Advanced Encryption Standard) blokkrejtjelező, melyet a 2.5. szakaszban tárgyalunk.

A DES struktúrája a 2.2. ábrán látható. A bemenet és a kimenet mérete 64 bit, a kulcs 56 bites. A rejttelező 16 rétegből áll, ahol a rétegek felépítése egy kicsit eltér a 2.1. ábra rétegeinek klasszikus felépítésétől. Itt az egyes rétegek bemenete két részből áll, jelöljük ezeket L_i -vel és R_i -vel. L_i és R_i mérete 32



2.2. ábra. A DES szerkezete

bit. A jobb oldali blokk, R_i áthalad egy kulcs vezérelt, S-dobozokból és különböző permutációkból felépülő F transzformáción. Az i -edik réteg kulcsát jelöljük K_i -vel. K_i 48 bites, és a kulcsütemező állítja elő az eredeti 56 bites kulcsból. Az F transzformáció kimenetét XOR-oljuk a bal oldali blokkal, L_i -vel, majd a két oldal blokkjait felcseréljük (kivéve az utolsó rétegben). Így kapjuk a következő réteg bemenetének két blokkját. Formálisan:

$$L_{i+1} = R_i \quad (2.1)$$

$$R_{i+1} = L_i \oplus F(R_i, K_i).$$

Ezt a struktúrát Feistel-struktúrának nevezik. Az az előnye, hogy így felépülő rejtjelező minden képpen invertálható, függetlenül attól, hogy F invertálható-e, vagy sem. Vegyük észre ugyanis, hogy L_i és R_i előállítható L_{i+1} -ből és R_{i+1} -ből anélkül, hogy F inverzét használnánk, hiszen (2.1)-ből:

$$L_i = R_{i+1} \oplus F(L_{i+1}, K_i) \quad \text{és} \quad R_i = L_{i+1}.$$

Sőt a DES tervezői még azt is elértek, hogy a dekódoló gép megegyezzen a kódolával, csupán a kulcsütemezőt kell fordítva működtetni, azaz a kódolásnál az első rétegben alkalmazott 48 bites kulcsot a dekódolásnál az utolsó rétegben kell használni, a kódolásnál a második rétegben alkalmazott 48 bites kulcsot a dekódolásnál az utolsó előtti rétegben kell használni, és így tovább. Ez egyébként annak a következménye, hogy az utolsó rétegben nem kell a két félblokkot felcserélni.

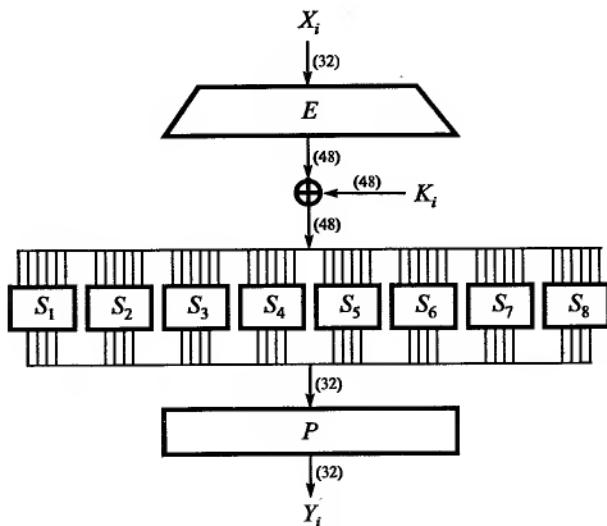
Vizsgáljuk most meg a DES építőelemeit részletesen!

A kezdeti és a végső permutáció (IP és FP). E két 64 bit bemenetű és 64 bit kimenetű bit permutációnak nincs különösebb szerepe, a DES elleni legerősebb támadásokat (differenciális és lineáris kriptanalízis) nem teszik nehezebbé.

A kulcsütemező. A kulcsütemező feladata, hogy az egyes rétegek számára előállítsa a megfelelő méretű kulcsot. Ezt a következőképpen teszi. Először is az 56 bites kulcsot két 28 bites részre vágja. Mindkét felet balra rotálja egyszer vagy kétszer, attól függően, hogy melyik réteg kulcsáról van éppen szó. A rotálások száma természetesen a DES szabványban rögzített. A rotálás után egy *PC* permutáció szerint kiválaszt az 56 bitből 48-at. A rotálások következtében minden rétegben más 48 bit kerül kiválasztásra, annak ellenére, hogy a választó *PC* permutáció minden rétegben azonos. minden bit kb. 14 rétegre lép be. Az összes rotálások száma pontosan 28, így egy blokkrejtelmezése után az 56 bites kulcs az eredeti állapotába áll vissza, és kezdődhet a következő blokk kódolása. Ennek elsősorban hardver megvalósításoknál van szerepe.

Az F függvény. Az *F* függvény felépítését a 2.3. ábra szemlélteti. A bemenet először egy expanziós transzformációt halad keresztül, mely a 32 bites vektorból egy 48 bites vektort állít elő. Ezt XOR-oljuk a 48 bites kulcs vektorral. A következő elem egy S-doboz réteg, melyben 8 darab 6 bitet 4 bitbe képző S-doboz található. Az S-doboz réteg kimenete így 32 bites. Az *F* függvény kiemenetét végül egy 32 bites P-doboz állítja elő.

Az expanziós transzformáció (E). Az *E* expanziós transzformáció a 32 bites bemeneti vektorból egy 48 bites vektort állít elő úgy, hogy néhány bemeneti bitet a kimeneten megdupláz, azaz két kimenetre is kivezet. A duplázott bitek a későbbiekben különböző S-dobozokba lépnek be. Így egyes bemeneti



2.3. ábra. A DES F rétegfüggvényének felépítése

bitek több S-dobozra is hatással vannak, ami fokozza a lavinahatást, hiszen a bemeneten történő változások így még gyorsabban terjednek tovább. Ennek következménye, hogy viszonylag kevés réteg után már minden bit minden bemeneti bittől függ, azaz már a néhány rétegre redukált DES is teljes¹.

Az S-dobozok (S_i). minden S-doboz 4 darab 4 bitet 4 bitbe helyettesítő táblázatból áll. Ezen táblázatok közül az S-doboz bemenetének két szélső bitje választja ki az aktuálisan alkalmazandót. Úgy is elképzelhetjük, hogy az S-dobozok 4 sorból és 16 oszloból álló táblázatok, és a bemenet két szélső (tehát az 1. és a 6.) bitje címzi meg a sort, a középső 4 (tehát a 2–5.) bit pedig az oszlopot.

A P-doboz (P). A P-doboz egy egyszerű bit permutáció. A P-doboz feladata a bitek összekeverése oly módon, hogy egy S-doboz kimeneti bitjei a következő rétegen más S-dobozok bemeneti bitjei legyenek.

¹ A teljesség definícióját lásd később az S-doboz tervezés kritériumait tárgyaló 2.2.1.2. szakaszban.

2.2. Helyettesítéses-permutációs rejtjelezők tervezése

Ebben a fejezetben azt vizsgáljuk, hogy mik az S- és P-doboz tervezés pontos kritériumai, és hogyan lehet a kritériumokat kielégítő transzformációkat tervezni. Először az S-doboz tervezéssel foglalkozunk, majd a P-doboz tervezést tárgyaljuk; végül kitérünk a rejtjelező rétegei számának megyálasztására.

2.2.1. S-doboz tervezés

A helyettesítéses-permutációs rejtjelezők legfontosabb építőelemei az S-dobozok. A rejtjelező ereje nagy mértékben függ S-dobozainak tulajdonságaitól, ezért ezek tervezése igen nagy körültekintést és tapasztalatot igényel. A megalapozott tervezés első lépése a tervezési kritériumok meghatározása. Sajnos nem ismert olyan elméleti módszer, amely pontos felvilágosítást ad arról, hogy egy S-doboznak milyen kritériumokat kell kielégítenie ahhoz, hogy a rejtjelező biztonságos legyen. A ma ismert kritériumok többsége nem elméleti megfontolásból született, hanem valamilyen ismertté vált támadás elleni védekezés feltételeit sikерült kritérium formájában megfogalmazni.

A kritériumok és a közöttük fennálló kapcsolatok részletesebb vizsgálata során kiderül, hogy az ismert kritériumok nem alkotnak kompatibilis halmozat, abban az értelemben, hogy vannak közöttük olyanok, melyeket egyszerre nem lehet kielégíteni, illetve bizonyos kritériumok kizártják az optimum elérését más kritériumok szempontjából. A tervező tehát kénytelen tapasztalataira támaszkodni, és az ismert kritériumok közül azokat kiválasztani, amelyeket egy adott alkalmazásban a legfontosabbnak tart, és amelyek nem zájják ki egymást.

A következő lépés a kritériumokat kielégítő S-dobozok keresése. Erre alapvetően két út áll rendelkezésre. Az egyik a véletlen választás, a másik a konstrukció. Az első esetben arról van szó, hogy véletlenül választunk S-dobozokat, majd az így nyert S-dobozok közül kiválasztjuk azokat, amelyek a tervezés során használt kritériumokat kielégítik. Utóbbi esetben valamilyen matematikai konstrukciós eljárást használunk a kívánt kritériumokat kielégítő S-dobozok közvetlen megkeresésére. A véletlen választást és a konstrukciót együtt is használhatjuk: kereshetünk például adott tulajdonságú Boole-függvényeket véletlen választással, majd ezekből megkonstruálhatjuk az S-dobozt.

A továbbiakban néhány fontos alapfogalom bevezetése után az ismertebb S-doboz tervezési kritériumokkal és a közöttük fennálló kapcsolatokkal fog-

lalkozunk, majd megvizsgáljuk a véletlen választás és a konstrukció lehetőségeit az S-doboz tervezésben.

2.2.1.1. Alapfogalmak

Az ezen alfejezetben használt jelöléseket a 2.1. táblázatban foglaltuk össze.

\mathcal{X}	az egész számok halmaza;
\mathcal{Z}_2^n	n hosszú bináris vektorok halmaza;
$x \oplus y$	x és y moduló 2 összege, ha $x, y \in \mathcal{Z}_2^1$, x és y bitenkénti moduló 2 összege, ha $x, y \in \mathcal{Z}_2^n$ és $n > 1$;
$x \odot y$	x és y moduló 2 szorzata, ha $x, y \in \mathcal{Z}_2^1$, x és y bináris skalárszorzata, ha $x, y \in \mathcal{Z}_2^n$ és $n > 1$, azaz $x \odot y = x_1 \odot y_1 \oplus x_2 \odot y_2 \oplus \dots \oplus x_n \odot y_n$ ahol x_i és y_i jelöli az x és y vektorok i -edik bitjét;
$A \odot x$	az $n \times m$ méretű bináris A mátrix és az $x \in \mathcal{Z}_2^m$ vektor szorzata, ahol az eredmény vektor mérete n , és komponensei a mátrix sorainak x -szel vett bináris skalárszorzataiként állnak elő;
$e_n^{(i)}$	n dimenziós egységektor, mely egyetlen egyest tartalmaz az i -edik pozícióban;

2.1. táblázat. A 2. fejezetben használt főbb jelölések összefoglalása

Boole-függvény. Boole-függvényen egy m bitet 1 bitbe képző $f : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^1$ függvényt értünk. Az így definiált Boole-függvény helyett gyakran használjuk az alább definiált $\hat{f} : \mathcal{Z}_2^m \rightarrow \{1, -1\}$ leképezést:

$$\hat{f}(x) = (-1)^{f(x)} = 1 - 2f(x). \quad (2.2)$$

Bevezetünk két speciális Boole-függvényt, amit a későbbiekbén gyakran használunk majd

- Legyen $L_{u,v} : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^1$ a következő alakú:

$$L_{u,v}(x) = u \odot x \oplus v \quad (2.3)$$

ahol $u \in \mathcal{Z}_2^m$ és $v \in \mathcal{Z}_2^1$.

- Legyen továbbá $\delta : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^1$ a következő függvény:

$$\delta(x) = \begin{cases} 1 & \text{ha } x = 0 \\ 0 & \text{egyébként.} \end{cases} \quad (2.4)$$

Boole-függvények súlya. Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^1$ Boole-függvény $w(f)$ súlya alatt az igazságtáblájában szereplő egyesek számát értjük. Formálisan:

$$w(f) = |\{x \in \mathbb{Z}_2^m : f(x) = 1\}| = \sum_{x \in \mathbb{Z}_2^m} f(x). \quad (2.5)$$

Boole-függvények távolsága. Két Boole-függvény távolsága nem más, mint azon bemenetek száma, melyekre a két függvény kimenete különbözik egymástól. Formálisan egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^1$ és egy $g : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^1$ Boole-függvény $d(f, g)$ távolsága a következő:

$$d(f, g) = |\{x \in \mathbb{Z}_2^m : f(x) \neq g(x)\}| = w(f \oplus g) \quad (2.6)$$

Boole-függvények korrelációja. Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^1$ és egy $g : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^1$ Boole-függvény $c(f, g)$ korrelációján az alábbi kifejezést értjük:

$$c(f, g) = \sum_{x \in \mathbb{Z}_2^m} \hat{f}(x) \hat{g}(x). \quad (2.7)$$

Boole-függvények autokorrelációs függvénye. Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^1$ Boole-függvény $r_f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}$ autokorrelációs függvényét a következőképpen definiáljuk:

$$r_f(a) = \sum_{x \in \mathbb{Z}_2^m} \hat{f}(x) \hat{f}(x \oplus a). \quad (2.8)$$

Boole-függvények Walsh-transzformáltja. Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^1$ Boole-függvény $F : \mathbb{Z}_2^m \rightarrow \mathbb{Z}$ Walsh-transzformáltján a következő függvényt értjük:

$$F(\omega) = \sum_{x \in \mathbb{Z}_2^m} f(x) (-1)^{x \odot \omega}. \quad (2.9)$$

A Walsh-transzformálóból a transzformáció inverzét alkalmazva kapjuk vissza az eredeti függvényt:

$$f(x) = \frac{1}{2^m} \sum_{\omega \in \mathbb{Z}_2^m} F(\omega) (-1)^{\omega \odot x} \quad (2.10)$$

A fenti definíció alapján számoljuk az \hat{f} függvény Walsh-transzformáltját is:

$$\begin{aligned}
\hat{F}(\omega) &= \sum_{x \in \mathcal{Z}_2^m} \hat{f}(x)(-1)^{x \odot \omega} \\
&= \sum_{x \in \mathcal{Z}_2^m} (1 - 2f(x))(-1)^{x \odot \omega} \\
&= \sum_{x \in \mathcal{Z}_2^m} (-1)^{x \odot \omega} - 2 \sum_{x \in \mathcal{Z}_2^m} f(x)(-1)^{x \odot \omega} \\
&= 2^m \delta(\omega) - 2F(\omega).
\end{aligned} \tag{2.11}$$

S-doboz. Egy $f : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^n$ leképzést S-doboznak nevezünk, ha $1 < n \leq m$. f -et gyakran tekintjük úgy, mint n darab Boole-függvény együttesét:

$$f(x) = (f_1(x), f_2(x), \dots, f_n(x)), \tag{2.12}$$

ahol az $f_i : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^1$ függvényeket komponens Boole-függvényeknek nevezzük.

Egy másik interpretáció szerint, f 2^{m-n} darab n bitet n bitbe képző helyettesítő tábla együttese. Ekkor f működése úgy képzelhető el, hogy a bemenet valamely $m-n$ bitje csupán arra szolgál, hogy az S-dobozban található 2^{m-n} transzformáció közül kiválasszon egyet, és a választott transzformáció állítja elő az S-doboz n bites kimenetét a bemenet maradék n bitjéből. A formális leírás egyszerűsége kedvéért tegyük fel, hogy a bemenet első $m-n$ bitje a transzformáció választó bit. Ekkor

$$f(x) = f(x_1, x_2, \dots, x_m) = f_{x_1, x_2, \dots, x_{m-n}}(x_{m-n+1}, x_{m-n+2}, \dots, x_m), \tag{2.13}$$

ahol tehát $f_{x_1, x_2, \dots, x_{m-n}} : \mathcal{Z}_2^n \rightarrow \mathcal{Z}_2^n$.

S-dobozok lineáris approximációs táblája. Egy $f : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^n$ S-doboz LAT_f lineáris approximációs tábláját (Linear Approximation Table – LAT) a következő kifejezés definiálja:

$$LAT_f(\alpha, \beta) = |\{x \in \mathcal{Z}_2^m : \alpha \odot x = \beta \odot f(x)\}| - 2^{m-1}, \tag{2.14}$$

ahol $\alpha \in \mathcal{Z}_2^m$ és $\beta \in \mathcal{Z}_2^n$.

$LAT_f(\alpha, \beta)$ tehát azt fejezi ki, hogy hány olyan x van, melyre a bemenet α által maszkolt bitjei és a kimenet β által maszkolt bitjei lineáris kapcsolatban vannak egymással. Könnyen ellenőrizhető, hogy $LAT_f(\alpha, 0) = 2^{m-1} \delta(\alpha)$,

azaz a lineáris approximációs tábla $\beta = 0$ -hoz tartozó oszlopa az $\alpha = 0$ -hoz tartozó sort kivéve mindenhol 0, $\alpha = 0$ -nál pedig 2^{m-1} .

A lineáris approximációs tábla hasonló szerepet tölt be az S-dobozoknál, mint a Walsh-transzformált a Boole-függvényeknél.

S-dobozok differencia eloszlási táblája. Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ S-doboz DDT_f differencia eloszlási tábláját (Differential Distribution Table – DDT) a következő kifejezés definiálja:

$$DDT_f(a, b) = |\{x \in \mathbb{Z}_2^m : f(x) \oplus f(x \oplus a) = b\}| \quad (2.15)$$

ahol $a \in \mathbb{Z}_2^m$ és $b \in \mathbb{Z}_2^n$.

$DDT_f(a, b)$ tehát azt fejezi ki, hogy a bemeneti differencia esetén a kimeneti differencia hány esetben b . Az S-doboz determinisztikus tulajdonsága miatt, 0 bemeneti differencia esetén a kimeneti differencia is biztosan 0, így $DDT_f(0, b) = 2^m \delta(b)$, azaz a differencia eloszlási táblázat $a = 0$ -hoz tartozó sora a $b = 0$ oszlopot kivéve mindenhol 0, $b = 0$ -nál pedig 2^m .

A bemeneti és a kimeneti differenciák közötti kapcsolatot Boole-függvények esetén az autokorrelációs függvény írta le, ezért a differencia eloszlási táblázat szerepe az S-dobozoknál a Boole-függvények autokorrelációs függvényének szerepéhez hasonlítható.

2.2.1.2. S-doboz tervezési kritériumok

Balansz tulajdonság. A balansz tulajdonság azt jelenti, hogy a transzformáció nem torzítja el egy egyenletes eloszlású bemenet gyakoriság-statisztikáját. Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ S-doboz akkor balansz, ha tetszőleges $y \in \mathbb{Z}_2^n$ esetén

$$|\{x \in \mathbb{Z}_2^m : f(x) = y\}| = 2^{m-n}. \quad (2.16)$$

Balansz S-dobozok lineáris approximációs táblájára a következő érdekes tételet mondhatjuk ki:

2.1. Tétel. Az $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ S-doboz akkor és csak akkor balansz, ha $LAT_f(0, \beta) = 0$ minden $\beta \in \mathbb{Z}_2^n, \beta \neq 0$ esetén.

Bizonyítás: Jelölje $c(y)$ azon bemenetek számát, melyekre a kimenet y , azaz $c(y) = |\{x \in \mathbb{Z}_2^m : f(x) = y\}|$. Ekkor nyilván

$$\sum_{y \in \mathbb{Z}_2^n} c(y) = 2^m.$$

Számoljuk ki $c(y)$ Walsh-transzformáltját:

$$\begin{aligned}
 C(\beta) &= \sum_{y \in \mathbb{Z}_2^n} c(y)(-1)^{y \odot \beta} \\
 &= \sum_{y \in \mathbb{Z}_2^n} c(y)(-1)^{y \odot \beta} + \sum_{y \in \mathbb{Z}_2^n} c(y) - 2^m \\
 &= \sum_{y \in \mathbb{Z}_2^n} c(y)(1 + (-1)^{y \odot \beta}) - 2^m \\
 &= \sum_{y \in \mathbb{Z}_2^n} 2c(y)I_{\{y \odot \beta = 0\}} - 2^m \\
 &= 2|\{x \in \mathbb{Z}_2^m : \beta \odot f(x) = 0\}| - 2^m \\
 &= 2LAT_f(0, \beta).
 \end{aligned}$$

Ezt felhasználva kapjuk, hogy

$$LAT_f(0, \beta) = \frac{1}{2}C(\beta) = \frac{1}{2} \sum_{y \in \mathbb{Z}_2^n} c(y)(-1)^{y \odot \beta}. \quad (2.17)$$

Ha f balansz, akkor $c(y) = 2^{m-n}$ minden $y \in \mathbb{Z}_2^n$ esetén. Ekkor a (2.17) kifejezést felhasználva, és feltételezve, hogy $\beta \neq 0$, azt kapjuk, hogy:

$$LAT_f(0, \beta) = \frac{2^{m-n}}{2} \sum_{y \in \mathbb{Z}_2^n} (-1)^{y \odot \beta} = 0,$$

mert $y \odot \beta$ ugyanannyi y -ra veszi fel az 1 értéket, mint amennyire a 0 értéket tetszőleges $\beta \neq 0$ esetén.

Fordítva, ha $LAT_f(0, \beta) = 0$ minden $\beta \neq 0$ esetén, akkor

$$\begin{aligned}
 c(y) &= \frac{1}{2^n} \sum_{\beta \in \mathbb{Z}_2^n} C(\beta)(-1)^{\beta \odot y} \\
 &= \frac{1}{2^n} \sum_{\beta \in \mathbb{Z}_2^n} 2LAT_f(0, \beta)(-1)^{\beta \odot y} \\
 &= \frac{2}{2^n} LAT_f(0, 0) \\
 &= \frac{2}{2^n} 2^{m-1} \\
 &= 2^{m-n}.
 \end{aligned}$$

□

Teljesség. Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ függvény teljes, ha minden kimeneti bitje minden bemeneti bittől függ. Más szavakkal ez azt jelenti, hogy az $f(x) \oplus f(x \oplus e_m^{(i)})$ függvény j -edik komponens Boole-függvényének igazságáblájában legalább egy darab 1 található, és ez minden $1 \leq i \leq m$ és $1 \leq j \leq n$ esetén igaz. Formálisan, minden $1 \leq i \leq m$ és $1 \leq j \leq n$ esetén teljesül a következő:

$$|\{x \in \mathbb{Z}_2^m : e_n^{(j)} \odot f(x) \neq e_n^{(j)} \odot f(x \oplus e_m^{(i)})\}| > 0 \quad (2.18)$$

Linearitás és affinitás. Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ függvény lineáris, ha felírható $f(x) = A \odot x$ alakban, és affin, ha felírható $f(x) = A \odot x \oplus b$ alakban, ahol A egy $n \times m$ -es bináris mátrix és $b \in \mathbb{Z}_2^n$. Mivel a lineáris (affin) S-dobozokból felépülő rejtjelező könnyen analizálható, ezért a lineáris (affin) S-dobozok kerülendők.

Lineáris struktúrák, lineáris dimenzió. Előfordulhat, hogy egy függvény nem lineáris (affin), mégis rendelkezik bizonyos parciális linearitással, azaz kimenete néhány bemeneti bittel lineáris (affin) kapcsolatban áll. Ezen parciális linearitás mérőszáma a lineáris dimenzió.

A lineáris dimenzió formális definíciójához először a lineáris struktúra definícióját vezetjük be. Az $a \in \mathbb{Z}_2^m$ bináris vektor lineáris struktúrája az $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ függvénynek, ha $f(x) \oplus f(x \oplus a)$ minden $x \in \mathbb{Z}_2^m$ esetén konstans. A 2.2. téTEL szerint egy függvény lineáris struktúrái alteret alkotnak a \mathbb{Z}_2^m térben. Ezen altér dimenzióját nevezzük f lináris dimenziójának.

2.2. Tétel. Ha a_1 és a_2 lineáris struktúrája az $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ függvénynek, akkor $a_1 + a_2$ is az.

Bizonyítás:

$$\begin{aligned} f(x) \oplus f(x \oplus a_1 \oplus a_2) &= f(x) \oplus f(x \oplus a_1) \oplus f(x \oplus a_1) \oplus f(x \oplus a_1 \oplus a_2) \\ &= (f(x) \oplus f(x \oplus a_1)) \oplus (f(y) \oplus f(y \oplus a_2)), \end{aligned}$$

ahol $y = x \oplus a_1$. Mivel a_1 és a_2 lineáris struktúrák, ezért a fenti összeg két tagja konstans, és így maga az összeg is az. \square

Nem-linearitás. Mivel egy helyettesítéses-permutációs rejtjelező egyetlen nem-lineáris eleme a helyettesítő réteg, ezért ez a kritérium központi szerepet játszik az S-dobozok tervezése során.

Egy $f : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^1$ Boole-függvény $N(f)$ nem-linearitásán az affin függvények halmazától vett távolságát értjük:

$$N(f) = \min_{u \in \mathcal{Z}_2^m, v \in \mathcal{Z}_2^1} d(f, L_{u,v}). \quad (2.19)$$

Felhasználva, hogy

$$\begin{aligned} \hat{F}(u) &= \sum_{x \in \mathcal{Z}_2^m} \hat{f}(x) (-1)^{x \odot u} \\ &= \sum_{x \in \mathcal{Z}_2^m} (-1)^{f(x) \oplus x \odot u} \\ &= |\{x \in \mathcal{Z}_2^m : f(x) = u \odot x\}| - |\{x \in \mathcal{Z}_2^m : f(x) \neq u \odot x\}| \\ &= 2^m - 2d(f, L_{u,0}), \end{aligned}$$

azt kapjuk, hogy

$$d(f, L_{u,0}) = 2^{m-1} - \frac{1}{2} \hat{F}(u).$$

Hasonlóan

$$d(f, L_{u,1}) = 2^{m-1} + \frac{1}{2} \hat{F}(u),$$

és így

$$N(f) = 2^{m-1} - \frac{1}{2} \max_{u \in \mathcal{Z}_2^m} |\hat{F}(u)|. \quad (2.20)$$

Egy S-doboz nem-linearitását komponens Boole-függvényeinek segítségével definiáljuk a következőképpen: Képezzük egy S-doboz komponens Boole-függvényeinek lehetséges lineárkombinációt. Az így kapott Boole-függvények közül válasszuk ki azt, amelyiknek nem-linearitása minimális. Ez a minimális nem-linearitás az S-doboz nem-linearitása:

$$N(f) = \min_{\beta \in \mathcal{Z}_2^n, \beta \neq 0} N(\beta \odot f). \quad (2.21)$$

Egy S-doboz nem-linearitását a következőképpen határozhatjuk meg lineáris approximációs táblájának ismeretében:

$$N(f) = 2^{m-1} - \max_{\alpha \in \mathcal{Z}_2^m, \beta \in \mathcal{Z}_2^n, \beta \neq 0} |LAT_f(\alpha, \beta)|. \quad (2.22)$$

Differenciális egyenletesség. A differenciális egyenletesség a nem-linearitás egy másik mérőszáma, mely a lineáris struktúráktól való távolsággal van összefüggésben. Ez a kritérium a differenciális kriptanalízissel kapcsolatban jelent meg. A differenciális kriptanalízis egy olyan támadási módszer, mely az S-dobozok azon tulajdonságát használja ki, hogy adott bemeneti differencia esetén a kimeneten nem minden differencia jelenik meg azonos valószínűséggel. A differenciális egyenletesség ennek az egyenetlenségnek a mérőszáma.

Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ S-doboz $D(f)$ differenciális egyenletességét a következőképpen definiáljuk:

$$D(f) = 2^m - \max_{a \in \mathbb{Z}_2^m, b \in \mathbb{Z}_2^n, a \neq 0} |\{x \in \mathbb{Z}_2^m : f(x) \oplus f(x \oplus a) = b\}|. \quad (2.23)$$

Figyelembe véve az S-doboz differencia eloszlási táblájának (2.15) definíóját láthatjuk, hogy

$$D(f) = 2^m - \max_{a \in \mathbb{Z}_2^m, b \in \mathbb{Z}_2^n, a \neq 0} DDT_f(a, b). \quad (2.24)$$

Felhívjuk a figyelmet (2.22) és (2.24) hasonlóságára.

Terjedési kritériumok, szigorú lavinahatás kritérium. Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ Boole-függvény kielégíti a k -ad fokú terjedési kritériumot, ha autokorrelációs függvényére igaz, hogy $r_f(a) = 0$ minden k -nál nem nagyobb súlyú $a \in \mathbb{Z}_2^m$ esetén, ahol $a \neq 0$ (más szavakkal, $1 \leq w(a) \leq k$). Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ S-doboz kielégíti a k -ad fokú terjedési kritériumot, ha minden komponens Boole-függvénye kielégíti azt.

Az első fokú terjedési kritériumot szokás szigorú lavinahatás kritériumnak (Strict Avalanche Criterion – SAC) is nevezni. Egy SAC tulajdonsággal rendelkező $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ Boole-függvényre tehát $r_f(e_m^{(i)}) = 0$ minden $1 \leq i \leq m$ esetén. Ez lényegében azt jelenti, hogy a bemeneten egy bitet megváltoztatva a kimenet $\frac{1}{2}$ valószínűsséggel változik meg.

Egy SAC tulajdonságot kielégítő S-doboz tetszőleges bemeneti bitjét megváltoztatva a kimenet minden egyes bitje $\frac{1}{2}$ valószínűsséggel változik meg. S-dobozok esetén, a SAC tulajdonság teljesülésének ellenőrzésére a diffúziós mátrix szolgál. Egy $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ S-doboz diffúziós mátrixa alatt azt az $m \times n$ méretű W_f mátrixot értjük, melyre

$$W_f^{(i,j)} = \frac{1}{2^m} |\{x \in \mathbb{Z}_2^m : e_n^{(j)} \odot f(x) \neq e_n^{(j)} \odot f(x \oplus e_m^{(i)})\}|. \quad (2.25)$$

A diffúziós mátrix $W_f^{(i,j)}$ eleme tehát azt mondja meg, hogy megváltoztatva az f függvény i -edik bemeneti bitjét a kimenet j -edik bitje milyen valószínűsséggel változik meg. Ha a diffúziós mátrix minden eleme $\frac{1}{2}$, akkor az S-doboz kielégíti a szigorú lavinahatás kritériumot.

2.2.1.3. A kritériumok közötti kapcsolatok

Mielőtt rátérnénk a kritériumokat kielégítő S-dobozok tervezési kérdéseire, röviden kitérünk néhány kritériumok között fennálló kapcsolatra.

Ha egy f S-doboz affin, akkor könnyen belátható, hogy $N(f)$ nem-linearitása 0. Ugyanakkor $N(f)$ úgy is lehet 0, ha az S-doboz maga nem affin, de komponens Boole-függvényeinek létezik olyan lineárkombinációja, mely affin Boole-függvényre vezet. Az ilyen S-dobozokat kriptográfiailag ugyanolyan gyengének tartjuk, mint az affin S-dobozokat, és ha lehet, kerüljük őket.

Vizsgáljuk most a nem-linearitás másik extrém értékét. Ha egy S-doboz minden komponens Boole-függvénye és azok minden nem triviális lineárkombinációja maximálisan nem-lineáris, akkor $|LAT_f(\alpha, \beta)| = 2^{\frac{m}{2}-1}$ minden α -ra és $\beta \neq 0$ -ra, és így $N(f) = 2^{m-1} - 2^{\frac{m}{2}-1}$. Ebből azonnal következik, hogy egy maximálisan nem-lineáris S-doboz nem lehet balansz, hiszen $|LAT_f(0, \beta)| \neq 0$.

Ha egy $f : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^n$ S-doboz kielégíti az m -ed fokú terjedési kritériumot (azaz minden komponens Boole-függvénye és azok minden lineárkombinációja kielégíti az m -ed fokú terjedési kritériumot), akkor $|LAT_f(\alpha, \beta)| = 2^{\frac{m}{2}-1}$ minden $\alpha \in \mathcal{Z}_2^m, \beta \in \mathcal{Z}_2^n, \beta \neq 0$ esetén, és így $N(f) = 2^{m-1} - 2^{\frac{m}{2}-1}$, vagyis f maximálisan nem-lineáris.

Egy $f : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^n$ S-doboz akkor éri el a legkisebb differenciális egyenletességet, ha létezik lineáris struktúrája. f differenciális egyenletessége akkor a legnagyobb, ha $DDT_f(a, b) = 2^{m-n}$ minden $a \neq 0$ és b esetén. Ez azt jelenti, hogy $f(x) \oplus f(x \oplus a)$ minden $a \neq 0$ -ra balansz. Ekkor $D(f) = 2^m - 2^{m-n}$. Belátható, hogy ekkor $|LAT_f(\alpha, \beta)| = 2^{\frac{m}{2}-1}$ minden $\alpha \in \mathcal{Z}_2^m, \beta \in \mathcal{Z}_2^n, \beta \neq 0$ esetén. A differenciálisan egyenletes S-dobozok tehát maximálisan nem-lineárisak.

A fenti állításokat a következő fontos téTELben foglaljuk össze:

2.3. Tétel. Az $f : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^n$ S-dobozra az alábbi állítások ekvivalensek:

- f nem-linearitása maximális, $N(f) = 2^{m-1} - 2^{\frac{m}{2}-1}$;
- f differenciálisan egyenletes, $D(f) = 2^m - 2^{m-n}$.

Továbbá a fenti állítások bármelyikéből következik, hogy f kielégíti az m -ed fokú terjedési kritériumot.

A 2.3. téTEL arra világít rá, hogy léteznek olyan S-dobozok, melyek a nem-linearitás, a differenciális egyenletesség és a terjedési kritériumok szempontjából is optimálisak. Ezeket a függvényeket *tökéletes (perfect) függvényeknek* nevezzük. Sajnos, mint korábban kiderült, a tökéletes függvények nem lehetnek balanszok. Érdekes kérdés annak vizsgálata, hogy vajon adható-e hatékony módszer a lehető legnagyobb nem-linearitású balansz függvények konstruálására.

2.2.1.4. S-doboz tervezés véletlen választással

Ebben a fejezetben a véletlen választáson alapuló S-doboz tervezés lehetőségeit vizsgáljuk. Fő kérdésünk tehát az, hogy az S-dobozok véletlen választása, majd különböző kritériumok szerinti szűrése, mely kritériumok esetén hatékony módszer. Ez a kérdés általában az adott kritériumot kielégítő S-dobozok számával van összefüggésben. Ha ugyanis egy kritériumot csak kevés S-doboz elégít ki, akkor nincsen arra remény, hogy véletlen választással pont ezeket találjuk meg. A véletlen választás tehát akkor hatékony módszer, ha a kritériumot kielégítő S-dobozok száma elég nagy.

Nem-linearitás. Affin $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ leképezések számára vonatkozóan könnyen kaphatunk eredményeket. Mivel az affin függvények $A \odot x \oplus b$ alakúak, ezért az affin függvények száma $2^{mn}2^n$, és így annak valószínűsége, hogy egy véletlenül választott S-doboz affin, $2^{(m+1)n}/(2^n)^{2^m}$. Például $m = 6, n = 4$ esetén 2^{-228} annak a valószínűsége, hogy egy véletlenül választott S-doboz affin. Ez jó, mert azt jelenti, hogy véletlen választással nagy valószínűséggel nem affin S-dobozra jutunk.

Tudjuk, hogy az affin S-dobozokon kívül vannak még olyan S-dobozok, melyek nem-linearitása 0, azok amelyekre a komponens Boole-függvényeknek létezik olyan nem nulla lineárkombinációja, mely affin. Ezek a függvények kriptográfiailag ugyanolyan gyengék, mint az affin S-dobozok. Mivel a 0 nem-linearitású S-dobozok halmaza tartalmazza az affin S-dobozok halmazát, ezért célszerű ezek számát is megvizsgálni.

2.4. Tétel. *Annak valószínűsége, hogy egy véletlenül választott $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ S-doboz nem-linearitása 0, felülről becsülhető a következő kifejezéssel:*

$$\frac{2^{m+n+1}}{2^{2^m}}.$$

Bizonyítás: Azon f függvények száma, melyekre valamelyen rögzített $\alpha \in \mathcal{Z}_2^m, \beta \in \mathcal{Z}_2^n, \beta \neq 0$ esetén minden x -re teljesül, hogy $\alpha \odot x = \beta \odot f(x)$ a következő:

$$|\{f : \alpha \odot x = \beta \odot f(x), \forall x \in \mathcal{Z}_2^m\}| = (2^{n-1})^{2^m}.$$

Ugyanis tetszőleges $x \in \mathcal{Z}_2^m$ esetén $\alpha \odot x$ értéke vagy 0 vagy 1, és tetszőleges nem nulla $\beta \in \mathcal{Z}_2^n$ esetén 2^{n-1} olyan $y \in \mathcal{Z}_2^n$ található, melyre $\beta \odot y = 0$, és 2^{n-1} olyan y melyre $\beta \odot y = 1$. Így minden $x \in \mathcal{Z}_2^m$ -re $f(x)$ értékét 2^{n-1} lehetséges érték közül választhatjuk, amiből a fenti állítás adódik. Hasonlóképpen

$$|\{f : \forall x \in \mathcal{Z}_2^m : \alpha \odot x \neq \beta \odot f(x)\}| = (2^{n-1})^{2^m},$$

és így

$$|\{f : |LAT_f(\alpha, \beta)| = 2^{m-1}\}| = 2(2^{n-1})^{2^m}.$$

Ezek alapján azon f függvények száma, melyekre létezik olyan $\alpha \in \mathcal{Z}_2^m, \beta \in \mathcal{Z}_2^n, \beta \neq 0$, hogy $|LAT_f(\alpha, \beta)| = 2^{m-1}$ teljesül, felülről becslőhető:

$$\begin{aligned} & |\{f : \exists \alpha \in \mathcal{Z}_2^m, \beta \in \mathcal{Z}_2^n, \beta \neq 0 : |LAT_f(\alpha, \beta)| = 2^{m-1}\}| \\ & \leq 2^m(2^n - 1)2(2^{n-1})^{2^m}. \end{aligned}$$

Mivel az összes $f : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^n$ függvények száma $(2^n)^{2^m}$, ezért a keresett valószínűséget felülről becslő formula a következő:

$$\frac{2^m(2^n - 1)2(2^{n-1})^{2^m}}{(2^n)^{2^m}} < \frac{2^{m+n+1}}{2^{2^m}}.$$

□

A 2.4. téTEL értelmében tehát igen kicsi annak a valószínűsége, hogy egy véletlenül választott S-doboz nem-linearitása 0. Például $m = 6, n = 4$ esetén ez a valószínűség kevesebb, mint 2^{-53} .

Differenciális egyenletesség. Egy S-doboz differenciális egyenletessége akkor a legkedvezőtlenebb, ha létezik lineáris struktúrája. Szerencsére a következő téTEL állítása szerint a lineáris struktúrával rendelkező S-dobozok száma kicsi.

2.5. Tétel. *Annak valószínűsége, hogy egy véletlenül választott $f : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^n$ S-doboznak létezik lineáris struktúrája, felülről becslőhető a következő kifejezéssel:*

$$\frac{2^{m+n}}{2^{n2^{m-1}}}.$$

Bizonyítás: Azon f függvények száma, melyekre valamilyen rögzített $a \in \mathbb{Z}_2^m, b \in \mathbb{Z}_2^n, a \neq 0$ esetén teljesül, hogy $f(x) \oplus f(x \oplus a) = b$ minden x -re b , a következő:

$$|\{f : \forall x \in \mathbb{Z}_2^m : f(x) \oplus f(x \oplus a) = b\}| = (2^n)^{2^{m-1}}.$$

Osszuk ugyanis kétfelé a lehetséges bemeneteket egy I_1 és egy I_2 halmazra, mégpedig úgy, hogy minden $x \in I_1$ -re $x \oplus a \in I_2$. Ekkor minden $x \in I_1$ -re $f(x)$ értéke szabadon választható, ugyanakkor $f(x \oplus a)$ értéke meghatározott, hiszen teljesülnie kell az $f(x) \oplus f(x \oplus a) = b$ egyenletnek. Így 2^{m-1} kimeneti értéket szabadon választhatunk a lehetséges 2^n -féle kimenetből, azaz a fenti állítás adódik.

Ezek alapján azon f függvények száma, melyekre létezik olyan $a \in \mathbb{Z}_2^m, b \in \mathbb{Z}_2^n, a \neq 0$, hogy $DDT_f(a, b) = 2^m$ teljesül, felülről becsülhető:

$$|\{f : \exists a \in \mathbb{Z}_2^m, b \in \mathbb{Z}_2^n, a \neq 0 : DDT_f(a, b) = 2^m\}| \leq (2^m - 1)2^n(2^n)^{2^{m-1}}.$$

Mivel az összes $f : \mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ függvények száma $(2^n)^{2^m}$, ezért a keresett valószínűséget felülről becslő formula a következő:

$$\frac{(2^m - 1)2^n(2^n)^{2^{m-1}}}{(2^n)^{2^m}} < \frac{2^{m+n}}{2^n 2^{m-1}}.$$
□

A 2.5. téTEL értelmében tehát igen kicsi annak a valószínűsége, hogy egy véletlenül választott S-doboz differenciális egyenletessége a lehető legkedvezőtlenebb. Például $m = 6, n = 4$ esetén ez a valószínűség kevesebb, mint 2^{-118} .

Terjedési kritériumok. Nem ismert, hogy egy véletlenül választott S-doboz milyen valószínűsséggel teljesíti a szigorú lavinahatás kritériumot. Erős sejtés, hogy ez a valószínűség exponenciálisan csökken, ahogy az S-doboz bemenetének mérete nő. Itt a véletlen választás tehát nem járható út.

Ezzel szemben olyan S-dobozt, melynek minden bemeneti és kimeneti bitje között „majdnem” teljesül a SAC, könnyű találni, ráadásul a jó választás valószínűsége nő, ahogy az S-doboz bemenetének mérete nő. Egy S-doboz „majdnem” kielégíti a SAC tulajdonságot, ha diffúziós mátrixa nem sokban különbözik egy SAC tulajdonságot kielégítő S-doboz diffúziós mátrixától, azaz a két diffúziós mátrix elemei közel vannak egymáshoz.

Fenti állításunkat számítógép segítségével generált véletlen S-dobozok diffúziós mátrixainak vizsgálatával ellenőrizhetjük. A 2.2. táblázatban a diffúziós mátrix maximális és minimális elemének várható értéke és szórása látható. Figyeljük meg, hogy a maximum és a minimum átlaga egyre közelebb van a kívánt 0.5-höz, ahogy a méret növekszik, továbbá a szórás nagyon kicsi, és gyorsan csökken a mérettel. Ez azt jelenti, hogy olyan S-doboz, melynek diffúziós mátrixában az elemek közel vannak $\frac{1}{2}$ -hez, könnyen generálható véletlen úton.

$m (= n)$	N	$E(max)$	$D^2(max)$	$E(min)$	$D^2(min)$
4	10000	0.82	0.0113	0.23	0.0055
6	10000	0.69	0.002	0.32	0.0019
8	5000	0.6	0.0004	0.4	0.0004
10	500	0.56	0.0001	0.44	0.0001
12	100	0.53	0.00002	0.47	0.0002

2.2. táblázat. Véletlen generált S-dobozok diffúziós mátrixa maximális és minimális elemének várható értéke és szórása. N a kísérletben generált S-dobozok számát jelöli

Egy másik nyitott probléma a perfekt S-dobozok számának pontos meghatározása vagy szigorú becslése. Az minden esetre sejthető, hogy a perfekt függvények igen kevesen vannak, így véletlen választással nem lehet őket megtalálni. Sajnos ebben az esetben még a majdnem perfekt S-dobozok véletlen választása is kivitelezhetetlennek tűnik. A perfektség kritériuma már túl erős a véletlen tervezéshez.

Balansz tulajdonság. Mivel a balansz $f : \mathcal{Z}_2^m \rightarrow \mathcal{Z}_2^n$ S-dobozok száma

$$\frac{2^m!}{(2^{m-n}!)^{2^n}},$$

ezért annak valószínűsége, hogy egy véletlenül választott S-doboz balansz, a következő:

$$\frac{2^m!}{(2^n)^{2^m} (2^{m-n}!)^{2^n}}. \quad (2.26)$$

Ez a valószínűség gyorsan csökken a bemenet és a kimenet méretével, így balansz S-dobozok egyszerű véletlen választással nehezen találhatók. Szenesre a következő algoritmussal egyszerűen lehet véletlen, balansz S-dobozokat közvetlenül generálni, így a probléma megoldható.

2.1. Algoritmus. Az algoritmus olyan véletlen, m bitet n bitbe képező S-dobozt generál, mely rendelkezik a balansz tulajdonsággal. A generált S-dobozt az y tömb tartalmazza, ahol $y[i]$ az S-doboz i bemenetre adott válasza (decimális alakban).

1. Legyen $y[i] = \text{UNDEF}$ minden i -re, ahol $0 \leq i \leq 2^m - 1$.
2. Legyen $k = 0$ és $c = 2^{m-n}$.
3. Válasszunk egy $0 \leq r \leq 2^m - 1$ véletlen számot, melyre $y[r] = \text{UNDEF}$.
4. Legyen $y[r] = k$.
5. Legyen $c = c - 1$.
6. Ha $c = 0$, akkor legyen $k = k + 1$ és $c = 2^{m-n}$.
7. Ha $k < 2^n$, akkor ugorjunk vissza a 3. lépéstre, egyébként vége.

Teljesség. Annak valószínűségére, hogy egy véletlenül választott balansz S-doboz nem teljes, a következő téTEL ad felső becslést:

2.6. Tétel. Annak valószínűsége, hogy a balansz $\mathbb{Z}_2^m \rightarrow \mathbb{Z}_2^n$ leképezések közül véletlenül választott f S-doboz nem teljes, felülről becsülhető a következő kifejezéssel:

$$nm \frac{\binom{2^{m-1}}{2^{m-2}} (2^{m-1}!)^2}{2^m!}.$$

Bizonyítás: Először kiszámoljuk annak valószínűségét, hogy egy véletlenül választott balansz S-doboz j -edik kimeneti bitje nem függ az i -edik bemeneti bittől. Osszuk a lehetséges kimeneteket két egyenlő halmazra, O_1 -re és O_2 -re úgy, hogy az O_1 -be kerülő vektorok j -edik bitje 0 legyen, míg az O_2 -be kerülőké 1. A bemenetkből képezzünk párokat úgy, hogy minden pár két tagja csak az i -edik bitben különbözzön. Ha a j -edik kimeneti bit nem függ az i -edik bemeneti bittől, akkor minden (x, y) párra igaz az, hogy ha $f(x) \in O_k$, akkor $f(y) \in O_k$, ahol $k = 1, 2$. Ennek alapján csoporthosszuk a párokat: azon (x, y) párok, melyekre $f(x), f(y) \in O_1$, kerüljenek az I_1 halmazba, a többi pedig az I_2 -be. Ha az S-doboz balansz, akkor I_1 és I_2 mérete megegyezik. Így I_1 -et $\binom{2^{m-1}}{2^{m-2}}$ -féléképpen lehet megválasztani. I_1 és O_1 illetve I_2 és O_2 között $2^{m-1}/(2^{m-n}!)^{2^{n-1}}$ balansz leképzés létezik, így az (i, j) bitben nem teljes balansz leképzések száma:

$$\binom{2^{m-1}}{2^{m-2}} \left(\frac{2^{m-1}!}{(2^{m-n}!)^{2^{n-1}}} \right)^2.$$

Felhasználva, hogy összesen

$$\frac{2^m!}{(2^{m-n}!)^2}$$

balansz leképezés van, azt kapjuk, hogy a keresett valószínűség

$$\frac{\binom{2^{m-1}}{2^{m-2}}(2^{m-1}!)^2}{2^m!}.$$

A tételek felsőbecslése innen már adódik. \square

2.2.1.5. S-doboz tervezés konstrukcióval

A 2.2.1.4. szakaszban láttuk, hogy véletlen választással könnyen jutunk balansz és teljes S-dobozokhoz. Ugyanakkor, a SAC tulajdonságot kielégítő vagy az erősen nem-lineáris S-dobozok száma igen kicsi, ezért ezen tulajdonságokat kielégítő S-dobozok generálására a véletlen választás nem járható út. Ilyenkor folyamodhatunk a konstrukcióra épülő tervezési hozzálláshoz. Ebben a szakaszban ezt szemléltetjük egy olyan konstrukciós eljárás bemutatásával, mellyel erősen nem-lineáris, balansz S-dobozokat lehet generálni.

A konstrukció első lépéseként a Maiorana-McFarland-módszerrel generálunk egy perfekt (maximális nem-linearitással rendelkező) S-dobozt, majd ezt módosítjuk oly módon, hogy az kielégítse a balansz tulajdonságot, és nem-linearitása ne csökkenjen jelentős mértékben.

1. lépés: Maiorana-McFarland-konstrukció. Vegyünk egy $f : \mathbb{Z}_2^{2n} \rightarrow \mathbb{Z}_2^n$ S-dobozt, melynek f_i komponens Boole-függvényei a következő alakúak:

$$f_i(x) = f_i(x^{(1)}, x^{(2)}) = \pi_i(x^{(1)}) \odot x^{(2)}, \quad (2.27)$$

ahol $i = 1, 2, \dots, n$, $x^{(1)}, x^{(2)} \in \mathbb{Z}_2^n$, és $\pi_i : GF(2^n) \rightarrow GF(2^n)$ a következő módon definiált függvény:

$$\pi_i(x^{(1)}) = \alpha^i \circ x^{(1)}, \quad (2.28)$$

ahol α a $GF(2^n)$ véges test egy primitív eleme, és \circ a $GF(2^n)$ feletti szorzást jelöli.

2. lépés: balansszá tétele. A fent definiált f S-dobozról tudjuk, hogy perfekt, azaz minden komponens Boole-függvénye és azok minden nem triviális lineáris kombinációja perfekt. Korábban megmutattuk azonban, hogy a perfekt S-dobozok nem elégítik ki a balansz tulajdonságot. Esetünkben f minden komponens Boole-függvénye és azok minden nem triviális lineáris kombinációja $2^{n-1} - 2^{n-1}$ súlyú. Ezért ahhoz, hogy f -et balansszá tegyük, minden komponens Boole függvényének igazságátlájában 2^{n-1} darab 0 értéket 1 értékre kell változtatnunk. Ráadásul ezt úgy kell megtennünk, hogy a komponens Boole-függvények nem triviális lineáris kombinációi is kielégítsek a balansz tulajdonságot, mert csak így lesz maga az f S-doboz is balansz.

Vegyük észre, hogy $f_i(x^{(1)}, x^{(2)}) = 0$ ha $x^{(2)} = 0$, minden i -re. Tekintsük tehát azokat a kimeneteket, amelyek olyan $(x^{(1)}, x^{(2)})$ bemenethez tartoznak, ahol $x^{(2)} = 0$. Ilyenből 2^n van. Ezek közül fogunk 2^{n-1} darabot 1-re változtatni. A változtatáshoz a $2^n \times 2^n$ méretű H_n Hadamard-mátrixot fogjuk használni, melyet a következőképpen definiálunk:

$$H_n = \begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & \neg H_{n-1} \end{pmatrix}, \quad (2.29)$$

ahol neg az elemenkénti negálás operátora, azaz $\neg H = \neg(h_{ij}) = (\neg h_{ij})$, és $H_0 = (0)$. Az első sort kivéve a fent definiált H_n mátrix minden sorában pontosan 2^{n-1} darab 1-es van. Ezen sorok közül választunk n darabot, melyeket h_1, h_2, \dots, h_n -nel jelölünk, és az f S-doboz komponens Boole-függvényeit a következőképpen módosítjuk:

$$f'_i(x^{(1)}, x^{(2)}) = \begin{cases} f_i(x^{(1)}, x^{(2)}) & \text{ha } x^{(2)} \neq 0 \\ f_i(x^{(1)}, x^{(2)}) \oplus h_i[x^{(1)}] & \text{ha } x^{(2)} = 0, \end{cases} \quad (2.30)$$

ahol $h_i[x^{(1)}]$ jelöli a h_i vektor $x^{(1)}$ -edik elemét.

Ekkor a következő tételet mondhatjuk ki:

2.7. Tétel. A fenti konstrukcióval nyert f' S-doboz balansz, és nem-lineáritása alulról becsülhető a következő kifejezéssel:

$$N(f') \geq N(f) - 2^{n-1} = 2^{2n-1} - 2^n.$$

Bizonyítás: A Hadamard-mátrix tulajdonságaiból adódik, hogy H_n minden kiválasztott h_1, h_2, \dots, h_n sorának és azok bármely lineáris kombinációjának súlya 2^{n-1} . Ebből következik, hogy f' minden komponens Boole-függvényének és azok minden lineáris kombinációjának súlya 2^{n-1} -gel több, mint

f komponens Boole-függvényeinek és azok lineáris kombinációjának súlya. Mivel f komponens Boole függvényeinek és azok lineáris kombinációjának súlya $2^{2n-1} - 2^{n-1}$ volt, ezért f' komponens Boole-függvényeinek és azok lineáris kombinációjának súlya 2^{2n-1} -re adódik, ami azt jelenti, hogy f' balansz.

Mivel f minden komponens Boole-függvényének és azok lineáris kombinációjának pontosan 2^{n-1} darab kimenetét változtattuk meg, ezért f -hez képest f' legfeljebb 2^{n-1} -gyel került közelebb az affin függvények halmazához. Mivel f perfekt, ezért nem-linearitása $2^{2n-1} - 2^{n-1}$. Innen a téTEL állításának második része adódik. \square

2.2.2. P-doboz tervezés

Jól ismert tény, hogy a rejtjelező erejét elsősorban az S-dobozok határoz zák meg. A P-doboz lineáris és minden rétegben azonos. A P-doboz feladata a diffúzió megvalósítása, annak biztosítása, hogy tetszőleges bemeneti bit megváltozásának hatása minden S-dobozhoz elérjen valamelyik rétegben. Azt a P-dobozt, amely kielégíti ezt a követelményt, nem-degenerált P-doboznak nevezzük.

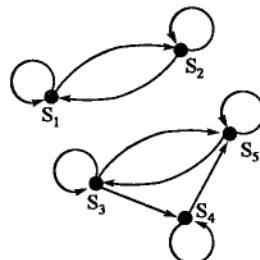
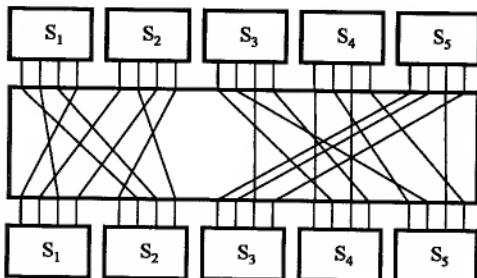
Az S-dobozoknál alkalmazott véletlen választás módszerét a P-doboz tervezésben is felhasználhatjuk. A következő téTELből kiderül, hogy véletlen választással igen nagy valószínűséggel nem-degenerált P-dobozhoz jutunk.

2.8. TéTEL. *Tegyük fel, hogy a rejtjelező egy rétege egy q S-dobozt tartalmazó S-doboz rétegből és egy P-dobozból áll. Az S-dobozok bemenete és kimenete legyen n bites. Ekkor annak a valószínűsége, hogy egy véletlenül választott P-doboz nem-degenerált, pontosan*

$$P_n(q) = 1 - \sum_{i=1}^{q-1} P_n(i) \frac{\binom{q-1}{i-1}}{\binom{qn}{in}},$$

ahol $P_n(1) = 1$ minden n -re.

Bizonyítás: Egy P-dobozt szemléltethetünk egy irányított gráffal, melynek q csúcsa van, és az i -edik csúcs akkor van összekötve a j -edik csúccsal, ha az i -edik S-doboz valamelyik kimeneti bitjét a P-doboz a következő réteg j -edik S-dobozának valamelyik bemeneti bitjéhez rendeli. Ha a P-doboz nem-degenerált, akkor a gráf bármely csúcsából vezet irányított út bármely másik csúcsba. Ennek szükséges és elégsges feltétele, hogy a gráf összefüggő legyen. A gráf reprezentáció szemléltetésére tekintsük a 2.4. ábrát.



2.4. ábra. Egy P-doboz reprezentációja irányított gráffal

Jelöljük $T_n(q)$ -val azon P-dobozok számát, melyekhez összefüggő gráf rendelhető a fent leírt módon. Ha $q = 1$, akkor az összes lehetséges $n!$ P-doboz gráfja összefüggő, hiszen ezen P-dobozokhoz olyan gráf tartozik, amelynek csak egyetlen csúcsa van. Így $T_n(1) = n!$. Vizsgáljuk most azt az esetet amikor $q > 1$. Ekkor a P-dobozokat osztályozhatjuk aszerint, hogy egy rögzített csúcs (például az első) mekkora méretű részgráfhoz tartozik. Itt a gráf mérete a csúcsok számát jelenti. Az i . osztály

$$\binom{q-1}{i-1} T_n(i)(qn-in)! \quad (2.31)$$

P-dobozt tartalmaz, mert pontosan ennyi azon P-dobozok száma, ahol az első csúcs egy i méretű összefüggő részgráfhoz tartozik. Ha összegezzük a (2.31) kifejezést a lehetséges i értékekre, akkor megkapjuk az összes P-dobozok számát, ami $(qn)!$. A következő kifejezéseket nyerjük tehát $T_n(q)$ -ra és $P_n(q)$ -ra:

$$\begin{aligned} T_n(q) &= (qn)! - \sum_{i=1}^{q-1} \binom{q-1}{i-1} T_n(i)(qn-in)! \\ P_n(q) &= \frac{T_n(q)}{(qn)!} \\ &= 1 - \sum_{i=1}^{q-1} \binom{q-1}{i-1} \frac{T_n(i)}{(in)!} \frac{(qn-in)!(in)!}{(qn)!} \\ &= 1 - \sum_{i=1}^{q-1} P_n(i) \frac{\binom{q-1}{i-1}}{\binom{qn}{in}}. \end{aligned}$$

□

A 2.3. táblázatban látható $P_n(q)$ értéke néhány tipikus n és q esetén. Mint látható, a véletlen választás már kis n és q értékekre is hatékony módszer nem-degenerált P-dobozok tervezésére.

n	q	$P_n(q)$
4	8	0.99977
4	16	0.99997
8	4	$1 - 3.85 \cdot 10^{-7}$
8	8	$1 - 1.81 \cdot 10^{-9}$

2.3. táblázat. $P_n(q)$ értéke néhány tipikus n és q esetén

2.2.3. A rétegek számának megválasztása

Shannon ötlete szerint, ha rejtjelezőnk megfelelően sok rétegből áll, és minden réteg más kulccsal van vezérelve, akkor a rejtjelező nehezen törhető, erős kódolást valósít meg, annak ellenére, hogy az egyes rétegek könnyen analízálható, gyenge transzformációkból épülnek fel. Ellentétben az S-doboz tervezés elméleti megalapozottságával, az irodalomban nem nagyon található olyan téTEL, mely a szükséges rétegek számára adna valamilyen alsó korlátot. Ennek oka valószínűleg az, hogy a rétegek számát nagy mértékben befolyásolja az alkalmazott rétegfüggvény és a kulcsütemező, ezért a probléma általános kezelése igen bonyolult. Két egyszerű téTEL azonban így is kimondhatunk.

Ha legalább a rejtjelező teljességét megköveteljük, akkor a következő alsó korlátot kapjuk a rétegek számára vonatkozóan:

2.9. Tétel. *Ha a rejtjelező teljes, akkor rétegeinek r számára a következő alsó korlát adható:*

$$r \geq \log_n q + 1,$$

ahol n az S-dobozok bemenetének és kimenetének mérete, q pedig az S-doboz réteg S-dobozainak száma.

Bizonyítás: A rejtjelező egy bemeneti bitje legfeljebb n S-dobozra lehet hatással a második rétegben, legfeljebb n^2 S-dobozra lehet hatással a harmadik rétegben, és így tovább. Végül egy bemeneti bit legfeljebb n^{r-1} S-dobozra lehet hatással az utolsó rétegben. A teljesség csak akkor teljesülhet, ha az S-dobozok q száma nem nagyobb, mint n^{r-1} , azaz $q \leq n^{r-1}$. Mindkét oldal logaritmusát véve a téTEL állítását kapjuk. \square

Egy kicsit erősebb alsó korláthoz jutunk, ha megköveteljük, hogy a rejtjelező kielégítse a lavinahatás kritériumot (feltesszük, hogy az S-dobozok kielégítik azt):

2.10. Tétel. *Ha a rejtjelező kielégíti a lavinahatás kritériumot, akkor rétegeinek r számára a következő alsó korlát adható:*

$$r \geq \frac{\log_2 N - 1}{\log_2 n - 1},$$

ahol n az S-dobozok bemenetének és kimenetének mérete, N pedig a teljes rejtjelező blokkmérete, azaz $N = qn$.

Bizonyítás: Mivel az S-dobozok kielégítik a lavinahatás kritériumot, ezért megváltoztatva egy bemeneti bitet, várhatóan a kimeneti bitek fele, azaz $\frac{n}{2}$ bit változik meg. Ezért az előző tétel bizonyításának gondolatmenetét követve, egy bit változása a rejtjelező bemenetén várhatóan $(\frac{n}{2})^r$ kimeneti bit változását okozza. Ha a rejtjelező kielégíti a lavinahatás kritériumot, akkor $(\frac{n}{2})^r \geq \frac{N}{2}$ kell, hogy teljesüljön. Mindkét oldal logaritmusát véve a tétel állítása következik. \square

Ha például $N = 64$, $n = 4$ és $q = 16$, akkor a rétegek száma legalább 5.

2.3. Differenciális és lineáris kriptanalízis

A rejtjelező tervezőjének tisztában kell lennie az alapvető kriptanalízis módszerekkel is, hiszen a várható támadások nagymértékben befolyásolhatják a tervezést. Ez olyannyira igaz, hogy a tervezési kritériumok nagy része valamilyen támadás elleni védekezés ellenszereként született. Ha meg tudjuk mondani, hogy egy rejtjelező mennyi erőfeszítés árán törhető fel egy adott támadási módszerrel, akkor ezzel lényegében a rejtjelezőt minősítettük, így a kriptanalízist minősítésre is használhatjuk. Itt most röviden összefoglaljuk a ma ismert két legerősebb támadás, a differenciális és a lineáris kriptanalízis alapjait.

2.3.1. Differenciális kriptanalízis

A differenciális kriptanalízis alapvetően egy választott nyílt szövegű támadás. Azon alapszik, hogy bizonyos bemeneti differenciáakra a kimeneti differenciák nem egyenletes eloszlásúak, azaz vannak olyan kimeneti differenciák, amelyek nagyobb valószínűsséggel fordulnak elő, mint mások. A támadás felhasználja még azt is, hogy a rejtjelező Feistel-típusú, tehát az utolsó réteg F függvényének bemenete ismert, ha a rejtjeles blokk ismert.

A támadás alapgondolata. DES típusú eszközöknél az F függvény egy E lineáris transzformációból, egy kulcs injekciós műveletből, egy S-doboz rétegből és egy P invertálható lineáris leképezésből, általában egy bitpermutációból áll (2.3. ábra). Formálisan $Y = F(X, K) = P \odot S(E \odot X \oplus K)$.

Először azt vizsgáljuk meg, hogy hogyan lehet megfejteni a kulcs egy részét, ha ismerjük az utolsó, r -edik réteg F függvényének X és X^* bemenetét, valamint ezen bemenetekhez tartozó $Y = F(X, K_r)$ és $Y^* = F(X^*, K_r)$ kimenetek $Y \oplus Y^*$ differenciáját. Ha a bemenetek ismertek, akkor a bemenetek differenciája is ismert. Mivel a kulcs injekció hatástan a differenciákra nézve, azaz $E \odot X \oplus E \odot X^* = E \odot X \oplus K_r \oplus E \odot X^* \oplus K_r$, ezért ismert az S-doboz réteg bemenetén megjelenő differencia. A kimenetek differenciájából a bitpermutáció invertálhatósága miatt meghatározható az S-doboz réteg kimenetén megjelenő differencia is. Így ismerjük az S-dobozok bemeneti és kimeneti differenciáit, amelyből meg tudjuk mondani, hogy melyek azok a párok, melyek megjelenhetnek az S-doboz réteg bemenetén. Ezen lehetséges párok, valamint az F bemenetén ténylegesen megjelenő pár ismeretében meg tudjuk mondani, hogy melyek a lehetséges kulcs értékek. Ezt az eljárást több X és X^* bemenetre, valamint a hozzájuk tartozó kimenetek $Y \oplus Y^*$ differenciájára végrehajtva minden megkapjuk a lehetséges kulcs értékeket. Ha elég sok párt megvizsgálunk, akkor csak az igazi kulcs fog minden megjelenni a lehetséges értékek között.

A probléma az, hogy általában nem ismerjük az utolsó réteg F függvényének kimenetén megjelenő differenciát. Azt azonban meg lehet mondani, hogy adott bemeneti differencia esetén az $(r - 1)$ -edik réteg kimenetén az egyes kimeneti differenciák milyen valószínűsséggel jelennek meg. Kereshetünk olyan bemeneti differenciát, melyre a kimeneti differenciák az egyenleírásból lényegesen eltérő valószínűsséggel jelennek meg az $(r - 1)$ -edik réteg kimenetén, és kiválaszthatunk ezek közül egy kimeneti differenciát, mely az egyenletesnél nagyobb valószínűsséggel jelenik meg. Az $(r - 1)$ -edik réteg kimeneti differenciájából és a rejtjeles blokkokból viszont már igen egyszerűen számítható az utolsó réteg F -jének kimeneti differenciája, és alkalmazhatjuk az előző algoritmust. Most is megkapjuk a lehetséges kulcs értékeit, de nem biztos, hogy köztük lesz az igazi kulcs. Előfordulhat ugyanis, hogy nem az általunk remélt kimeneti differencia jelenik meg az $(r - 1)$ -edik réteg kimenetén. Ha azonban sokszor próbálkozunk, akkor minden kulcs közel egyenletes valószínűsséggel fog megjelenni, kivéve az igazi kulcsot, ami az egyenletesnél nagyobb valószínűsséggel jelenik meg a lehetséges kulcs értékek között, és így az igazi kulcs kiválasztható.

2.2. Algoritmus. A differenciális kriptanalízis fő lépései tehát a következők:

1. Keressünk egy olyan $(\Delta P, \Delta C')$ differencia párt, melyre ΔP bemeneti differencia esetén az $(r - 1)$ -edik réteg kimenetén nagy valószínűséggel $\Delta C'$ differencia jelenik meg.
2. Vegyük M darab (P, P^*) nyílt szöveg párt, ahol a párokat alkotó blokkok differenciája éppen ΔP .
3. Feltételezzük, hogy az $(r - 1)$ -edik réteg kimeneti differenciája $\Delta C'$, és felhasználva az r -edik réteg kimenetén megjelenő (C, C^*) rejtett szöveg párt, valamint ezek differenciáját, megkapjuk az utolsó réteg F függvényének $C^{(R)}$ és $C^{*(R)}$ bemenetét és $\Delta C^{(L)} \oplus \Delta C^{(L)}$ kimeneti differenciáját. Ezek ismeretében minden nyílt szöveg párra határozzuk meg az utolsó réteg lehetséges kulcsainak halmazát.
4. Számítsuk meg, hogy az egyes kulcsok hányszor jelentek meg lehetséges kulcsként. A leggyakrabban megjelenő kulcs az igazi.

Differencia-átmenetek valószínűségének meghatározása. A támadáshoz szükséges nyílt szöveg párok száma lényegében attól függ, hogy adott ΔP bemeneti differencia esetén mekkora valószínűséggel jelenik meg az $(r - 1)$ -edik réteg kimenetén a feltételezett $\Delta C'$ differencia. Minél nagyobb ez a valószínűség, annál kevesebb nyílt szöveget van szükség. Ezért jó lenne tudni, hogy adott bemeneti differencia esetén, az egyes kimeneti differenciák milyen valószínűséggel jelennek meg az $(r - 1)$ -edik réteg kimenetén. Jelöljük ΔZ_i -vel az i -edik réteg bemenetén megjelenő differenciát. Ekkor a következő feltételes valószínűségre vagyunk kíváncsiak:

$$\Pr\{\Delta Z_r = \zeta_r | \Delta Z_1 = \zeta_1\}. \quad (2.32)$$

Ha a rejtjelező Markov-típusú (a legtöbb DES típusú eszköz ilyen), akkor igaz a következő:

$$\Pr\{\Delta Z_r = \zeta_r | \Delta Z_1 = \zeta_1\} = \sum_{\zeta_2, \zeta_3, \dots, \zeta_{r-1}} \prod_{i=1}^{r-1} \Pr\{\Delta Z_{i+1} = \zeta_{i+1} | \Delta Z_i = \zeta_i\}. \quad (2.33)$$

A kérdés tehát az, hogy hogyan lehet a

$$\Pr\{\Delta Z_{i+1} = \zeta_{i+1} | \Delta Z_i = \zeta_i\} \quad (2.34)$$

feltételes valószínűséget kiszámolni, vagyis megmondani, hogy adott bemeneti differencia esetén adott kimeneti differencia milyen valószínűsséggel jelenik meg egy réteg vonatkozásában.

Ha az i -edik réteg bemeneti differenciája $\zeta_i = (\zeta_i^{(L)}, \zeta_i^{(R)})$, kimeneti differenciája pedig $\zeta_{i+1} = (\zeta_{i+1}^{(L)}, \zeta_{i+1}^{(R)})$, akkor a Feistel-struktúra miatt az F függvény bemeneti differenciája $\Delta X' = \zeta_i^{(R)} = \zeta_{i+1}^{(L)}$, kimeneti differenciája pedig $\Delta Y' = \zeta_i^{(L)} \oplus \zeta_{i+1}^{(R)}$. Mivel a kulcs injekció hatástanon a differenciákra nézve, ezért az S-doboz réteg bemenetén a $\Delta X = E \odot \Delta X'$, kimenetén pedig a $\Delta Y = P^{-1} \odot \Delta Y'$ differencia jelenik meg. Tegyük fel, hogy az S-doboz réteg k darab $s_j : \mathcal{X}_2^m \rightarrow \mathcal{X}_2^n$ S-dobozból áll. Ekkor a j -edik S-doboz bemeneti differenciája Δx_j , kimeneti differenciája Δy_j , ahol $\Delta X = (\Delta x_1, \Delta x_2, \dots, \Delta x_k)$, $\Delta x_j \in \mathcal{X}_2^m$, és $\Delta Y = (\Delta y_1, \Delta y_2, \dots, \Delta y_k)$, $\Delta y_j \in \mathcal{X}_2^n$. Annak a valószínűsége, hogy a j -edik S-doboz kimeneti differenciája Δy_j , feltéve, hogy a bemeneti differenciája Δx_j , könnyen megadható az S-doboz differencia eloszlási táblájának segítségével:

$$\begin{aligned}\Pr\{\Delta x_j \xrightarrow{s_j} \Delta y_j\} &= \frac{1}{2^m} |\{x \in \mathcal{X}_2^m : s_j(x) \oplus s_j(x \oplus \Delta x_j) = \Delta y_j\}| \quad (2.35) \\ &= \frac{1}{2^m} DDT_{s_j}(\Delta x_j, \Delta y_j).\end{aligned}$$

Ezt felhasználva megkapjuk a keresett valószínűséget:

$$\begin{aligned}\Pr\{\Delta Z_{i+1} = \zeta_{i+1} | \Delta Z_i = \zeta_i\} &= \Pr\{\Delta Y' = \zeta_i^{(L)} \oplus \zeta_{i+1}^{(R)} | \Delta X' = \zeta_i^{(R)} = \zeta_{i+1}^{(L)}\} \\ &= \Pr\{\Delta X' \xrightarrow{F_i} \Delta Y'\} \quad (2.36) \\ &= \Pr\{\Delta X \xrightarrow{s_1, s_2, \dots, s_k} \Delta Y\} \\ &= \prod_{j=1}^k \Pr\{\Delta x_j \xrightarrow{s_j} \Delta y_j\} \\ &= \prod_{j=1}^k \frac{1}{2^m} DDT_{s_j}(\Delta x_j, \Delta y_j).\end{aligned}$$

Nagy valószínűségű differencia sorozat keresése. Sajnos a (2.33) kifejezésben szereplő szumma miatt (2.32) értékének pontos meghatározása a gyakorlatban szinte lehetetlen. Szerencsére nekünk az is elég, ha találunk egy olyan $\zeta_1, \zeta_2, \dots, \zeta_r$ sorozatot, melyre

$$\prod_{i=1}^{r-1} \Pr\{\Delta Z_{i+1} = \zeta_{i+1} | \Delta Z_i = \zeta_i\} \quad (2.37)$$

elég nagy, azaz jelentősen eltér az egyenletestől, hiszen ez már biztosítja azt, hogy ζ_1 bemeneti differencia esetén az $(r - 1)$ -edik réteg kimenetén a ζ_r differencia nagy valószínűséggel jelenik meg, és a támadás ezt használja ki. Feladatunkat tehát a következőképpen fogalmazhatjuk meg:

Keressük azt a $\zeta_1, \zeta_2, \dots, \zeta_r$ differencia sorozatot, melyre

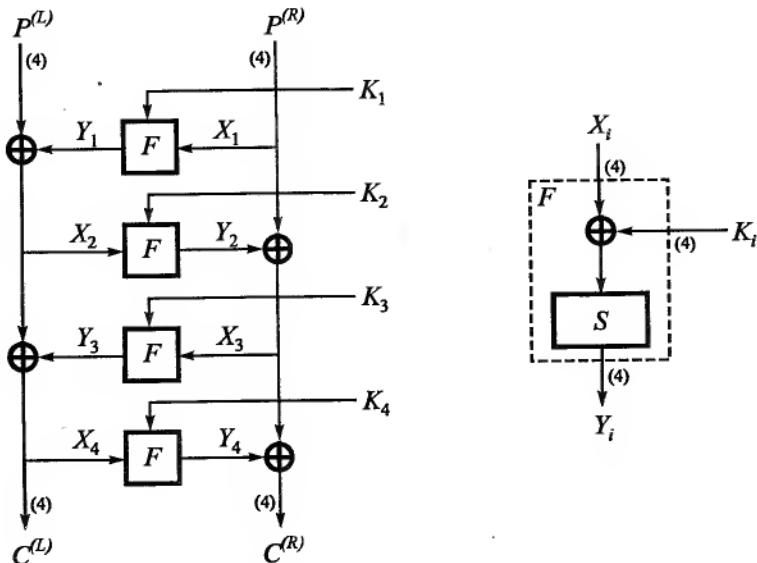
- (i) $\zeta_{i+1}^{(L)} = \zeta_i^{(R)}$ minden $i = 1, 2, \dots, r - 1$ esetén, továbbá
- (ii) $\prod_{i=1}^{r-1} P_i$ maximális, ahol $P_i = \Pr\{\zeta_i^{(R)} \xrightarrow{F_i} \zeta_i^{(L)} \oplus \zeta_{i+1}^{(R)}\}$.

Tippek és triükkök. A könnyebb érthetőség kedvéért eddig nem foglalkoztunk vele, most viszont megemlíjtük, hogy a 2.2. algoritmus 4. pontja kicsit elnagyolt. Nevezetesen arról van szó, hogy ha minden lehetséges kulcs számára fenntartunk egy számlálót, akkor az algoritmus tárígénye borzasztóan nagy lesz, és a gyakorlatban nem is alkalmazhatjuk. Gondolunk például a DES-re, ahol a rétegkulcs 48 bites, és így 2^{48} darab számlálóra lenne szükségünk. Ezért általában nem figyeljük az összes kulcsbitet, helyette csak néhány kulcsbitre, tipikusan egy kiragadott S-dobozba belépő kulcsbitekre koncentrálunk. Ekkor persze a rejtejes blokkokból és az $(r - 1)$ -edik réteg kimeneti differenciájából is csak azokat a biteket használjuk fel, amelyeknek az adott S-dobozra hatásuk van. Ezzel csökkenthetjük az algoritmus tárígényét, ugyanakkor belátható, hogy ekkor több nyílt szöveg – rejttett szöveg párra van szükség a sikeres támadáshoz. Itt tehát kompromisszumra kénytelenünk.

A szükséges nyílt szöveg – rejttett szöveg párok számát ún. quartettek használatával csökkenthetjük. Tegyük fel, hogy rendelkezésünkre áll egy nagy valószínűségű $\zeta_1, \zeta_2, \dots, \zeta_r$ és egy nagy valószínűségű $\zeta'_1, \zeta'_2, \dots, \zeta'_r$ differencia sorozat, és ezen sorozatokat külön-külön használva a rejtelező feltöréséhez M számú ζ_1 illetve ζ'_1 differenciájú nyílt szöveg pár, azaz $2M$ számú nyílt szöveg szükséges. Ügyesen választott nyílt blokkokból álló quartetteket használva párhuzamosan használhatjuk fel a két nagy valószínűségű differencia sorozatot. Tekintsük például a $(P, P \oplus \zeta_1, P \oplus \zeta'_1, P \oplus \zeta_1 \oplus \zeta'_1)$ négyest, mely felfogható két nyílt szöveg párnak, ahol a párokban a differencia ζ_1 , ugyanakkor meghatároz két másik pár is, ahol a differencia ζ'_1 . Egy ilyen quartett minden két differencia sorozat számára két nyílt szöveg párát biztosít. Mivel egyszerre használjuk a két differencia sorozatot, ezért minden kettőnek $\frac{M}{2}$ számú nyílt szöveg pár kell a sikeres támadáshoz. $\frac{M}{4}$ quartett minden két sorozat igényét egyszerre kielégíti. Ez azt jelenti, hogy M darab nyílt blokk elegendő a támadáshoz.

Kérdés, hogy hogyan lehet védekezni a differenciális kriptanalízis ellen. Mivel az előzőek alapján a rejtjelező differenciális viselkedése az S-dobozok differenciális viselkedésétől függ, ezért a kérdésre az S-doboz tervezésben kell keresni a választ. Valóban, ha az S-dobozok differencia eloszlási táblájában nincsenek kiríván magas értékek, akkor bármilyen bemeneti differencia esetén bármilyen kimeneti differencia közel egyenlő valószínűséggel jelenik meg, és így nem lesz olyan bemeneti-kimeneti differencia pár, amelyre (2.32) jelentősen eltér az egyenletestől, tehát a differenciális kriptanalízis nem alkalmazható. Úgy gondolnánk, hogy a célnak legjobban a perfekt S-dobozok felelnek meg, melyek differencia eloszlási táblája teljesen egyenletes. De sajnos a perfekt S-dobozok nem balanszok, ami pedig egy alapvető követelmény. Így azt mondhatjuk, hogy olyan S-dobozokat érdemes keresni, amelyek majdnem perfektek, differencia eloszlási táblájuk majdnem egyenletes, nincs bennük kiríván magas érték, de egyúttal más kriptográfiai tulajdonságokat, mint például a balansz tulajdonságot megőrzik.

2.1. Példa. A differenciális kriptanalízist a 2.5. ábrán látható, demonstrációs célokra szolgáló mini rejtjelezőn szemléltetjük, melynek S-dobozát és differencia eloszlási tábláját rendre a 2.4. és a 2.5. táblázat tartalmazza.



2.5. ábra. A példaként használt mini rejtjelező felépítése

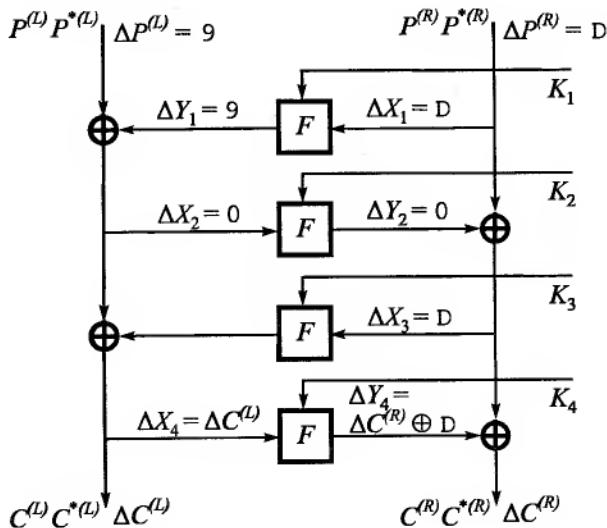
bemenet	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
kimenet	A	D	3	8	1	B	0	5	2	F	C	9	4	7	6	E

2.4. táblázat. A mini rejtjelező S-doboza

16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	2	0	4	0	2	2	0	2	2	0	2	0	0	0
0	2	2	0	0	2	2	0	0	4	0	0	0	0	4	0	0
0	2	2	2	2	0	0	0	0	0	2	4	0	0	2	0	0
0	0	0	2	0	0	4	2	2	0	2	2	0	2	0	0	0
0	2	2	0	0	2	2	0	2	0	0	2	2	0	0	0	2
0	2	2	2	2	0	0	0	4	0	2	0	0	0	2	0	2
0	0	0	0	0	0	0	2	4	0	2	2	4	0	2	0	2
0	2	2	2	0	0	2	2	0	2	0	2	2	0	0	2	2
0	0	0	2	2	2	2	0	0	0	2	0	0	0	2	4	0
0	2	2	0	4	2	2	4	0	0	0	0	0	0	0	0	0
0	4	0	2	0	0	0	2	0	0	2	0	2	2	0	2	0
0	0	0	2	2	2	2	0	0	0	2	0	4	0	2	0	0
0	0	0	0	0	0	0	0	8	0	0	0	4	4	0	0	2
0	0	4	2	0	0	0	2	0	0	2	0	2	2	0	2	0
0	0	0	0	4	0	0	4	2	0	0	2	2	0	0	0	2

2.5. táblázat. A mini rejtjelező S-dobozának differencia eloszlási táblázata

A rejtjelező S-dobozát és annak differencia eloszlási táblázatát megvizsgálva észrevehetjük, hogy $DDT_S(D, 9) = 8$, ami azt jelenti, hogy $\Pr\{D \xrightarrow{S} 9\} = \frac{1}{2}$. Ezért ha a rejtjelező bemeneti differenciájának a $9D$ -t választjuk, akkor a 2.6. ábrán látható differencia átmenetekhez jutunk. Az első réteg F függvényének bemeneti differenciája D , kimeneti differenciája $\frac{1}{2}$ valószínűsséggel 9. A második réteg F függvényének bemeneti differenciája így $\frac{1}{2}$ valószínűsséggel 0. Ekkor persze a kimeneti differencia is 0. Így a harmadik réteg F függvényének bemeneti differenciája $\frac{1}{2}$ valószínűsséggel D , azaz az utolsó réteg F függvényének kimeneti differenciája $\frac{1}{2}$ valószínűsséggel $\Delta C^{(R)} \oplus D$. Mivel a rejtjeles blokkok ismertek, ezért az utolsó réteg F függvényének ismerjük $C^{(L)}$ és $C^{*(L)}$ bemeneteit és $\frac{1}{2}$ valószínűsséggel ismerjük kimeneti differenciáját, ami alapján meghatározhatjuk az utolsó réteg lehetőséges kulcsait. Több $9D$ differenciájú nyílt blokk párt is megvizsgálva számolájuk az egyes kulcsok előfordulásának számát. A leggyakrabban megjelenő kulcs az igazi.



2.6. ábra. Differencia-átmenetek 9D bemeneti differenciával esetén

A fenti gondolatokat felhasználva elvégeztünk egy kísérletet. A rejtjelező kulcsának az ABBA értéket választottuk, így az utolsó réteg kulcsa az A volt. Rejtjeleztünk 20 véletlenszerűen választott nyílt szöveg párt, ahol a párok elemeinek differenciája a 9D volt. A rejtjeles blokkokat felhasználva és feltételezve, hogy az utolsó réteg F függvényének kimeneti differenciája $\Delta C^{(R)} \oplus D$, minden párra meghatároztuk az utolsó réteg lehetséges kulcsainak halmazát. Számoltuk, hogy az egyes kulcsok hányszor jelentek meg lehetséges kulcsként. Az eredményeket a 2.6. táblázat mutatja, melyből jól látható, hogy az A fordult elő legtöbbször.



bemenet	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
kimenet	0	0	2	4	2	3	0	2	3	3	8	3	1	0	2	1

2.6. táblázat. Lehetséges kulcsok előfordulásának gyakorisága 9D bemeneti differenciával esetén

2.3.2. Lineáris kriptanalízis

A lineáris kriptanalízis alapvetően egy ismert nyílt szövegű támadás DES típusú rejtjelezők ellen. A támadás célja a kulcs vagy a kulcs egy részének megfejtése egy effektív lineáris közelítés és nagy számú nyílt szöveg - rejtett szöveg pár segítségével. Központi kérdés az effektív lineáris közelítés megkeresése, ezért először ezt vizsgáljuk majd. Előtte azonban bebizonyítjuk a következő lemmát, amit a későbbiekben gyakran fel fogunk használni.

2.11. Lemma (Piling-up lemma). *Legyenek X_1, X_2, \dots, X_n független valószínűségi változók, melyekre $X_i \in \{0, 1\}$ és $\Pr\{X_i = 0\} = p_i = \frac{1}{2} + q_i$ minden i -re. Ekkor*

$$\Pr\left\{\bigoplus_{i=1}^n X_i = 0\right\} = \frac{1}{2} + 2^{n-1}q_1q_2\dots q_n. \quad (2.38)$$

Bizonyítás: A lemma állítását n szerinti teljes indukcióval bizonyítjuk be. Először vizsgáljuk az $n = 2$ esetet:

$$\begin{aligned} \Pr\{X_1 \oplus X_2 = 0\} &= \Pr\{X_1 = 0, X_2 = 0\} + \Pr\{X_1 = 1, X_2 = 1\} \\ &= p_1p_2 + (1-p_1)(1-p_2) \\ &= \left(\frac{1}{2} + q_1\right)\left(\frac{1}{2} + q_2\right) + \left(\frac{1}{2} - q_1\right)\left(\frac{1}{2} - q_2\right) \\ &= \frac{1}{2} + 2q_1q_2. \end{aligned}$$

Most tegyük fel, hogy az állítás $n - 1$ -re igaz. Legyen

$$Y = \bigoplus_{i=1}^{n-1} X_i.$$

Az indukciós feltétel miatt, $\Pr\{Y = 0\} = \frac{1}{2} + 2^{n-2}q_1q_2\dots q_{n-1} = \frac{1}{2} + Q$, valamint

$$\begin{aligned} \Pr\left\{\bigoplus_{i=1}^n X_i = 0\right\} &= \Pr\{Y \oplus X_n = 0\} \\ &= \frac{1}{2} + 2Qq_n \\ &= \frac{1}{2} + 2^{n-1}q_1q_2\dots q_n. \end{aligned}$$

□

A rejtjelező lineáris közelítése. A linearizálást az S-dobozok lineáris közelítésével kezdjük, majd kiterjesztjük az egész rejtjelezőre.

Minden $s : \mathcal{X}_2^m \rightarrow \mathcal{X}_2^n$ S-doboz és $\alpha \in \mathcal{X}_2^m, \beta \in \mathcal{X}_2^n$ esetén felírhatjuk a következő lineáris kifejezést:

$$\alpha \odot x \oplus \beta \odot s(x) = 0. \quad (2.39)$$

Ez az s függvény egy lineáris közelítése, mely

$$\begin{aligned} p &= \frac{1}{2^m} |\{x \in \mathcal{X}_2^m : \alpha \odot x \oplus \beta \odot s(x) = 0\}| \\ &= \frac{1}{2^m} (LAT_s(\alpha, \beta) + 2^{m-1}) \\ &= \frac{1}{2} + \frac{LAT_s(\alpha, \beta)}{2^m} \\ &= \frac{1}{2} + q \end{aligned}$$

valószínűséggel áll fenn.

Egy k S-dobozból álló S-doboz réteg felfogható úgy is, mint $S : \mathcal{X}_2^{km} \rightarrow \mathcal{X}_2^{kn}$ leképezés, ahol $S(X') = (s_1(x_1), s_2(x_2), \dots, s_k(x_k))$, $X' = (x_1, x_2, \dots, x_k)$, $x_i \in \mathcal{X}_2^m$, $s_i : \mathcal{X}_2^m \rightarrow \mathcal{X}_2^n$, $i = 1, 2, \dots, k$. Ha minden s_i -re van lineáris közelítésünk:

$$\alpha_i \odot x_i \oplus \beta_i \odot s_i(x_i) = 0, \quad (2.40)$$

mely $\frac{1}{2} + q_i$ valószínűséggel áll fenn, akkor az S-doboz réteget közelíthetjük az

$$A' \odot X' \oplus B' \odot S(X') = 0 \quad (2.41)$$

kifejezéssel, ahol $A' = (\alpha_1, \alpha_2, \dots, \alpha_k)$, $B' = (\beta_1, \beta_2, \dots, \beta_k)$. (2.41) teljesülésének valószínűsége a Piling-up lemma alapján $Q' = \frac{1}{2} + 2^{k-1}q_1q_2 \dots q_k$.

Megjegyezzük, hogy ha minden $i = 1, 2, \dots, k$ -ra $\alpha_i = 0$ és $\beta_i = 0$, akkor minden i -re $q_i = \frac{1}{2}$, és így $Q' = 1$, azaz az S-doboz réteg egy olyan lineáris közelítését nyertük, mely biztosan fennáll. Később, mikor a rejtjelező minden rétegét közelítjük majd, szívesen alkalmazzuk a fenti biztos közelítést a lehető legtöbb rétegben, mert ez növeli a teljes rejtjelezőre kapott közelítés fennállásának valószínűségét. Persze minden rétegben nem alkalmazhatjuk, mert akkor a rejtjelező egy olyan közelítéséhez jutnánk, amely ugyan 1 valószínűséggel fennáll, de egyetlen bemeneti, kimeneti és kulcsbitet sem tartalmaz, és így használhatatlan.

Itt ragadjuk meg az alkalmat, hogy bevezessük az aktív S-doboz fogalmát, amit később még használni fogunk. Ha valamely i -re $\alpha_i \neq 0$ vagy $\beta_i \neq 0$, akkor azt mondjuk, hogy az i -edik S-doboz aktív a közelítés során.

Felhasználva, hogy az S-doboz réteg bemenetén megjelenő X' -re $X' = E \odot X \oplus K$, továbbá az S-doboz réteg kimenetén megjelenő Y' -re $Y' = P^{-1} \odot Y$, a (2.41) lineáris közelítést kiterjeszhetjük az egész F függvényre:

$$A' \odot (E \odot X \oplus K) \oplus B' \odot P^{-1} \odot Y = 0, \quad (2.42)$$

azaz

$$A \odot X \oplus B \odot Y \oplus G \odot K = 0, \quad (2.43)$$

ahol $A = A' \odot E$, $B = B' \odot P^{-1}$ és $G = A'$. (2.43) valószínűsége megegyezik (2.41) valószínűségével. Ez tehát azt jelenti, hogy az F függvény bemenetének, kimenetének, és az F függvénybe belépő kulcsnak az A , a B , illetve a G vektorok által maszkolt bitjei Q' valószínűséggel lineáris kapcsolatban állnak egymással. Mivel A értéke általában meghatározza G értékét, ezért (2.43) helyett röviden csak annyit írunk, hogy $\Pr\{A[F]B\} = Q'$.

Az utolsó lépés a közelítés kiterjesztése az összes rétegre. Ebben fel fogjuk használni, hogy a rejtelező Feistel-típusú, azaz, hogy az i -edik réteg (L_i, R_i) bemenetére és (L_{i+1}, R_{i+1}) kimenetére:

$$\begin{aligned} L_{i+1} &= R_i \\ R_{i+1} &= L_i \oplus F(R_i, K_i). \end{aligned}$$

ahol K_i az i -edik réteg kulcsa. Megjegyezzük, hogy a fenti jelölést használva az utolsó réteg után L_{r+1} és R_{r+1} értékét fel kell cserélni.

Ha mind az r réteg F függvényére van lineáris közelítésünk:

$$A_i \odot X_i \oplus B_i \odot Y_i \oplus G_i \odot K_i = 0, \quad (2.44)$$

ahol $X_i = R_i = L_{i+1}$, $Y_i = L_i \oplus R_{i+1} = X_{i-1} \oplus X_{i+1}$, $i = 1, 2, \dots, r$, továbbá a közelítések olyanok, hogy minden $i = 2, 3, \dots, r - 1$ esetén teljesül, hogy $B_{i-1} \oplus A_i \oplus B_{i+1} = 0$, akkor az r darab egyenlet összeadása után azt kapjuk, hogy

$$\begin{aligned} B_1 \odot X_0 \oplus (A_1 \oplus B_2) \odot X_1 \oplus (A_r \oplus B_{r-1}) \\ \odot X_r \oplus B_r \odot X_{r+1} \oplus \bigoplus_{i=1}^r G_i \odot K_i = 0, \end{aligned} \quad (2.45)$$

azaz a teljes rejtjelező lineáris közelítését nyerjük:

$$\pi \odot P \oplus \gamma \odot C = \kappa \odot K, \quad (2.46)$$

ahol $\pi = (B_1, A_1 \oplus B_2)$, $\gamma = (B_r, A_r \oplus B_{r-1})$, továbbá P jelöli a nyílt blokkot, $P = (X_0, X_1)$, és C jelöli a rejtjeles blokkot, $C = (X_{r+1}, X_r)$. K a rejtjelező teljes kulcsa, κ pedig a kulcsütemező, és a G_i -k ismeretében könnyen meghatározható. Ha (2.44) $Q'_i = \frac{1}{2} + Q_i$ valószínűséggel állt fenn, akkor (2.46) valószínűsége $\frac{1}{2} + 2^{r-1} Q_1 Q_2 \dots Q_r = \frac{1}{2} + Q$.

A lineáris közelítés használhatósága a $|Q|$ értékétől függ. Minél nagyobb $|Q|$ értéke, annál effektívebb a lineáris közelítés, ugyanis annál jobban eltér $\frac{1}{2}$ -től annak a valószínűsége, hogy (2.46) fennáll.

A támadás algoritmusa. A (2.46) lineáris közelítés ismeretében meghatározhatunk egy bitet a kulcsból, nevezetesen $\kappa \odot K$ értékét. Ezt a következő algoritmus segítségével tehetjük meg:

2.3. Algoritmus. Az algoritmus $\kappa \odot K$ értékét próbálja meg kitalálni. Az algoritmus lépései a következők:

1. Vegyük N darab nyílt blokk – rejtett blokk párt.
2. Legyen T azon párok száma, melyekre (2.46) bal oldala 0.
3. Ha $T > N/2$, akkor $Q > 0$ esetén $\kappa \odot K = 0$ -ra, míg $Q \leq 0$ esetén $\kappa \odot K = 1$ -re döntünk, egyébként $Q > 0$ esetén $\kappa \odot K = 1$ -re, míg $Q \leq 0$ esetén $\kappa \odot K = 0$ -ra döntünk.

Az algoritmus annál sikeresebben működik, minél nagyob N és $|Q|$ értéke. Mint látható, ezzel a módszerrel csak egy kulcsbitet lehet kitalálni, ami még nem elegendő. Egy kis továbbgondolással azonban a módszer alkalmass több kulcsbit megfejtésére is. Az ötlet azon alapszik, hogy a Feistel-típusú rejtjelezőknél az utolsó réteg F függvényének bemenete ismert, hisz az nem más, mint a rejtjeles blokk jobb fele. Felhasználva egy $(r-1)$ réteges lineáris közelítést:

$$\pi \odot P \oplus \gamma \odot C' = \kappa \odot K, \quad (2.47)$$

ahol C' az $(r-1)$ -edik réteg kimenete, és azt a tényt, hogy $C' = (L_r, R_r) = (L_{r+1} \oplus F(R_{r+1}, K_r), R_{r+1})$, azt kapjuk, hogy

$$\pi \odot P \oplus \gamma \odot C \oplus \phi \odot F(R_{r+1}, K_r) = \kappa \odot K, \quad (2.48)$$

ahol $\phi = \gamma^{(L)}$, ami γ bal felét jelöli, és $C = (L_{r+1}, R_{r+1})$. (2.48) fennállásának valószínűsége megegyezik (2.47) valószínűséggel. (2.48) bal oldalát ismert

nyílt blokk – rejtett blokk párok esetében kiszámolhatjuk, ha K_r helyére behelyettesítünk egy értéket. Ha K_r helyére rossz értéket helyettesítünk, akkor a bal oldal közel $\frac{1}{2}$ valószínűséggel 0, ellenben helyes érték behelyettesítése esetén a bal oldal $\frac{1}{2}$ -től eltérő valószínűséggel lesz 0, feltéve, hogy (2.48) elég effektív. Így a következő algoritmust alkalmazhatjuk, melyel egynél több kulcsbitet is kitalálhatunk:

2.4. Algoritmus. A több kulcsbit megfejtését végző algoritmus lépései a következők:

1. Vegyük N darab nyílt blokk – rejtett blokk párt.
2. K_r helyére helyettesítsük be az összes lehetséges $K_r^{(i)}$ értéket ($i = 1, 2, \dots$), és jelöljük T_i -vel azon párok számát, melyekre (2.48) bal oldala 0.
3. Legyen $T_{max} = \max_i T_i$ és $T_{min} = \min_i T_i$.
4. Ha $|T_{max} - N/2| > |T_{min} - N/2|$, akkor a T_{max} -hoz tartozó K_r értéket fogadjuk el, és $Q > 0$ esetén $\kappa \odot K = 0$ -ra, míg $Q \leq 0$ esetén $\kappa \odot K = 1$ -re döntünk. Ha $|T_{max} - N/2| \leq |T_{min} - N/2|$, akkor a T_{min} -hez tartozó K_r értéket fogadjuk el, és $Q > 0$ esetén $\kappa \odot K = 1$ -re, míg $Q \leq 0$ esetén $\kappa \odot K = 0$ -ra döntünk.

Effektív lineáris közelítés keresése. Most bevezetjük a maszk sorozat fogalmát, amelynek segítségével az effektív lineáris közelítés keresésének problémáját a nagy valószínűségű differencia sorozat keresésének problémájához hasonlóan fogalmazhatjuk meg. Jelöljük az i -edik réteg bemenetén alkalmazott maszkot $\xi_i = (\xi_i^{(L)}, \xi_i^{(R)})$ -rel. Maszk sorozat alatt az egyes rétegek bemenetén alkalmazott maszkok $\xi_1, \xi_2, \dots, \xi_r$ sorozatát értjük. Ha azt szeretnénk, hogy az egyes rétegek F függvényének bemeneti és kimeneti maszkjaira minden $i = 2, 3, \dots, r-1$ esetén teljesüljön a $B_{i-1} \oplus A_i \oplus B_{i+1} = 0$ összefüggés, és az egyes rétegek lineáris közelítéseit összeadva a rejtjelező (2.46) szerinti lineáris közelítéséhez jussunk, akkor az $A_i = \xi_i^{(R)} \oplus \xi_{i+1}^{(L)}$, illetve $B_i = \xi_i^{(L)} = \xi_{i+1}^{(R)}$ egyenleteknek kell teljesülniük. Ezeket figyelembe véve a legjobb ($r-1$) rétegű lineáris közelítés megkeresésének feladata a következőképpen fogalmazható meg:

Keressük azt a $\xi_1, \xi_2, \dots, \xi_r$ maszk sorozatot, melyre

(i) $\xi_i^{(L)} = \xi_{i+1}^{(R)}$ minden $i = 1, 2, \dots, r-1$ esetén, továbbá

(ii) $\prod_{i=1}^{r-1} |Q_i|$ maximális, ahol $Q_i = \Pr\{\xi_i^{(R)} \oplus \xi_{i+1}^{(L)} [F_i] \xi_{i+1}^{(R)}\} - \frac{1}{2}$.

Figyeljük meg az effektív lineáris közelítés keresése feladatának és a nagy valószínűségű differencia sorozat keresése feladatának nagy mértékű formai hasonlóságát.

Tippek és trükkök. A 2.4. algoritmus második pontja első látásra elszomorító, hiszen az összes lehetséges K_r kipróbálása elég reménytelen dolog. Szerencsére $\phi \odot F(R_{r+1}, K_r)$ értéke nem függ K_r összes bitjétől, sőt általában csak kevés bittel van kapcsolatban. Lényegében ϕ -től függ, hogy hánny értéket kell behelyettesíteni K_r -be. Ha például ϕ olyan maszkot eredményez az S-doboz réteg kimenetén, amely csak egyetlen S-dobozt tesz aktívvá, akkor $2^m K_r$ érték jöhét számításba, ahol m az S-doboz bemenetének mérete.

A lineáris kriptanalízis alkalmazhatósága azon múlik, hogy létezik-e effektív lineáris közelítés. Mivel a rejtjelező lineáris közelítésének valószínűsége visszavezethető az S-dobozok közelítésének valószínűségére, ezért ha az S-dobozok nem közelíthetők effektíven, akkor a teljes rejtjelező sem. Az S-dobozok lineáris közelíthetőségét pedig lineáris approximációs táblájuk határozza meg. Ha a lineáris approximációs táblában nincs kirívónan nagy abszolút értékű elem, akkor nincs igazán effektív lineáris közelítés sem. A tökéletes megoldásnak itt is perfekt S-dobozok látszanak, de az előző szakszabban már tisztázottuk, hogy a tökéletes egyenletesség miért nem jó mégsem. A megoldást tehát ismét a majdnem perfekt S-dobozok jelentik, melyek nem-linearitása a lehető legnagyobb, de más kriptográfiai tulajdonságaik (pl. a balansz tulajdonság) is megvannak.

2.2. Példa. A lináris kriptanalízist a 2.1. példa mini rejtjelezőjén szemléltetjük. Ehhez szükségünk lesz a mini rejtjelező S-dobozának lineáris approximációs táblájára, melyet a 2.7. táblázat tartalmaz.

Az S-doboz lineáris approximációs táblázatát megfigyelve látható, hogy $LAT_S(9, 6) = 4$, azaz az S-doboz

$$9 \odot x \oplus 6 \odot s(x) = 0$$

lineáris közelítése $q'_1 = \frac{1}{2} + \frac{1}{4}$ valószínűsséggel fennáll. A lináris közelítést kiterjesztve az F függvényre a következő egyenlethez jutunk:

$$9 \odot X_1 \oplus 9 \odot K_1 \oplus 6 \odot Y_1 = 0,$$

mely ugyancsak q'_1 valószínűsséggel áll fenn. A második és a harmadik rétegben a

8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	4	0	0	2	-2	-2	-2	4	0	0	0	0	2	-2	2	2
0	-2	-2	0	0	2	-2	-4	0	2	2	0	0	-2	2	-4	
0	2	2	-4	2	0	0	-2	0	-2	2	0	-2	0	-4	-2	
0	0	0	0	2	2	-2	-2	-4	0	-4	0	2	-2	-2	2	
0	0	-4	0	0	0	-4	0	0	-4	0	0	0	4	0	0	
0	2	-2	-4	-2	4	0	2	0	2	2	0	2	0	0	2	
0	2	-2	0	4	2	2	0	0	2	-2	0	-4	2	2	0	
0	-2	2	0	4	2	-2	4	0	-2	2	0	0	-2	2	0	
0	-2	-2	0	2	0	4	-2	0	-2	2	4	2	0	0	2	
0	0	0	-4	0	-4	0	0	-4	0	0	0	0	0	4	0	
0	0	0	0	-2	2	2	-2	0	-4	0	-4	-2	-2	2	2	
0	2	-2	0	-2	0	0	2	0	-2	-2	4	-2	-4	0	-2	
0	-2	-2	0	0	-2	-2	0	0	2	2	0	-4	-2	-2	4	
0	0	4	0	-2	2	-2	-2	0	0	0	4	-2	2	2	2	
0	4	0	4	0	0	0	0	-4	0	4	0	0	0	0	0	

2.7. táblázat. A mini rejtjelező S-dobozának lineáris approximációs táblája

$$4 \odot X_2 \oplus 4 \odot K_2 \oplus 8 \odot Y_2 = 0$$

$$5 \odot X_3 \oplus 5 \odot K_3 \oplus 2 \odot Y_3 = 0$$

közeliítéseket használjuk, melyek $q'_2 = q'_3 = \frac{1}{2} - \frac{1}{4}$ valószínűséggel állnak fenn.

Felhasználva, hogy $X_1 = P^{(R)}$, $Y_1 = P^{(L)} \oplus X_2$, $Y_2 = P^{(R)} \oplus C^{(R)} \oplus F(C^{(L)}, K_4)$, $X_3 = C^{(R)} \oplus F(C^{(L)}, K_4)$ és $Y_3 = C^{(L)} \oplus X_2$, valamint összeadva a fenti lineáris közelítéseket a rejtjelező következő lineáris közelítéséhez jutunk:

$$\begin{aligned} 6 \odot P^{(L)} \oplus 1 \odot P^{(R)} \oplus 2 \odot C^{(L)} \oplus D \odot C^{(R)} \oplus D \odot F(C^{(L)}, K_4) \\ \oplus 9 \odot K_1 \oplus 4 \odot K_2 \oplus 5 \odot K_3 = 0 \end{aligned}$$

Tömörebben:

$$61 \odot P \oplus 2D \odot C \oplus D \odot F(C^{(L)}, K_4) = 9450 \odot K,$$

mely $Q' = \frac{1}{2} + 2^2 \frac{1}{4} (-\frac{1}{4}) (-\frac{1}{4}) = \frac{1}{2} + \frac{1}{16}$ valószínűséggel igaz. Ezt a közelítést használhatjuk a 2.4. algoritmusban. ♣

2.3.3. A DES tervezése során alkalmazott tervezési kritériumok

Miután a differenciális kriptanalízis nyilvánosan is ismertté vált (a kilencvenes évek elején), a DES tervezői elismerték, hogy a DES tervezésekor (a hetvenes évek elején!) ők már ismerték ezt a támadási módszert, és mind az S-dobozokat, minden a rétegek számát úgy választották meg, hogy a DES ellenálljon a differenciális kriptanalízisen alapuló támadásnak. Bár nem árulták el, hogy vajon ez volt-e a legerősebb támadás, aminek tudatában voltak, és ami ellen felkészítették a DES-t, az S- és P-doboz tervezés során használt kritériumokat felfedték.

A DES S-doboz tervezési kritériumai a következők voltak:

- minden S-doboz bemenete 6, kimenete 4 bites legyen. (Az akkor technológia korlátai maximálisan ekkora méretű S-dobozokat engedtek meg. Ezek a méretek lehetővé tették, hogy a DES-t egyetlen chip-be integrálják.)
- Egyetlen S-doboz egyetlen kimeneti bitje se legyen közel a bemeneti bitek valamely lineáris függvényéhez. (Tehát a nem-linearitás legyen nagy.)
- Ha rögzítjük a két szélső bit értékét, és csak a bemenet középső négy bit-jét változtatjuk folyamatosan, akkor a kimeneten minden 4 bites vektor pontosan egyszer jelenjen meg. (Azaz az S-dobozban található 4 darab 4 bitet 4 bitbe helyettesítő tábla mindegyike legyen balansz. Ekkor persze maga az S-doboz is balansz, vagyis minden 4 bites kimeneti vektor pontosan négyszer jelenik meg, ha a bemeneten minden lehetséges értéket végigpörgetünk.)
- Ha az S-doboz bemenetén egyetlen bitet megváltoztatunk, akkor a kimeneten legalább két bit értéke változzon meg (lavinahatás).
- Ha az S-doboz bemenetén a két középső bitet megváltoztatjuk, akkor a kimeneten legalább két bit értéke változzon meg.
- Ha két bemeneti vektor első két bitje különböző, utolsó két bitje azonos, akkor a megfelelő kimeneti vektorok nem lehetnek azonosak.
- Tetszőleges, nem nulla bemeneti differencia esetén, az adott differenciával rendelkező 32 bemeneti vektorpár közül legfeljebb nyolchoz tartozhat azonos kimeneti differencia. (Azaz a differenciális egyenletesség legyen nagy.)
- Az előzőhöz hasonló kritérium, de egyszerre három S-dobozra.

A DES P-doboz tervezési kritériumai a következők voltak:

- A P-doboz legyen olyan, hogy minden S-doboz négy kimeneti bitje közül kettőt a következő réteg S-dobozainak középső bitjeihez, kettő pedig szélső (táblázat választó) bitekhez továbbítson.
- minden S-doboz négy kimeneti bitje a következő rétegen hat különböző S-dobozra legyen hatással.
- Ha egy S-doboz valamely kimeneti bitje egy másik S-doboz valamely középső bitjéhez van vezetve, akkor ez utóbbi S-doboz egyetlen kimenete sem lehet az előző S-doboz középső bemeneteihez vezetve.

2.4. Algebrai zártsgág és többszörös rejtjelezés

Tekintsük azon egyműveletes $T = \{E, \cdot\}$ algebrai struktúrát a rejtjelező transzformációk E halmazán, ahol a „.” művelet két transzformáció egymás utáni alkalmazása. Nevezzük ezt a műveletet szorzásnak.

2.12. Definíció. *T zárt, ha $E \cdot E = E$, azaz tetszőleges két, E halmazbeli transzformáció szorzata E -beli transzformáció. Formálisan: bármely K_1 és K_2 esetén létezik K_3 , hogy*

$$E_{K_1} \cdot E_{K_2} = E_{K_3}.$$

Nem nehéz belátni, hogy T csoporttulajdonságokkal rendelkezik:

2.13. Tétel. *A T zárt algebrai struktúra csoport.*

Bizonyítás: Azt kell belátni, hogy T tartalmazza az identitást, s minden elemnek létezik az inverze. Az egyszerűbb jelölés kedvéért E_i álljon E_{K_i} helyett. Tetszőleges E_1, E_2, E_3 esetén, ahol $E_1 \neq E_3$ fennáll, hogy

$$E_1 \cdot E_2 \neq E_3 \cdot E_2. \quad (2.49)$$

Ha ugyanis (2.49) nem állna fenn, E_2 inverzával jobbról szorozva minden oldalt ellentmondásra jutnánk. Következésképpen

$$\{E_1 \cdot E_2 : E_1 \in E\} = E. \quad (2.50)$$

Ebből következik, hogy tetszőleges E_2 transzformációhoz létezik olyan E_2^* transzformáció, amelyre $E_2^* \cdot E_2 = E_2$, ahol minden oldalt jobbról E_2 inverzával szorozva, a jobb oldalon az I identitás transzformációt kapjuk, azaz $E_2^* = I$, s így $I \in E$.

Ha tetszőleges $E_2 \in E$ transzformációt balról szorzunk a különböző E -beli elemekkel, (2.50) alapján a szorzat végigfutja az összes E -beli elemet, közöttük az I identitást is. Azaz létezik $E'_2 \in E$ transzformáció, amelyre $E'_2 \cdot E_2 = I$. E'_2 az E_2 transzformáció inverze ($E'_2 = E_2^{-1}$). \square

A zártsgág tehát esetünkben további algebrai strukturáltságot von maga után. Egy blokkrejtjelező algebrai zártsgágának ellentéte az az eset, amikor E a $\{0, 1\}^n$ feletti permutációk halmazának egy véletlenszerűen választott részhalmaza. Ezen utóbbi esetben két E -beli transzformáció egymás utáni alkalmazása nagy valószínűsséggel nem E -beli permutációt eredményez.

A zártsgából következő csoporttulajdonság kedvezőtlen, mert az ún. *középen találkozás* algoritmikus támadásra ad lehetőséget. A támadás alapja a valószínűségszámításból már jól ismert születésnapi paradoxon (lásd 4.2. szakasz).

2.4.1. Középen találkozás támadás zárt struktúrájú rejtjelező ellen

A támadás célja az éppen alkalmazottal ekvivalens transzformáció konstruálása. A támadáshoz szükséges, hogy a támadó rendelkezzen a pillanatnyilag használt kulccsal előállított nyílt – rejtett blokkpárok valahogyan megszerzett halmazával. Jelöljük ezt a halmazt Q -val, ahol tehát $Q = \{(x_1, y_1), (x_2, y_2), \dots, (x_s, y_s)\}$ és $y_\ell = E_K(x_\ell)$. $E_K \in E$ a nem ismert, pillanatnyilag alkalmazott transzformáció, ahol K jelöli a nem ismert kulcsot. A támadás megértését segíti a 2.7. ábra.

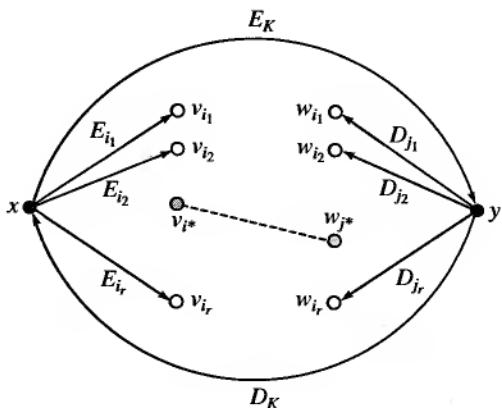
A Q halmazból kiveszünk egy (x, y) párt, s a nyílt x blokkot r számú véletlenszerűen választott kulccsal kódoljuk: $v_{i_\ell} = E_{i_\ell}(x)$, $\ell = 1, 2, \dots, r$. Hasonlóan az y rejtett blokkot r számú véletlenszerűen választott kulccsal dekódoljuk: $w_{j_\ell} = D_{j_\ell}(y)$, $\ell = 1, 2, \dots, r$. Megvizsgáljuk, hogy van-e a $V = \{v_{i_1}, v_{i_2}, \dots, v_{i_r}\}$ és a $W = \{w_{j_1}, w_{j_2}, \dots, w_{j_r}\}$ halmaznak közös eleme. Ha van, s például $v_{i_*} = w_{j_*}$ egy ilyen közös elempár, akkor sejtésünk az, hogy

$$E_K = E_{i_*} \cdot E_{j_*}. \quad (2.51)$$

Ezt a sejtésünket ellenőrizzük további Q halmazbeli párokon. Azaz megvizsgáljuk, hogy fennáll-e az

$$E_K(x_\ell) = E_{j_*}(E_{i_*}(x_\ell)) \quad (2.52)$$

egyenlőség $\ell = 2, \dots, s$ -re. Ha igen, akkor a Q halmaz mérete szerinti biztonsággal igazoltuk sejtésünket. Ha valamelyik Q -beli nyílt-rejtett páron sejtünk megdől, akkor egy másik közös elempárt vizsgálunk, ha van további;



2.7. ábra. A középen találkozás támadás szemléltetése

ha azonban nincsen, akkor előről újrakezdjük az eljárást, újabb $2r$ kulcsot sorsolva.

Kérdés, hogy mekkora legyen r értéke, illetve a Q halmaz mérete. Az r értékkel kacsolatosan a választ a születésnapi paradoxon adja. A Φ és Ψ halmazok legyenek transzformációk következő halmazai:

$$\Phi = \{E_{i_1}, E_{i_2}, \dots, E_{i_r}\}$$

$$\Psi = \{D_{j_1} \cdot E_K, D_{j_2} \cdot E_K, \dots, D_{j_r} \cdot E_K\},$$

ahol a $\Phi \subset E$ állítás nyilvánvaló, továbbá, mivel a csoport tulajdonság miatt $E_{j_\ell}^{-1} \in E$, azaz $D_{j_\ell} \in E$, $\ell = 1, 2, \dots, r$ is fennáll, ezért $\Psi \subset E$ is fennáll. Ha Φ és Ψ részhalmazoknak van közös eleme, azaz ha például $E_{i^*} = D_{j^*} \cdot E_K$, akkor minkét oldalt balról E_{j^*} -gal szorozva $E_K = E_{j^*} \cdot E_{i^*}$ adódik, azaz a nem ismert E_K aktuális transzformációt két véletlenül sorsolt transzformáció szorzataként előállítottuk.

A születésnapi paradoxon változata alapján annak valószínűsége, hogy Φ és Ψ halmazok metszete nem üres, elegendően nagy egy sikeres támadáshoz, ha r értéke az E elemszáma négyzetgyöökének nagyságrendjébe esik. Ez ekvivalensen a kulcsok halmaza méretének négyzetgyöökét jelenti. Például egy 60 bites kulcsméret esetén 2^{30} nagyságrendről van szó.

A Q nagyságrendjét illetően azt kell észrevennünk, hogy egy (2.52) szerinti sejtés-ellenőrzési lépésen egy téves sejtés $2^{-|\{0,1\}^n|}$ valószínűséggel

megy át, azaz kisszámú ellenőrző lépés nagy biztonságú ellenőrzést jelent. Az eljárás praktikusságának korlátját tehát az r értéke adja.

Az algebrai zártság tehát veszedelmes tulajdonság lehet. Ugyanakkor ki-
csi az esélye, hogy egy rejtjelező kódoló ilyen tulajdonsággal rendelkezzen. A fenti gondolatoknak praktikus haszna, ezért elsősorban nem közvetlenül a kódoló transzformációk esetén, hanem a többszörös rejtjelezés esetén van.

2.4.2. Többszörös rejtjelezés

Több különböző kódolási eljárás kombinálásától azt reméljük, hogy egy erő-
sebb kódolóra jutunk. Az egyik legközvetlenebb ezen irányba tett lépés a
kétszeres kódolás:

$$\begin{aligned} y &= E_{K_2}(E_{K_1}(x)) \\ x &= D_{K_1}(D_{K_2}(y)), \end{aligned}$$

ahol $E_1, E_2 \in E$.

Jelölje az egylépéses kódolás kulcshosszát k . A kulcshossz a kétszeres kódolással formailag kétszeresére nőtt, hiszen két kulcsot használunk. Formailag, mert ténylegesen csak akkor van így, ha a kétszeres kódolással a kétszeres kulcshossznak megfelelő számú különböző transzformációt állítanánk elő. Ha a kódolónk csoporttulajdonságú, akkor ez biztosan nincs így, hiszen ez esetben $E_{K_2}(E_{K_1}(x)) = E_{K_3}(x)$, valamely $E_{K_3} \in E$ esetén. Ekkor tehát egyáltalán nem változott a különböző transzformációk száma, azaz kétszeres kódolással csak ugyanazon transzformációkat tudjuk előállítani, mint egyszeres kódolással.

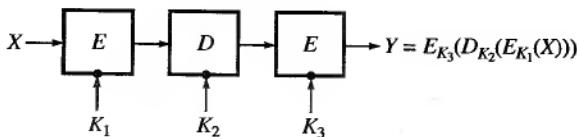
A kétszeres kódolás általános hiányossága az, hogy a kimerítő kipróbálásos támadásnál lényegesen egyszerűbb kínálkozik, ugyanis a kétszeres kódolás lehetővé tesz középen találkozás támadást.

Tudjuk, hogy két egylépéses kódoló transzformáció szorzata a nem ismert transzformáció. Az x nyílt blokkot kódoljuk egylépéses kódolással az összes különböző kulccsal, s hasonlóan az y rejtett blokkot (x pájrát) dekódoljuk egylépésesek dekódolással az összes különböző kulccsal. A kódolás és dekódolás eredmények halmazának metszetét vizsgáljuk. Ha egy $E_{K_i}(x)$ kódolási lépés eredménye egybeesik egy $D_{K_j}(y)$ dekódolási lépés eredményével, akkor a keresett ismeretlen kétlépéses transzformációra a sejtésünk $E_{K_i} \cdot E_{K_j}$, amit további nyílt-rejtett blokkokon történő kipróbálással ellenőrzünk. Nyilvánvaló, hogy ez esetben a metszetben benne van a keresett pár is.

Tehát, az egyszeres rejtjelezés elleni kimerítő kulcskeresés átlagos műveletigénye $|K|/2$, a kétszeres rejtjelezés elleni kimerítő kulcskeresésé pedig – várakozásainkkal ellentétben – nem ezen műveletszám négyzete, hanem csak annak kétszerese.

2.4.3. 3DES

A DES-ről bebizonyították, hogy nem csoport. Ezért van értelme többszörös kódolással erősíteni, és ezzel rövid kulcsméretet megnövelni. Az előző fejezetben láttuk, hogy kétszeres DES rejtjelezéssel nem érünk el jelentős javulást. A következő lehetőség tehát a háromszoros kódolás. Az így nyert rejtjelezőt 3DES-nek nevezik, és a gyakorlatban igen elterjedten használják. A 3DES-t EDE és DED konfigurációban lehet használni, ahol az E betű a kódoló, a D betű pedig a dekódoló transzformációra utal. A 3DES-EDE vázlatát mutatja a 2.8. ábra.



2.8. ábra. A 3DES EDE konfigurációban

A 3DES EDE és DED konstrukcióinak fő motivációja, hogy $K_1 = K_2 = K_3$ esetben az egyszeres DES-t adják vissza, így a 3DES-t támogató eszközök kulcsegyeztetéssel olyan rendszerekkel is tudnak kommunikálni, amelyek csak az egyszeres DES-t támogatják. Ha $K_1 = K_3$, ekkor két kulcsos 3DES-ről beszélünk, melynek effektív kulcshossza tehát 112 bit. Ha a K_1 , K_2 , és K_3 kulcsok mind különbözők, akkor három kulcsos 3DES-ről beszélünk, melynek effektív kulcshossza 168 bit.

2.5. Az AES blokkrejtjelező

A Rijndael szimmetrikus kulcsú blokkrejtjelező algoritmus a DES algoritmust lecserélő új rejtjelező szabvány az USA-ban (Advanced Encryption Standard – AES). Feladata – továbbra is – a bizalmas kereskedelmi információk rejtjelezéssel történő védelme. Számos versengő algoritmus közül a Rijndael lett a győztes, mivel több követelményt kiegyensúlyozottan valósít meg. Ezen követelmények legfontosabbika természetesen a biztonság,

de nagy súllyal számított a hatékony implementálhatóság (sebesség) különböző platformokon, valamint a blokkméretek (üzenet, kulcs) variálhatósága, azaz a flexibilitás. A biztonság kritérium tekintetében számos versenytársa lényegében azonos minőséget nyújtott; a Rijndael előnye az, hogy ezt a biztonságot hatékonyabban tudja megvalósítani. A biztonság vonatkozásában fontos bírálati szempont volt az algoritmus részletekben is világos, áttekinthető struktúrája, s bizonyíthatósága. Az AES tiszta algebrai struktúrája $GF(2^8)$ aritmetikát alkalmaz, bájt a legkisebb kezelt egység. Az üzenet és kulcs bemenetpár és a rejtjeles üzenet kimenet között $GF(2^8)$ feletti nemlineáris egyenletrendszer írható fel.

Az AES iteratív blokkrejtjelező, amelynek rétegfelépítése követi a shannoni helyettesítéses-permutációs elveket, így az iterációs rétegekben (round) megtalálható a bájt elemekkel dolgozó nemlineáris S-boxokból álló és a permutációs alréteg is. Az alrétegek külön-külön invertálhatók.

Az üzenet (rejtett üzenet) és a kulcs N_b , illetve N_k számú 32 bites (4 bájt) szóból áll. Ennek megfelelően az AES blokkméret választéka 128, 192, illetve 256 bit. Az iterációs rétegek számára N_r jelölést használva, $N_r = 10$, ha $N_b = N_k = 4$; $N_r = 12$, ha N_b és N_k 4 és 6 közötti (de nem 4 mindegyikük); egyéb esetekben $N_r = 14$. (Összehasonlításul a DES paraméterei, az adott jelölésben: $N_b = 2$, $N_k \cong 2$ (56 bit), $N_r = 16$.)

Egy iterációs réteg (round) szerkezete az alábbi pseudo C kód szerinti:

```
Round(State, RoundKey) {
    ByteSub(State);
    ShiftRow(State);
    MixColumn(State);
    AddRoundKey(State);
}
```

A ByteSub bájt helyettesítő transzformáció alréteg nyújtja a nemlineari-tást (S-boxok alrétege). Az alréteg bemeneti $4 \cdot N_b$ bájtrája bájtonként alkalmazzuk az $S : GF(2^8) \rightarrow GF(2^8)$ helyettesítő transzformációt:

$$S(x) = Bx^{-1} + b,$$

ahol B egy 8×8 dimenziós invertálható bináris mátrix, b egy 8 bites vektor, x^{-1} az x inverze (bináris koordinátákban). Így az S helyettesítő transzformáció (S-box) a nemlinearitását ezen utóbbi invertálásból nyeri. Az S transzfor-máció egyszerűen invertálható: $S^{-1}(y) = (B^{-1}(y - b))^{-1}$.

A ShiftRow transzformáció-alréteg a permutációt végzi. Bemenetét – gondolatban – rendezzük egy bájt-elemű mátrixba, amely mátrixnak 4 sora és N_b oszlopa van, s ahol az egyes oszlopokba sorban az egyes szavak 4 bájtja kerül. Ezen mátrixot tekintve, az egyes sorokat ciklikusan forgatjuk, soronként különböző módon: az i -edik sorra ($0 \leq i \leq 3$) alkalmazott balról-jobbra forgatásra C_i jelölést alkalmazva:

$$C_0 = 0$$

$$C_1 = 1$$

$$C_2 = \begin{cases} 2 \text{ ha } N_b = 4 \text{ vagy } N_b = 6 \\ 3 \text{ egyébként} \end{cases}$$

$$C_3 = \begin{cases} 3 \text{ ha } N_b = 4 \text{ vagy } N_b = 6 \\ 4 \text{ egyébként.} \end{cases}$$

Ezen transzformációk inverze - nyilván - a forgatás irányának megváltoztatásával adódik.

A MixColumn transzformáció-alréteg a permutációval együtt a diffúziós hatásért felelős. Bemenetét szintén 4 soros bájt-elemű $D = \{d_{i,j}\}$ mátrixként tekintve, a MixColumn transzformáció ezen mátrix oszlopait transzformálja, oszloponként. A j -edik oszlophoz ($0 \leq j \leq N_b - 1$) természetes módon rendelt $D_j(x) = d_{3,j}x^3 + d_{2,j}x^2 + d_{1,j}x + d_{0,j}$ polinomot és egy rögzített $E(x) = e_3x^3 + e_2x^2 + e_1x + e_0$, ($e_3 = 0x03$, $e_2 = 0x01$, $e_1 = 0x01$, $e_0 = 0x02$) polinomot szorzunk össze mod ($x^4 + 1$):

$$D_j(x)E(x) \pmod{x^4 + 1},$$

azaz elvégezzük a szokásos polinomszorzást, majd a szorzat 4, 5, 6 fokszámú tagjait redukáljuk, rendre 0, 1, 2 fokszámúvá. Ez a moduláris szorzás invertálható: $x^4 + 1$ nem irreducibilis polinom $GF(2^8)$ felett, de $E(x)$ úgy lett választva, hogy létezzen az inverze, azaz létezik $E^{-1}(x)$ polinom, amelyre $E^{-1}(x)E(x) = 1 \pmod{x^4 + 1}$.

Az AddRoundKey transzformáció-alréteg végzi a rétegkulcs lineáris bevitelét, a rétegkulcsot bitenként mod 2 hozzáadva a transzformáció bemeneti blokkja megfelelő bitjeihez.

A kulcsütemezési algoritmus (KeyExpansion) bemenete az AES rejtjelező $K = K_0, \dots, K_{N_k-1}$ kulcsa, kimenete a $W = (W_0, \dots, W_{N_b(N_r+1)-1})$ kiterjesztett kulcs, ahol W_i, K_i 32 bites szavak. A kulcsütemezési algoritmus a SubByte

és a RotByte transzformációkat alkalmazza, ahol

$$\begin{aligned} \text{SubByte}(a, b, c, d) &= (S(a), S(b), S(c), S(d)) \\ \text{RotByte}(a, b, c, d) &= (b, c, d, a). \end{aligned}$$

A kulcsütemezési algoritmus két verziója közül az első az $N_k \leq 6$, a második az $N_k > 6$ esetben kerül felhasználásra. Mindegyiknek két fázisa van: az inicializálási (Initialization), és az ezt követő kiterjesztési (Expansion) fázis. Az inicializálás mindenkorban: $W_i = K_i, i = 0, \dots, N_k - 1$.

Az első verzió ($N_k \leq 6$) kiterjesztési fázisa az alábbi:

$$\begin{aligned} \text{tmp} &= W_{i-1} \\ \text{if } i \pmod{N_k}, \text{ then } \text{tmp} &= \text{SubByte}(\text{RotByte}(\text{tmp})) \oplus Rcon_{\lfloor i/N_k \rfloor} \\ W_i &= W_{i-N_k} \oplus \text{tmp}, \end{aligned}$$

ahol $Rcon_i = (RC_i, 0, 0, 0)$, $RC_i = RC_{i-1} = x^{i-1}$ és x egy $GF(2^8)$ -beli elem.
A második verzió ($N_k > 6$) kiterjesztési fázisa az alábbi:

$$\begin{aligned} \text{tmp} &= W_{i-1} \\ \text{if } i \pmod{N_k}, \text{ then } \text{tmp} &= \text{SubByte}(\text{RotByte}(\text{tmp})) \oplus Rcon_{\lfloor i/N_k \rfloor} \\ \text{else if } (i \bmod N_k == 4), \text{ then } \text{tmp} &= \text{SubByte}(\text{tmp}) \\ W_i &= W_{i-N_k} \oplus \text{tmp}. \end{aligned}$$

A kódolandó üzenet és az első iterációs réteg között egy külön AddRoundKey kulcsbeviteli művelet is szerepel.

A Rijndael-rejtjelező szabvánnyal való elfogadása óta, a kriptoanalízis szakértők egyik értékes célpontja, ugyanúgy, ahogy közel harminc évvel ezelőtt a DES rejtjelező volt. Az algoritmus ellenállt a differenciális és lineáris analízis kísérleteknek. Az eddig ismert legjobb támadás a Square attack, amely például 256 bites ($N_b = N_k = 8$) rejtjelező esetén $N_r = 8$ iterációs réteget feltételezve 2^{204} kódolási lépéssel, $2^{128} - 2^{119}$ választott nyílt szöveggel lenne sikeres (sikernek számít az, ha az adott módszer műveletigénye a kiemelő kulcskeresés műveletigénye alatt marad). Összehasonlításul, az adott blokkméret esetén, az AES $N_r = 14$ iterációs réteget használ.

2.6. Feladatok

2.1. Feladat. Tekintsünk egy lineáris bináris blokkrejtjelezőt. Adja meg az ismert nyílt-rejtett párokon alapuló támadást a rejtjelező ellen! Van-e megkötés a párokra?

2.2. Feladat*. Határozza meg a nemlinearitás mértékét az $f : \{0,1\}^2 \rightarrow \{0,1\}$ Boole leképezés esetére: $00 \rightarrow 1, 10 \rightarrow 0, 01 \rightarrow 1, 11 \rightarrow 1!$

2.3. Feladat. Tekintsen egy $f : \{0,1\}^2 \rightarrow \{0,1\}^2$ S-dobozt. Jelölje f_1 az első, illetve f_2 második output bitre vonatkozó transzformáció-komponenst (Boole függvényt). Ha f_1 nemlinearitása N_1 , akkor mekkora f nemlinearitása,

1. ha $f_2(x) = f_1(x) \oplus 1$
2. ha $f_2(x) = 1$.

2.4. Feladat*. Igazolja, hogy ha $f : \{0,1\}^{n-1} \rightarrow \{0,1\}$ tetszőleges Boole-függvény, továbbá $g(x, x_n) = f(x) \oplus x_n$, ahol $x \in \{0,1\}^{n-1}$ és $x_n \in \{0,1\}$, akkor g nemlinearitása f nemlinearitásának kétszerese!

2.5. Feladat*. Adja meg az alábbi 4 bitet 4 bitbe képező transzformáció diffúziós mátrixát:

1. $y_i = x_i \oplus x_{i+1}$
2. $y_i = x_i \odot x_{i+1}$,

ahol $i = 0, 1, 2, 3$, továbbá az indexben mod 4 számolunk!

$((x_i, y_j))$ input-output bitpár közötti $0 \leq w_{ij} \leq 1$ diffúzió azon esetek arányát méri a transzformáció összes különböző x inputját vizsgálva, amikor az x_i input bit megváltozása esetén az y_j output bit is megváltozik.)

2.6. Feladat*. Igazolja, hogy ha egy (x_i, y_j) input-output bitpár között a diffúzió 1 értékű, akkor az $y_i = f(x)$ függvénykapcsolat x_i -ben lineáris!

2.7. Feladat*. Igazolja, hogy lineáris kódolás maximális diffúziót szolgáltat!

2.8. Feladat*. Tekintsük a következő 4 bites helyettesítő transzformációkat: $f : \{0,1\}^4 \rightarrow \{0,1\}^4$. Véletlen választás esetén mi a valószínűsége, hogy invertálható transzformációt kapunk?

2.9. Feladat. Feistel-típusú blokkrejtjelezőt tekintünk, azaz az i -edik réteg szerkezete $L_i = R_{i-1}, R_i = L_{i-1} + f(R_{i-1}, k_i)$, ahol $[L_{i-1}, R_{i-1}]$ az input blokk, $[L_i, R_i]$ az output blokk, továbbá k_i a réteg kulcsa. Tekintsünk egy 2 rétegű

rejtjelezőt, ahol legyen $f(A, k) = A + k \pmod{2}$, továbbá az egyes rétegek kulcsa függetlenül sorsolt.

Támadható-e hatékony algoritmussal a rejtjelező, ismert nyílt szöveg alapú támadással, ahol a támadó a k_1, k_2 kulcsokat szeretné meghatározni? Ha igen, adja meg az algoritmust!

2.10. Feladat. Legyen \bar{m} az m bináris üzenetblokk bitenkénti komplemente. $E_k(m)$ jelölje m DES kódolását k kulcs mellett. Igazoljuk, ha $c = E_k(m)$, akkor $\bar{c} = E_{\bar{k}}(\bar{m})$, azaz egyszerre komplementálva az üzenet, rejtett szöveg és kulcs blokkokat, köztük a rejtjelező kódolási leképezés kapcsolat változatlanul fennáll. Ehhez kapcsolódó további kérdés az, hogy ezen komplementálás tulajdonság mennyiben jelent támadási könnyebbésséget egy kimerítő kulcskereséses támadás esetén, ha:

1. ismert nyílt szövegű támadást,
2. választott szövegű támadást tekintünk?

2.11. Feladat. Tegyük fel, hogy DES rejtjelezést használtunk 64 bites üzenet blokkok rejtjelezésére, amelyek 8 bites karakterekből állnak, s a 8. bit paritás bit.

1. Elvben végrehajtható-e kulcskereséses támadás csak rejtett szövegek megfigyelésére alapozva (azaz várhatóan sikerül-e kiválasztani a keresett kulcsot)? Hány rejtjeles blokkot kellene megfigyelni ehhez?
2. Ha az üzenet blokknak csak az utolsó bitje paritásbit, 50 rejtjeles blokk megfigyelése elegendő-e a feladat sikeres végrehajtásához?

(A DES kódolást és dekódolást véletlen függvényként modellezheti.)

2.12. Feladat. Két DES transzformációt egymás után használunk:

$y = E_{k_2}(E_{k_1}(x))$, ahol k_1, k_2 két véletlen kulcs.

1. Hogyan támadná meg a rejtjelezőt, ha ismert nyílt-rejtett szövegpárok állnak rendelkezésre? Becsülje meg a támadás komplexitását, számításigény és tárkapacitás vonatkozásában, ahol számításigényt a kódoló/dekódoló transzformációk számában, a tárkapacitást a tárolandó blokkok számában mérjük!
2. Hasonlítsa az eredményt a naív kimerítő kereséses támadás erőforrásigényéhez.

3. Az idő-memória erőforrás mérleget hogyan lehetne módosítani, ha kevesebb memóriát kívánunk felhasználni több számításigény rovására?
4. A középen találkozás támadás kétszeres DES kódolás elleni erőforrásigényét szeretné valaki megnövelni olyan módon, hogy a következő háromszoros kódolást alkalmazza: $y = E_{k_2}(E_{k_1}(E_{k_1}(x)))$ Sikerül-e a célját elérni? Elemezze a támadás erőforrásigényét!

2.13. Feladat. Vizsgálja meg a kétkulcsos háromszoros DES kódolás (EDE) középen találkozásos támadás erőforrásigényét választott nyílt szövegű támadás mellett!

3.

Nyilvános (aszimmetrikus) kulcsú rejtjelezők

A nyilvános kulcsú kriptográfia alapja, hogy a kódoló és a dekódoló transzformáció, illetve az azokat paraméterező nyilvános és titkos kulcsok nem azonosak, sőt egyiket a másikból kiszámítani nagy komplexitású feladat, azaz praktikusan lehetetlen. Ezért a két kulcs közül az egyiket, általában a kódoló kulcsot, nyilvánosságra lehet hozni. Ez azért hasznos, mert ezáltal leegyszerűsödik a kulccscsere probléma, ugyanis a titkos kommunikáció megvalósításához nincsen szükség egy titkos (szimmetrikus) kapcsolatkulcs létrehozására a küldő és a vevő között; elegendő a rejteni kívánt üzenetet a vevő nyilvános kódoló kulcsával rejtjelezni. Megjegyezzük azonban, hogy gondoskodni kell a nyilvános kulcsok hitelességéről. Más szavakkal, a küldőnek meg kell tudnia győződni arról, hogy a használni kívánt nyilvános kulcs valóban a vevő nyilvános kulcsa, és nem egy harmadik, feltehetően rossz-szándékú fél. A nyilvános kulcsú kriptográfia tehát oly módon egyszerűsíti a kulccscsere problémát, hogy titkos csatorna helyett hiteles csatorna létezését követeli meg a vevő és a küldő között, melyen a vevő eljuttathatja nyilvános kulcsát a küldőnek.

Az alábbiakban ismertetjük az egyik leggyakrabban használt nyilvános kulcsú rejtjelező módszert, az RSA algoritmust, és áttekintjük annak ismert gyengeségeit. Mint látni fogjuk, az RSA kulcsok generálásához nagy prímszámokra lesz szükségünk. Ezért röviden összefoglaljuk a prímtesztelés módszereit is.

Az utolsó szakaszban a nyilvános kulcsú kriptográfia egy viszonylag új ágával, az elliptikus görbék algebrájára épülő kriptografiával foglalkozunk. Az elliptikus görbe kriptográfia gyakorlati jelentősége, hogy a kulcsok ál-

talában tömörebb módon reprezentálhatóak, mint a hagyományos nyilvános kulcsú algoritmusok (pl. az RSA) esetében, és így módon az elliptikus görbékre épülő algoritmusok kis tároló kapacitással rendelkező eszközökön (pl. smart kártyákon) is implementálhatóak.

3.1. Az RSA algoritmus

Az RSA algoritmus három algoritmusból, a kulcsválasztás, a kódolás és a dekódolás algoritmusából áll. A kulcsválasztást a rendszer minden felhasználója elvégzi:

1. Véletlenszerűen választunk két nagy, p és q prímszámot ($p \neq q$). Jelenleg „nagynak” a legalább 500 bit bináris méretű számokat tekintik.
2. Kiszámítjuk az $N = pq$ modulust és a $\varphi(N) = (p-1)(q-1)$ szorzatot. Választunk továbbá egy e számot, amelyik mind $(p-1)$ -hez, mind $(q-1)$ -hez, tehát $\varphi(N)$ -hez relatív prím.
3. Kiszámítjuk az e inverzét modulo $\varphi(N)$, azaz keresünk egy d számot, amelyre $1 < d < \varphi(N)$ és $ed = 1 \pmod{\varphi(N)}$.

A kulcsválasztás a 3. lépéssel be is fejeződik. A rendszerszerű üzemeltetés során azonban egy központi, minden felhasználó által hozzáférhető nyilvántartásra van szükség. Ennek érdekében a kulcs nyilvános részét, esetünkben az (N, e) számpárt nyilvánosságra hozzuk, a kulcs titkos részét, a (d, p, q) számhármast s ezzel együtt a $\varphi(N)$ számot titokban tartjuk.

Ha A felhasználó B felhasználónak akar üzenni, akkor A felhasználó kikeresi a B nyilvános kulcsát, az $e = e_B$ és $N = N_B$ számokat. A felhasználó előkódolja az üzenetet. A nyílt szöveg valamelyen karakterkészlet elemeiből áll. A rejtelezéshez ezt a karaktersorozatot olyan nemnegatív egészek sorozatává kell átalakítani, amelyek mindegyike kisebb, mint N_B . Ez egy kölcsönösen egyértelmű transzformáció, amelyet minden felhasználó ismer. A rejtelezést az előkódolt üzenet egymás utáni számain A felhasználó egyenként hajtja végre. Ha az előkódolt szöveg következő száma x , akkor a megfelelő rejtejes szám:

$$y = E_B(x) = x^{e_B} \pmod{N_B}.$$

A B felhasználó megkap egy rejtejes üzenetet. Ez az üzenet szükségszerűen a 0 és $N_B - 1$ közti egész számok egy y_1, y_2, \dots sorozata. A dekódolást a számsorozat elemein külön-külön hajtja végre. Legyen a rejtejes szöveg a következő szám: y . Ekkor az „előkódolt” üzenet megfelelő száma

$$x = D_B(y) = y^{d_B} \pmod{N_B}.$$

A visszaállított, „előkódolt” x_1, x_2, \dots sorozatból az előkódolás inverzét alkalmazva adódik a valódi nyílt szöveg.

Lássuk be az RSA dekódolási algoritmus helyességét.

3.1. Tétel. *Bármely x egész számra fennáll, hogy*

$$(x^e)^d = x \pmod{N}.$$

Bizonyítás: Mivel d az $e \bmod \varphi(N)$ inverze, ezért valamely v egész számra $ed = v\varphi(N) + 1$, s így a tétel bizonyításához az

$$x = x^{v\varphi(N)+1} \pmod{N} \quad (3.1)$$

modulo egyenlőséget kell belátni. Először tetszőleges u prímre igazoljuk, hogy tetszőleges x és s egész esetén

$$x = x^{s(u-1)+1} \pmod{u} \quad (3.2)$$

fennáll: ha u prím nem osztója x -nek, akkor a Fermat-tétel szerint

$$(x^{u-1})^s = 1^s = 1 \pmod{u},$$

tehát (3.2) fennáll, ha pedig u osztója x -nek, akkor nyilván

$$x = 0 = x^{s(u-1)+1} \pmod{u}.$$

Visszatérve az általános esetre, mivel $p - 1 \mid \varphi(N)$ és $v - 1 \mid \varphi(N)$, ezért (3.2) felhasználásával

$$\begin{aligned} x &= x^{v\varphi(N)+1} \pmod{p} \\ x &= x^{v\varphi(N)+1} \pmod{q} \end{aligned}$$

fennállnak. Innen

$$p \mid x^{v\varphi(N)+1} - x$$

és

$$q \mid x^{v\varphi(N)+1} - x$$

teljesülnek, amelyből $N \mid x^{v\varphi(N)+1} - x$ is következik. \square

A kulcsválasztásnak elfogadható időn belül kivitelezhetőnek kell lennie. Ezért meg kell vizsgálni, hogy milyen nagy sebességgel tudunk „nagy” prímszámokat generálni. Prímelőállítással a fejezet további részében még foglalkozunk. Továbbá könnyű eldönteni, hogy $\varphi(N)$ és az e kitevőjelölt relatív

prímek-e, amely vizsgálatához az euklideszi algoritmus használható. Az e szám inverzét is gyorsan meg tudjuk határozni. Erre a célra szintén az euklideszi algoritmus használható. Az algoritmus legfeljebb kétszer annyi lépést igényel, mint a modulus logaritmusá.

A kódolásnak és a dekódoló kulcs ismeretében a dekódolásnak is gyorsnak kell lennie. Ez azt jelenti, hogy a modulo aritmetikában gyorsan kell tudni hatványozni. Az ismételt négyzetreemelés és szorzás módszerét alkalmazva az e -edik hatvány kiszámítása legfeljebb $2\log_2(e)$ (illetve $2\log_2(d)$) számú modulo N szorzással megoldható.

Az alábbiakban egy számpéldával szemléltetjük a moduláris aritmetika alkalmazását az RSA algoritmus esetén.

3.1. Példa. Legyen $p = 73$, $q = 151$, így $N = 73 \cdot 151 = 11023$, $\varphi(N) = (73 - 1)(151 - 1) = 10800$. Az e paramétert $e = 11$ -re választhatjuk, mivel $(10800, 11) = 1$, hiszen $10800 = 2^4 \cdot 3 \cdot 5^2 \cdot 9$. Az e inverzét mod $\varphi(N)$ az $e \cdot s + \varphi(N) \cdot t = 1$ előállításból kaphatjuk ($d = s (\text{mod } \varphi(N))$), amelyhez az euklideszi algoritmussal juthatunk:

$$\begin{aligned} 10800 &= 11 \cdot 981 + 9 \\ 11 &= 9 \cdot 1 + 2 \\ 9 &= 2 \cdot 4 + 1 \\ 2 &= 1 \cdot 2 + 0, \end{aligned}$$

ahonnan

$$\begin{aligned} 9 &= 10800 - 981 \cdot 11 \\ 2 &= 11 - 9 = 11 - (10800 - 981 \cdot 11) = -10800 + 982 \cdot 11 \\ 1 &= 9 - 2 \cdot 4 = 10800 - 981 \cdot 11 - 4(-10800 + 982 \cdot 11) = \\ &= 5 \cdot 10800 - 4909 \cdot 11, \end{aligned}$$

így

$$-4909 \cdot 11 = 1 \pmod{10800},$$

ezért

$$d = 10800 - 4909 = 5891 \pmod{10800}.$$

Nyilvánosságra hozzuk az $e = 11$, $N = 11023$ egészeket, s titokban tartjuk a $p = 73$, $q = 151$, $d = 5891$ egészeket.

Tegyük fel, hogy az $x = 17$ nyílt üzenetet kívánjuk kódolni, ekkor

$$y = 17^{11} \pmod{11023},$$

ahonnan $y = 1782$. A dekódolás az

$$x = 1782^{5891} \pmod{11023} \quad (3.3)$$

művelettel történik. Ezen modulo hatvány kiszámítását egyszerűsíti a „négyzetreemelés és szorzás” módszere. A kitevőt az

$$5891 = 2^0 + 2^1 + 2^8 + 2^9 + 2^{10} + 2^{12}$$

összegre bonthatjuk a bináris ábrázolása alapján. Ennek alapján a (3.3) hatványt az alábbi alakba célszerű átírni:

$$\begin{aligned} 1782^{2^{12}} \cdot 1782^{2^{10}} \cdot 1782^{2^9} \cdot 1782^{2^8} \cdot 1782^{2^1} \cdot 1782 \\ = ((\dots((1782)^2)^2 \cdot 1782)^2 \cdot 1782)^2 \cdot 1782)^2 \cdot 1782)^2 \cdot 1782 \cdot 1782. \end{aligned}$$

A kiértékelést a legbelső moduláris négyzetreemeléssel kezdjük, azaz az első néhány lépés:

$$1782^2 = 900 \pmod{11023}$$

$$900^2 = 5321 \pmod{11023}$$

$$5321 \cdot 1782 = 2242^2 = 76 \pmod{11023},$$

...

amely lépéseket követve, végül a 17-et kapjuk vissza. Így tehát ahelyett, hogy (3.1) mechanikus kiszámítását, azaz a

$$((\dots(17891789)1789)1789)\dots)1789$$

szorzáshoz szükséges 5890 darab modulo szorzást végeztük volna el, megúsztuk 17 darab modulo szorzással. Könnyen ellenőrizhető, hogy ezzel a módszerrel legfeljebb a kitevő kettes alapú logaritmusa kétszerese számú modulo szorzásra van szükség.



3.2. Az RSA biztonsága

Ha létezne hatékony algoritmus az egész számok faktorizációjára, akkor ennek birtokában az RSA rejtjelezés hatékonyan fejthető lenne: az N modulus p és q faktorja felhasználásával kiszámolnánk a dekódoló kulcsot. Nem tudjuk azonban biztosan, hogy az RSA törés feladata valóban redukálható a

faktorizációs feladatra. Ha nem lenne redukálható, az azt jelentené, hogy az RSA hatékonyan törhető lehetne akkor is, ha valóban nehéz feladat az egész számok faktorizációja. Megjegyezzük, hogy az $e = 2$ kitevő esete (azaz a négyzetreemelés mod N) speciális, mivel ebben az esetben bebizonyítható, hogy akkor és csak akkor létezik hatékony inverziós algoritmus (azaz mod N négyzetgyökvonó algoritmus), ha létezik hatékony algoritmus az egész szám faktorizáció feladatra. Vegyük észre, hogy az $e = 2$ kitevő esete nem RSA kódolás. A kérdéskörre és az említett bizonyításra, továbbá az RSA leképezésre, mint csapda egyirányú függvényre a bizonyított biztonság részben részletesen visszatérünk.

Az RSA implementációja és protokollokbeli felhasználása során sok részletén műlhet a biztonság. Számos hatékony támadás ismert az üzenethalmaz, a prímpár, a kódoló és dekódoló kulcs speciális választása, valamint nem biztonságos protokollok kapcsán. Mindezt az alábbiakban három problémával illusztráljuk.

3.2.1. Fermat-faktorizáció: p és q közeli prímek

Nem elégsges, hogy két, különböző nagy prímszámot választottunk véletlenszerűen, ugyanis sikeres támadásra van lehetőségünk, ha a p és q prímek távolsága „nem elég nagy”. Fermattól származó egyszerű faktorizációs lehetőség adódik az N modulusra. Tekintsünk egy n pozitív összetett egész számot, ahol $n = a \cdot b$, $a > b$. Legyen $t = (a + b)/2$ és $s = (a - b)/2$, azaz $a = t + s$, $b = t - s$. Innen $n = t^2 - s^2 = (t + s)(t - s)$ adódik. Ha $a - b$ különbség „kicsi”, akkor s is „kicsi”, azaz $t \approx n^{1/2}$. Találgassuk t értékét a következőképpen: számítsuk ki az

$$t_1 = \lfloor n^{1/2} \rfloor + 1, \quad t_2 = \lfloor n^{1/2} \rfloor + 2, \dots$$

értékeket, s ellenőrizzük hogy $t_i^2 - n$ különbség négyzetszámot ad-e. Ha igen, akkor a különbség egyenlő s^2 -tel. A megkapott t és s alapján már könnyen kiszámítható a és b .

Például faktorizáljuk $n = 200819$ -et. n lefelé kerekített négyzetgyöke 448. $t_1 = 449$, $449^2 - 200819 = 782$ nem négyzetszám, $t_2 = 450$, $450^2 - 200819 = 1681 = 41^2$ azonban már négyzetszám, ahonnan $200819 = 450^2 - 41^2 = (450 + 41)(450 - 41) = 491 \cdot 409$.

3.2.2. Kicsi kódoló kulcsok problémája

Tegyük fel, hogy egy központ r távoli ügynöknek azonos titkos üzenetet küld RSA rejtjelezéssel, ahol az ügynökök nyilvános RSA kulcsát azonosra választották, azon az alapon, hogy az RSA rejtjelezés biztonsága nem múlik ezen kulcs választásán. A csatornákat lehallgató támadó az alábbi rejtjelű üzeneteket szerzi meg:

$$\begin{aligned} y_1 &= x^e \pmod{m_1} \\ y_2 &= x^e \pmod{m_2} \\ &\dots \\ y_r &= x^e \pmod{m_r}, \end{aligned} \tag{3.4}$$

ahol $r \geq e$. Feltehetjük, hogy az m_1, m_2, \dots, m_r modulusok páronként relatív prímek, ugyanis ügynökönként véletlenszerűen választottuk a prímpárt, így ezen feltevés teljesülése gyakorlatilag biztos. A támadó a kínai maradékok tételeit alkalmazhatja a (3.4) kongruencia-rendszer megoldására. Ezzel ugyanis hatékonyan és egyértelműen kiszámíthatja $z = x^e \pmod{(m_1 \cdot m_2 \dots m_r)}$ maradékot, s vegyük észre, hogy $x < \min_i \{m_i\}$ miatt $0 < x^e < m_1 \cdot m_2 \dots m_r$, így z maradék e -edik gyökét kiszámítva magát a keresett x nyílt szöveget megkapja.

3.2.3. Közös modulus problémája

Tudjuk, hogy az RSA kódolás igen számításigényes, ezért arra a gondolatra juthatunk, hogy célhardverben legyártatjuk egy választott N modulus (egy adott p, q prímpár) esetére a $\text{mod } N$ hatványozó aritmetikát. Mivel azonban többfelhasználós a rendszer, a központ úgy különbözteti meg az egyes felhasználókat, hogy az adott N modulus mellett különböző nyilvános kulcsokat választ számukra, amelyekből különböző dekódoló kulcsok adódnak. A központ ezek után egy x titkos üzenetet küld A és B felhasználónak. A támadó lehallgatja a csatornákat. Ezzel a támadó megismeri az $y_A = x^{e_A} \pmod{N}$ és $y_B = x^{e_B} \pmod{N}$ rejtett szövegeket. Tegyük fel, hogy $(e_A, e_B) = 1$. Így létezik t és s egész, hogy $t \cdot e_A + s \cdot e_B = 1$, ahol $t \cdot s < 0$, mivel $e_A > 0$, $e_B > 0$. Az általánosság korlátozása nélkül tegyük fel, hogy $t < 0$, azaz $t = -1 \cdot |t|$. Továbbá feltehető, hogy $(y_A, N) = 1$, ezért létezik az $y_A^{-1} \pmod{N}$ inverz. Innen $(y_A^{-1})^{|t|} \cdot (y_B)^s = (x^{e_A})^t \cdot (x^{e_B})^s = x^{te_A + se_B} = x \pmod{N}$ alapján a támadó, kizártlag nyilvános információk és a lehallgatott rejtett szövegpár felhasználásával, dekódolni képes a titkos üzenetet.

3.3. Prímszámok keresése

A kiinduló probléma az, hogy eldöntsük egy adott nagy egész számra, hogy prím-e (PRÍM döntési probléma). Hatékony megoldó algoritmus utáni kutatás az ókortól fogva izgalomban tartja a számelmélezeket. A legfrissebb eredmény szerint a probléma eldönthető az input méretében polinomiális futási idejű, determinisztikus algoritmussal is. Az ezt megelőző 25 év során is sikerült polinomiális idejű algoritmusokat találni, ezek azonban véletlen biteket is igényelnek számításaikhoz. (A bizonyított biztonság részben adunk egy rövid bevezetést az algoritmusok komplexitáselméletébe, az ottani felhasználás mértékéig. A nevezett legfrissebb eredmény azt jelenti, hogy $PRIM \in P$, míg megelőzően azt tudtuk, hogy $PRIM \in co-RP$). A legfontosabb, gyakorlatban alkalmazott algoritmus továbbra is az egyik randomizált polinomiális idejű algoritmus, a Miller–Rabin prímtesztelő algoritmus.

A legegyszerűbb prímteszt során megvizsgáljuk, hogy a tesztelt szám négyzetgyökénél kisebb számok között van-e valódi osztó. Az algoritmus egyszerű, determinisztikus, de nem hatékony, mivel a számításigény bemenet méretében exponenciális.

Az RSA kapcsán szeretnénk egy hatékony (azaz polinomiális idejű) algoritmust annak eldöntésére, hogy véletlenül kisorsolt egész szám prím-e vagy sem. Megnyugtató, hogy a prímek elég sűrűn találhatók az egész számok között, s így, ha véletlenszerűen választunk a páratlan egész számok közül, az alkalmazások szempontjából elegendően nagy valószínűsséggel prímszámot választunk. Ezek után már csak az a kérdés, hogy hogyan vesszük észre, hogy prímet sorsoltunk ki. Ezen utóbbi feladatra szolgálnak a prímtesztelő algoritmusok. A prímek sűrűségével kapcsolatban bizonyítás nélkül utalunk a számelmélet ismert tételeire, amely kimondja, hogy $\Pi(n)$, azaz az n pozitív egésznél kisebb prímek számának nagyságrendje:

$$\Pi(n) \cong \frac{n}{\ln n}. \quad (3.5)$$

Számpéldaként tekintsünk egy $N = pq = 10^{200}$ nagyságrendű modulust, azaz $10^{100} (= 2^{330})$ nagyságrendű prímeket választunk. Annak a valószínűsége, hogy egy véletlenszerűen választott $2^{331} < s < 2^{332}$ (azaz 332 bites) szám prím, (3.5) alapján becsülhető:

$$\frac{\Pi(2^{332}) - \Pi(2^{331})}{2^{332} - 2^{331}} \cong \frac{2^{331} \frac{1}{331 \cdot \ln 2}}{2^{331}} \cong \frac{1}{230}.$$

Továbbá, mivel az adott számtartomány fele páros szám, amelyek nem prímek, elég csak a páratlanok közül válogatnunk, s ezzel $1/115$ -re duplázzuk meg a találati valószínűséget. Így tehát átlagosan 115 próbálkozásunkonként jutunk prímszámhöz a 10^{100} nagyságrendű számok között.

Véletlenül sorsolt számok közül a prímek valószínűségi alapon történő kiszűrésére több prímtesztelő algoritmus is létezik. Az alapelveket a Fermat prímtesztelő algoritmus kapcsán mutatjuk be.

3.3.1. A Fermat-prímteszt

3.2. Definíció. Egy n összetett szám (Fermat-)álprím egy b bázisra nézve, ha

$$b^{n-1} = 1 \pmod{n}, \quad (3.6)$$

ahol $b \in W_n$, $W_n = \{z : 1 < z < n \text{ és } (z, n) = 1\}$.

A 3.2. definíció a „kis” Fermat-tételen alapul, amely szerint, ha n prímszám, akkor (3.6) teljesül. Ha n összetett szám, de mégis teljesíti a (3.6) egyenletet, akkor álprím.

A prímteszt tehát a következő: ha egy véletlen sorsolt n egész szám egy b bázissal nem teljesíti a (3.6) egyenletet, akkor biztosan nem prím, míg ha teljesíti, akkor prímgyanús, és továbbvizsgálandó egy következő bázissal. Ha a tesztelt szám M egymás utáni vizsgálat során prímgyanúsnak bizonyult, akkor nagy valószínűséggel prímszám.

Kérdés: van-e olyan összetett szám, amely tetszőleges szóbajövő bázisra „átmegy” a Fermat-próbán? Sajnos van, s ezen számokat Carmichael-számoknak nevezük. Bizonyosan a legkisebb ilyen szám az 561.

Felmerül továbbá a kérdés, hogy mekkora a valószínűsége annak az eseménynek, hogy egy n összetett szám, amely nem Carmichael-szám, egy véletlenszerűen választott b , $b \in W_n$ bázissal kielégíti a (3.6) egyenletet.

3.3. Tétel. Ha egy n összetett szám nem Carmichael-szám, akkor legfeljebb $n/2$ különböző b , $b \in W_n$ bázissal elégíti ki a (3.6) egyenletet.

A 3.2. definíció alapján közvetlenül ellenőrizhető, ha n álprím b bázisra, akkor álprím $b^{-1} \pmod{n}$ bázisra is. Továbbá, ha n álprím b_1 és b_2 bázisokra, akkor álprím ezek modulo n szorzatára azaz $b_1 \cdot b_2 \pmod{n}$ bázisra is.

Legyen $B = \{b_1, b_2, \dots, b_s\}$ azon bázisok halmaza, amelyekre n álprím. B halmaz nem üres, mivel n nem Carmichael-szám. Legyen $b' \in W_n \setminus B$. Könnyen látható, hogy

$$b'B = \{b'b_1, b'b_2, \dots, b'b_s\} \pmod{n}$$

halmaz elemeire, mint bázisokra n nem álprím. Ha ugyanis $b'b_i \pmod{n}$ bázisra álprím lenne, akkor $(b'b_i)b_i^{-1} \pmod{n} = b'$ bázisra is annak kelene lennie, ami ellentmondás. Következésképpen azon bázisok száma, melyekre n nem álprím, legalább $n/2$. \square

3.3. tétel következménye: Ha egy n összetett szám nem Carmichael-szám, továbbá $b, b \in W_n$ egy véletlenszerűen választott bázis, akkor annak valószínűsége, hogy n „átmegy” a Fermat-próbán M alkalommal, kisebb, mint 2^{-M} .

3.3.2. Miller–Rabin-prímteszt

A teszt leírásához néhány jelölést vezetünk be. Adott $n > 1$ páratlan egész számhoz legyenek $u = u(n)$ és $v = v(n)$ azon egészek, melyekre u páratlan és

$$n - 1 = 2^v u.$$

Legyen R_n azon $b \in W_n$ számok halmaza, amelyekre vagy

$$b^u = 1 \pmod{n}, \quad (3.7)$$

vagy létezik olyan $0 \leq j < v$, melyre

$$b^{2^j u} = -1 \pmod{n}. \quad (3.8)$$

Mivel tetszőleges x egészre

$$\begin{aligned} (x^{n-1} - 1) &= (x^{(n-1)/2} + 1)(x^{(n-1)/2} - 1) \\ &= (x^{(n-1)/2} + 1)[(x^{(n-1)/4} + 1)(x^{(n-1)/4} - 1)] \\ &\quad \dots \\ &= (x^{(n-1)/2} + 1)(x^{(n-1)/4} + 1) \dots (x^{(n-1)/2^v} + 1)(x^{(n-1)/2^v} - 1) \end{aligned}$$

azonosság fennáll, ezért ha n prím, akkor $n | (b^{n-1} - 1)$ fennáll, tetszőleges $b, b \in W_n$ bázisra, ezért vagy (3.7) vagy (3.8) fennáll, azaz ekkor $R_n = W_n$. Következésképpen n biztosan összetett, ha valamely b bázisra $b \notin R_n$.

Ennek alapján a Miller–Rabin-teszt a következő: Véletlenszerűen választunk egy b bázist. A próba eredménye: n prímgyanús, ha $b \in R_n$, illetve n összetett, ha $b \notin R_n$. Bizonyított, hogy annak a valószínűsége, hogy véletlenszerűen választott bázis mellett egy összetett szám átmegy a Miller–Rabin-teszten, legfeljebb $1/4$. Ha az M darab bázist egymástól függetlenül választjuk meg, akkor a téves prímgenerálás valószínűsége 2^{-2M} .

Természetesen merül fel a kérdés: Miért kell véletlenszerűen megválogatni a bázisokat? Nem létezik-e univerzálisan jó determinisztikus alapszámhalmaz. A kérdés megválaszolása még várat magára, bár jelentek meg érdekes eredmények. Így például, ha a négy számból álló $\{2, 5, 7, 13\}$ bázishalmazt használjuk, akkor a Miller–Rabin-próbán egyetlen összetett szám sem megy át $n < 25109$ intervallumban, míg a $\{2, 3, 5, 7\}$ bázishalmaz ebben az intervallumban bázisonként legfeljebb egyet enged meg.

3.4. Elliptikus görbe kriptográfia

Az elliptikus (és hiperelliptikus) görbék algebrai elméletének nyilvános kulcsú kriptográfiai alkalmazása viszonylag rövid múltra tekint vissza. Ugyanakkor, a rendkívül aktív kutatómunka gyors fejlődést eredményezett. A fő ok az, hogy ezen görbék rendkívül sok véges Abel-csoport konstruálására alkalmasak, amelyekre biztonságos kriptográfiai primitívek építhetők. Ezen csoportok analógok a véges test multiplikatív csoportjával. Ugyanakkor sok változatban előfordulnak, továbbá belőlük jóval kisebb kulcsméretek mellett azonos biztonságú nyilvános kulcsú algoritmusok konstruálhatók. A jelen fejezet csak bevezetőt kíván adni a téma körbe.

3.4. Definíció (elliptikus görbe). *Egy F test feletti E elliptikus görbét az*

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6, \quad a_i \in F \quad (3.9)$$

egyenlet definiálja.

Az $E(F)$ görbét azon $(x, y) \in F^2$ pontok alkotják, amelyek kielégítik a (3.9) egyenletet, továbbá a görbéhez tartozik egy absztrakt pont, a „végtelesen” fekvő O pont. Ezenkívül még egy úgynévezett simasági feltételnek is eleget kell tennie a (3.9) definíció szerinti görbénél.

Jelölje \bar{F} az F algebrai lezárást (\bar{F} az F azon legszűkebb bővített teste, amely felett tetszőleges F -feletti polinom lineáris faktorokra bontható). A simasági feltétel szerint nem lehet az $E(\bar{F})$ görbénél olyan $(x, y) \in E(\bar{F})$ pontja, amely minden parciális deriváltját kielégíti, azaz amely kielégíti az

$$\begin{aligned} a_1Y &= 3X^2 + 2a_2X + a_4 \\ 2Y + a_1X + a_3 &= 0 \end{aligned} \quad (3.10)$$

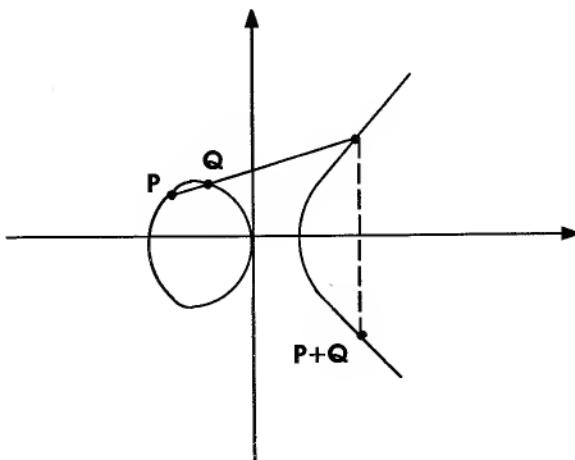
egyenleteket. Az F test karakteristikája szerint is csoportosítják a görbéket: 2, 3 karakteristikájú, illetve a nem 2, 3 karakteristikájú testek feletti

görbékre. Az utóbbi esetben $X \rightarrow X - \frac{1}{3}a_2$ változócserevel (3.9) alábbi egyszerűbb alakjára juthatunk:

$$Y^2 = X^3 + aX + b \quad (3.11)$$

ahol $a, b \in F, \text{char}(F) \neq 2, 3$. Ez esetben a simasági feltétel pontosan akkor teljesül, ha a (3.11) jobboldali polinomjának csak egyszeres gyökei vannak, ami viszont teljesül, ha a $-(4a^3 + 27b^2)$ determináns nem zérus.

Az elliptikus görbe pontjai megfelelően definiált művelettel Abel-csoportot képeznek. A valós test feletti esetre mutatjuk be a műveletképzést, analóg módon definiálhatók a műveleti szabályok véges testek esetére is. Tekintsünk valós számtest feletti E elliptikus görbét. Legyen P és Q két pont az E görbén (3.1. ábra).



3.1. ábra. Elliptikus görbe pontjainak összeadása

1. Ha $P = O$, akkor $-P = O$. Tetszőleges Q pont esetén $O + Q = Q$. Azaz O a csoport zérus eleme. Az alábbiakban tegyük fel, hogy $P \neq O, Q \neq O$.
2. Ha $P = (x, y)$, akkor $-P = (x, -y)$: a (3.11) formula alapján látható, hogy ha P a görbe pontja, akkor $-P$ is rajta van a görbén.
3. Ha $Q = -P$ (azaz egymás x tengelyre vett tükröképe), akkor definíció szerűen $-P + Q = O$.
4. Ha P és Q pontok x koordinátája különböző, akkor a két ponton áthaladó $l = \overline{PQ}$ egyenes egy, ezen két ponttól különböző R pontban metszi az E

görbét, kivéve, ha az egyenes P pontban érinti a görbét, mert ez esetben R definíciója $R = P$. Ezek után P és Q összegének definíciója $P + Q = -R$, azaz R pont x tengelyre vett tükröképe (3.1. ábra). Ha $P = Q$, akkor az l egyenes a görbét a P pontban érinti. Legyen R az a pont, ahol az l egyenes metszi a görbét, amivel $P + P = 2P = -R$.

Mindebből az is következik, hogy azon három pont összege, amelyben egy egyenes metszi a görbét, zérus: ha az egyenes nem megy át az O ponton, akkor $(P + Q) + R = -R + R = O$, ha pedig átmegy az O ponton, akkor $(P + Q) + O = (P + (-P)) + O = O + O = O$.

Az alábbiakban az (x, y) koordinátás alakban megadott pontokra mutatjuk meg a fentiekben definiált összeadás művelet eredményeként kapható pont koordinátáinak számítását. Legyen $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $P + Q = (x_3, y_3)$. Legyen az l egyenes egyenlete $y = \alpha \cdot x + \beta$, ahol $\alpha = (y_2 - y_1)/(x_2 - x_1)$, és $\beta = y_1 - \alpha \cdot x_1$. Egy $(x, \alpha \cdot x + \beta) \in l$ pont pontosan akkor fekszik a görbén, ha kielégíti a (3.11) egyenletet, azaz ha

$$(\alpha \cdot x + \beta)^2 = x^3 + a \cdot x + b. \quad (3.12)$$

Ezen utóbbi egyenletet kielégítő gyökök között van x_1 és x_2 , mivel $P = (x_1, \alpha \cdot x_1 + \beta)$, $Q = (x_2, \alpha \cdot x_2 + \beta)$. A harmadfokú (3.12) egyenlet átrendezésével látható, hogy másodfokú tagjának együtthatója α^2 . Ugyanakkor azt is tudjuk, hogy egy n -edfokú főpolinom gyökeinek összege az $(n-1)$ -edfokú tagja együtthatójának (-1) -szerese. Így a harmadik gyök első koordinátájára $x_3 = \alpha^2 - x_1 - x_2$ adódik. Mivel $P + Q = (x_3, -(\alpha \cdot x_3 + \beta))$, ezért

$$\begin{aligned} x_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2, \\ y_3 &= -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1} \right) (x_1 - x_3). \end{aligned} \quad (3.13)$$

A $P = Q$ esetben a számítás hasonló. Ez esetben α a dy/dx derivált a P pontban. Implicit függvény deriválási szabályát alkalmazva $\alpha = (3x_1^2 + a)/2y_1$. Ezzel $2P = (x_3, y_3)$ pont koordinátáira a következő formulákat kapjuk:

$$\begin{aligned} x_3 &= \left(\frac{3x_1 + a}{2y_1} \right)^2 - 2x_1, \\ y_3 &= -y_1 + \left(\frac{3x_1 + a}{2y_1} \right) (x_1 - x_3). \end{aligned} \quad (3.14)$$

A (3.9–3.12) szabályok közvetlen következményeként megmutatható a kommutatív csoport axiómáinak mindegyikének teljesülése, kivéve az asszociativitást, amelynek igazolásához további algebrai eredmények ismerete szükséges.

3.2. Példa. Tekintsük az $Y^2 - 2Y = X^3 - X^2$ valós együtthatós görbét és rajta a $P = (0, 0)$ pontot. Határozzuk meg a $2P$ pont koordinátáit!

$Y \rightarrow Y - 1$, $X \rightarrow X + 1/3$ változócserevel a görbét a (3.11) alakra hozhatjuk: $Y^2 = X^3 + \frac{1}{3}X + \frac{1}{27}$, amely görbén P képe $Q = (1, -1/3)$. A (3.14) formulák alkalmazásával $2Q = (23, -550 + 1/3)$. ♣

További példaként tekintsük a $GF(p)$, p prím, véges test feletti (3.11) alakú elliptikus görbékét. A (3.13) és (3.14) formulák – mod p műveletekkel – érvényben maradnak.

3.3. Példa. Tekintsük az $Y^2 = X^3 + 3X + 1$ elliptikus görbét $GF(11)$ felett. Azon $P = (x, y)$ pontok vannak a görbén, amelyekre $x^3 + 3x + 1 \pmod{11}$ kvadratikus maradék. Végigvizsgálva a szóbajövő $x = 0, 1, \dots, 10$ értékeket a 3.1. táblázat foglalja össze a görbe pontjainak koordinátáit.

A görbénél tehát 18 pontja van. A Hasse-tétel szerint a görbe pontjainak $\#E$ darabszámára

$$P + 1 - 2\sqrt{p} \leq \#E \leq p + 1 + 2\sqrt{p}$$

korlátok érvényesek, ami a példabeli $p = 11$ esetén a $6 \leq \#E \leq 18$ korlátokat jelenti. Tekintsük a görbe $P = (1, 4)$ pontját, ahonnan $2P = (x, y) = (1, 4) +$

x	$x^3 + 3x + 1 \pmod{11}$	y
0	1	1, 10
1	5	4, 7
2	4	2, 9
3	4	2, 9
4	0	0
5	9	8, 3
6	4	2, 9
7	2	—
8	9	8, 3
9	9	8, 3
10	8	—

3.1. táblázat. Az $Y^2 = X^3 + 3X + 1$, $GF(11)$ elliptikus görbe pontjai

(1.4) a (3.14) formula alkalmazásával:

$$\alpha = (3 \cdot 1^2 + 3)(2 \cdot 4)^{-1} = 6 \cdot 7 = 9 \pmod{11},$$

ahonnan

$$\begin{aligned}x_3 &= 9^2 - 1 - 1 = 2 \pmod{11} \\y_3 &= 9 \cdot (1 - 2) - 4 = 9 \pmod{11},\end{aligned}$$

így $2P = (2, 9)$. ♣

Elliptikus görbék kriptográfiai alkalmazása a generált csoportban a diszkrét logaritmusképzés feladat nehézségére alapoz. A diszkrét logaritmusképzés feladat elliptikus görbékre a következő:

3.5. Definíció (diszkrét logaritmusképzés feladat elliptikus görbékre).

G = E csoportban adott Q ∈ E és P ∈ E pontok esetén megkeresni azt az i egész számot, amelyre iQ = P, feltéve, hogy ilyen létezik.

Ennek megfelelően a Diffie–Hellman (DH) kulccscsere algoritmus (az algoritmus részletes leírását lásd a 8.4. szakaszban) elliptikus görbékre a következő:

Diffie–Hellman-protokoll elliptikus görbékre

- | | |
|------|----------------------------|
| (1) | $A \rightarrow B : k_{AQ}$ |
| (2) | $B \rightarrow A : k_{BQ}$ |
| (3a) | $A : P = k_A(k_{BQ})$ |
| (3b) | $B : P = k_B(k_{AQ})$ |
-

ahol egy $E, GF(q)$ feletti elliptikus görbe ($E/GF(q)$) egy előre megállapodott Q pontja a nyilvános kulcs, továbbá k_A az A , illetve k_B a B fél által választott véletlen egész szám az $\{1, 2, \dots, |G|\}$ halmazból. Az E elliptikus görbe így kiszámított P pontja lesz a közös kulcs, amelyet előre megállapodott módon konvertál egy bináris sorozatba a két fél. Egy hallgatózó támadó a $\{G, k_{AQ}, k_{BQ}, Q\}$ ismeretében szeretné meghatározni $P = k_A k_B Q$ pontot, amivel a Diffie–Hellman számítási probléma nehézségével kerül szembe.

Az ElGamal rejtjelezés a DH-kulccscsere kis módosításával megkapható. Ha A fél szeretne B félnek egy titkos M üzenetet küldeni, akkor a DH-kulccscsere (3a) és (3b) lépése helyett a

...

$$(3) \quad A \rightarrow B : rQ \mid M + r(k_B Q)$$

lépés kerül végrehajtásra, ahol r egy további véletlen egész szám az $\{1, 2, \dots, |G|\}$ halmazból.

A diszkrét logaritmusképzés problémára épülő alkalmazások, mint a Diffie–Hellman-kulccscsere protokoll, illetve az ElGamal nyilvános kulcsú rejtjelező algoritmus, eredetileg a $GF(p)$ illetve a $GF(2^n)$ véges testek $p - 1$, illetve $2^n - 1$ méretű multiplikatív csoportjára épültek. Míg azonban ezen csoportok esetén léteznek szubexponenciális algoritmusok a logaritmusképzésre, addig az elliptikus görbék esetén ilyen nem ismert. Pontosabban még $GF(q)$ feletti diszkrét logaritmusképzésre a jó algoritmusok futási ideje $e^{O(\sqrt{\ln q \ln \ln q})}$, addig ($E/GF(q)$) esetén a diszkrét logaritmusképzés ideje nem gyorsabb, mint egy általános csoportra: $e^{O(\ln q)}$.

3.5. Feladatok

3.1. Feladat. Egy játék RSA algoritmus esetén: $p = 5, q = 11$ prímeket választottuk. Adja meg a legkisebb kódoló kulcsot, s az ehhez tartozó dekódoló kulcsot! Írja alá az $x = 2$ üzenetet!

3.2. Feladat. Tekintsük az RSA rejtjelezést $p = 23, q = 71$ prímekkel.

1. Hány olyan x üzenet van ($1 < x < N$), amelynek nincs közös faktorja a titkos prímekkel?
2. Mekkora annak a valószínűsége, hogy egy véletlenszerű x üzenet N faktORIZÁCIÓJÁRA ad lehetőséget?

3.3. Feladat. Támadjon meg egy RSA rejtjelezőt, amelyről a nyilvános $N = 4003997$ modulus, valamint az $e = 379$ kulcs mellett megtudja a $\varphi(N) = 3999996$ értéket is.

1. Számítsa ki a d dekódoló kulcsot!
2. Adja meg az $N = pq$ prímfaktorjait!

3.4. Feladat. Ha két, RSA rejtjelezéssel kommunikáló pár közötti különböző lehetséges üzenetek száma kicsi halmoz, támadásra ad lehetőséget. Tegyük fel, hogy a támadó ismeri az üzenethalmazt, de nem ismeri az RSA dekódoló kulcsot, s támadásra szánja el magát.

1. Adja meg a támadás menetét!
2. Javasoljon algoritmikus védekezési módot!

3.5. Feladat*. Egy y rejtjelezett blokkot figyelünk meg. Feladat az x nyílt blokk meghatározása. Az x vonatkozásában semmi a priori ismeretünk nincsen, az tetszőleges lehet. Elvileg meghatározható-e x , ha

1. RSA
2. DES

kódolással készült? Indokoljon, illetve adjon módszert! Segít-e, ha ismert formátumú az üzenet?

3.6. Feladat. Ha gyors RSA kódolást kívánunk megvalósítani, miért előnyös az $e = 3$, illetve általában az $e = 2^t + 1$ alakú választás? Adja meg a számításigényt a szükséges szorzások számában!

3.7. Feladat. Fermat-álprím-e

1. $n = 87$ a $b = 5$ bázisra nézve?
2. $n = 33$ a $b = 32$ bázisra nézve?
3. Általánosítsa a 2.) esetet! Vonja le a következtetést!

3.8. Feladat. Ha egy RSA kódolást a rejtjelezett blokkon ismételten sokszor elvégzünk, akkor szerencsével a következő támadást hajthatjuk végre:

Megfigyelünk egy y rejtjeles blokkot, majd kódoljuk azt a nyilvános kulccsal, s aztán az eredményt újra kódoljuk, és így tovább. Ha az i -edik újrakódolás után újra visszakapjuk y rejtjeles blokkot, azaz $y = (x^{e^{i-1}})^e$, akkor – az invertálhatóság kapcsán – nyilván $x = x^{e^{i-1}}$, azaz egyértelműen megkapjuk a nyílt blokkot. Mutassuk meg, hogy az $m = 35$ modulusú játék-RSA ezen támadás szempontjából nem biztonságos semmilyen szóbajövő e nyilvános kitevő mellett sem!

3.9. Feladat. Tekintsünk egy RSA rejtjelezőt $e = 3$ nyilvános kulccsal. Legyen a blokkhossz 128 byte (1024 bit). Tegyük fel, hogy rövidek az üzeneteink, hosszuk nem nagyobb, mint 40 byte, s a nagyobb helyiértékek felé nullákkal egész blokkokra egészítjük ki azokat. Van aki azt állítja, hogy a rejtett blokkokat lehallgatva fejti az üzeneteket. Hihetetlen? Milyen tanulságot vonna le?

3.10. Feladat. Az RSA kódolás is blokk kódolás, ezért felmerül az ismert blokk kódolás módok alkalmazásának lehetősége az RSA felhasználásával

(a blokkrejtjelezési módok leírását lásd az 5. fejezetben.). A CFB, illetve az OFB mód alkalmazhatók-e?

3.11. Feladat. Az RSA algoritmus a kulcsgenerálásnál a $\varphi(N) = (p-1) \cdot (q-1)$ modulust alkalmazza. Használhatnánk-e ehelyett a $z = l.k.k.t.(p-1, q-1)$ modulust?

3.12. Feladat. Igazolja, ha az alábbi két feladat valamelyikére van hatékony megoldó algoritmusunk, akkor van a másikra is:

1. dekódoló exponens kiszámítása ($N = pq, e$) nyilvános adatokból,
2. $N = pq$ szorzat faktorizálása.

3.13. Feladat. Ha az üzenetek tere kicsi, a támadás megelőzésére szokásos megoldás a nyílt üzenet friss véletlen elemmel bővítése a rejtjelezést megelőzően. Tegyük fel, hogy az üzenethossz egy RSA blokk méretű. Hogyan végezzük a véletlen elemmel bővítést a rejtjelezés előtt:

1. tömörítsük az üzenetek halmazát, s egy blokknál rövidebb bithosszal írjuk le azokat, majd egészítsük ki véletlen bitekkel teljes blokkra, vagy
2. bővítsük egy teljes véletlen blokkal két blokk hosszúvá az üzenetet, majd alkalmazzunk CBC módú láncolásos rejtjelezést (a CBC leírását lásd az 5. fejezetben)? Ekkor két blokkot kell továbbítanunk, de a módszer, amennyiben biztonságos, használható lenne tetszőleges blokkszámú nyílt szöveg esetére.

3.14. Feladat. Legyen x_1 és x_2 két üzenet, továbbá y_1 és y_2 a megfelelő két rejtjeles blokk, amit RSA kódolással kaptunk. Érvényes a következő multiplikatív tulajdonság:

$$(x_1 x_2)^e = y_1 y_2 \bmod N,$$

azaz az üzenetek szorzatának rejtjeles képe a rejtjeles képek $\bmod N$ szorzata. Egy támadó dekódolni szeretné egy $x^e = y \bmod N$ rejtett szöveget. Tegyük fel, hogy a megtámadott dekódol tetszőleges neki küldött, számára nem bizalmas tartalmú üzenetet. Hogyan hajthat végre a támadó sikeres támadást az y dekódolására?

3.15. Feladat. A 3.14. feladatban említett multiplikatív tulajdonság kapcsán felmerül a kérdés, hogy lehet-e olyan balszerencsés üzenetformátumot megadni, amelynél két üzenet szorzata szintén – a formátum szerint – elfogad-

ható üzenetre vezet? Ugyanis ekkor, lévén az üzenetek szorzatának rejtjeles képe a rejtjeles képek szorzata, az üzenetformátum megkötést is kijátszhatná a támadó.

3.16. Feladat*. Az e, d, N RSA paramétereket ismerve, faktorizálható-e az N modulus?

3.17. Feladat. Egy S megbízható szerveren – azért, hogy egyfél modulo aritmetikát kelljen csak használni – az A_1, A_2, A_3, \dots felhasználók mindenkihez azonos N modulussal felépített RSA kódolót telepítenek. Természetesen az (e_i, d_i) nyilvános-rejtett kulcs párokból a szerver különbözőket oszt ki a különböző felhasználók számára. Nyilván nem oszthatja ki a p és q prímelek. Ennek ellenére veszélyes az azonos modulus választás. Miért?

3.18. Feladat. Mekkora veszélyt jelent az az elvi lehetőség, hogy az RSA rejtjelezés eredménye megegyezik a nyílt szöveggel, azaz nem történik meg a rejtés?

3.19. Feladat. Szeretnénk megtudni A felhasználó $N = pq$ RSA modulusa titkos faktorjait. Felelőtlenül jár-e el A, ha „baráti” kérésünkre egy b számnak megmondja egyik $\text{mod } N$ szerinti négyzetgyökét.

3.20. Feladat. Ha az A felhasználó egy az RSA algoritmuson alapuló aláíráshoz egy m_A , míg a B felhasználó felé történő, ugyancsak RSA alapú rejtjelezéshez m_B modulust alkalmaz, ahol $m_A > m_B$, akkor rejtjelezéssel védett aláírt blokk esetén fellép a blokkméret-probléma: ha egy M üzenetblokkot $E_B[D_A(M)]$ módon kódolunk, akkor gondba kerülünk a kódolásnál, mivel a $D_A(M)$, mint egész szám nagyobb, mint az m_B modulus. Két megoldáson gondolkozunk:

1. Megoldást jelentene-e a kétféle transzformáció modulus méretének megfelelő sorrendű alkalmazása, azaz esetünkben A először rejtjelezzen, majd írjon alá?
2. A rendszerben a modulusokat a következő alakban választjuk: legyen a modulus a következő bináris alakú: $100\dots0xx\dots x$, ahol a legmagasabb helyiértékű 1 bitet k darab 0 bit követi. Mutassuk meg, hogy ekkor legfeljebb $(1/2)^k$ a valószínűsége annak, hogy a blokkméret-probléma előforduljon!

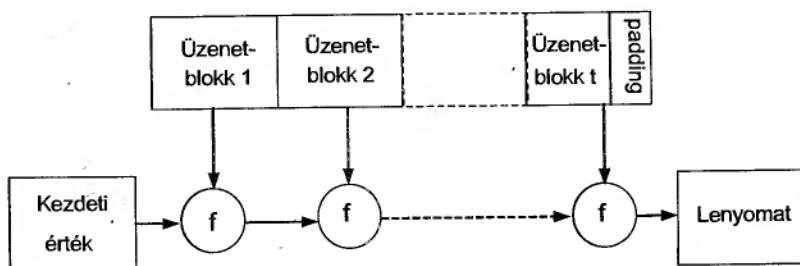
4.

Kriptográfiai hash függvények

Egy $H : \{0,1\}^* \rightarrow \{0,1\}^n$ hash függvény tetszőleges hosszúságú űsképtérbeli bináris sorzatot rögzített, n hosszúságú bináris sorozatba képez. A leggyakoribb alkalmazás okán, egy űsképtérbeli M elemet M üzenetnek (illetve dokumentumnak), a megfelelő $H(M)$ képtérbeli elemet hash értéknek (illetve lenyomatnak) nevezzük.

4.1. Hash függvény fajták és biztonsági kritériumok

A Merkle-től származó iterációs konstrukció alapja egy $f : \{0,1\}^{2n} \rightarrow \{0,1\}^n$ iterációs (kompressziós) függvény (4.1. ábra).



4.1. ábra. Iterált kriptográfiai hash függvény

Az $M \in \{0,1\}^*$ üzenetet, amelyre a lenyomatot képezzük, n bites blokkokra bontjuk. Legyen

$$M = [M_1, M_2, \dots, M_t]$$

ez a felbontás, ahol M_i az i -edik üzenetblokk, és az üzenet t blokkból áll. Ha az utolsó blokkba eső üzenetdarab n bitnél rövidebb lenne, akkor feltöljük n bit hosszúságúra (kiegészítés, padding). Az iterációs számítás

$$H_i = f(M_i, H_{i-1}),$$

$i = 1, 2, \dots, t$ lépésekben történik, ahol H_0 egy – nyilvános – kezdeti értéket vesz fel. A H (iterált) hash függvény a

$$H(M, H_0) = H_t$$

hash értéket szolgáltatja. A továbbiakban, ha hangsúlyozni kívánjuk a hash függvény iteratív voltát, kiemeljük a H_0 kezdeti értéket is.

A hash függvények két fő fajtáját különböztetjük meg: az egyirányú és az ütközésmentes hash függvényeket.

Az egyirányú tulajdonság az egyirányú függvények formális definíciója szerinti. A hash függvény specialitása, hogy az ōsképtere nem rögzített hoszsúszágú bináris sorozat. Az egyirányúság jelentése az, hogy könnyű feladat kiszámítani $H(M)$ lenyomatot bármely adott M üzenetre, de nehéz feladat adott üzenethez tartozó lenyomatra vezető üzenetet előállítani (az egyirányú függvény formális definícióját a bizonyított biztonság elmélete rész első fejezete tartalmazza). Az egyirányú hash függvények szokásos neve még az ōsképpellenálló, továbbá szokásos rövidítése az angol nevéből OWF (One Way Hash Function).

4.1. Példa. Ha nem lenne nehéz egy adott lenyomathoz ōsképet találni, akkor – elvileg – nyilvános kulcsú aláírást tudna hamisítani egy lehallgató támadó: a támadó előállítja a lenyomatot, amelyhez az aláírás is rendekezésre áll, majd ezen lenyomathoz előállít egy csaló ōsképet (dokumentumot). Fontos azonban észrevenni, hogy – amint az szokásos – egy dokumentum formátummal rendelkezik, s így elenyésző lesz annak a valószínűsége, hogy az előállított ōskép még a formátum követelménynek is eleget tegyen. ♣

4.1. Definíció (gyengén ütközés-ellenálló hash függvény). *A H hash függvény gyengén ütközés-ellenálló vagy második ōskép ellenálló M üzenetre, ha nehéz feladat előállítani egy $M' (\neq M)$ üzenetet (második ōsképet), amelynek azonos a lenyomata, azaz melyre $H(M') = H(M)$.*

4.2. Definíció (ütközés-ellenálló hash függvény). *Egy H hash függvény ütközés-ellenálló (CRHF, Collision Resistant Hash Function), ha nehéz olyan $M, M', M' \neq M$ üzenetpárt találni, amelyre $H(M') = H(M)$.*

4.2. Példa. Ha nem lenne nehéz ütköző párt előállítani, akkor maga az üzenetet küldő fél kísérelhetne meg csalást olyan módon, hogy még aláírás előtt előállít két, azonos lenyomatú dokumentumot, ahol az egyik a tartalékban maradó csaló célú dokumentum, a másik az aláíráusra és elküldésre szánt. A csaló dokumentumot abban az esetben mutatja fel a támadó, ha a körülmények úgy alakulnak az aláírást követően, hogy az elküldött, aláírt dokumentum tartalma számára már nem kedvező. ♣

Az ütközésmentesség tulajdonságból nem következik az ōsképek párai lenyomatainak „valamiféle függetlensége”. Statisztikai értelemben, adott üzenetpár esetén, illesmi nem is kerül szóba determinisztikus leképezések-nél, de még olyan mérőszámok vonatkozásban sem, mint a kicsi korreláltság, ahol M és M' bináris $2r$ hosszúságú üzenetek korrelációját $dist(H(M), H(M')) - r$ definálja, ahol $dist(x,y)$ a Hamming-távolság. Az ütközésmentesség tulajdonság csak azt követeli meg, hogy nehéz legyen olyan üzenetpárt találni, amelyre ezen korreláció maximális.

4.3. Definíció (szabad kezdőérték). *Iteratív hash függvényt tekintve, abban az esetben, ha a H_0 kezdőérték szabad megválasztása mellett állítunk elő ōsképet, második ōsképet vagy ütköző üzenetpárt, akkor azt mondjuk, hogy szabad kezdőérték melletti ōsképet, második ōsképet vagy ütköző üzenetpárt állítottunk elő.*

Megjegyezzük, hogy az angol nyelvű irodalomban a „pseudo” jelzőt használják e célból (pseudo-preimage, second pseudo-preimage, pseudo-collision). Nylván valamivel könnyebb feladat a szabad kezdőérték melletti támadás, ugyanakkor már akkor sem tekintjük a hash függvényt biztonságosnak, ha a szabad kezdőérték melletti támadás sikeres ellene.

4.4. Definíció (univerzális egyirányú hash függvény). *Abban az esetben beszélünk univerzális egyirányú hash függvényről (UOWHF), ha lenyomat-képző függvények egy $\{h_k\}_{k \in K}$ indexelt halmaza áll rendelkezésre, ahol az indexek egy K halmazból veszik az értékeit, továbbá a tagfüggvényeknek azonos ōsképterük, illetve képterük van, továbbá:*

Ha először választanunk kell egy M üzenetet, s ezután kerül csak véletlenszerűen kisorsolásra egy h_k függvény, akkor nehéz feladat olyan $M' (\neq M)$ üzenetet találni, amelyre $h_k(M') = h_k(M)$.

Az UOWHF a CRHF egy vonzó alternatívája, mivel valamivel egyszerűbb hatékony és biztonságos UOWHF előállítása, másrészt számos alkalmazásban (pl. digitális aláírás) elegendő biztonságot nyújthat. A Cramer–Shoup rejtjelező konstrukció kapcsán láthatjuk egy felhasználását a bizonyított biztonságú nyilvános kulcsú rejtjelezőkről szóló fejezetben.

4.5. Definíció (kulcsolt hash függvény). *Kulcsolt hash függvény tetszőleges hosszúságú M üzenetet és rögzített hosszúságú K kulcsot rögzített hosszúságú $H(k, M)$ lenyomatba képezi le az alábbi tulajdonságokkal:*

1. könnyű kiszámítani $H(k, M)$ lenyomatot adott M és k esetén,
2. k kulcs ismerete nélkül nehéz kiszámítani a $H(k, M)$ értéket tetszőleges M üzenet esetén, még az esetben is, ha ismert $(M_i, H(k, M_i))$, $M \neq M_i$, $i = 1, 2, \dots$ párok általnak rendelkezésre,
3. nehéz a k kulcs kiszámítása $(M_i, H(k, M_i))$, $i = 1, 2, \dots$ párok ismeretében,
4. k kulcs ismerete nélkül fennáll az egyirányúság és a gyenge ütközésmennesség, azaz
 - (a) ha adott y lenyomat, nehéz feladat olyan M üzenetet találni, amelyre $H(k, M) = y$,
 - (b) ha adott M és $H(k, M)$, nehéz olyan $M' (\neq M)$ üzenetet (második ősképet) találni, amelyre $H(k, M') = H(k, M)$.

A kulcsos hash függvény természetes alkalmazása az üzenethitelesítés célú kriptográfiai ellenőrzőösszeg képzése, amellyel rejtjelező kódolásra támaszkodás (pl. CBC-MAC) nélkül lehet biztonságos ellenőrzőösszeget képezni.

4.3. Példa. Ha kulcsolt hash alapján szeretnénk kriptográfiai ellenőrzőösszeget készíteni, körültekintően kell eljárnunk. Egy tipikus hiba a következő: ha a H_0 kezdővektort adja a kulcs, akkor könnyen generálható egy ellenőrzőösszeg új üzenethez; legyen az új üzenet a megfigyelt üzenet bármely folytatása. ♣

Alább vizsgáljuk az ütközésmennesség, a gyengén ütközésmennesség, valamint az egyirányúság tulajdonságok kapcsolatát. Kiderül, hogy az ütközésmennesség a legerősebb tulajdonság. Az első állítás az ütközésmennesség és a gyengén ütközésmennesség tulajdonság szoros kapcsolatára vonatkozik.

4.6. Tétel. *Egy H hash függvény akkor és csak akkor ütközésmentes, ha nehéz olyan M üzenetet találni, amelyre nem gyengén ütközésmentes.*

Bizonyítás: Ha van hatékony algoritmusunk, amely H függvényhez felmutat ütköző párta, akkor ezen pár bármelyik tagja nem gyengén ütközésmentes, hiszen már ismerjük az ütköző pájrát.

Megfordítva, ha van hatékony algoritmusunk, amely előállít egy nem gyengén ütközésmentes M üzenetet, ez – definíció szerint – azt jelenti, hogy nem nehéz feladat előállítani hozzá egy azonos lenyomatú $M' (\neq M)$ üzenetet. Ez viszont azt jelenti, hogy M, M' ütköző üzenetpár előállítása nem nehéz feladat. \square

A téTEL egyik következménye, hogy ütközésmentes hash függvény esetén a nem gyengén ütközésmentes üzenetek részaránya elhanyagolhatóan kicsi. Az alábbi téTEL az ütközésmentesség és az egyirányúság tulajdonság kapcsolatát tárja fel:

4.7. Tétel. *Jelölje X illetve Y véges halmaz a H függvény ősképterét, illetve képterét, ahol $|Y| < |X|$. Tegyük fel, hogy létezik egy hatékony I invertáló algoritmus, amely tetszőleges lenyomatra, mint bemenetre, kiszámít egy üzenetet. Ekkor megadható egy hatékony algoritmus, amely legalább $1 - |Y|/|X|$ valószínűséggel ütköző üzenetpárt tud előállítani.*

Bizonyítás: Tegyük fel, hogy H nem egyirányú. Legyen I egy hatékony invertáló algoritmus. Válasszunk véletlenül egy M üzenetet, s számítsuk ki az $M' = I(H(M))$ ősképet. Ha $M' \neq M$, akkor sikeresült ütköző pár előállítani. Vizsgáljuk a $P(M' \neq M)$ sikervalószínűséget. Jelölje X illetve Y véges halmaz a H függvény ősképterét illetve képterét. Jelölje $\{M\}$ a $H(M)$ lenyomathoz tartozó ősképek halmazát. Válasszunk ki egy-egy reprezentáns elemet egy-egy ilyen ősképhalmazból. Legyen ezen reprezentánsok halmaza V , s a jelölésünknek megfelelően, ha v egy reprezentáns elem, akkor $\{v\}$ jelölje a v által reprezentált ősképhalmazt. Nyilván

$$P[M' \neq M | M] = (|\{M\}| - 1)/|\{M\}|,$$

ahonnan egyenletes eloszlást feltételezve az üzenetek halmaza felett:

$$\begin{aligned} P[M' \neq M] &= \frac{1}{|X|} \sum_M P[M' \neq M | M] = \frac{1}{|X|} \sum_M (|\{M\}| - 1)/|\{M\}| \\ &= \frac{1}{|X|} \sum_{v \in V} \sum_{M \in \{v\}} (|\{v\}| - 1)/|\{v\}| = \frac{1}{|X|} \sum_{v \in V} (|\{v\}| - 1) \\ &= \frac{1}{|X|} (|X| - |Y|) = 1 - |Y|/|X|. \end{aligned}$$

Mivel az iterációs függvény tömörítő, ezért már egy blokknyi üzenet lenyomata képzésekor is nagy a siker valószínűsége. Ha például felére tömörít az iterációs függvény, akkor egy üzenetblokk esetén 1/2 valószínűség adódik az ütköző pár találati sikerre. \square

Az iterációs lenyomatképzés előnye, hogy tetszőleges hosszú üzenet lenyomatának képzését, s ezen lenyomatképzés kriptográfiai tulajdonságait visszavezethetővé teszi az iterációs függvény megfelelő tulajdonságaira. Felmerül a kérdés, hogy az f kompressziós függvény mely tulajdonságai eredményeznek biztonságos iteratív hash leképezést, nevezetesen egyirányú, illetve ütközésmentesség tulajdonságot. Azt nem nehéz látni, hogy gyenge kompressziós függvényekre alapozott iterációs függvény is gyenge lesz, pl. ha f nem ütközésmentes, akkor egy blokk hosszú üzeneteket tekintve a hash leképezés sem ütközésmentes (szabad H_0 választásra semmiképp). Viszont nem feltétlen terjed ki az iterációs hash függvényre a kompressziós függvény egy jó tulajdonsága. Az ütközésmentesség vonatkozásában szerencsére ismeretes egy igen egyszerű konstrukció, amely ezt a kiterjesztést garantálja:

4.8. Tétel (Damgard–Merkle-(DM)-kiegészítés). *Tegyük fel, hogy van egy f kompressziós függvényünk ütközésellenálló tulajdonsággal. Ha az $M = [M_1, M_2, \dots, M_{r-1}]$ üzenetet kiegészítjük egy M_r blokkal, amely tartalmazza az M üzenet bithosszát (nulla bitekkel kiegészítve egész blokkhosszra), akkor az f kompressziós függvényre épülő iterációs hash függvény is ütközésmentes lesz.*

Bizonyítás: Indirekt. Tegyük fel, hogy hatékonyan elő tudunk állítani ütköző M, M' üzenetpárt, ahol most az üzenetbe még nem értük bele a kiegészítést. Két esetet különböztetünk meg: az első esetet, amikor azonos bithosszú a pár két tagja, illetve a második esetet, amikor különböző.

Az első esetben $m = [M_1, M_2, \dots, M_r], m' = [M'_1, M'_2, \dots, M'_r]$ párra áll elő azonos kimenet: az r -edik üzenetblokk azonos (az azonos bithosszot tartalmazza), ezért H CRHF tulajdonsága miatt ez csak úgy lehetséges, ha a közbenső, $(r-1)$ -edik iterációs lépés kimenete is azonos; ha az $(r-1)$. üzenetblokk is azonos, akkor H CRHF tulajdonsága miatt ez ismét csak úgy lehetséges, ha a közbenső, $(r-2)$ -edik iterációs lépés kimenete is azonos, és így tovább. Ha – ilyen módon visszafelé haladva az iterációban – a j -edik blokk az első különböző a két üzenetben, akkor ellentmondásra jutunk, mivel ezen lépésekben ütközés állna elő a kompressziós függvényre.

A második esetben $m = [M_1, M_2, \dots, M_r], m' = [M'_1, M'_2, \dots, M'_s]$ párra áll elő azonos kimenet, ahol az utolsó blokkok különbözők (a különböző bit-

hosszot tartalmazzák). Mivel a r -edik üzenetblokk különböző, ezért semmi-képp sem állhatott elő ütközés a kimeneten, ellenkező esetben H ütközésmentes tuladonságával kerülnénk ellentmondásba. \square

4.2. A születésnapi paradoxon

A születésnapi paradoxonnal kapcsolatos kérdés a következő: „Mekkora a valószínűsége annak az eseménynek, hogy emberek egy véletlenszerűen választott r fős csoportjában van legalább két személy, akik azonos napon születtek?” A paradoxon, vagyis az ember józan sejtésével nehezen érthető tény az, hogy ahhoz, hogy a kérdéses valószínűség $\frac{1}{2}$ körüli érték legyen, elegendő $r = 23$ személyből álló csoportot választani. Ugyanakkor $r = 253$ személyből álló csapat szükséges ahhoz, hogy $\frac{1}{2}$ körüli valószínűsége legyen annak, hogy a választott r személy között van egy, aki egy adott napon született.

A kérdéses P valószínűség elemi kombinatorikus úton meghatározható:

$$P = 1 - \frac{\binom{365}{r} \cdot r!}{365^r},$$

amely jól közelíthető a

$$P = 1 - e^{-\frac{r^2}{2m}}$$

kifejezéssel ($m = 365$). Az $r = \sqrt{m}$ választás esetén, $P \approx 1 - e^{-\frac{1}{2}} \approx 0.4$.

Tehát ha van egy nagy méretű halmazunk, amelynek elemeit egy egész számmal kifejezett jellemzővel címkezzük fel, amely jellemző m különböző értékű lehet, akkor m négyzetgyökének nagyságrendjébe eső méretű részhalmazt választva már nagy annak valószínűsége, hogy a választott részhalmazban van legalább kettő, azonos tulajdonsággal felcímeksző elem.

Egy kissé módosított formában is fogjuk használni a paradoxont, s ez a következő kérdéshez kapcsolódik: „Mekkora a valószínűsége annak, hogy egy U nagy méretű alaphalmazból véletlenszerűen választva V és W r méretű részhalmazokat, azok metszete nem üres?”

A kérdéses P' valószínűség és közelítése az alábbi:

$$P' = 1 - \frac{\binom{m}{2r} \cdot (2r)!}{\left(\binom{m}{r} \cdot r!\right)^2} \approx 1 - e^{-\frac{3r^2}{m}}.$$

Az $r = \sqrt{m}$ választás esetén, a $P' \approx 1 - e^{-3} \approx 0.95$ közelítést kapjuk.

A születésnapi paradoxon jelentőségét a hash függvények esetében az adja, hogy ez alapján becsülhető egy ütközés találásának valószínűsége. Ha

ugyanis egy H hash függvény kimenete n bites, akkor a születésnapi paradoxon alapján $2^{\frac{n}{2}}$ véletlenül választott üzenet között $\frac{1}{2}$ -hez közel valószínűséggel lesz kettő, melynek ugyanaz a hash értéke, azaz ütköző párt alkot. Az ütközés-ellenállóság garantálásához tehát a kimenetet úgy kell méretezni, hogy $2^{\frac{n}{2}}$ megfelelően nagy legyen, és így az ütköző párok véletlen választással történő keresése gyakorlatilag kivitelezhetetlen feladattá váljon. Ez ma tipikusan $n = 160$ bites kimenetet jelent.

4.3. Bizonyítható biztonságú konstrukciók

Az MD-kiegészítést már tekinthetjük egynek azon konstrukciók közül, amelyek bizonyítható tulajdonságú hash függvényekre vezetnek. A bizonyítható biztonság vonatkozásban fő kutatási irányok és tervezési módszerek az alábbi csoportokba sorolhatók:

1. információelméleti módszerek;
2. komplexitáselméleti módszerek;
3. (ideális) kriptográfiai primitívekre épülő hash függvények;
4. dedikált módszerek.

Az információelméleti megközelítés a tökéletes hitelesség elmélet keretében konstruál hash függvényeket, s a cél a tökéletes rejtjelezés megfelelője a hitelesség feladatra. A konstrukciók kombinatorikai alapúak, diszkrét valószínűségszámítási technikákat is alkalmazva.

A komplexitáselméleti módszerek a redukciós technikát alkalmazzák a bizonyításaiakban, a tervezett hash függvény biztonsága támadhatóságát redukálják egy standard nehéz feladat hatékony megoldhatóságára. Az alábbiakban egy példát mutatunk erre a módszerre, amely a diszkrét logaritmus-kepzés nehézségére épít.

Kis számításigényű primitívekből építkezés esetén gyors hash függvények tervezhetők, ugyanakkor kérdés a primitívek kriptográfiai tulajdonságaival kapcsolatos feltételezések (modell) helyessége. Továbbá az adott primitív (tipikusan szimmetrikus kulcsú blokkrejtjelező) – eredeti használata miatt – olyan tulajdonságokkal is rendelkezhet, ami az új felhasználási helyén már veszélyes lehet, s nem utolsósorban az így kapható hash függvények nem optimalizálhatók futási időre, lévén az építőelemeik futási ideje adott. Az első ilyen (publikált) tervezés a DM (Davies–Meyer) hash függvény volt, amelyre alább szintén visszatérünk.

A bizonyítható biztonságot nyújtó tervezések minden ideig nem vezettek gyakorlatban is jól használható (kriptográfiai) hash függvényekre. A gyakorlatban széles körben használt hash függvények (pl. MD család, SHA-1, RIPEMD-160, SNEFRU) minden közvetlen, dedikált tervezések, amelyeknél a tervező

1. közvetlenül figyelembe tudja venni az erőforráskorlátokat (futási idő, tárkapacitás), erre is optimalizál;
2. igyekszik elkerülni olyan konstrukciós hibákat, amelyek a tervezés idejéig ismert támadások sikeressége alapjául szolgáltak.

Ugyanakkor minden ideig – sajnálosan – nem állnak rendelkezésre olyan (publikált) tervezési módszerök, amelyek a dedikált tervezések esetén bizonyítható hash tulajdonságokra vezetnének. Ez nem azt jelenti, hogy a dedikált tervezések során kritériumok teljesítésére ne lenne törekvés, mint például hatékony diffúzió, nemlinearitás. Ugyanakkor ezen kritériumok formálisan nem kapcsolhatók össze a hash függvények olyan (komplexitás-elmélet indítatású) biztonsági kritériumaival, mint amilyen az egyirányúság vagy az ütközésmenetesség. A dedikált tervezések kriptográfiai kritériumai lényegében elsősorban arra irányulnak, hogy egy támadó ne állíthasson fel olyan egyenletrendszert (viszonylag kevés változóban, linearitásokkal), amelyek alapján őskép, második őskép vagy ütköző pár számítása erőforrásai lehetőségein belülre kerülne.

4.3.1. Egy komplexitás-elméleti konstrukció

Az alábbi konstrukció diszkrét logaritmusképzés nehézségére épül. A konstrukció az alábbi:

Legyen p egy nagy prímszám, amelyre $q = (p - 1)/2$ is prímszám. Legyen α és β a $GF(p)$ két primitív eleme. p, α, β nyilvánosak. A $\log_{\alpha} \beta$ diszkrét logaritmust nem hozzuk nyilvánosságra, és feltételezzük, hogy kiszámítása nehéz. Legyen egy $H : \{0, \dots, q - 1\}^2 \rightarrow GF(p) \setminus 0$ hash függvény az alábbi:

$$H(x_1, x_2) = \alpha^{x_1} \beta^{x_2} \pmod{p}. \quad (4.1)$$

4.9. Tétel (Chaum–van Heijst–Pfitzman). A (4.1) formulával definiált hash függvény ütközésmentes, ha $\log_{\alpha} \beta$ diszkrét logaritmus kiszámítása nehéz.

Bizonyítás: A bizonyítás indirekt. Tegyük fel, hogy rendelkezésünkre áll egy $[(x_1, x_2), (x_3, x_4)]$, $(x_1, x_2) \neq (x_3, x_4)$ üzenetpár, amely ütköző, azaz amelyre

$H(x_1, x_2) = H(x_3, x_4)$. Így fennáll $\alpha^{x_1}\beta^{x_2} = \alpha^{x_3}\beta^{x_4} \pmod{p}$ összefüggés, következésképpen az

$$\alpha^{x_1-x_3} = \beta^{x_4-x_2} \pmod{p} \quad (4.2)$$

összefüggés. Legyen

$$d = l.n.k.o.(x_4 - x_2, p - 1).$$

Mivel $p - 1 = 2q$, ahol q prim, ezért $d \in \{1, 2, q, p - 1\}$. A d ezen négy lehetséges értékét vizsgáljuk a továbbiakban.

Tegyük fel, hogy $d = 1$: Legyen $y = (x_4 - x_2)^{-1} \pmod{p - 1}$. (4.2) fizetélyelemben $\beta = \beta^{(x_4-x_2)y} = \alpha^{(x_1-x_3)y} \pmod{p}$, ezért a $\log_{\alpha} \beta$ logaritmust ki tudjuk számítani a következőképpen:

$$\log_{\alpha} \beta = (x_1 - x_3)(x_4 - x_2)^{-1} \pmod{p - 1}.$$

Tegyük fel, hogy $d = 2$: Mivel $p - 1 = 2q$, ahol q páratlan, ezért $l.n.k.o.(x_4 - x_2, q) = 1$. Legyen $z = (x_4 - x_2)^{-1} \pmod{q}$. Mivel $(x_4 - x_2)z = kq + 1$ valamely k egészre, ezért

$$\beta^{(x_4-x_2)z} = \beta^{kq+1} = (-1)^k \beta = \pm \beta \pmod{p},$$

ugyanis $\beta^q = -1 \pmod{p}$. Innen adódik, hogy

$$\beta^{(x_4-x_2)z} = \alpha^{(x_1-x_3)z} = \pm \beta \pmod{p},$$

azaz vagy

$$\log_{\alpha} \beta = (x_1 - x_3)z \pmod{p - 1},$$

vagy

$$\log_{\alpha} \beta = (x_1 - x_3)z + q \pmod{p - 1}.$$

Könnyen ellenőrizhető, hogy melyik az igazi a kettő közül, azaz ez esetben is kiszámítottuk a $\log_{\alpha} \beta$ logaritmust.

Tegyük fel, hogy $d = q$: Mivel $0 \leq x_2 \leq q - 1$ és $0 \leq x_4 \leq q - 1$, így $-(q - 1) \leq x_4 - x_2 \leq q - 1$. Ez azonban nem lehetséges, hiszen feltételezésünk szerint $l.n.k.o.(x_4 - x_2, p - 1) = q$.

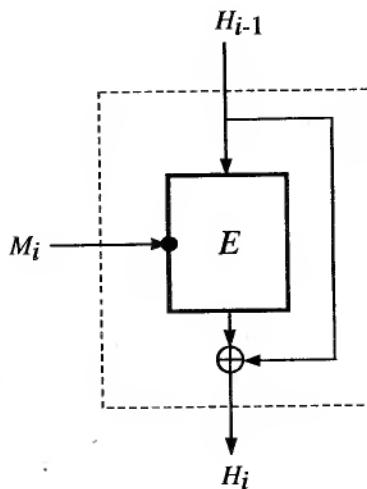
Végül tegyük fel, hogy $d = p - 1$: Ez $0 \leq x_2, x_4 \leq q - 1$ miatt csak akkor lehetséges, ha $x_2 = x_4$. Ekkor $\alpha^{x_1}\beta^{x_2} = \alpha^{x_3}\beta^{x_2} \pmod{p}$, s ezért $\alpha^{x_1} = \alpha^{x_3} \pmod{p}$, azaz $x_1 = x_3$ is fennállnak. Azonban ekkor $(x_1, x_2) = (x_3, x_4)$ következne. Tehát ez az eset sem lehetséges. \square

4.3.2. Davies–Meyer-(DM)-séma

Tekintsük a

$$H_i = f(M_i, H_{i-1}) = E_{M_i}(H_{i-1}) + H_{i-1} \quad (4.3)$$

alakú iterációs függvényt. Ez a Davies–Meyer-(DM)-séma (4.2. ábra). Ennek megfelelő struktúrát alkalmaz számos elterjedt hash függvény, így például az MD-x család (MD: Message Digest), amely esetben az E rejtjelező transzformáció helyett egy F invertálható transzformáció szerepel. A bemenet előrecsatolásával azt kívánjuk elérni, hogy a tömörítő függvény ne legyen könnyen invertálható az $[M_i, H_{i-1}]$ bemenetére. Ellenkező esetben az iterációs láncban visszafelé is tudnánk számolni, s ennek alapján középen találkozásos születésnapi paradoxon támadáson alapuló (rögzített H_0 mellett) űskép kiszámítását tudnánk elvégzni.



4.2. ábra. Előrecsatolással képzett iterációs függvény

A DM séma egyirányúságának bizonyítását R. Merkle adta meg, aki az egyik úttörő kutató volt a kriptográfiai hash függvények konstrukciós alapjainak kidolgozásában. Az E rejtjelezőt ideális modellel, egy titkos véletlen függvénnnyel, helyettesítjük, s ezen „idealizált” építőelemet tartalmazó konstrukcióra bizonyítjuk azt a tulajdonságot, amit az eredeti konstrukcióra sz-

rettünk volna belátni. (Rokon bizonyítási technikát mutatunk be a bizonyított biztonság elmélet véletlen orákulumos technikája kapcsán.)

4.10. Tétel (Merkle). A DM séma egyirányú hash függvényt ad, ha az F transzformáció véletlen leképezéssel modellezhető, ahol a támadó erőforrására nem tesziink korlátozást.

Bizonyítás: (Vázlat.) Tegyük fel, hogy az E transzformáció helyett egy véletlen leképezés áll, amely a különböző, rögzített $[M_i, H_{i-1}]$ bemeneteit független, a kimenet téren egyenletes eloszlású valószínűségi változókba képezi le. Ekkor az XOR összeadás miatt, az f kompresziós függvény kimenetén ulyanez a tulajdonság érvényben marad. Az f bemenete és kimenete statisztikailag független bármely bemeneti eloszlás mellett. Az invertáló támadó feladata, hogy egy y kimenet ismeretében egy hozzá tartozó x bemenetet kiszámoljon. Ehhez használhat x -től különböző bemenetekhez kért [bemenet, kimenet] párokat a polinomiális korlátton belüli számban. A különböző bemenetekhez független kimenetek tartoznak, így ezen információ mellett is csak elhanyagolható valószínűséggel (bemenet dimenzióban exponenciálisan kicsi valószínűséggel) lehet sikeres. \square

Ezen egyirányúság bizonyítás tehát nem az eredeti (konkrét) f leképezésre, hanem annak egy idealizált modelljére vonatkozik. Ha a véletlen függvényt kriptográfiai álvéletlen függvénnyel helyettesítjük (lásd az álvéletlen függvényekről szóló 16. fejezetet), a fenti tétel érvényben marad, amennyiben a polinom erőforráskorlátú támadót tételezünk fel.

Megjegyezzük, hogy az ütközés tulajdonság vonatkozásban véletlen leképezés modell mellett a születésnapi paradoxon felhasználásával juthatunk analóg állításra.

4.4. Feladatok

4.1. Feladat*. Tekintsünk egy $h : X \rightarrow Y$ hash függvényt. minden $y \in Y$ -ra jelölje $h^{-1}(y)$ az y ősképeinek halmazát:

$$h^{-1}(y) = \{x : h(x) = y\}.$$

Legyen ε annak a valószínűsége, hogy két véletlen választott (nem feltétlen különböző) X -beli elem, x_1 és x_2 ütközik, azaz $h(x_1) = h(x_2)$. Bizonyítsa be, hogy

$$\varepsilon \geq \frac{1}{|Y|},$$

és az egyenlőség akkor és csak akkor áll fenn, ha minden $y \in Y$ -ra

$$|h^{-1}(y)| = \frac{|X|}{|Y|}.$$

Segítség: Ismert, hogy minden $a_1, a_2, \dots, a_n \geq 0$ esetén

$$\frac{1}{n} \sum_{i=1}^n a_i^2 \geq \left(\frac{1}{n} \sum_{i=1}^n a_i \right)^2,$$

ahol az egyenlőség akkor és csak akkor áll fenn, ha minden a_i egyenlő.

4.2. Feladat. Az iterációs függvény „erejénél” nem nagyobb a hash függvény „ereje”, azaz az előbbire végrehajtható hatékony támadás kiterjeszthető a hash függvény elleni hatékony támadássá. Mutassuk meg ezt a CRHF tulajdonságára!

4.3. Feladat. Egy iterációs hash függvény esetén mutassuk meg, hogy egy $H(m, H_0) = H(m', H'_0)$, $m \neq m'$ pszeudo ütközés előállítása nem nehéz feladat!

4.4. Feladat. Tekintsünk egy ideális H hash függvényt, amely n bit méretű hash értéket ad, s tekinthetjük véletlenszerűnek a leképezést. Mutassuk meg, hogy az ütközéshez szükséges számítások várható értékére jó közelítés $O(2^{n/2})$! (Segítség: Tekintsen t különböző bemenetet, s a hozzájuk tartozó hash értéket. A bemenet párokra kapott hash értékek azonosságához rendeljen 0/1 értékű indikátort. Adja össze ezen indikátor valószínűségi változókat, számítsa ki a várható értéket, s állítsa be 1 közeli értékre a várható ütközésszámot a t megfelelő megválasztásával.)

4.5. Feladat. Tekintse a

$$h(x) = \begin{cases} 1 & |x| \text{ ha } |x| = n \\ 0 & |g(x)| \text{ egyébként} \end{cases}$$

függvényt, ahol $g : \{0,1\}^* \rightarrow \{0,1\}^n$ CRHF tulajdonságú. Igazolja $h(x)$ felhasználásával, hogy egy függvény CRHF tulajdonsága még nem garantálja az OWHF tulajdonságot!

4.6. Feladat. Tekintse az alábbi iterációs függvényű lenyomatkészítő eljárását:

$$H_i = f(M_i, H_{i-1}) = E_{M_i}(H_{i-1}). \quad (4.4)$$

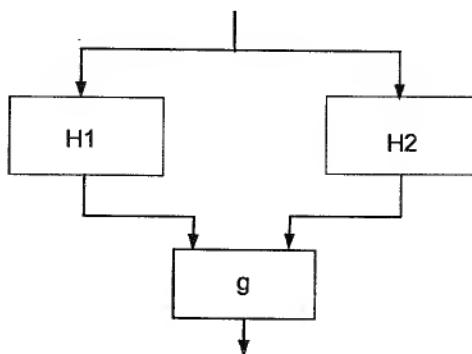
Mutassa meg, hogy ezen lenyomatkészítő nem OWHF!

(Segítség: Alkalmazza a születésnapi paradoxonon alapuló középen találkozás támadást, kettévágva az üzenetblokk-sorozatot; egyik oldalról H_0 -tól előre iterálva, a másik oldaról a H hash értéktől visszafelé iterálva.)

4.7. Feladat. n bites hash lenyomatot képező iteratív hash függvényből $n' = 2n$ bites lenyomatot képzőt szeretnénk előállítani olyan módon, hogy a két utolsó iteráció eredményét konkatenáljuk, azaz $[H_{t-1}, H_t]$ $2n$ bites lenyomatot használunk az n bites H_t helyett. Hatásos-e ezen dimenzióövelő megerősítési ötlet a születésnapi támadás sikérének csökkentésére?

4.8. Feladat. Legyenek adottak a $H1$ és $H2$ ütközés-ellenálló hash függvények (CRHF).

1. Bizonyítsa be, hogy a konkatenáció műveettel kapott $H' = [H1, H2]$ hash függvény is CRHF! Tehát ez a dimenzióövelés hatásos.
2. Egy $g : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$ segédfüggvénytel H'' = g(H1, H2) dimenzió visszaszűkítést végrehajtva egy az eredeti $H1$ és $H2$ komponens függvények bemenet-kimenet dimenziójának megfelelő hash függvényre jutunk (4.3. ábra). Milyen előnyei lehetnek egy ilyen konstrukciónak?



4.3. ábra. Hash függvények kaszkádosítása

4.9. Feladat. S kriptográfiai ellenőrzőösszeget egy H iterációs hash függvény, egy k szimmetrikus kulcs felhasználásával készítünk egy M üzenetre az alábbi módokon:

1. $S = H([k, M])$, azaz az üzenetet egy kulcsblokk prefix-szel bővítjük,
2. $S = H([M, k])$, azaz az üzenetet egy kulcsblokk suffix-szel bővítjük,
3. $S = H([k, M, k])$, azaz a kulcsblokkot alkalmazzuk mind prefixként, mind suffixként.

Biztonságosak-e ezen konstrukciók?

4.10. Feladat. Ha mind a titkosság, mind pedig az adatintegritás biztosítása szükséges, egy lehetséges módszer a rejtjelezés és a lenyomatkészítés együttes alkalmazása, az

$$E_k(m | H(m)) \quad (4.5)$$

kódolás, azaz az üzenetet lenyomattal meghosszabbítjuk, majd rejtjelezünk.

1. Képezzük a lenyomatot olyan módon, hogy az üzenetblokkokat XOR összeadással összeadjuk, továbbá a rejtjelezés legyen CBC módú (a CBC mód leírását lásd az 5. fejezetben). Helyes-e a védelemnek ez a módja?
2. Segít-e az előző problémán az, ha a lenyomatképzést a jól ismert lineáris CRC-vel (Cyclic Redundancy Code) végezzük?

II.

Kriptográfiai alapprotokollok

5.

Blokkrejtjelezési módok

Egy blokkrejtjelező n bit hosszú nyílt szöveg blokkokat kódol n bit hosszú rejtejes blokkokba. n tipikus értéke 64 vagy 128. Sok alkalmazásban ennél jóval hosszabb (vagy rövidebb) üzeneteket szeretnénk rejtjelezni. Ezért szükségünk van olyan eljárásokra, melyek lehetővé teszik tetszőleges hosszúságú üzenetek kódolását egy n bites blokkrejtjelező segítségével. Ezeket az eljárásokat blokkrejtjelezési módoknak nevezzük.

Hosszú üzenetek rejtjelezésére az első gondolatunk az lehet, hogy az üzenetet n bit hosszúságú blokkokra osztjuk, ezen blokkokat egymástól függetlenül rejtjelezzük, majd az így nyert rejtejes blokkokat egymás után fűzzük és így kapjuk a rejtejes üzenetet. Az 5.1. alfejezetben részletesen ismertetjük ezt az eljárást, melyet ECB (Electronic Codebook) módnak neveznek. Látni fogjuk, hogy az ECB mód számos nemkívánatos tulajdonsággal rendelkezik, ezért nem tanácsos hosszabb, több blokkból álló üzenetek rejtjelezésére használni. A gyakorlatban ezt a módot egy blokkból álló üzenetek (pl. csatolatkulcsok) rejtjelezésénél alkalmazzuk.

Több blokkból álló üzenetek rejtjelezésére gyakran a CBC (Cipher Block Chaining) módot használjuk. Az ECB módozathoz hasonlóan a nyílt üzenetet ebben a módban is n bit hosszúságú blokkokra osztjuk, de az ECB móddal ellentétben az így nyert blokkokat nem egymástól függetlenül rejtjelezzük. CBC módban az i -edik nyílt blokkot bitenként XOR-oljuk az $(i-1)$ -edik rejtejes blokkal, majd az eredményt rejtjelezve kapjuk az i -edik rejtejes blokkot. Így az i -edik rejtejes blokk értéke nemcsak az i -edik nyílt blokktól függ, hanem az előző rejtejes blokktól is, és azon keresztül az összes előző

nyílt blokktól. Ennek a láncolási technikának számos előnye van, melyekről bővebben az 5.2. alfejezetben szólunk.

Előfordulhat, hogy a nyílt üzenet hossza nem többszöröse a blokkrejtjelező n blokkhosszának, ami azt jelenti, hogy az utolsó nyílt blokk hossza kisebb, mint n . Ebben az esetben CBC (és ECB) módban az utolsó nyílt blokkot ki kell egészíteni. Ezt az eljárást kitöltésnek (padding) nevezük. A kitöltésnek természetesen olyannak kell lennie, hogy a rejtjeles üzenet dekódolása során egyértelműen felismerhető és eltávolítható legyen. Erre számos megoldás létezik, ezek közül néhányat az 5.2.2. alfejezetben ismertetünk.

Interaktív alkalmazásokban (pl. távoli terminál) gyakran a blokkrejtjelező n blokkhosszánál sokkal rövidebb üzeneteket (biteket vagy pár bitból álló karaktereket) kell továbbítani, mégpedig olyan ütemben, ahogy az üzenetek keletkeznek. Ez azt jelenti, hogy a küldő nem tudja megvárni, hogy egy teljes blokkhosszt kitevő mennyiségű üzenet rendelkezésre álljon. Egyik megoldás, hogy minden rövid üzenetet kitöltsük, majd a kitöltött üzenetet ECB vagy CBC¹ módban rejtjezzük. Ez azonban a sávszélesség felesleges pazarlásához vezet, hiszen az átküldött hasznos bitek száma csak töredéke a ténylegesen átküldött bitek számának. Ebben az esetben tehát a CBC mód nem elég hatékony, és helyette célravezetőbb a CFB (Cipher Feedback) az OFB (Output Feedback), vagy a CTR (Counter) módok valamelyikének alkalmazása. Ezen módok közös jellemzője, hogy a blokkrejtjelezőt kulcsfolyamatos rejtjelezővé alakítják. Mindegyik módnak megvan a saját előnye és hátránya, ezekről bővebben az 5.3., az 5.4. és az 5.5. alfejezetekben szólunk.

Léteznek még más blokkrejtjelezési módok is, de a fent említett öt mód (ECB, CBC, CFB, OFB és CTR) lefedi a gyakorlati alkalmazásokban előforduló problémák túlnyomó részét. Ezért ezeket a blokkrejtjelezési módokat szabványosították, és ajánlott ezen szabványos módok alkalmazása minden esetben, amikor ez lehetséges.

5.1. Az ECB mód

ECB módban történő rejtjelezéskor, a nyílt üzenetet először kitöltjük, hogy mérete a blokkrejtjelező n blokkhosszának többszöröse legyen. Jelöljük a kitöltött nyílt üzenetet P -vel. Ezután P -t n bites blokkokra osztjuk. Jelölje P_1, P_2, \dots, P_N az így nyert nyílt blokkokat. Az i -edik rejtjeles blokkot, C_i -t, a

¹ CBC mód esetén a kitöltött üzenetet XOR-oljuk az előző üzenethez tartozó rejtjeles blokkal, majd az eredményt rejtjelezve kapjuk az aktuális üzenethez tartozó rejtjeles blokkot.

következőképpen kapjuk:

$$C_i = E_K(P_i) \quad i = 1, 2, \dots, N, \quad (5.1)$$

ahol E_K jelöli a blokkrejtjelező kódoló függvényét a K kulccsal paramétereze. A dekódolás hasonlóan egyszerű:

$$P_i = E_K^{-1}(C_i) \quad i = 1, 2, \dots, N, \quad (5.2)$$

ahol E_K^{-1} jelöli a blokkrejtjelező dekódoló függvényét a K kulccsal paramétereze (azaz az E_K függvény inverzét).

Biztonság. A fenti kifejezésekben azonnal látszik, hogy azonos nyílt blokkhoz minden blokk ugyanazzal a K kulccsal lett kódolva. Ez azt jelenti, hogy ha adott két különböző P és P' nyílt szöveghez tartozó C és C' rejtjeles szöveg, akkor egy támadó a K kulcs ismerete nélkül is azonosítani tudja P és P' egyforma blokkjait C és C' egyforma blokkjainak azonosításával. A legtöbb gyakorlati alkalmazásban gyakran előfordul, hogy különböző nyílt üzenetek azonos blokkokat tartalmaznak. Ez azért van, mert az üzenetek általában szabályos szerkezetűek, és sok redundanciát tartalmaznak (pl. gyakran ismétlődő fejlec mezőket, nullák vagy szóköz karakterek hosszú láncát, stb.). Az ECB mód tehát nem rejti megfelelően a nyílt szövegen található mintákat, megkönyítve ezzel a támadó dolgát.

Ha a támadó valamelyen módon nagy számú nyílt blokk – rejtjeles blokk párhoz jut, akkor ezekből egy szótárat (vagy kódkönyvet) építhet. Ezt a szótárat aztán később megfigyelt rejtjeles szövegek részleges dekódolásához használhatja a K kulcs ismerete nélkül. Ehhez olyan blokkokat kell keresnie a rejtjeles szövegen, amelyekhez már létezik bejegyzés a szótárban. Ezeket a blokkokat a szótár segítségével a támadó dekódolni tudja.

További probléma, hogy egy rejtjeles üzenet blokkjai egymással tetszőleges módon felcserélhetők, törlhetők vagy más rejtjeles üzenetekből kimásolt blokkokkal helyettesíthetők. Az ECB mód nem teszi lehetővé az ilyen jellegű blokkcserék, törlések és helyettesítések megbízható detektálását, mivel a rejtjeles blokkok nem függnek a szomszédos blokkoktól.

Bár a fent említett problémák a nyílt üzenet rejtjelezés előtti tömörítésével és integritásvédelmi mechanizmusok (pl. kriptográfiai ellenőrzőösszeg) alkalmazásával enyhíthetők, az ECB mód mégsem javasolt egynél több blokkból álló nyílt üzenet rejtjelezésére. Helyette a következő alfejezetben ismertetett CBC mód használata ajánlott.

Hatékonyság. Az ECB mód előnye, hogy a kódolás és dekódolás sebessége megegyezik a blokkrejtjelező sebességével, valamint az, hogy a kódolás és a dekódolás is párhuzamosítható. Továbbá, az ECB mód lehetővé teszi a nyílt blokkokhoz történő véletlen hozzáférést a rejtjeles szövegben. Ez azt jelenti, hogy a rejtjeles szöveg tetszőleges sorszámu blokkját dekódolni tudjuk a többi blokk dekódolása nélkül, sőt a dekódolt blokk módosítása, majd újrakódolása sem érinti a többi blokkot. Ez igen előnyös tulajdonság például adatbázisfájlok rejtjelezésénél, ami a rekordok hatékony lekérdezését és módosítását teszi lehetővé.

Hibaterjedési tulajdonságok. A biztonsággal és hatékonysággal kapcsolatos tulajdonságok mellett meg szokták még említeni egy adott blokkrejtjelezési mód ún. hibaterjedési tulajdonságait. Egészen pontosan arra vagyunk kíván- csiak, hogy a rejtjeles üzenet átvitele során keletkezett egy bites hibának milyen hatása van a dekódolt nyílt üzenetre. Az egy bites hiba lehet egy bit értékének változása, valamint egy bit törlése vagy beszúrása a rejtjeles üzenetbe. Az utóbbit kerethibának (framing error) nevezzük.

ECB mód esetén a rejtjeles szöveg egy bitjének megváltozása csak a bittet tartalmazó rejtjeles blokkhoz tartozó nyílt blokkra van hatással, mégpedig oly módon, hogy az adott nyílt blokk teljes egészében használhatatlan pénzfeldobás sorozattá válik (a blokk minden bitje $1/2$ valószínűséggel lesz helyes dekódolás után). Ezzel szemben egy bit beszúrása vagy törlése a hiba helye után található blokkhatárok elcsúszását eredményezi (innen a kerethiba elnevezés), és így módon hatással van a beszúrást, illetve törlést tartalmazó rejtjeles blokkhoz tartozó nyílt blokkra és az összes azt követő nyílt blokkra is. Ezek a blokkok használhatatlan pénzfeldobás sorozattá válnak.

5.2. A CBC mód

A CBC mód működését az 5.1 és az 5.2. ábra szemlélteti. Az 5.1. ábra a kódolást, az 5.2. ábra pedig a dekódolást ábrázolja. Az ECB módhoz hasonlóan a kódoláshoz a kitöltött nyílt üzenetet n bit hosszúságú blokkokra osztjuk. Ezután az i -edik rejtjeles blokkot úgy állítjuk elő, hogy az i -edik nyílt blokkot XOR-oljuk az $(i-1)$ -edik rejtjeles blokkal, majd az eredményt kódoljuk a blokkrejtjelezővel. A dekódolásnál az i -edik rejtjeles blokkot dekódoljuk, majd az eredményt XOR-oljuk az $(i-1)$ -edik rejtjeles blokkal, és így kapjuk az i -edik nyílt blokkot.

Jelölje ismét P_1, P_2, \dots, P_N a nyílt blokkokat, és C_1, C_2, \dots, C_N a rejtjeles blokkokat. A kódolást tehát a következőképpen írhatjuk le formálisan:

$$C_1 = E_K(P_1 \oplus IV) \quad (5.3)$$

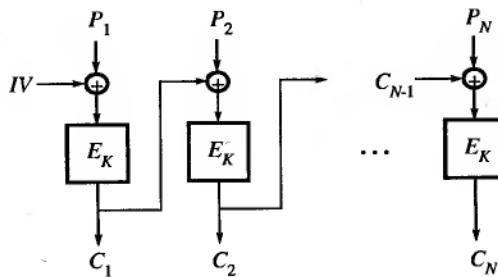
$$C_i = E_K(P_i \oplus C_{i-1}) \quad i = 2, 3, \dots, N, \quad (5.4)$$

ahol E_K jelöli a blokkrejtjelező K kulccsal paraméterezett kódoló függvényét, \oplus a bitenkénti XOR művelet, és IV egy kezdeti változó (initializing variable). Az IV kezdeti változó az előző rejtjeles blokk szerepét tölti be az első nyílt blokk kódolásánál, ahol még nem áll rendelkezésre egy valódi előző rejtjeles blokk. A dekódolás képlete a következő:

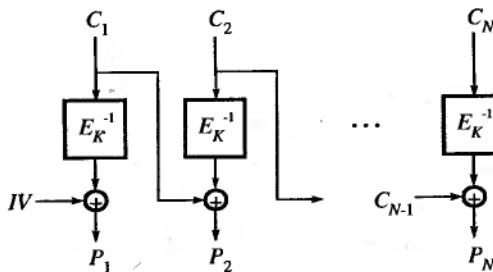
$$P_1 = E_K^{-1}(C_1) \oplus IV \quad (5.5)$$

$$P_i = E_K^{-1}(C_i) \oplus C_{i-1} \quad i = 2, 3, \dots, N, \quad (5.6)$$

ahol E_K^{-1} jelöli a blokkrejtjelező K kulccsal paraméterezett dekódoló függvényét.



5.1. ábra. Kódolás CBC módban



5.2. ábra. Dekódolás CBC módban

Mint látható, az első blokk dekódolásánál szükség van IV -re, így gondoskodni kell ennek a vevőhöz történő eljuttatásáról. Ez megoldható úgy, hogy IV -t ECB módban rejtjelezzük, majd a kódolt IV -t az átküldött rejtjeles üzenet elő csatoljuk. IV rejtjelezéséhez ugyanazt a kulcsot használjuk, mint az üzenet rejtjelezéséhez. A vevő először IV -t dekódolja, majd ennek ismeretében a rejtjeles üzenetből (5.5) és (5.6) szerint visszaállítja a nyílt üzenetet.

Valójában IV titkossága nem követelmény, ügyelni kell azonban IV integritásának védelmére. Erre azért van szükség, mert (5.5) szerint IV értéke közvetlen hatással van az első visszaállított nyílt blokkra. IV bitjeinek módosításával tehát egy támadó az első nyílt blokk adott bitjeit manipulálni (invertálni) tudja. A rejtjelezés alkalmazása ezt megakadályozza, mert a rejtjelezett IV módosítása a támadó által nem predikálható változásokat eredményez a dekódolt IV -ben, és így a visszaállított első nyílt blokkban is.

5.2.1. A CBC mód tulajdonságai

Biztonság. A CBC mód egyik legfontosabb tulajdonsága, hogy a C_i rejtjeles blokk értéke nemcsak a P_i nyílt blokktól függ, hanem az azt megelőző $P_{i-1}, P_{i-2}, \dots, P_1$ blokkoktól és IV -től is². Ez (5.3) és (5.4) felhasználásával a következő módon adódik:

$$\begin{aligned} C_i &= E_K(P_i \oplus C_{i-1}) \\ &= E_K(P_i \oplus E_K(P_{i-1} \oplus C_{i-2})) \\ &\quad \dots \\ &= E_K(P_i \oplus E_K(P_{i-1} \oplus \dots E_K(P_1 \oplus IV) \dots)). \end{aligned} \tag{5.7}$$

Ennek a tuljadonságának több kedvező hatása is van. Egyrészt azonos nyílt blokkokhoz nagy valószínűséggel különböző rejtjeles blokkok tartoznak. Ez akkor is igaz, ha az azonos nyílt blokkok egy nyílt üzenet részei, és akkor is, ha két különböző nyílt üzenethez tartoznak. Pontosabban, ha két különböző P és P' nyílt üzenetben valamely P_i és P'_j blokkok azonosak, akkor az ezen blokkokhoz tartozó C_i és C'_j rejtjeles blokkok nagy valószínűséggel csak akkor lesznek azonosak, ha $i = j$ és minden $k = 1, 2, \dots, i-1$ esetén $P_k = P'_k$, valamint $IV = IV'$ (azaz P és P' minden i -nél kisebb sorszámu blokkja, valamint a két kezdeti változó értéke is megegyezik). Ebből az is látszik, hogy ha teljesen azonos nyílt üzeneteket különböző IV -t alkalmazva rejtjelezünk,

² Meg kell azonban jegyezni, hogy ez a függés csak az előző rejtjeles blokkon, C_{i-1} -en keresztül valósul meg.

akkor különböző rejtjeles üzenetekhez jutunk. *IV* üzenetenkénti változtatása megoldható *IV* véletlen módon történő generálásával, vagy egy sorozatszám *IV*-ként történő használatával.

A CBC mód egy másik kedvező tulajdonsága, hogy a C_{i-1} blokk P_i -hez történő XOR-olása növeli a blokkrejtjelező bemenetének entrópiáját, és ezzel nehezebbé teszi a kriptanalítikus támadásokat. Az entrópia-növekedést az okozza, hogy a blokkrejtjelező tulajdonságaiból adódóan a rejtjeles blokkok a támadó számára lényegében pénzfeldobás sorozatok.

Az ECB móddal ellentétben, a CBC mód korlátozott mértékben lehetősséget nyújt a rejtjeles blokkok felcserélésének, törlésének és beszúrásának detektálására. Ez azért van, mert egy C_i rejtjeles blokkot csak akkor lehet helyesen dekódolni, ha azt a helyes C_{i-1} blokk előzi meg. Két blokk, C_i és C_j átvitel során történő felcserélésekor azonban C_i dekódolásánál C_{j-1} -et használja a vevő. Az így eredményül kapott $E_K^{-1}(C_i) \oplus C_{j-1}$ egy pénzfeldobás sorozat lesz, amit a vevő könnyen kiszűrhet mint egy várt struktúrától való durva eltérést. Meg kell azonban jegyezni, hogy a nyílt üzenet maga is tartalmazhat véletlen blokkokat (pl. egy véletlen módon generált tranzakció azonosítót), ezért a véletlen blokkok detektálására alapozott integritás-védelem nem minden megbízható. Általában is igaz, hogy ha az alkalmazás megkívánja az üzenetek integráltságának védelmét, akkor erre a célra speciális integrálás-védelmi mechanizmusokat (lásd 6. fejezet) ajánlott használni, és nem tanácsos kizárálag a CBC mód által nyújtotta lehetőségekre hagyatkozni.

Ehhez kapcsolódóan említiük meg az úgynevezett „kivág-és-beszúr” (cut-and-paste) támadást CBC módban rejtjelezett üzenetek ellen. Legyen $P = P_1|P_2|\dots|P_N$ és $P' = P'_1|P'_2|\dots|P'_M$ két nyílt üzenet. Mindkettőt ugyanazzal a K kulccsal (de nem feltétlenül ugyanazzal a kezdeti változóval) CBC módban rejtjelezve kapjuk a $C = C_1|C_2|\dots|C_N$ és a $C' = C'_1|C'_2|\dots|C'_M$ rejtjeles üzeneteket. Egy támadó C' -ból kivágja a j -től k -ig ($1 \leq j \leq k \leq M$) terjedő blokkokat, és a kivágott blokksorozatot beszúrja C -be az i -edik és $(i+1)$ -edik blokkok közé. Az eredményül kapott $C_1|\dots|C_i|C'_j|\dots|C'_k|C_{i+1}|\dots|C_N$ üzenetet dekódolva a vevő a $P_1|\dots|P_i|*|P'_{j+1}|\dots|P'_k|*|P_{i+2}|\dots|P_N$ nyílt üzenetet állítja vissza, ahol a $*$ pénzfeldobás sorozatból álló véletlen blokkokat jelöl. Ha az adott pozíciókban a véletlen blokkok megengedettek, akkor a vevő elfogadja a támadó által fabrikált üzenetet.

A CBC mód egy ismert gyengesége a következő: Ha két rejtjeles blokk, C_i és C'_j megegyezik, akkor (5.4) használatával kapjuk, hogy

$$C_{i-1} \oplus C'_{j-1} = P_i \oplus P'_j. \quad (5.8)$$

Mivel a támadó C_{i-1} -et és C'_{j-1} -et ismeri, ezért (5.8) alapján két nyílt blokk XOR összegére vonatkozóan jut információhoz. Innen a nyílt üzenetek redundanciáját felhasználva, a támadó sikkerrel kísérelheti meg P_i és P'_j megfejtését. A gyakorlatban ennek a támadásnak az szab korlátot, hogy a rejtjeles blokkok egyenletes eloszlásúak, ezért annak a valószínűsége, hogy kettő megegyezik, a születésnapi paradoxon alapján becsülhető. A támadás akkor hatékony, ha a megfigyelt rejtjeles blokkok száma $2^{\frac{n}{2}}$ nagyságrendjébe esik. A tipikus $n = 64$ esetben ez több gigabajt adat megfigyelését igényli.

Hatókonyság. A CBC módban történő kódolás és dekódolás sebessége lényegében megegyezik a blokkrejtjelező sebességével, mert a blokkonként végrehajtott XOR művelet által okozott késleltetés elhanyagolható a blokk-rejtjelezéshez szükséges időhöz képest. A CBC mód hátránya, hogy a kódolás nem párhuzamosítható. Egy másik hátrány, hogy egy nyílt blokk módosítása az összes ō követő blokk újrakódolását igényli. Ezzel szemben a dekódolás párhuzamosítható és egy nyílt blokkhoz történő módosítás nélküli hozzáférés (olvasás) nem érinti a többi blokkot. Az ECB móddal ellentétben tehát a nyílt blokokhoz történő véletlen hozzáférés a rejtjeles szövegben csak olvasás esetén hatékony.

Hibaterjesí tulajdonságok. Ha a rejtjeles üzenet átvitele során az i -edik rejtjeles blokk j -edik bitje megváltozik, akkor ez hatással lesz az i -edik és az $(i+1)$ -edik nyílt blokk visszaállítására. Egészen pontosan az i -edik visszaállított nyílt blokk pénzfeldobás sorozat lesz, míg az $(i+1)$ -edik visszaállított nyílt blokkban csak a j -edik bit lesz hibás. Az $(i+2)$ -edik nyílt blokkra a hiba hatása már nem terjed ki. Az egy bithibából való felépülés képességét önszinkronizációnak is nevezzük. A CBC mód tehát ebben az értelemben önszinkronizáló.

A CBC mód azon tulajdonságát, hogy az i -edik rejtjeles blokk j -edik bitjének megváltoztatása az $(i+1)$ -edik visszaállított nyílt blokkban csak a j -edik bit megváltozását eredményezi, egy támadó bizonyos esetekben sikeresen használhatja ki egy adott nyílt blokk bitjeinek manipulálására. Ezért még egyszer kihangsúlyozzuk a kriptográfiai integritásvédő mechanizmusok alkalmazásának szükségesességét.

Az ECB módban hasonlóan egy bit elvesztése vagy beszúrása a rejtjeles üzenetből, illetve üzenetbe a blokkhatárok elcsúszását eredményezi, és a hiba helyétől kezdve minden visszaállított nyílt blokk használhatatlanná válik. Bitvesztésből és -beszúrásból tehát nem áll helyre a rendszer.

5.2.2. Üzenetek kitöltése (padding)

Előfordulhat, hogy a rejtjelezni kívánt nyílt üzenet bitekben mért hossza nem többszöröse a rejtjelező n blokkhosszának. Ekkor az üzenetet ki kell tölteni néhány további bit hozzáadásával oly módon, hogy a kitöltéssel együtt az üzenet hossza n többszöröse legyen. Ez sok esetben csak az utolsó, csonka blokk kitöltését jelenti, de néha egy teljes blokk hozzáadásával is járhat. Sőt, vannak olyan kitöltési sémák is, melyek több, véletlenszámú blokkot is hozzáadnak az üzenethez kitöltésként, ezzel próbálva az üzenet valódi hosszát elrejteni egy forgalmat analizáló támadó elől.

Akármennyi bitet is alkalmazunk kitöltésként, egy dolgot minden esetben biztosítani kell: a vevőnek képesnek kell lennie a kitöltésként alkalmazott bitek azonosítására, majd azok eltávolításával az eredeti nyílt üzenet visszaállítására. Ennek elérése érdekében sok kitöltő sémában a kitöltés utolsó bájtja a kitöltés hosszára vonatkozó információt tartalmaz. Ezen sémák használata esetén minden üzenetnek tartalmaznia kell kitöltést, még azoknak is, amelyek hossza n többszöröse. Ha ugyanis megengednénk olyan üzeneteket, melyek nem tartalmaznak kitöltést, akkor egy adott üzenet vétele esetén a vevő nem tudná egyértelműen megállapítani, hogy az üzenet tartalmaz-e kitöltést vagy sem. Ha viszont megköveteljük, hogy minden üzenet tartalmazzon kitöltést, akkor ilyen probléma nem lép fel.

A továbbiakban néhány a gyakorlatban előforduló kitöltési sémát ismeretünk. Feltesszük, hogy minden üzenet 8 bites bájtokból áll, és $n = 8b$, azaz a rejtjelező blokkhossza b bájt.

5.1. Algoritmus. Legyen a rejtjelezni kívánt utolsó nyílt blokk hossza ℓ bájt ($1 \leq \ell \leq b$). Ha $\ell < b$, akkor az utolsó blokkot $b - \ell$ darab $b - \ell - 1$ értékű bájttal egészítjük ki. Ha $\ell = b$, akkor egy teljes blokkot hozzáadunk az üzenethez, és a hozzáadott blokk minden bájtjának értékét $b - 1$ -re állítjuk.

Az 5.1. algoritmusnak megfelelő helyes kitöltések például a következő bájtsorozatok: 00 vagy 01|01 vagy 02|02|02 stb. Hibás kitöltésnek számít például a 01|02|02 bájtsorozat. Helyes kitöltés esetén tehát az utolsó dekódolt blokk utolsó bájtja a további (az utolsó bájtot megelőző) kitöltő bájtok számát tartalmazza. A kitöltés ellenőrzéséhez a vevő a következőt teszi: ha az utolsó bájt k , akkor ellenőrzi, hogy az utolsó bájtot megelőző k bájt értéke k . Ha a kitöltés helyes, akkor az utolsó $k + 1$ bájt eltávolításával állítható vissza az eredeti üzenet.

A fenti séma változatai, mikor a kitöltő bájtok mindegyike 00 értékű, vagy mikor a kitöltés a $00|01|\dots|b-\ell-1$ bájtsorozat. Az utolsó bájt, csakúgy mint az alapváltozatban, a kitöltés hosszát tartalmazza.

Az 5.1. algoritmus által definiált kitöltő séma minden megnöveli az üzenet hosszát. Néhány alkalmazásban ez nincs megengedve, és arra van szükség, hogy a rejtjeles üzenet hossza pontosan megegyezzen a nyílt üzenet hosszával (pl. pontosan ugyanarra a memóriaterületre szeretnénk visszaírni, ahol előzőleg a nyílt üzenetet tároltuk). Ekkor alkalmazható a következő algoritmus:

5.2. Algoritmus. Legyen a rejtjelezni kívánt utolsó nyílt blokk hossza ℓ bit ($1 \leq \ell \leq n$). Ha $\ell = n$ (az utolsó blokk egy teljes blokk), akkor nem csinálunk semmit. Ha $\ell < n$ (az utolsó blokk egy csonka blokk), akkor az utolsó teljes blokkhoz tartozó C_{N-1} rejtjeles blokkot újra kódoljuk, majd az eredményül kapott $E_K(C_{N-1})$ blokk első ℓ bitjét XOR-oljuk az utolsó, csonka nyílt blokk ℓ bitjével, és így kapjuk az utolsó, csonka rejtjeles blokkot.

Az 5.2. algoritmus egy gyengesége, hogy egy támadó manipulálni (invertálni) tudja az utolsó, csonka, visszaállított nyílt blokk bitjeit az utolsó, csonka rejtjeles blokk bitjeinek megváltoztatásával. A következő algoritmust használva ez a veszély nem áll fenn, mert a nyílt üzenet minden bitje keresztül megy az E_K rejtjelező függvényen.

5.3. Algoritmus. Jelöljük az utolsó, teljes nyílt blokkot P_{N-1} -gyel, valamint az utolsó, csonka nyílt blokkot p -vel. Legyen p hossza ℓ bit, ahol $\ell < n$. Az utolsó, teljes rejtjeles blokkot, C_{N-1} -et és az utolsó, csonka rejtjeles blokkot, c -t, a következő módon állítjuk elő:

$$V = E_K(P_{N-1} \oplus C_{N-2}), \quad (5.9)$$

$$C_{N-1} = E_K((p|z) \oplus V), \quad (5.10)$$

$$c = V \text{ első } \ell \text{ bitje}, \quad (5.11)$$

ahol z egy $n - \ell$ hosszú csupa 0 bitet tartalmazó bitsorozat. A dekódolás a következő módon történik:

$$W = E_K^{-1}(C_{N-1}) \oplus (c|z), \quad (5.12)$$

$$w = W \text{ utolsó } n - \ell \text{ bitje}, \quad (5.13)$$

$$p = W \text{ első } \ell \text{ bitje}, \quad (5.14)$$

$$P_{N-1} = E_K^{-1}(c|w) \oplus C_{N-2}. \quad (5.15)$$

Most belátjuk, hogy a dekódoló algoritmus helyesen működik. Jelöljük V utolsó $n - \ell$ bitjét v -vel. Ekkor (5.11) alapján $V = c|v$. (5.10) definícióját (5.12) kifejezésbe helyettesítve kapjuk, hogy

$$\begin{aligned} W &= E_K^{-1}(C_{N-1}) \oplus (c|z) \\ &= (p|z) \oplus V \oplus (c|z) \\ &= (p|z) \oplus (c|v) \oplus (c|z) \\ &= (p \oplus c \oplus c)|(z \oplus v \oplus z) \\ &= p|v. \end{aligned}$$

Innen következik, hogy W első ℓ bitje valóban p , továbbá $w = v$. Ez utóbbit felhasználva kapjuk, hogy

$$\begin{aligned} E_K^{-1}(c|w) \oplus C_{N-2} &= E_K^{-1}(c|v) \oplus C_{N-2} \\ &= E_K^{-1}(V) \oplus C_{N-2} \\ &= P_{N-1} \oplus C_{N-2} \oplus C_{N-2} \\ &= P_{N-1}. \end{aligned}$$

5.2.3. Vaudenay támadása

Tegyük fel, hogy valamely alkalmazás CBC módú rejtjelezést és az 5.1. algoritmusban leírt kitöltési sémák valamelyikét használja. Mikor a vevő kap egy rejtjeles üzenetet, akkor dekódolja azt, majd ellenőrzi, hogy a kitöltés helyes-e. Az ellenőrzés eredményétől függően a vevő folytatja az üzenet további feldolgozását, vagy eldobja az üzenetet. Sok esetben a vevő viselkedése megfigyelhető egy külső támadó számára is. A támadó ezt kihasználhatja rejtjeles üzenetek megfejtésére a kulcs ismerete nélkül. A módszert Serge Vaudenay publikálta 2002-ben.

5.1. Példa. A web tranzakciók védelmére gyakran alkalmazott TLS 1.0 protokoll CBC módú rejtjelezést és az 5.1. algoritmusban leírt kitöltési sémát használja. Tegyük fel, hogy egy támadó egy véletlen módon generált üzenetet küld egy TLS szervernek. A szerver megpróbálja az üzenetet CBC módban dekódolni. Ha a dekódolt üzenetben a kitöltés hibás, akkor a szerver a DECRYPTION_FAILED hibaüzenettel válaszol. Ha a kitöltés helyes, a szerver folytatja az üzenet feldolgozását a kitöltés eltávolításával és az üzenet-hitelesítő kód (MAC) ellenőrzésével. Az üzenet véletlen volta miatt a MAC

majdnem biztosan hibás lesz, és a szerver a BAD_RECORD_MAC hibaüzenettel válaszol. A szerver hibaüzeneteiből tehát a támadó következtetni tud arra, hogy az adott véletlen üzenet dekódolása helyes kitöltést eredményez, vagy sem.



A vevő (pl. egy TLS szerver) viselkedését úgy modellezük, hogy felte tesszük, hogy a támadó rendelkezésére áll egy orákulum, amely minden neki küldött üzenetet dekódolni próbál egy adott K kulccsal CBC módban, majd egy 1 bites választ ad a támadónak, ahol a válasz értéke attól függ, hogy a dekódolt üzenetben a kitöltés helyes volt-e, vagy sem. A támadás során a támadó többször orákulumhívással annyi információt gyűjt, ami lehetővé teszi a célüzenet teljes megfejtését.

Megjegyezzük, hogy a támadás nemcsak a CBC mód tulajdonságait használja ki, hanem a rejtjelezés előtt alkalmazott kitöltés tulajdonságait is. Így a támadás nem minden típusú kitöltés esetén használható. Ez azonban nem csökkenti a jelentőségét, már csak azért sem, mert számos, a gyakorlatban elterjedt kitöltés esetén (pl. az SSL/TLS vagy az IPSec protokollokban használt kitöltési sémák esetén) nagyszerűen működik. Mi az 5.1. algoritmusban leírt kitöltési sémát feltételezve magyarázzuk el a támadást.

A továbbiakban feltesszük, hogy a blokkrejtjelező blokkhossza 8 bájt ($n = 64$ bit). Megjegyezzük azonban, hogy ez a feltevés nem korlátozó abban az értelemben, hogy a Vaudenay-féle támadás tetszőleges blokkhossz esetén kivitelezhető.

Az utolsó bájt(ok) megfejtése. Adott egy $Y = E_K(X)$ rejtjeles blokk, ahol $Y = y_1|y_2|\dots|y_8$ és $X = x_1|x_2|\dots|x_8$, mindenkor 8 hosszú bájtsorozat. Szeretnénk X utolsó bájträgt, x_8 -at megfejteni a K kulcs ismerete nélkül. Ezt a következőképpen tehetjük meg: Választunk egy véletlen generált bájtokból álló $R = r_1|r_2|\dots|r_8$ blokkot, majd elküldjük az $R|Y$ két blokkból álló üzenetet az orákulumnak. Az orákulum ezt egy CBC módban rejtjelezett üzenetnek fogja fel, és megpróbálja dekódolni. (5.6) alapján a visszaállított nyílt üzenet utolsó blokkja $X \oplus R = x_1 \oplus r_1|x_2 \oplus r_2|\dots|x_8 \oplus r_8$ lesz. Az orákulum ellenőri, hogy az $X \oplus R$ blokk helyes kitöltést tartalmaz-e, és az eredménynek megfelelő egy bites választ ad. Ha a kitöltés hibás, akkor r_8 -at megváltoztatjuk, és újra próbálkozunk. Ezt addig ismétljük, amíg az orákulum végül helyes kitöltést jelez. Ekkor tudjuk, hogy $X \oplus R$ végződése csak 00 vagy 01|01 vagy 02|02|02 ... lehet. Mivel az R véletlen választott bájtokból áll, ezért ezen lehetséges esetek közül a 00 végződés a legvalószínűbb (a

$k|k|\dots|k$ végződés valószínűsége $2^{-8(k+1)}$). Mikor tehát az orákulum helyes kitöltést jelez, tudjuk hogy $x_8 \oplus r_8$ nagy valószínűséggel 00, azaz $x_8 = r_8$.

Természetesen előfordulhat, hogy a kisebb valószínűségű események valamelyike következik be, azaz az $X \oplus R$ utolsó bájtja mégsem 00. Néhány további orákulumhívással azonban könnyen megállapítható a kitöltés k hossza. Tegyük fel tehát, hogy tudjuk, hogy $X \oplus R$ helyes kitöltést tartalmaz, de nem tudjuk pontosan, hogy mi is az a kitöltés. Először r_1 -et módosítjuk, majd az így nyert $R'|Y$ üzenettel újra meghívjuk az orákulumot. Ha az orákulum továbbra is azt jelzi, hogy $X \oplus R'$ helyes kitöltést tartalmaz, akkor az első bájt nem lehet része a kitöltésnek. Ekkor r_2 -t módosítjuk, és újra meghívjuk az orákulumot. Ha még mindig helyes a kitöltés, akkor az r_3 -at módosítjuk, és így tovább. Ezt addig folytatjuk, még végül az orákulum azt jelzi, hogy a kitöltés hibás. Tegyük fel, hogy ez az R j -edik bájtjának módosításánál következik be. Ekkor tudjuk, hogy $X \oplus R$ j -edik bájtja a kitöltés első bájtja, azaz $k = 8 - j$. Innen következik, hogy $x_j \oplus r_j | x_{j+1} \oplus r_{j+1} | \dots | x_8 \oplus r_8 = 8 - j | 8 - j | \dots | 8 - j$, vagyis $x_j | x_{j+1} | \dots | x_8 = (8 - j) \oplus r_j | (8 - j) \oplus r_{j+1} | \dots | (8 - j) \oplus r_8$.

5.2. Példa. Tegyük fel, hogy $X = DE | AD | BE | EF | DE | AD | BE | EF$ és $R = 01 | 23 | 45 | 67 | DD | AE | BD | EC$. Ekkor $X \oplus R = DF | 8E | FB | 88 | 03 | 03 | 03 | 03$ helyes kitöltést tartalmaz. Az 5.1. táblázat azt mutatja, hogy r_1, r_2, \dots egymás utáni megváltoztatásával hogyan deríthető ki a kitöltés hossza. A táblázat j -edik sora a j -edik bájt megváltoztatásával kialakult helyzetet mutatja. Mint látható, az 5. sorban az $X \oplus R'$ blokk által tartalmazott kitöltés hibássá válik. Innen tehát tudjuk, hogy $X \oplus R$ 5. bájtja a kitöltés első bájtja, azaz $X \oplus R$ a $03 | 03 | 03 | 03$ bájtsorozattal végződik. Mivel tudjuk, hogy R a $DD | AE | BD | EC$ bájtsorozattal végződik, ezért megállapíthatjuk, hogy $x_5 | x_6 | x_7 | x_8 = 03 \oplus DD | 03 \oplus AE | 03 \oplus BD | 03 \oplus EC = DE | AD | BE | EF$. ♣

Az utolsó bájt(ok) megfejtését végző eljárást a következőképpen foglalhatjuk össze:

5.4. Algoritmus. Adott egy $Y = E_K(X)$ rejtjeles blokk. Az algoritmus megfejtí X utolsó néhány bájtját (a legtöbb esetben egyet, ha szerencsénk van többet is) a K kulcs ismerete nélkül. Ehhez egy \mathcal{O} orákulumot használ, mely minden neki küldött üzenetet CBC módban dekódol a K kulccsal, majd egy 1 bites választ ad. \mathcal{O} válasza 1, ha a dekódolt üzenetben a kitöltés helyes, és 0, ha a kitöltés hibás. Az algoritmus lépései a következők:

R'	$X \oplus R'$	\mathcal{O}
00 23 45 67 DD AE BD EC	DE 8E FB 88 03 03 03 03	1
00 22 45 67 DD AE BD EC	DE 8F FB 88 03 03 03 03	1
00 22 44 67 DD AE BD EC	DE 8F FA 88 03 03 03 03	1
00 22 44 66 DD AE BD EC	DE 8F FA 89 03 03 03 03	1
00 22 44 66 DC AE BD EC	DE 8F FA 89 02 03 03 03	0

5.1. táblázat. Az \mathcal{O} orákulum által adott válasz az $R'|Y$ hívásra R' különböző értékei esetén. Ha a dekódolt üzenet kitöltése helyes, akkor a válasz 1, ellenkező esetben 0

1. Válasszunk 8 véletlen bájtot, ezeket jelöljük r_1, \dots, r_8 -cal. Legyen $i = 0$.
2. Legyen $R = r_1|r_2|\dots|r_7|(r_8 \oplus i)$.
3. Ha $\mathcal{O}(R|Y) = 0$, akkor legyen $i = i + 1$, és folytassuk a 2. lépéssel.
4. Legyen $r_8 = r_8 \oplus i$.
5. $j = 1$ -től 7-ig ismételjük a következő lépéseket:
 - (a) Legyen $R' = r_1|\dots|r_j \oplus 1| \dots |r_8$.
 - (b) Ha $\mathcal{O}(R'|Y) = 0$, akkor állunk meg. Az algoritmus eredménye:

$$x_j|x_{j+1}|\dots|x_8 = (8-j) \oplus r_j|(8-j) \oplus r_{j+1}| \dots |(8-j) \oplus r_8$$
.
6. Állunk meg. Az algoritmus eredménye: $x_8 = r_8$.

Egy teljes blokk megfejtése. Adott egy $Y = E_K(X)$ rejtjeles blokk, ahol $Y = y_1|y_2|\dots|y_8$ és $X = x_1|x_2|\dots|x_8$. Az 5.4. algoritmus segítségével X utolsó (néhány) bájtját meg tudjuk fejteni. Tegyük fel tehát, hogy x_j, x_{j+1}, \dots, x_8 értékét ismerjük valamely $1 < j \leq 8$ esetén. Ezt felhasználva szeretnénk x_{j-1} értékét meghatározni. Nyilván, ha ezt meg tudjuk tenni, akkor az eljárást ismételve végül X minden bájtját meg tudjuk fejteni.

Az előzőekhez hasonlóan most is két blokkból álló üzenetekkel hívjuk meg az orákulumot, ahol az első blokk egy általunk választott R blokk, a második blokk pedig Y . Vegyük észre, hogy mivel X utolsó $9 - j$ bájtja ismert, R -et pedig mi választjuk, ezért az orákulum által visszaállított nyílt szöveg utolsó blokkja ($X \oplus R$) utolsó $9 - j$ bájtjának értékét teljes mértékben befolyásolni tudjuk. A módszer lényege tehát az lesz, hogy ezen $9 - j$ bájt értékét $(9 - j)$ -re állítjuk, majd a $(j - 1)$ -edik bájtot addig változtatjuk (r_{j-1} változtatásával), míg az orákulum helyes kitöltést nem jelez. Ez pontosan akkor történik meg, mikor $x_{j-1} \oplus r_{j-1}$ is a $9 - j$ értéket veszi fel. Innen kapjuk, hogy $x_{j-1} = (9 - j) \oplus r_{j-1}$.

5.3. Példa. Előző példánkat folytatva tegyük fel, hogy $X = \text{DE} | \text{AD} | \text{BE} | \text{EF} | \text{DE} | \text{AD} | \text{BE} | \text{EF}$, és tudjuk, hogy $x_5|x_6|x_7|x_8 = \text{DE} | \text{AD} | \text{BE} | \text{EF}$. R első három bájtját tetszőlegesen választjuk, r_4 -et pedig 00-ra állítjuk. Legyen például $r_1|r_2|r_3|r_4 = 01|23|45|00$. R utolsó négy bájtját a következő módon választjuk: $r_5|r_6|r_7|r_8 = 04 \oplus \text{DE} | 04 \oplus \text{AD} | 04 \oplus \text{BE} | 04 \oplus \text{EF} = \text{DA} | \text{A9} | \text{BA} | \text{EB}$. Ezzel a beállítással $X \oplus R$ értéke $\text{DF} | 8E | \text{FB} | \text{EF} | 04 | 04 | 04 | 04$ lesz, tehát az $R|Y$ üzenetre az orákulum hibás kitöltést jelez. Ezután r_4 értékét addig változtatjuk, amíg az orákulum végül helyes kitöltést jelez. Ezt az 5.2. táblázat szemlélteti. Példánkban ez $r_4 = \text{EB}$ esetén következik be. Ekkor tehát tudjuk, hogy $x_4 \oplus \text{EB} = 04$, ahonnan $x_4 = 04 \oplus \text{EB} = \text{EF}$. ♣

$R = r_1 \dots r_4 \dots r_8$	$X \oplus R$	\mathcal{O}
$01 23 45 00 \text{DA} A9 \text{BA} \text{EB}$	$\text{DF} 8E \text{FB} \text{EE} 04 04 04 04$	0
$01 23 45 01 \text{DA} A9 \text{BA} \text{EB}$	$\text{DF} 8E \text{FB} \text{EE} 04 04 04 04$	0
...
$01 23 45 \text{EB} \text{DA} A9 \text{BA} \text{EB}$	$\text{DF} 8E \text{FB} 04 04 04 04 04$	1

5.2. táblázat. Az \mathcal{O} orákulum által adott válasz az $R|Y$ hívásra r_4 különböző értékei esetén. Ha a dekódolt üzenet kitöltése helyes, akkor a válasz 1, ellenkező esetben 0

A teljes blokk megfejtését végző eljárást így foglalhatjuk össze:

5.5. Algoritmus. Adott $Y = E_K(X)$ rejtejes blokk, ahol $Y = y_1|y_2|\dots|y_8$ és $X = x_1|x_2|\dots|x_8$. Tegyük fel, hogy x_j, x_{j+1}, \dots, x_8 értékét ismerjük valamely $1 < j \leq 8$ esetén. Az algoritmus ezt felhasználva meghatározza x_{j-1} értékét, ugyanazt az \mathcal{O} orákulumot használva, mint az 5.4. algoritmus. Az algoritmus lépései a következők:

1. Legyen $r_j|r_{j+1}|\dots|r_8 = (9-j) \oplus x_j|(9-j) \oplus x_{j+1}|\dots|(9-j) \oplus x_8$.
2. Legyen $r_1|r_2|\dots|r_{j-2}$ tetszőleges, és legyen $r_{j-1} = 00$.
3. Legyen $R = r_1|\dots|r_{j-1}|\dots|r_8$.
4. Ha $\mathcal{O}(R|Y) = 0$, akkor legyen $r_{j-1} = r_{j-1} + 1$, és folytassuk a 3. lépéssel.
5. Állunk meg. Az algoritmus eredménye: $x_{j-1} = r_{j-1} \oplus (9-j)$.

Egy teljes üzenet megfejtése. Adott egy N blokkból álló $C_1|C_2|\dots|C_N$ rejtejes üzenet, melyet egy $P_1|P_2|\dots|P_N$ nyílt üzenetből CBC módban történt rejtjelezéssel állítottak elő, egy K kulcsot és egy IV kezdeti változót használva. A P_i ($i > 1$) nyílt blokkot a következőképpen állíthatjuk elő a K kulcs

ismerete nélkül: először C_i -t dekódoljuk az 5.4. és az 5.5. algoritmusok segítségével, majd az eredményt XOR-oljuk C_{i-1} -gyel. A P_1 blokkot csak akkor tudjuk megfejteni, ha IV -t ismerjük. Ha IV -t ECB módban a K kulccsal rejtjezték, és a rejtjeles üzenethez csatolták, akkor IV is megfejthető az 5.4. és az 5.5. algoritmusok segítségével.

A támadás komplexitása. Az 5.4. és az 5.5. algoritmusok mindegyikének végrehajtása átlagosan 128 orákulumhívást igényel. Egy blokk dekódoláshoz az 5.4. algoritmust egyszer, az 5.5. algoritmust pedig nagy valószínűsséggel hétszer kell végrehajtani. Így egy blokk dekódolása átlagosan $8 \cdot 128 = 1024$ orákulum hívást igényel. Egy N blokkból álló üzenet megfejtéséhez tehát átlagosan $1024 \cdot N$ orákulum hívásra van szükség. Ez azt jelenti, hogy a módszer nagyon hatékony.

A támadás kikiúszöbölése. A támadás több módon is kivédhető. Egyik módszer lehet például a vevő viselkedésének rejtése. Ennek egyik módja, ha a vevő nem küld hibaüzeneteket, vagy ha hibaüzeneteket küld, akkor azokat rejtjelez, és a válaszidejét véletlenné teszi. Utóbbira azért van szükség, mert a kitöltés ellenőrzése biztosan rövidebb ideig tart, mint az azt általában követő üzenetintegritás-ellenőrzés. Így, még ha a hibaüzenetek rejtjelezve is vannak, a vevő válaszidejének méréséből a támadó következtetni tud arra, hogy hibás kitöltést jelző hibaüzenetről vagy az üzenetintegritás sérülését jelző hibaüzenetről van szó. Utóbbi esetben a támadó tudja, hogy a dekódolt üzenetben a kitöltés helyes volt.

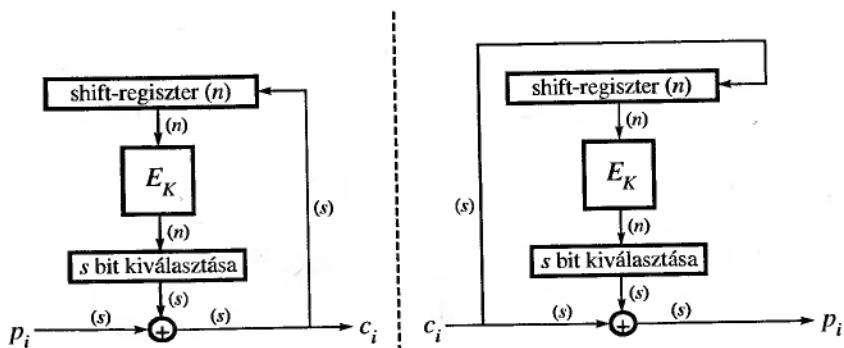
Egy másik módszer a támadás meghiúsítására egy olyan kitöltési séma alkalmazása, amely esetén a támadás nem működik. Ilyen például az a séma, ahol a kitöltő bájtokat véletlen módon választjuk, és az utolsó bájt jelzi a kitöltés hosszát. Ennél a sémánál pusztán a kitöltés vizsgálatával nem lehet eldönthető, hogy a kitöltés helyes-e, ezért a vevő sosem küld olyan hibaüzenetet, ami hibás kitöltést jelez.

Végül a legbiztosabb módszer a támadás elkerülésére, ha az üzenethitelesség ellenőrzése mindenkitől megelőzi a kitöltés ellenőrzését. A támadó által fabrikált üzenetek ugyanis majdnem biztosan nem mennek át a hitelesítési eljárásban, és így a vevő mindenkitől olyan hibaüzenettel válaszol, ami az üzenethitelesség sérülését jelzi. Ez tehát azt jelenti, hogy először kell a nyílt üzenetet kitölteni, és utána kell a már kitöltött üzeneten az üzenethitelesítő kódot (MAC) számolni. A nyílt üzenet biztonságos formátuma tehát a következő: nyílt üzenet | kitöltés | MAC(nyílt üzenet | kitöltés).

5.3. A CFB mód

Vannak olyan alkalmazások, melyekben a rejtjelező blokkhosszánál rövidebb üzeneteket (pl. karaktereket vagy akár biteket) kell rejtjelezni. Ráadásul az alkalmazás késleltetésre vontakozó követelményei olyanok lehetnek, hogy a küldőnek nincs ideje megvárni, míg e rövid üzenetekből összegyűlik egy blokkra való. Rövid üzenetenként egy blokkot elküldeni pazarló lenne, így a CBC mód nem jöhét szóba. Szeretnénk viszont továbbra is valamilyen CBC-hez hasonló láncolási technikát alkalmazni, mely segíti az üzenetek sorrendhelyes vételét és az üzenetek visszajátszásának detektálását. A megoldás a CFB mód alkalmazása.

A CFB mód működését az 5.3. ábra szemlélteti. Tegyük fel, hogy s bites karaktereket szeretnénk rejtjelezni. Ekkor az i -edik karakter p_i rejtjelezéséhez egy n bites belső shift-regiszter tartalmát rejtjelezzük a blokkrejtjelezővel, majd az eredmény első s bitjét XOR-oljuk p_i bitjeivel, és így kapjuk az i -edik rejtjeles karaktert, c_i -t. Ezt egyfelől elküldjük a vevőnek, másfelől jobbról beléptetjük a shift-regiszterbe. A következő karakter rejtjelezésekor tehát már a shift-regiszter új tartalmát fogjuk rejtjelezni. A dekódolás hasonlóan történik. A shift-regiszter aktuális tartalmát rejtjelezzük, majd az eredmény első s bitjét XOR-oljuk a c_i rejtjeles karakter bitjeivel, és így kapjuk vissza a p_i nyílt karaktert. Ezután c_i -t jobbról beléptetjük a shift-regiszterbe. Figyeljük meg, hogy a kódolásnál és a dekódolásnál is a blokkrejtjelező E_K kódoló függvényét használjuk. Vegyük észre továbbá, hogy a rendszer lényegében egy önszinkronizáló, kulcsfolyamatos rejtjelezőként működik.



5.3. ábra. Kódolás és dekódolás CFB módban

Mielőtt az első karaktert kódolnánk, a shift-regisztert fel kell tölteni egy kezdeti értékkel, amit a CBC módszer hasonlóan itt is *IV*-nek nevezünk. Az első rejtjeles karakter dekódolásához a vevőnek is szüksége van a küldő által használt *IV*-re, így azt valamilyen módon el kell juttatni a vevőhöz. Mivel CFB módban az *IV* a rejtjelező függvényen keresztül lép be a folyamatba, ezért az *IV* nyíltan is elküldhető a vevőnek. Egyszerűbb az *IV* ismerete nem segíti a támadót, mert a blokkrejtjelező által használt *K* kulcs nélkül úgysem tudja megfejteni az üzenetet. Vegyük észre, hogy a shift-regiszter tartalma úgyis ismertté válik a támadó számára az első *n/s* karakter feldolgozása után, hiszen a támadó által is megfigyelhető rejtjeles karaktereket léptetjük be a shift-regiszterbe. A rendszer biztonsága tehát nem a shift-regiszter állapotának titokban tartására épül, és ez a kezdeti állapotra éppen úgy igaz, mint a későbbi állapotokra. Másrészt a blokkrejtjelező tulajdonságai miatt, az *IV* bitjeinek módosítása a támadó által nem predikálható változásokat eredményez az első (és esetleg néhány további) visszaállított nyílt karakterben. A támadó tehát az *IV* manipulálásával nem tudja a nyílt szöveg kiválasztott bitjeit manipulálni.

Az *IV* vevőhöz történő eljuttatásának legegyszerűbb módja az, amikor a küldő az üzenet karakterei előtt, nyíltan küldi el az *IV*-t a vevőnek. A vevő nem alkalmaz külön eljárást az *IV* feldolgozására, hanem ugyanazt a dekódoló algoritmust futtatja, mint a rejtjeles karakterek vételénél. Így a vevő karakterenként lépteti be az *IV*-t a shift-regiszterébe. A dekódoló kimenetén eközben generálódó karaktereket a vevő eldobja. Mire az első rejtjeles karakter megérkezik, a vevő shift-regisztere már helyes állapotban van, és így a rejtjeles karakter dekódolása sikeres lesz. Ez az eljárás a CFB mód önszinkronizáló tulajdonságát használja ki, azaz azt, hogy *n/s* rejtjeles karakter helyes vétele után a dekódoló helyes állapotba kerül, függetlenül attól, hogy milyen állapotban volt a karakterek vétele előtt.

Megjegyezzük, hogy bár a CFB mód bevezetését a rövid üzenetek rejtjelezésének képessége motiválta, semmi nem szól az ellen, hogy teljes blokkok rejtjelezésére is használjuk. Más szóval, lehetséges az *s = n* eset. Ekkor az ECB és CBC módok mintájára, a CFB kódoló működését a következő módon írhatjuk le:

$$C_1 = P_1 \oplus E_K(IV), \quad (5.16)$$

$$C_i = P_i \oplus E_K(C_{i-1}) \quad i = 2, 3, \dots \quad (5.17)$$

A dekódoló működését pedig a következő egyenletek definiálják:

$$P_1 = C_1 \oplus E_K(IV), \quad (5.18)$$

$$P_i = C_i \oplus E_K(C_{i-1}) \quad i = 2, 3, \dots \quad (5.19)$$

Biztonság. A CFB mód hasonló láncolási tulajdonságokkal rendelkezik, mint a CBC mód: az i -edik rejtjeles karakter értéke nemcsak az i -edik nyílt karaktertől függ, hanem az összes azt megelőző nyílt karaktertől és az IV -től is. Ez a függés azonban csak az előző n/s rejtjeles karakteren keresztül valósul meg. Ez tehát azt jelenti, hogy egy adott rejtjeles karakter helyes dekódolásához elegendő a megelőző n/s rejtjeles karakter helyes értékének ismerete. Mindenesetre, ha két azonos nyílt szöveget különböző IV -ket alkalmazva rejtjelezünk, akkor a rejtjeles szövegek különbözők lesznek. Továbbá, rejtjeles karakterek módosítása, törlése, beszúrása és felcserélése a hiba helye után (mindaddig amíg a hibás karakterek a shift-regiszterben jelen vannak) pénzfeldobás sorozatokból álló nyílt karakterek visszaállítását eredményezi, tehát sok esetben detektálható.

Hatékonyság. CFB módban a kódolás és a dekódolás sebessége a blokkrejtjelező sebességének s/n -szerese. Ez $s = n$ esetén egyenlőséget jelent, $s < n$ esetén viszont a CFB módú kódolás sebessége kisebb, mint a blokkrejtjelező sebessége, ami egyszerűen abból adódik, hogy a blokkrejtjelező kimenetének csak töredék részét használjuk. A CBC módhoz hasonlóan a dekódolás CFB módban is párhuzamosítható, a kódolás viszont nem. Tetszőleges rejtjeles karaktert dekódolni lehet a többi karakter dekódolása nélkül, ha viszont a dekódolt karaktert módosítani is akarjuk, akkor az összes őt követő karaktert újra kell rejtjelezni. A nyílt karakterekhez való véletlen hozzáférés a rejtjeles szövegben tehát csak olvasás esetén hatékony.

Hibaterjedési tulajdonságok. Tegyük fel, hogy egy rejtjeles karakter egy bitje megváltozik. Mivel a hibás rejtjeles karakter bekerül a shift-regiszterbe, a hiba hatása kiterjed, és mindaddig érezhető lesz, amíg a hibás karakter a shift-regiszterben van. Egészen pontosan, a j -edik bit hibája egy rejtjeles karakterben a j -edik bit hibáját okozza az adott rejtjeles karakterből visszaállított nyílt karakterben, az azt követő n/s visszaállított nyílt karakter pedig használhatatlan pénzfeldobás sorozat lesz. Egy támadó tehát egy adott nyílt karakter bitjeit manipulálni tudja, bár az adott karaktert követően megjelenő véletlen karakterek azonosítása a támadás detektálásához vezethet. A véletlen karakterek azonban lehet, hogy túl későn érkeznek, és addigra a támadó már elérte a célját. Továbbá, az utolsó átküldött karakter minden-

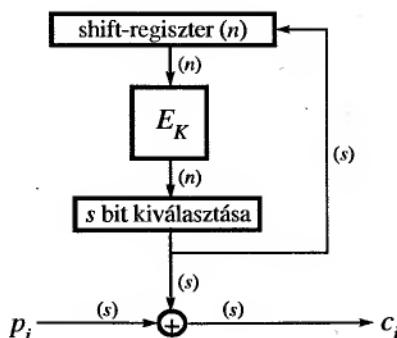
támadható, mert azt nem követi semmi, amiből a támadásra következtetni lehetne.

CFB mód használata esetén bitvesztésből és -beszúrásból származó hibák-ból is felépül a rendszer, mert a hiba után érkező helyes karakterek „kitolják” a hibát a shift-regiszterből, és n/s lépés után visszaáll a helyes állapot.

5.4. Az OFB mód

A CFB mód hátránya, hogy egy egy bites hiba a rejtejes szövegben több dekódolt karaktert tesz használhatatlanná. Előfordulhat, hogy az alkalmazás jellege nem teszi lehetővé ezen hibák dekódolás előtti javítását (pl. a vett karaktereket azonnal fel kell dolgozni, de nem hatékony minden átküldött rejtejes karakterhez hibajavítást lehetővé tevő redundáns biteket csatolni). Ezért ebben az esetben, ha a bithiba valószínűsége nagy (azaz a csatorna záros), akkor a CFB mód nem használható. Ilyenkor OFB vagy CTR módot használhatunk. Ezen módok egyik közös jellemzője, hogy a blokkrejtjelezőt egy szinkron kulcsfolyamatos rejtjelzővé alakítják, melyben a rejtjelező belső állapota (a shift-regiszter tartalma) nem függ sem a nyílt, sem a rejtejes karakterektől, és így egy rejtejes karakterben bekövetkező bithiba hatása nem terjed ki más karakterekre.

Az OFB mód működését az 5.4. ábra szemlélteti. Az ábrán a kódolás vázlata látható. Mivel szinkron kulcsfolyamatos rejtjelezők esetében a dekódolás sémája megegyezik a kódolással (csupán annyi a különbség, hogy



5.4. ábra. Kódolás OFB módban. A dekódolás ugyanilyen séma szerint történik, csak a rejtejes karakterek lépnek be, és a nyílt karakterek lépnek ki

a bemeneten rejtjeles karakterek lépnek be, és a kimeneten nyílt karakterek jönnek ki), ezért a dekódoló működését sem OFB sem CTR mód esetén nem illusztráljuk külön ábrán.

Az OFB kódoló vázlatára feltűnő hasonlóságot mutat a CFB kódoló vázlatával. A két rendszer között annyi a különbség, hogy OFB esetén nem a rejtjeles karakter van visszacsatolva, hanem a blokkrejtjelező kimenete (Pontosabban általános esetben a kimenet egy része). Az i -edik nyílt karakter, p_i rejtjelezése tehát a CFB módon hasonlóan történik: a shift-regiszter aktuális tartalmát rejtjelezzük, majd az eredmény első s bitjét XOR-oljuk p_i -vel (melyről feltesszük, hogy szintén s bites), és így kapjuk az i -edik rejtjeles karaktert, c_i -t. Ezután a blokkrejtjelező kimenetének egy részét (tipikusan ugyanazt az s bitet, melyet p_i -hez XOR-oltunk) beléptetjük a shift-regiszterbe. Dekódolásnál ugyanez történik: a shift-regiszter aktuális tartalmát rejtjelezzük, majd az eredmény első s bitjét XOR-oljuk c_i -vel, és így kapjuk vissza p_i -t. Ezután a blokkrejtjelező kimenetének egy részét (ugyanazt a részt mint kódolásnál) beléptetjük a shift-regiszterbe.

A CFB módon hasonlóan, az első karakter kódolása és dekódolása előtt a shift-regisztereket egy kezdeti értékkel (IV) kell feltölteni.

A visszacsatolás mérete nem feltétlenül kell, hogy megegyezzen a karakterek s méretével. Sőt, alább látni fogjuk, hogy tanácsos a blokkrejtjelező teljes n bites kimenetét visszacsatolni, mert ez nagymértékben megnöveli a generált kulcsfolyam periódusát. Az n bites OFB működését a következő formulák írják le:

$$X_i = E_K(X_{i-1}), \quad (5.20)$$

$$x_i = X_i \text{ első } s \text{ bitje}, \quad (5.21)$$

$$c_i = p_i \oplus x_i, \quad (5.22)$$

$$p_i = c_i \oplus x_i, \quad (5.23)$$

ahol $i = 1, 2, \dots$ és $X_0 = IV$.

Biztonság. Az IV titkosságát nem kell biztosítani. Ügyelni kell azonban arra, hogy minden nyílt üzenet (ahol üzenet alatt most csak egy karakter-sorozatot értünk) kódolásához különböző IV -t használunk. Ellenkező esetben minden nyílt üzenetet pontosan ugyanazzal a kulcsfolyammal fogjuk kódolni, és egy támadó a két rejtjeles üzenetet egymással XOR-olva a nyílt üzenetek XOR összegét kapja, melyből az üzenetek redundanciáját ismerve sikeresen kiszérel-

heti meg az üzenetek megfejtését. Az IV -k különbözőségét például úgy lehet biztosítani, hogy az üzenetek sorszámát használjuk IV -nek.

A CFB móddal ellentétben, az OFB mód esetén az i -edik rejtjeles karakter értéke csak az i -edik nyílt karaktertől (és a shift-regiszter tartalmától) függ. Ennek ellenére, a CFB módon hasonlóan, a rejtjeles karakterek törlését, beszúrását és sorrendjének megváltoztatását korlátozott mértékben mégis detektálni lehet. Ez azért van, mert egy karakter helyes visszaállításához a shift-regiszter tartalmának meg kell egyeznie a karakter kódolásánál használt shift-regiszter tartalommal. Ha azonban c_i -t felcseréljük c_j -vel, akkor c_i dekódolásánál a shift-regiszter tartalma a c_j előállításánál használt tartalom lesz. Ebből kifolyólag a c_i -ből visszaállított nyílt karakter bitjei pénzfeldobás sorozatot alkotnak.

Ha a visszacsatolt bitek száma pontosan n , akkor (5.20) alapján a shift-regiszter X_i állapotára az $X_i = f(X_{i-1})$ teljesül, ahol f egy permutáció. Ebben az esetben egy véletlen X_0 állapotból indulva, a generált X_0, X_1, X_2, \dots állapot-sorozat periódusának hossza egyforma valószínűséggel lesz $1, 2, \dots, 2^n$. Az átlagos periódushossz tehát 2^{n-1} .

Ezzel szemben, ha a visszacsatolt bitek száma kisebb, mint n , akkor a shift-regiszter X_i állapotára továbbra is az $X_i = f(X_{i-1})$ teljesül, de könnyen látható, hogy ekkor f nem permutáció. Ebben az esetben az analízis bonyolultabb. Az egyszerűség kedvéért ezért f -et úgy modellezünk mint egy véletlen függvényt. Ebben a modellben f minden lépésben 2^n lehetséges állapot közül választja a következő állapotot, amíg egy már előfordult állapot újra fel nem bukkan. Ekkor az átlagos periódushosszt a születésnapi paradoxon alapján tudjuk megbecsülni. Mivel az f függvény minden lépésben 2^n lehetséges állapot közül választ, ezért $2^{\frac{n}{2}}$ lépés után $\frac{1}{2}$ -nél nagyobb valószínűséggel olyan állapotot választ, ami egyszer már előfordult. Így a generátor átlagos periódushosszára a $2^{\frac{n}{2}}$ körüli becslés adható. A hosszabb periódus elérése érdekében tehát tanácsos a blokkrejtjelező teljes kimenetét visszacsatolni.

Hatókonyiság. A CFB módon hasonlóan, a kódolás és dekódolás sebessége a blokkrejtjelező sebességének s/n -szerese. Míg CFB módban a c_i rejtjeles karakter dekódolásához elegendő a $c_{i-n/s}, c_{i-n/s+1}, \dots, c_{i-1}$ rejtjeles karaktereket beléptetni a shift-regiszterbe, addig OFB mód esetén az IV -ből indulva elő kell állítani a c_i kódolásánál használt shift-regiszter tartalmat. Ezt nem lehet párhuzamosítani, ezért a CFB móddal ellentétben, sem a kódolás, sem a dekódolás nem párhuzamosítható. Ugyanakkor, mivel a generált kulcsfolyam nem függ a nyílt szövegtől, ezért az off-line módon előre kiszá-

mítható még mielőtt a nyílt szöveg rendelkezésre állna. Ez nagymértékben gyorsítja a kódolás és dekódolás műveletét. A megfelelő shift-regiszter állapot elérése után tetszőleges rejtejes karakter dekódolható, módosítható és újrakódolható a többi karakter változtatása nélkül. Az OFB mód tehát támogatja a nyílt karakterekhez való véletlen hozzáférést a rejtejes szövegben, mind olvasás, mind írás esetén.

Hibaterjedési tulajdonságok. Mint az már említettük, a CFB móddal ellenetben, az OFB mód nem terjeszti ki a rejtejes szövegben keletkezett bithibát több visszaállított nyílt karakterre. Egy rejtejes karakter *j*-edik bitjének változása csak a visszaállított nyílt karakter *j*-edik bitjére van hatással. Ez egyfelől előny, mert az OFB módot zajos átviteli csatorna esetén is használhatóvá teszi. Másrészt viszont hátrány, mert egy támadó a visszaállított nyílt szöveg bitjeit manipulálni tudja a megfelelő rejtejes bitek módosításával.

Bitvesztésből és -beszúrásból származó hibából nem épül fel a rendszer, mert a hiba helyét követően a karakterhatárok elcsúsznak, és minden további karakter hibásan áll vissza. Ezért OFB mód használata esetén speciális újraszinkronizáló pontokat (markereket) kell beiktatni a karakterfolyamba, melyek lehetővé teszik a szinkronból történt esetleges kiesésekből való felépülést.

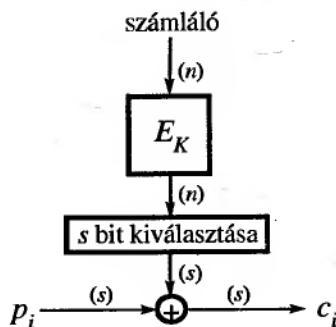
5.5. A CTR mód

A CTR mód működése nagyon hasonlít az OFB módhoz. A különbség annyi, hogy nincs visszacsatolás, helyette a shift-regiszter által tárolt értéket minden lépésben eggyel növeljük. Ezt úgy is felfoghatjuk, mintha egy számláló aktuális értékét rejtezeznénk a blokkrejtjelezővel. A CTR mód működését az 5.5. ábrán szemléltetjük.

A működés hasonlóságából fakadóan a CTR mód tulajdonságai nagyban hasonlítanak az OFB mód tulajdonságaihoz. Ezért itt most csak a lényeges különbségekre térünk ki.

Biztonság. CTR mód esetén a generátor periódusát könnyű megállapítani, hiszen azt a számláló mérete határozza meg. n bites számláló esetében a periódushossz 2^n . A biztonsággal kapcsolatos többi tulajdonság megegyezik az OFB mód tulajdonságaival.

Hatékonyság. Az OFB móddal ellentétben, CTR módban mind a kódolás, mind a dekódolás párhuzamosítható, hiszen az *i*-edik karakter feldolgozásá-



5.5. ábra. Kódolás CTR módban. A dekódolás ugyanilyen séma szerint történik, csak a rejtjel karakterek lépnek be, és a nyílt karakterek lépnek ki

hoz szükséges shift-regiszter tartalmat a számláló értékének megfelelő beállításával azonnal elő tudjuk állítani. A hatékonysággal kapcsolatos többi tulajdonság megegyezik az OFB mód tulajdonságaival.

5.6. Összefoglalás

Az egyes blokkrejtjelezési módok előnyeit és hátrányait a következőképpen foglalhatjuk össze:

ECB mód

biztonság:

- a nyílt szöveg mintáit nem rejt megfelelően
- a blokkrejtjelező bemenete nem randomizált
- szótár (kódkönyv) alapú támadás lehetséges
- a rejtjel blokkok felcserélhetők, törölhetők, helyettesíthetők

hatékonyság:

- + a sebesség megegyezik a blokkrejtjelező sebességével
- + a kódolás és a dekódolás is párhuzamosítható
- + nyílt blokokhoz (pl. adatbázis rekordokhoz) való véletlen hozzáférés olvasás és írás esetén is
- nem lehet előzetes off-line számításokkal gyorsítani

hibaterjedés:

- egy bithiba a rejtjel szövegben egy egész nyílt blokkot érint
- bitbeszűrásból vagy -törlésből származó hibából nem épül fel

CBC mód

biztonság:

- + a nyílt szöveg mintáit rejtő blokkal való XOR-olás
- + a blokkrejtjelező bemenetét randomizálja az előző rejtjeles blokkal való XOR-olás
- + azonos nyílt szövegeket különböző IV-vel rejtjelezve különböző rejtjeles szövegeket kapunk
- + korlátozott mértékben detektálni lehet a rejtjeles blokkok felcserélését, törlését, helyettesítését
- kivág- és-beszúr támadások lehetségesek
- azonos rejtjeles blokkokhoz tartozó nyílt blokkok XOR összegételfedi
- Vaudenay-féle támadásra adhat lehetőséget

hatékonyság:

- + a sebesség lényegében megegyezik a blokkrejtjelező sebességével
- /+ a kódolás nem párhuzamosítható / a dekódolás párhuzamosítható
- +/- nyílt szöveg blokkokhoz való véletlen hozzáférés csak olvasás esetén
- nem lehet előzetes off-line számításokkal gyorsítani

hibaterjedés:

- egy bithiba a rejtjeles szövegen hatással van egy teljes, nyílt szöveg-blokkra és egy bitre a következő blokkban
- bitbeszúrásból vagy -törlésből származó hibából nem épül fel

CFB mód

biztonság:

- + a nyílt szöveg mintáit rejtő
- + a blokkrejtjelező bemente véletlen
- + azonos nyílt szövegeket különböző IV-vel rejtjelezve különböző rejtjeles szövegeket kapunk
- + korlátozott mértékben detektálni lehet a rejtjeles karakterek felcserélését, törlését, helyettesítését
- utolsó karakter bitjei manipulálhatók

hatékonyság:

- a sebesség a blokkrejtjelező sebességének s/n -szerese
- /+ a kódolás nem párhuzamosítható / a dekódolás párhuzamosítható
- +/- nyílt karakterekhez való véletlen hozzáférés csak olvasás esetén
- + előzetes off-line számítás korlátozott mértékben lehetséges (shift-regiszter tartalmát rejtjelezni lehet a következő rejtjelezendő karakter beérkezése előtt)

hibaterjedés:

- egy bit hiba a rejtjeles szövegben $n/s + 1$ visszaállított karaktert érint, amiből n/s használhatatlan pénzfeldobás sorozattá válik
- + bitbeszúrásból vagy -törlesből származó hibából is felépül

OFB/CTR mód

biztonság:

- + a nyílt szöveg mintáit rejtí
- + azonos nyílt szövegeket különböző IV-vel rejtjelezve különböző rejtjeles szövegeket kapunk
- különböző nyílt szövegeket azonos IV-vel rejtjelezve a nyílt szövegek megfejthetőek
- +/- korlátozott mértékben detektálni lehet a rejtjeles karakterek felcserélését, törlését, helyettesítését, de nem olyan mértékben, mint CFB mód esetén (a korlátozott hibaterjedés miatt)
- OFB módban n -nél kevesebb bites visszacsatolás esetén a generátor periódushossza jelentősen csökken
- + CTR módban a generátor periódushossza a számláló méretétől függ
- a visszaállított nyílt karakterek bitjei manipulálhatók

hatékonyság:

- a kódolás és dekódolás sebessége a blokkrejtjelező sebességének s/n -szerese
- + a kulcsfolyam előzetes off-line számításával a működés gyorsítható
- /+ OFB módban a kódolás nem párhuzamosítható, a dekódolás párhuzamosítható
- + CTR módban a kódolás és a dekódolás is párhuzamosítható
- + nyílt karakterekhez való véletlen hozzáférés olvasás és írás esetén is

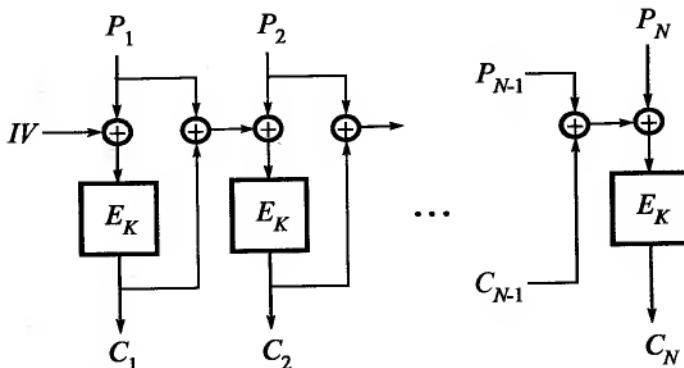
hibaterjedés:

- + egy bithiba a rejtjeles szövegben egy bithibát generál a visszaállított nyílt szövegben
- bitbeszúrásból vagy -törlesből származó hibából nem épül fel

5.7. Feladatok

5.1. Feladat. Az 5.6. ábra egy blokkrejtjelező (pl. DES) használatát mutatja PCBC (Plain and Cipher Block Chaining) módban.

1. Adja meg a dekódoló sémáját!
2. Tegyük fel, hogy a C_i és C_{i+1} blokkokat egy támadó felcseréli egymással. Mutassa meg, hogy ennek csak a P_i és P_{i+1} nyílt blokkokra van hatása!
3. Önszinkronizáló-e a PCBC mód? Miért?



5.6. ábra. Blokkrejtjelező használata PCBC módban

5.2. Feladat. Tekintsük a DES OFB módját. Ez a mód kulcsfolyamos rejtjelezést valósít meg. Fontos, hogy a kulcsfolyam ciklusa nagy legyen. Mekkora ezen ciklus várható értéke, ha a visszacsatolt bitek m számára

1. $m = 64$ (azaz a teljes blokkméret vissza van csatolva),
2. $m < 64$?

Milyen következtetést vonna le a kapott értékekkel?

Segítség: (1) Modellezze a rejtjelezést, mint 64 bites blokkok egy véletlen permutációját. (2) Modellezze a rejtjelezést, mint egy véletlen függvényt a 64 bites állapotok halmazán, s alkalmazza a születésnapi paradoxont.

5.3. Feladat. Véletlen bithibázású csatornán rejtjelezetten továbbítjuk üzenetünket CBC blokkrejtjelező módban. A véletlen hibázás ellen hibajavító kódolást alkalmazunk. Végezzük ezt a kódolást a rejtjelezést megelőzően.

1. Helyesen járunk-e el a fenti módon a hibák javításával kapcsolatosan?
2. Mi a válasza, ha CBC mód helyett OFB módban rejtjelezünk?

6.

Üzenethitelesítés

Az üzenethitelesítés feladata a kommunikációs csatornán átküldött üzenetek hitelességének és integritásának biztosítása. Pontosabban, az üzenethitelesítés lehetővé teszi az üzenet vevője számára a küldő identitásának ellenőrzését és az átvitel során az üzenetben bekövetkezett változások (melyek származhatnak véletlen hibából vagy szándékos módosításból) detektálását.

Az üzenethitelesítést leggyakrabban üzenethitelesítő kódok (Message Authentication Code – MAC) alkalmazásával valósítjuk meg. Egy üzenethitelesítő kódra gondolhatunk úgy, mint egy kriptográfiai ellenőrző összegre, amit a küldő az üzenet elküldése előtt kiszámít és az üzenethez csatol. A csatornán átvitelre kerül az üzenet és az üzenet ellenőrző összege is. A vevő mindenkorral veszi, majd ellenőrzi az ellenőrző összeget. Ha az ellenőrzés sikeresen jár, akkor a vevő meg lehet győződve arról, hogy az üzenet sértelesen és valóban a vétel (pl. az üzenetben megjelölt) feladó küldte. Ellenkező esetben, az üzenet integritása az átvitel során megsérült, és mivel ez lehet rosszindulatú módosítás következménye is, ezért a vevő nem fogadja el az üzenetet.

Formáját és funkcióját tekintve tehát egy üzenethitelesítő kód valamiféle hibadetektáló kódhoz (pl. CRC) hasonlítható. Van azonban egy nagyon fontos különbség az üzenethitelesítő és a hibadetektáló kódok között. Nevezetesen, a hibadetektáló kódok csak a zajos csatornán bekövetkezett véletlen hibák detektálására alkalmasak, és nem képesek egy rosszindulatú támadó által végrehajtott szándékos módosítások detektálására. Ez egyszerűen azért van, mert a támadó az üzenet módosítása után a módosított üzenethez kitudja számolni az új hibadetektáló kódot. A vevő tehát a módosított üzenetet

és a hozzátartozó helyes hibadetektáló kódot kapja meg, és így nem veszi észre a módosítást.

Ezzel szemben az üzenethitelesítő kódok nemcsak a véletlen hibákat, hanem a rosszindulatú módosításokat is képesek detektálni. Ezen képességük abból adódik, hogy a hibadetektáló kóddal ellentétben, az üzenethitelesítő kód értéke nemcsak magától az üzenettől függ, hanem egy a küldő és a vevő által megosztott titkos információtól (kulcsról) is. A támadó ezen titok hiányában nem tudja kiszámítani a módosított üzenethez tartozó helyes üzenethitelesítő kódot, és így a módosítás nem maradhat észrevéltelen. Ezen túlmenően a vevő tudja, hogy helyes üzenethitelesítő kód kiszámítására csak a küldő (és természetesen maga a vevő) alkalmas. Ezért meg lehet győződve arról, hogy minden helyes üzenethitelesítő kóddal vett (és nem saját magától származó) üzenet csakis a küldőtől származhat.

A fenti gondolatok az üzenethitelesítés következő modelljéhez vezetnek: Az A küldő és a B vevő rendelkezik egy közös K_{AB} kulccsal. K_{AB} -t rajtuk kívül más nem ismeri. Az m üzenet elküldése előtt A kiszámolja az m -hez tartozó $\mu = MAC_{K_{AB}}(m)$ üzenethitelesítő kódot, ahol $MAC_{K_{AB}}$ a K_{AB} kulccsal paraméterezett üzenethitelesítő függvény. A továbbiakban a μ üzenethitelesítő kódot röviden MAC értéknek, a $MAC_{K_{AB}}$ üzenethitelesítő függvényt pedig röviden MAC függvénynek fogjuk nevezni. A elküldi, B pedig megkapja a MAC értékkal kiegészített $m|\mu$ üzenetet. K_{AB} ismeretében B kiszámolja $MAC_{K_{AB}}(m)$ -et, és az eredményt összehasonlíta μ -vel. Egyenlőség esetén B elfogadja az üzenetet, ellenkező esetben eldobja azt.

A MAC függvényteljes szemben az alábbi követelményeket támasztjuk:

1. A MAC_K függvény olyan szűkítő transzformáció legyen, mely tetszőleges hosszúságú üzeneteket fix hosszúságú, n bites MAC értékbe képez.
2. A K kulcs ismeretében tetszőleges m üzenethez könnyű legyen kiszámolni $MAC_K(m)$ -et.
3. A K kulcs ismeretének hiányában viszont legyen $MAC_K(m)$ kiszámítása nehéz feladat, még akkor is, ha nagy számú $(m_i, MAC_K(m_i))$ pár áll rendelkezésre, ahol természetesen $m \notin \{m_i\}$.
4. A K kulcs meghatározása legyen nehéz feladat még nagy számú $(m_i, MAC_K(m_i))$ pár ismerete esetén is.

Megjegyezzük, hogy ha egy MAC függvény eleget tesz a 3. követelménynek, akkor kielégíti a 4. követelményt is. Ha ugyanis a 4. követelményt nem elégíténé ki, akkor egy támadó meg tudná határozni a K kulcsot és annak ismeretében tetszőleges üzenethez tudna MAC értéket generálni, azaz a MAC

függvény nem elégíthetné ki a 3. követelményt sem. Fordítva azonban nem áll fenn az implikáció, ugyanis elméletileg elképzelhető, hogy a K kulcs ismerete nem szükséges ahhoz, hogy egy üzenethez helyes MAC értéket generáljon a támadó. Erre a 6.1. és 6.2. alfejezetben látunk majd példákat.

Az 1. és a 2. követelmény a hash függvényekhez hasonló tulajdonságokat követel meg a MAC függvénytől. Tulajdonképpen a MAC függvényt úgy is felfoghatjuk mint egy kulcsvezérelt hash függvényt. Figyeljük meg azonban, hogy a MAC függvénnyel szemben támasztott további követelmények különböznek a hash függvények egyirányúságra vagy ütközésmentességre vonatkozó követelményeitől. Mindenesetre a hash függvényekhez való hasznatosság ötletet adhat hash függvényekre épülő MAC függvények konstruálására. Ezt a tervezési lehetőséget a 6.2. alfejezetben mutatjuk be részletesbben. Előtte azt vizsgáljuk meg, hogy egy CBC módban használt blokkrejtjelezőből hogyan lehet MAC függvényt építeni, és ez milyen veszélyekkel járhat.

6.1. A CBC MAC

A blokkrejtjelező egy alapvető kriptográfiai primitív, melyet sok protokollban használunk építőelemként. A 4.3.2. szakaszban láttuk, hogy hogyan lehet blokkrejtjelezőből hash függvényt konstruálni. Mivel a MAC függvények sokban hasonlítanak a hash függvényekhez, ezért az a gondolatunk támadhat, hogy blokkrejtjelezőből építünk MAC függvényt. Természetesen abban reménykedünk, hogy az így nyert MAC függvény biztonságát az alkalma-zott blokkrejtjelező biztonságára tudjuk visszavezetni. Ennek a gondolatnak a legelterjedtebb megtétesítője az ún. CBC-MAC konstrukció.

A CBC-MAC függvény működésének gyors megértéséhez képzeljük el, hogy az üzenetet CBC módban rejtjelezük egy blokkrejtjelezővel (lásd 5.2. alfejezet), azzal a módosítással, hogy a rejtjeles blokkokat az utolsó kivételével minden előbbihez hozzáadjuk. A MAC függvény kimenete az utolsó rejtjeles blokk lesz.

A CBC-MAC pontos működését a következőképpen írhatjuk le: Adott egy m üzenet és egy K kulcs, ahol a K mérete megegyezik a használni kívánt blokkrejtjelező kulcsméretével. Az m üzenetet először kiegészítjük, hogy mérete a blokkrejtjelező n blokkméretének többszöröse legyen. Mivel magát az m -et is el fogjuk küldeni a vevőnek, ezért a kitöltés lehet nagyon egyszerű, például az üzenet megfelelő számú 0 bittel történő kiegészítése. Jelöljük a kiegészített üzenetet P -vel. P -t n bites blokkokra osztjuk, melyeket P_1, P_2, \dots, P_N -nel jelölünk. Legyen $IV = 0$ (azaz a csupa 0 bitből álló blokk).

A CBC mód működését definiáló (5.3) és (5.4) kifejezések, valamint a megadott IV és K felhasználásával számítsuk ki az utolsó rejtjeles blokkot, C_N -t. Ez lesz a CBC-MAC függvény kimenete, azaz $MAC_K(m) = C_N$. Ha rendelkezésre áll egy $K' \neq K$ másik kulcs is, akkor a MAC függvény kimenetére opcionálisan nem C_N -et vezetjük, hanem $E_K(E_{K'}^{-1}(C_N))$ -et, ahol E jelöli a blokkrejtjelező kódoló, E^{-1} pedig a dekódoló függvényét.

Tekintsük most át a MAC függvénnyel szemben támasztott követelmények listáját. A CBC-MAC függvény tetszőleges méretű üzenethez egy n bites MAC értéket generál, ahol n az alkalmazott blokkrejtjelező blokkmérete. A K kulcs ismeretében a MAC számítása egyszerű, a CBC kódolással megegyező módon történik. A K kulcs meghatározása megfigyelt üzenet – MAC párokból lényegében a blokkrejtjelező feltörését jelenti, tehát megfelelő erősséggű blokkrejtjelező esetén ez biztosan nehéz feladat. Hasonlóképpen nehéznek tűnik a MAC meghatározása egy adott üzenethez, ha a kulcs nem ismert.

A CBC-MAC elleni támadások lehetősége a CBC mód gyengeségeiben rejlik. Nevezetesen, ha adott két nyílt szöveg, P és P' , akkor a $P|P'$ nyílt szöveget egy K kulccsal és egy IV kezdeti változóval CBC módban rejtjelezve a $C|C'$ rejtjeles szöveget kapjuk, ahol C a P -hez tartozó rejtjeles szöveg, melyet CBC módú rejtjelezéssel nyerünk P -ből a K kulcsot és az IV kezdeti változót használva, C' pedig a P' -höz tartozó rejtjeles szöveg, melyet szintén CBC módú rejtjelezéssel nyerünk P' -ből a K kulcsot és C utolsó blokkját, mint kezdeti változót használva. A CBC mód ezen tulajdonságát kihasználva egy támadó néhány megfigyelt üzenet – MAC pár segítségével olyan üzenetet tud konstruálni, melynek ki tudja számítani (vagy ismeri) a MAC értékét. Az alább tárgyalta támadások adaptívan választott üzeneteket használnak, azaz feltesszük, hogy a támadó képes megszerezni néhány általa választott üzenet MAC értékét, és ebből próbálja meg kiszámítani az általa konstruált üzenet MAC értékét.

Tegyük fel, hogy a támadó ismeri egy m üzenet μ MAC értékét. Tegyük fel továbbá, hogy a támadó meg tudja szerezni a μ -höz mint egyblokkos üzenethez tartozó μ' MAC értékét. Mivel μ egyblokkos, ezért $\mu' = E_K(\mu \oplus IV) = E_K(\mu)$ (felhasználva, hogy CBC-MAC esetén $IV = 0$). Ekkor a támadó – minden további számítás nélkül – ismeri az m üzenet és a csupa 0 bitből álló blokk összefűzésével nyert $m|0$ üzenethez tartozó MAC értékét, hiszen az nem más, mint $MAC_K(m|0) = E_K(0 \oplus \mu) = E_K(\mu) = \mu'$.

A következő példa azt mutatja, hogy a támadó nemcsak egy csupa 0 bitből álló blokkal kiegészített üzenethez, hanem egy tetszőleges blokkal ki-

egészített üzenethez is meg tudja szerezni a helyes MAC értéket. Tegyük fel, hogy a támadó ismeri egy m_1 és egy m_2 üzenet μ_1 és μ_2 MAC értékét. Tegyük fel továbbá, hogy a támadó meg tudja szerezni a $\mu_1 \oplus \mu_2 \oplus z$ blokkal megtoldott $m_1|\mu_1 \oplus \mu_2 \oplus z$ üzenethez tartozó μ' MAC értékét, ahol z egy tetszőleges blokk. Könnyen látszik, hogy $\mu' = E_K(\mu_2 \oplus z)$. Ekkor a támadó ismeri az $m_2|z$ üzenet MAC értékét, hiszen az nem más mint $MAC_K(m_2|z) = E_K(z \oplus \mu_2) = \mu'$.

A fenti támadások nem működnek, ha az alkalmazás olyan, hogy minden üzenetnek ismert, fix mérete van. Ekkor ugyanis a megtoldott üzeneteket hibás méretük miatt nem fogadja el a vevő. Ha a fix méretet nem lehet biztosítani, akkor a CBC-MAC azon opcióját használva védekezhetünk, mely szerint nem a CBC módú rejtelezéssel nyert utolsó blokkot, hanem annak egy továbbkódolt változatát tekintjük a CBC-MAC függvény kimenetének. Annak átgondolását, hogy ezen opció használata valóban megakadályozza-e a fenti támadásokat, az olvasóra bízzuk.

6.2. Hash függvényre épülő MAC függvények

A MAC függvényekre vonatkozó 1. követelmény szerint a MAC függvények szűkítő transzformációnak kell lennie. Ebben a tekintetben a MAC függvények a hash függvényekre hasonlítanak, ezért megpróbálkozhatunk hash függvény alapú MAC függvény konstrukciókkal. Ügyelünk kell azonban arra, hogy a hash függvényektől megkövetelt tulajdonságok (úgy mint az egyirányúság és az ütközésmentesség) nem feltétlenül biztosítják a MAC függvényekkel szemben támasztott további követelmények teljesülését.

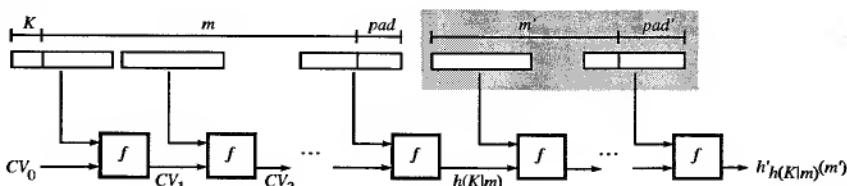
A legegyszerűbben úgy kovácsolhatunk hash függvényből MAC függvényt, hogy a kulcsot hozzáfűzzük az üzenethez, majd a kulccsal megtoldott üzenetnek kiszámítjuk a hash értékét, és az eredményt tekintjük az eredeti üzenet MAC értékének. Attól függően, hogy a kulcsot az üzenet elő vagy mögé fűzzük, alapvetően két módszert különböztethetünk meg, melyeket *titok prefix* és *titok suffix* módszereknek hívunk.

Titok prefix módszer. Nevéből adódóan a titok prefix módszer az üzenet elő fűzi a kulcsot. Az m üzenet MAC értékét tehát a következő módon számoljuk:

$$MAC_K(m) = h(K|m), \quad (6.1)$$

ahol h jelöli a hash függvényt, melyre a MAC konstrukció épül.

Az így nyert MAC függvény h tulajdonságaiból adódóan szűkítő transzformáció és a teljes input ismeretében (azaz a K kulcsot is ismervé) a MAC érték könnyen számolható. A hash függvény egyirányúsága miatt abban reménykedünk, hogy egy adott MAC érték ismeretében a K kulcs megfejtése nehéz feladat. Vegyük azonban észre, hogy a hash függvény egyirányúsága azt jelenti, hogy egy adott MAC érték ismeretében a teljes input (kulcs és üzenet) megfejtése nehéz feladat. A mi esetünkben azonban az input egy része – az üzenet – ismert. Kérdéses tehát, hogy ez a konstrukció milyen garanciát biztosít a kulcs megfejtése ellen. További probléma, hogy ha h egy iteratív hash függvény, akkor a támadó egy m üzenet MAC értékének ismeretében könnyen tud olyan üzeneteket fabrikálni, melyek MAC értékét ki tudja számolni. Ezt a gyengeséget az alábbiakban részletezzük.



6.1. ábra. A titok prefix módszer elleni támadás

A könnyebb megértés érdekében tekintsük a 6.1. ábrát. Jelöljük h tömörítő függvényét f -vel. Az f függvénynek két bemenete van, az egyik az aktuális iterációs lépésben feldolgozandó üzenetblokk, a másik pedig a láncváltozó (chaining variable – CV) aktuális értéke. Az f függvény kimenete a láncváltozó következő értéke, illetve az utolsó iterációs lépés esetén magának az üzenetnek a hash értéke. Legyen a feldolgozott üzenetblokkok mérete b bit. Mivel a kulccsal mint prefixeszel megtoldott $K|m$ üzenet hossza nem feltétlenül többszöröse b -nek, ezért valamilyen pad kitöltést alkalmazunk (mely tartalmazhat MD erősítést). A $h(K|m)$ hash értéket tehát úgy kapjuk, hogy a $K|m|pad$ kitöltött üzenetet f iteratív alkalmazásával blokkonként feldolgozzuk.

Jelöljük a láncváltozó kezdeti értékét CV_0 -val. CV_0 egy a hash függvény specifikációjában rögzített konstans. Elvileg azonban bármilyen értéket használhatnánk kezdeti láncváltozóként. Jelöljük h'_{cv} -vel azt a hash függvényt, melyet h -ból úgy kapunk, hogy CV_0 helyett cv -t használunk kezdeti láncváltozóként. Ekkor nyilván $h(x) = h'_{CV_0}(x)$.

Tekintsük most az $m|pad|m'$ üzenetet, ahol m' tetszőleges bitsorozat lehet. Az $m|pad|m'$ üzenethez tartozó MAC értéket a következőképpen számolhatjuk ki (lásd 6.1. ábra):

$$\begin{aligned} MAC_K(m|pad|m') &= h(K|m|pad|m') \\ &= h'_{h(K|m)}(m') \\ &= h'_{MAC_K(m)}(m'). \end{aligned}$$

Ez azt jelenti, hogy $MAC_K(m)$ ismeretében az $m|pad|m'$ üzenet MAC értéke a K kulcs ismerete nélkül is kiszámítható.

Titok suffix módszer. Egy másik lehetőség hash függvény alapú MAC függvény konstruálására a titok suffix módszer. Ekkor a kulcsot az üzenet mögé fűzzük, majd a kulccsal így kiegészített üzenetnek kiszámoljuk a hash értékét, és az eredményt tekintjük az üzenet MAC értékének:

$$MAC_K(m) = h(m|K). \quad (6.2)$$

A titok suffix módszer tulajdonságai hasonlítanak a titok prefix módszer tulajdonságaihoz azzal a különbséggel, hogy a titok prefix módszernél részletesen tárgyalt támadás ebben az esetben nem kivitelezhető. A módszer egy gyengesége, hogy ha h egy iteratív hash függvény, akkor a kulcsot csak a MAC érték kiszámításának utolsó lépésében használjuk fel. Ez problémához vezethet, ha a hash függvény nem ütközésmentes (pl. az alkalmazás jellegéből adódóan a MAC érték hosszára vonatkozóan korlátozásaink vannak). Tegyük fel ugyanis, hogy a támadó (a születésnapi paradoxont használva) talált egy m és egy m' üzenetet, melyekre $h(m) = h(m')$. Legyen az m és az m' kiszámításánál alkalmazott kitöltés (mely tartalmazhat MD erősítést is) rendre pad és pad' . Könnyen látszik, hogy ekkor $h(m|pad|K) = h(m'|pad'|K)$, hiszen $m|pad$ és $m'|pad'$ feldolgozása után a hash függvény belső láncváltozójának értéke a két esetben megegyezik. Ebből következik, hogy az $m|pad$ és az $m'|pad'$ üzenetekhez tartozó MAC értékek megegyeznek. Más szóval, ha a támadó megszerzi az egyik MAC értékét, akkor azt fel tudja használni a másik MAC értékeként is.

További módszerek. A titok prefix és a titok suffix módszereket kombinálva, a következő MAC konstrukcióra jutunk:

$$MAC_K(m) = h(K|m|K). \quad (6.3)$$

Ez a módszer, melyet szemléletesen szendvics módszernek is nevezhetünk, elég biztonságosnak tűnik. Egy további variáció lehet két különböző kulcs K és K' alkalmazása:

$$MAC_{(K,K')}(m) = h(K|m|K'). \quad (6.4)$$

Egy másik biztonságos megoldás a K kulcs néhány bitjének felhasználása minden üzenetblokk feldolgozásánál. Tegyük fel, hogy a hash függvény b bites blokkokban dolgozza fel az inputot. Legyen $b = b_1 + b_2$ ($b_1, b_2 > 0$). Az üzenetet osszuk b_1 bites blokkokra, és minden blokkhoz fűzzünk hozzá b_2 bitet a kulcsból. A b_1 bites üzenetblokkból és a b_2 kulcsbitből álló b bites blokkokat dolgozzuk fel a hash függvénnel. Az így eredményül kapott hash értéket tekintjük az üzenet MAC értékének. A kulcsbitek kiválasztását valamelyen, a vevő számára is ismert, kulcsütemező algoritmus végzi. Ezen algoritmussal szemben hasonló követelményeket támaszthatunk, mint a réteges szerkezetű blokkrejtjelezőknél alkalmazott kulcsütemezővel szemben (pl. minden kulcsbit közel azonos számú blokkban legyen felhasználva). A módszer hátránya, hogy az üzenetet rövidebb blokkokban dolgozzuk fel, mint a hash függvény blokkmérete, és így több iterációra van szükség (azaz a hash függvény tömörítő függvényét többször kell meghívni). Hosszú üzenetek esetén ez jelentősen csökkentheti a MAC függvény hatékonyságát.

Végül megemlíünk egy a hash függvény és a rejtjelezés kombinációjából származó MAC függvény konstrukciót. Ennek működése a következő: számoljuk ki az üzenet hash értékét, majd rejtjelezzük a hash értéket a K kulccsal:

$$MAC_K(m) = E_K(h(m)). \quad (6.5)$$

Ezen módszer ugyanazzal a gyengeséggel rendelkezik, mint a titok suffix módszer, azaz ha h nem ütközésmentes, akkor a támadó könnyen tud találni két üzenetet, melyeknek ugyanaz a MAC értékük.

HMAC. A HMAC a gyakorlatban igen elterjedten használt hash függvényre épülő MAC függvény, melynek részletes leírása a 2104-es számú Internet RFC-ben található. A HMAC függvényt a következő kifejezés definiálja:

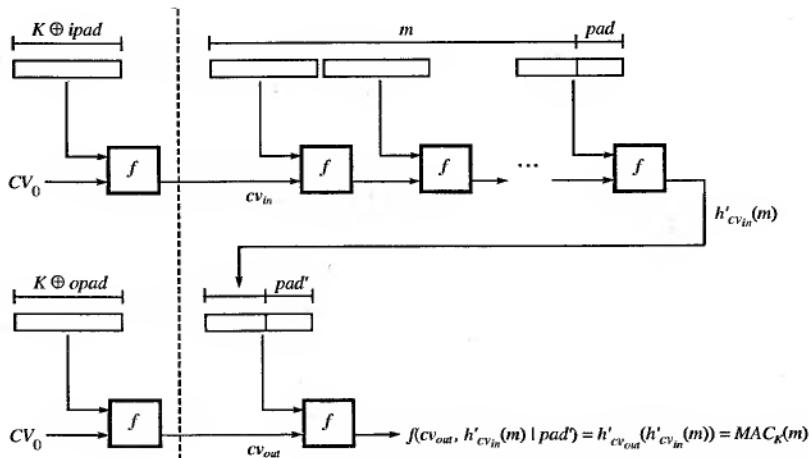
$$MAC_K(m) = h(K^+ \oplus opad | h(K^+ \oplus ipad | m)), \quad (6.6)$$

ahol

- h egy iteratív hash függvény, ami az üzenetet B bájtos blokkokban dolgozza fel,

- $ipad$ (inner pad) egy B bájt hosszú konstans blokk, melyben minden bájt értéke 36,
- $opad$ (outer pad) egy B bájt hosszú konstans blokk, melyben minden bájt értéke 5C és
- K^+ a K kulcs csupa 0 bittel kiegészítve, hogy hossza elérje a B bájtot.

A K kulcs tetszőleges hosszúságú lehet. B értéke tipikusan 64, így K általában rövidebb, mint B bájt. Ha mégis hosszabb lenne, akkor először kiszámoljuk $h(K)$ -t, és ezt használjuk kulcsként. A HMAC által definiált MAC érték mérete az alkalmazott h hash függvény kimenetének méretétől függ. Ha például h az MD5 hash függvény (HMAC-MD5), akkor a kiszámított MAC mérete 128 bit (16 bájt), míg a SHA-1 hash függvény használata esetén (HMAC-SHA1) a MAC mérete 160 bit (20 bájt).



6.2. ábra. A HMAC számítása

A HMAC egy előnyös tulajdonsága, hogy a MAC érték kiszámításának komplexitása lényegében megegyezik az üzenet hash értéke kiszámításának komplexitásával. Ezt a 6.2. ábra szemlélteti. Jelöljük az alkalmazott h hash függvény tömörítő függvényét f -fel. Legyen továbbá $cv_{in} = f(CV_0, K^+ \oplus ipad)$ és $cv_{out} = f(CV_0, K^+ \oplus opad)$. Vegyük észre, hogy cv_{in} és cv_{out} értéke független az m üzenettől, melynek a MAC értékét számoljuk. Ez azt jelenti, hogy cv_{in} és cv_{out} off-line módon előre számítható, még mielőtt m rendelkezésre állna. Jelöljük ismét h'_{cv} -vel azt a hash függvényt, melyet h -ból úgy

kapunk, hogy CV_0 helyett cv -t használunk kezdeti láncváltozóként. Ekkor

$$\begin{aligned} MAC_K(m) &= h(K^+ \oplus opad \mid h(K^+ \oplus ipad \mid m)) \\ &= h'_{cv_{out}}(h'_{cv_{in}}(m)) \\ &= f(cv_{out}, h'_{cv_{in}}(m) \mid pad'), \end{aligned}$$

ahol felhasználtuk, hogy $h'_{cv_{in}}(m)$ mérete kisebb, mint B bájt, ezért a külső hash függvény alkalmazása lényegében az f tömörítő függvény egyszeri meghívását jelenti. A fenti kifejezésből jól látszik, hogy a HMAC számítása az üzenet hash értékének kiszámítását és az f függvény egyszeri alkalmazását igényli.

A HMAC egy másik előnyös tulajdonsága, hogy biztonságát intenzíven tanulmányozták és bizonyították. A HMAC egy érdekes tulajdonsága például, hogy a (külső) hash függvény bemente egyáltalán nem ismert a támadó számára, hiszen az nem tartalmazza közvetlenül az üzenetet (mint a titok prefix és titok suffix módszereknél), hanem csak a belső hash függvény kiemenetét.

A jelenleg ismert legerősebb támadás a HMAC ellen ütközések keresésére épül a születésnapi paradoxont felhasználva. Ez azonban a tipikusan használt méretek mellett a gyakorlatban kivitelezhetetlen támadás. Ha például az alkalmazott hash függvény az MD5 függvény, mely 128 bites hash értéket generál, akkor a születésnapi paradoxon értelmében a támadónak 2^{64} számú üzenethez kell megszereznie a helyes MAC értéket ahhoz, hogy egy ütközést találjon. Ráadásul ezen MAC értékeket egyazon K kulccsal kell generálni. Figyeljük meg azt is, hogy míg a hash függvények elleni támadásoknál a támadó off-line dolgozva kereshet ütközéseket, addig a HMAC esetén az ütközések kereséséhez a MAC értékeket meg kell szereznie, hiszen a kulcs ismerete nélkül azokat off-line módon nem tudja kiszámolni. A támadó próbálkozhat üzenet – MAC párok lehallgatásával, ám 2^{64} üzenet – MAC pár ilyen módon történő megszerzése a mai technológia mellett több százezer évet venne igénybe (és a K kulcsnak mindvégig azonosnak kellene maradnia).

6.3. Feladatok

6.1. Feladat. Külföldön dolgozunk, s alkalmanként szeretnénk rejtetten párbeszédet folytatni otthoni barátunkkal. Tilos azonban rejtjelezni a határon átlépő üzeneteket. Hitelesítő protokollok használata viszont nem tiltott (amikor is egy nyílt szöveg nyílt marad). Van megoldás? Ha megtalálta, adjon

egy példát egy rövid párbeszéd védelmére, továbbá elemezze a megoldás hátrányait (nehézségeit) is!

(Segítség: Az eredeti párbeszédbe tűzzön bele olyan plusz sorokat, amelyek kibogozhatatlanná teszik a párbeszéd eredeti pontos értelmét. Már csak arra kell rájönnie, hogy hogyan lesz képes a partnere kiválogatni az eredeti sorokat, s itt jól jöhet egy, a feladat szerint legális hitelesítő protokoll alkalmazása.)

6.2. Feladat. Egy cég informatikai központja szoftverek egy-egy példányát szétesztja távoli egységei informatikai részlegeinek. Szeretné a szoftverek sértetlenségét biztosítani, s alkalmasnak (például hetente) szeretné ellenőrizni azok helyességét, amely feladat megoldásához azonban nem kíván titkos kulcsra kapcsolódó eljárásokat alkalmazni, például azért, mert a korrekt kulcsgondozás költséges feladat, s erre nem kíván erőforrásokat lekötni. Lehetséges-e megoldás ilyen feltételek mellett?

6.3. Feladat. Tekintsük az alábbi integritásvédelmi kódolást, ahol rejtjelezés és MAC kombinációját használjuk:

$$E_k(m|MAC_K(m)). \quad (6.7)$$

Legyen CBC módú mind, rejtjelezés, mind a MAC számítás, ahol az egyik funkcióra (IV, k) , míg a másikra (IV', k') (inicializáló vektor, kulcs) páros kerül alkalmazásra. Van-e veszélye annak, ha $IV = IV'$, $k = k'$ „egyszerűsítő” választással élünk? Mi a tanulság?

6.4. Feladat. m számú véletlen bináris blokkból álló üzenetüket rejtjelezve és integritásvédelemmel szeretnénk továbbítani. Integritásvédelemből azt a gyors módszert választjuk, hogy az m darab üzenetblokkot mod 2 összegezzük, s így egy ellenőrzőösszeg blokkot nyerünk (azaz az ellenőrzőösszeg blokk i -edik bitje az üzenetblokkok i -edik bitjeinek a mod 2 összege). Ezután az $m + 1$ darab blokkot blokkonként ECB módban rejtjezzük.

1. Lehallgató támadó sikeres lehet-e? És kimerítő kulcskeresést feltételezve?
2. Aktív támadó sikeres lehet-e?

6.5. Feladat. Mutassa meg, hogy a CBC-MAC hashing nem teljesíti az egyirányúság követelményét a k kulcsot ismerő fél számára, azaz tetszőleges MAC ellenőrzőösszeghez képes csaló üzenetet előállítani!

6.6. Feladat. Azt mondjuk, hogy egy MAC algoritmus választott szövegű támadás ellen védett, ha egy támadó rendelkezésére áll a MAC az általa választott üzenetekre, mégsem képes ennek alapján kiszámítani egy MAC lenyomatot egy új üzenetre. Mutassa meg, hogy a CBC-MAC nem védett választott szövegű támadás ellen!

6.7. Feladat. Mutassa meg, hogy a CBC-MAC nem védett választott szövegű támadás ellen akkor sem, ha úgy kívánjuk megerősíteni, hogy a MAC számítás előtt kiegészítjük az üzenetet egy olyan blokkal, amely az üzenet hosszát tartalmazza. A támadó képességéről azt tessük fel, hogy üzeneteire MAC lenyomatot kaphat.

6.8. Feladat. Két kommunikáló fél felváltva küld üzenetcsomagokat egymásnak. El szeretné kerülni a sorszámgondozást. Mindkét fél rendelkezik egy-egy forrással, ahonnan véletlen elemeket kaphat. MAC kódú üzenethitelesítésüket szeretnék tranzakció hitelesítés szintre feljavítani, azaz biztosítani az időbeli frissesség- és duplikátlatlanság-ellenőrizhetőség képességet is. Javasoljon egy egyszerű protokollt e célra!

6.9. Feladat. Vizsgálja meg, hogy a rejtelezés biztosítja-e az adatintegritás védelmet az alábbi esetekben:

1. additív kulcsfolyamos rejtelezés,
2. ECB módú blokkos rejtelezés,
3. a nyílt szöveg redundancia-mentes!

6.10. Feladat. Ha mind a titkosság, mind pedig az adatintegritás biztosítása szükséges, egy lehetséges módszer a rejtelezés és az MDC (Message Digest Code) együttes alkalmazása, azaz $E_k(m|MDC(m))$ kódolás. Itt MDC alatt tetszőleges nyilvános lenyomatképzőt értünk. Például ide tartozik egy lineáris CRC lenyomat, vagy akár egy kriptográfiai hash függvény is. Van-e különbség az alábbi két lineáris MDC alkalmazhatósága vonatkozásában: ha az MDC az üzenetblokkok bitenkénti mod2 összege, illetve, ha az MDC egy hibadetekciós kódolási gyakorlatban alkalmazott CRC (ciklikus redundancia)?

6.11. Feladat. Tekintsük az $m|E_k(MDC(m))$ integritásvédő kódolást. Milyen tulajdonsággal kell ez esetben az MDC-nek rendelkeznie?

7.

Digitális aláírás

Az előző fejezetben tárgyalt üzenethitelesítő kódok nagyon hasznos szolgáltatásokat nyújtanak: lehetővé teszik a csatornán átküldött üzenetek (véletlen és szándékos) módosításának detektálását és az üzenetek küldőjének hitelesítését. Az üzenethitelesítő kódok hátránya azonban az, hogy ezeket a szolgáltatásokat csak a vevő számára biztosítják. A vevő egy kívülálló harmadik felet már nem tud meggyőzni arról, hogy egy vett üzenet sértetlen, és a küldőtől származik. Ez azért van, mert az üzenethitelesítő kód értéke egy olyan titkos kulcsról függ, melyet a küldő és a vevő is ismer. A harmadik fél tehát nem tudja biztosan eldönteni, hogy az adott üzenethitelesítő kódot a küldő vagy a vevő generálta. Ez azt jelenti, hogy a küldő bármikor letagadhatja, hogy egy üzenetet küldött a vevőnek, és a vevő nem tudja bebizonyítani, hogy a küldő hazudik. Más szóval, az üzenethitelesítő kódok nem biztosítanak (*eredet*) *letagadhatatlanság* szolgáltatást.

Egy megbízható harmadik fél (Trusted Third Party – TTP) bevonásával ez a probléma megoldható. Tegyük fel, hogy az A küldő és a B vevő egy-egy titkos kulcsot oszt meg a TTP-vel, melyet rendre K_A -val és K_B -vel jelölünk. A minden m üzenetet a TTP-n keresztül küld el B -nek a következő protokollt használva:

-
- | | | |
|-----|-----------------------|---------------------------------------|
| (1) | $A \rightarrow TTP :$ | $A \mid B \mid m \mid MAC_{K_A}(B m)$ |
| (2) | $TTP \rightarrow A :$ | $A \mid m \mid MAC_{K_B}(A m)$ |
-

A a K_A kulcsot használva kiszámolja a $B|m$ üzenet MAC értékét, és azzal együtt küldi el az üzenetet a TTP-nek. Így a TTP a K_A kulcs ismeretében le tudja ellenőrizni az (1)-es üzenet hitelességét. Ha az ellenőrzés sikeres, akkor a TTP biztos lehet benne, hogy az m üzenet A -tól származik és B -nek szól.

A TTP ezek után a K_B kulcs alkalmazásával új MAC értéket számol, ezúttal az $A|m$ üzenethez. Így B ellenőrizni tudja, hogy a (2)-es üzenet a TTP-től jött. A azonosítója az üzenetben azt jelzi B számára, hogy az üzenet eredeti forrása A . Mivel B megbízik a TTP-ben, ezért biztos abban, hogy a TTP állítása igaz, és az üzenet valóban A -tól származik. Más szóval, a protokoll biztosítja a küldő hitelesítését a vevő számára. Vegyük észre azonban, hogy a fenti protokoll által a TTP A és B kommunikációjának tanújává vált, és bárki számára bizonyítani tudja, hogy A küldte m -et B -nek.

Bár a célt eléri, a fenti protokollnak számos hátránya van. Egyszerűen a TTP-nek aktívan részt kell vennie minden egyes üzenetküldésben. Ez szükségszerűen az üzenetek késleltetését eredményezi. Ezen kívül, egy nagyobb rendszer esetén a kommunikáló párok száma is nagy lehet, és így a TTP könnyen a rendszer szűk keresztmetszetévé válhat. Továbbá, a TTP rendelkezésre állását is biztosítani kell, hiszen ha a TTP valamelyen hiba folytán leáll, akkor semilyen további kommunikáció nem lehetséges a rendszerben. Egy másik nagy hátrány, hogy B csak a TTP aktív részvételével tudja bizonyítani C -nek, hogy az m üzenetet A küldte. A bizonyíték tipikusan a $MAC_{K_C}(A|m)$ MAC érték lehetne, melyet a TTP számol ki. Látható azonban, hogy ezt csak C tudja ellenőrizni. Tehát a TTP-nek annyi MAC értéket kell kiszámolnia, ahány félnek szüksége van a bizonyítékra.

Egy *aszimmetrikus* mechanizmus alkalmasabb a letagadhatatlanság szolgáltatás megvalósítására, melynek segítségével csak az üzenet küldője állíthatja elő az üzenet eredetére vonatkozó bizonyítékot (így azt hamisítani nem lehet), de a rendszer bármely résztvevője (köztük a vevő is) ellenőrizni tudja azt. Ezt a mechanizmust digitális aláírásnak nevezzük, mivel tulajdon-ságai nagymértékben hasonlítanak a hagyományos aláírás tulajdonságaihoz. Egy fontos különbség a digitális és a hagyományos aláírás között az, hogy a digitális aláírás nem az üzenet anyagi hordozójához (pl. papír) kötődik, hanem magához az üzenethez. Így nemcsak az üzenet eredetére vonatkozóan nyújt garanciát, hanem segítségével az üzenet tartalmában az aláírás generálása után bekövetkezett módosításokat is detektálni lehet. Összefoglalva tehát: a digitális aláírás egy olyan mechanizmus, mely biztosítja az üzenetek integritását és hitelességét, valamint az üzenetek eredetének letagadhatatlanságát.

A digitális aláírás fent említett aszimmetrikus tulajdonsága miatt, a jól ismert digitális aláírássémák minden nyilvános kulcsú kriptográfiára épülnek. Egy digitális aláírásséma két komponensből áll: egy aláírás-generáló S algoritmusból és egy aláírás-ellenőrző V algoritmusból. Az aláírás-generáló al-

goritmus az aláíró fél privát kulcsával van paraméterezve, míg az ellenőrző algoritmus az aláíró fél nyilvános kulcsát használja paraméterként. Jelölésben ezt úgy érzékelhetjük, hogy az aláíró fél azonosítóját alsó indexbe írjuk, azaz az A privát kulcsával paraméterezett aláíró algoritmust S_A -val, az A nyilvános kulcsával paraméterezett ellenőrző algoritmust pedig V_A -val jelöljük. Az S_A algoritmus bemenete az aláírni kívánt m üzenet, kimenete pedig $\sigma = S_A(m)$ digitális aláírás. A V_A ellenőrző algoritmus bemenete egy m üzenet és egy σ aláírás, kimenete pedig *true*, ha σ A érvényes aláírása m -en, és *false* egyébként:

$$V_A(m, \sigma) = \begin{cases} \text{true} & \text{ha } \sigma = S_A(m) \\ \text{false} & \text{egyébként.} \end{cases} \quad (7.1)$$

Értelemszerűen, az ellenőrzést végző fél akkor fogadja el az aláírást hitelesenek, ha az ellenőrző algoritmus kimenete *true*.

Természetesen az ellenőrzés végrehajtásához az ellenőrző félnek ismenie kell az aláíró fél nyilvános kulcsát, mert ez szükséges az ellenőrző algoritmus helyes paraméterezéséhez (azaz V_A használatához). Az aláíró fél hiteles nyilvános kulcsának megszerzése külön feladat, amit a gyakorlatban legtöbbször valamilyen nyilvános kulcs infrastruktúra (Public Key Infrastructure – PKI) segítségével oldanak meg. Ezekkel a kérdésekkel a 8.5. szakaszban foglalkozunk részletesebben. Ebben a fejezetben feltesszük, hogy az aláírás ellenőrzését végző fél ismeri az aláíró fél nyilvános kulcsát.

7.1. Támadások osztályozása

Kriptográfiai szempontból egy digitális aláírássáma legfontosabb tulajdonsgága a biztonsága. Mint azt már megszoktuk, a biztonság fogalma szorosan kapcsolódik a támadási modellhez, mely rögzíti feltevéseinket a támadó célját és képességeit illetően. A digitális aláírás esetében a támadó célja általában aláírások hamisítása. Részletesebben a következő célokat különböztetjük meg:

- *Teljes feltörés:* A digitális aláírássáma teljes feltöréséről beszélünk, amikor a támadó képes kiszámítani az aláíró privát kulcsát, vagy egy olyan algoritmust talál, mely funkcionálisan ekvivalens az aláíró privát kulcsával paraméterezett aláírás-generáló algoritmussal. Teljes feltörés esetén tehát a támadó tetszőleges általa választott üzenetre képes aláírást generálni az aláíró nevében.

- **Szelektív hamisítás:** Ebben az esetben a támadó célja érvényes aláírás generálása az aláíró nevében egy adott üzenetre vagy üzenetek egy adott családjára.
- **Egzisztenciális hamisítás:** Ebben az esetben a támadó célja, hogy legalább egy üzenetre érvényes aláírást generáljon az aláíró nevében. Fontos megjegyezni, hogy a szelektív hamisítással ellentében, itt az üzenet, melyhez a támadó az aláírást generálja, nem a támadó által választott üzenet; a támadó bármilyen üzenethez generálhatja az aláírást.

A támadó képességeit tekintve a lehetséges támadásokat a következőképpen osztályozhatjuk:

- **Csak a nyilvános kulcs ismert:** A leggyengébb képességekkel rendelkező támadó csak nyilvánosan hozzáférhető információk birtokában van, azaz a támadó csak az aláíró fél nyilvános kulcsát ismeri.
- **Ismert üzenet alapú támadás:** Ebben az esetben a támadó rendelkezésére áll néhány üzenet – aláírás pár, de az üzeneteket, melyekre az aláírások ismertek, nem a támadó választja.
- **Választott üzenet alapú támadás:** Ebben az esetben feltezzük, hogy a támadó képes néhány általa választott üzenethez megszerezni a hiteles aláírást, és ezeket felhasználva szeretne aláírásokat hamisítani újabb üzenetekhez (melyekhez természetesen még nem ismeri az aláírást).
- **Adaptívan választott üzenetekre épülő támadás:** A legerősebb képességekkel rendelkező támadóról feltezzük, hogy képes az aláírót orákulum-ként használni. Pontosabban, ismét azt feltételezzük, hogy a támadó képes általa választott üzenetekhez megszerezni a hiteles aláírást, de most azt is megengedjük, hogy az üzeneteket a korábban megszerzett aláírások függvényében, adaptívan válassza a támadó.

Szeretnénk, ha a digitális aláírásséma ellenállna a legerősebb képességekkel rendelkező támadónak és a leggyengébb célra irányuló támadásoknak is. Más szóval, egy digitális aláírássémát akkor tekintünk biztonságosnak, ha az ellenáll az egzisztenciális hamisításra irányuló, adaptívan választott üzenetekre épülő támadásoknak.

7.2. Lenyomat aláírása (a „hash-and-sign” paradigm)

Tekintsünk egy nyilvános kulcsú titkosító rendszert. Az A résztvevő kódoló transzformációját (melyet A nyilvános kulcsa határoz meg) jelöljük E_A -val, az ehhez tartozó dekódoló transzformációt (melyet A privát kulcsa határoz meg) pedig D_A -val. Ha minden m üzenetre teljesül az $E_A(D_A(m)) = m$ egyenlőség, akkor az adott nyilvános kulcsú rendszerből a következőképpen készíthetünk digitális aláírássémát. Az aláírás-generáló S_A algoritmus legyen a titkosító rendszer D_A dekódoló transzformációja ($S_A(m) = D_A(m)$), az aláírás-ellenőrző V_A algoritmust pedig definiálja a következő kifejezés:

$$V_A(m, \sigma) = \begin{cases} \text{true} & \text{ha } E_A(\sigma) = m \\ \text{false} & \text{egyébként} \end{cases} \quad (7.2)$$

Más szavakkal, az aláírást úgy generáljuk, hogy az m üzeneten a titkosító rendszer dekódoló transzformációját alkalmazzuk, az aláírást pedig úgy ellenőrizzük, hogy a σ aláíráson a kódoló transzformációt alkalmazzuk, és az eredményt összehasonlítjuk az eredeti m üzenettel. Ha az aláírás hiteles, azaz $\sigma = D_A(m)$, akkor a titkosító rendszer fenti tulajdonsága miatt $E_A(\sigma) = m$ egyenlőségnek teljesülnie kell.

Kihangsúlyozzuk, hogy a fenti digitális aláírás konstrukció csak abban az esetben működik, ha a nyilvános kulcsú titkosító rendszer rendelkezik azzal a tulajdonsággal, hogy $E_A(D_A(m)) = m$ minden m -re teljesül. Ilyen például az RSA rendszer, melyre $E_A(D_A(m)) = (m^d)^e \bmod N = (m^e)^d \bmod N = D_A(E_A(m)) = m$. Ez azonban nem minden nyilvános kulcsú titkosító rendszernél van így. Az ElGamal rendszernél például a kódoló és a dekódoló transzformáció nem ugyanazon a halazon van értelmezve, és így a fenti tulajdonság nem teljesül.

A fenti digitális aláírásséma hátránya, hogy az aláírás mérete függ az üzenet méretétől (hosszabb üzeneteken képzett aláírás hosszabb). Gyakorlati okokból azonban célszerű ezt a függést megszüntetni. Ezt úgy tehetjük meg, hogy egy alkalmas nyilvános egirányú hash függvény felhasználásával, nem az eredeti üzeneten, hanem annak hash értékén alkalmazzuk a dekódoló transzformációt. Ekkor az aláírt üzenet $m | D_A(h(m))$ alakú.

A hash érték aláírása általában is előnyös, nemcsak a fenti, nyilvános kulcsú titkosító rendszerből kialakított aláírássémák esetében. Ha ugyanis az üzenet helyett csak annak (általában kisebb méretű) hash értékén alkalmazzuk az aláírás-generáló algoritmust, akkor annak futási ideje rövidebb lesz. Ráadásul az aláírás-generálásának ideje így majdnem független az üzenet

méretétől¹. Ekkor az aláírt üzenet $m \mid S_A(h(m))$ alakú. Ezt nevezzük „hash-and-sign” paradigmának.

Ha egy támadó megfigyel egy $m \mid S_A(h(m))$ üzenetet, és talál egy olyan m' üzenetet, melyre $h(m') = h(m)$, akkor az $S_A(h(m))$ aláírást felhasználhatja az m' üzenet hiteles aláírásaként, hiszen ekkor nyilván $S_A(h(m')) = S_A(h(m))$ is teljesül. Csakhogy ha h egyirányú hash függvény, akkor a támadó gyakorlatilag nem találhat azonos hash értékre vezető m' üzenetet (ezt a második ōskép ellenállóság (2nd preimage resistance) tulajdonság biztosítja). Megjegyezzük, hogy a gyakorlatban a támadó feladatát tovább nehezíti, hogy nemcsak a $h(m') = h(m)$ egyenlőségnek kell teljesülnie, hanem m' -nek értelmes – a támadó céljainak megfelelő – csaló tartalmú üzenetnek kell lennie.

Yuval születésnapi támadása. Láttuk tehát, hogy a támadó gyakorlatilag nem tud egy megfigyelettel azonos hash értékű üzenetet találni. Ha például a hash függvény kimenete 64 bites, akkor a támadó sikerének valószínűsége 2^{-64} . Ha azonban a támadó rá tudja venni A -t, hogy adjon aláírást egy általa összeállított – akár ártatlannak tűnő tartalmú – üzenetre (választott üzenet alapú támadás), akkor a születésnapi paradoxont használva a támadó lényegesen hatékonyabb támadás végrehajtására is képes.

Tegyük fel, hogy a támadó elő tud állítani 2^{32} ártatlan, és ugyanennyi csalásra felhasználható, értelmes üzenetet. Az ártatlan üzenetek halmazát jelöljük M -mel, a csaló üzenetek halmazát pedig M' -vel. A születésnapi paradoxon miatt nagy a valószínűsége annak, hogy van legalább egy ($m \in M, m' \in M'$) pár, melyre $h(m) = h(m')$. A támadás menete innen már nyilvánvaló: a támadó kéri A aláírását az ártatlan m üzenetre, és ezzel megszerzi A hiteles aláírását a csaló m' üzenetre.

Elsőre talán kivitelezhetetlennek tűnik ilyen nagy számú ($2 \cdot 2^{32} \approx 8$ milliárd) ártatlan és csaló üzenet generálása és tárolása, ám ez sokkal könnyebb, mint hinnénk. Szemléltetésül tekintsük az alábbi példát:

Tisztelt ... Úr!

Ezen levelemmel bemutatom Önnel ... urat, aki ... naptól ... napig megbízottam ... ügyletekben. Jogosult ... összeghatárig ... illetve ... vásárlására. Az összegeket ... pénznemben bocsássa rendelkezésére ... címletekben. A kettőnk közti elszámolás ... ütemezéssel történik, stb.

¹ Az üzenet hash értékének kiszámítása továbbra is függ az üzenet méretétől, de a hash számítása még így is több nagyságrenddel gyorsabb, mint az aláírás generálása, ezért az aláírás-generáló algoritmus futási ideje dominál.

A kipontozott helyekre többféle lehetőség közül választva szavakat nagyszámú üzenetváltozatot állíthatunk elő. Például, ha 33 kipontozott helyet hagyunk, s mindegyikre két lehetőség szerint választunk, már 2³³ üzenetváltozatunk van. Egy-egy változatot 33 bittel tudunk kódolni. Vessük ezt össze egy 1 oldalas, 2 kbyte méretű szöveg 16384 bitjével, ami felére tömörítve is 8000 bit nagyságrendjébe esik. Mindegyik üzenethez 64 bites hash érték tartozik, így egy üzenethez $33 + 64 + 1 = 98$ bitet rendelünk, ahol a plusz egy bit jelzi, hogy melyik halmazból való az üzenet (ártatlan vagy csaló). A párosítást végezzük úgy, hogy a teljes, 2³³ méretű halmaz elemeit rendezzük sorba hash értékek szerint, s ha egybeesést (ütközést) találunk a rendezés során, akkor a halmazindikátor bit alapján eldöntjük, hogy különböző halmazból valók-e az ütköző üzenetek. A rendezéshez $33 \cdot 2^{33}$ nagyságrendű számítás szükséges, ami a mai technológiával még egy otthoni PC-n sem kivitelezhető feladat. A tárigény 10 Gbyte nagyságrendű, ami szintén biztosítható házi PC szintű technológiával.

A fenti támadást maga A is végezheti, előre bebiztosítva maga számára a csalás lehetőségét. Ha a támadást nem ő végzi, akkor a támadó munkáját nehezíthetjük azzal, hogy minden egyes aláírás előtt az aláírandó dokumentumhoz illesztünk egy véletlenül sorsolt bináris sorozatot. Ezt a véletlen kiélezést a támadó nem képes predikálni, így nem tudja a sablon üzenetváz ismeretében az előkészítő számításokat elvégezni.

A védekezés egy másik módja ütközésmentes hash függvény használata, mely nagyobb hash méretet (a mai technológia mellett tipikusan minimum 128 bitet) jelent.

7.3. Példák digitális aláírássémákra

Az RSA aláírásséma. Legyen A nyilvános RSA kulcsa (e, N) , ahol e a nyilvános exponens, N pedig az RSA modulus, A privát RSA kulcsa pedig d . Ekkor egy m üzenet aláírásához A kiszámítja $m^d \bmod N$ -t, azaz

$$S_A(m) = m^d \bmod N.$$

Az m üzenet σ aláírásának ellenőrzéséhez $\sigma^e \bmod N$ értékét kell m -hez hasonlítani, azaz

$$V_A(m, \sigma) = \begin{cases} \text{true} & \text{ha } \sigma^e = m \\ \text{false} & \text{egyébként.} \end{cases}$$

Természetesen az RSA aláírássémát is lehet hash-and-sign módon használni. Ekkor az aláírás-generáló és -ellenőrző algoritmusok definíciója:

$$S_A(m) = (h(m))^d \bmod N$$

$$V_A(m, \sigma) = \begin{cases} \text{true} & \text{ha } \sigma^e = h(m) \\ \text{false} & \text{egyébként.} \end{cases}$$

Az ElGamal aláírásséma. Az ElGamal aláírásséma ismertetését az ElGamal kulcsgenerálás lefrásával kezdjük. Válasszunk egy nagy p prímszámot, és keressük meg a Z_p^* multiplikatív csoport egy g generátor elemét. Válasszunk egy $1 \leq a \leq p - 2$ számot, és számítsuk ki $A = g^a \bmod p$ -t. A nyilvános kulcs legyen a (p, g, A) hármás, a privát kulcs pedig a .

Az m üzenet aláírása a következő módon történik. Először választunk egy $1 \leq r \leq p - 2$ véletlenszámot, melyre $\text{l.n.k.o.}(r, p - 1) = 1$, majd meghatározzuk r inverzét mod $(p - 1)$, amit r^{-1} -gyel jelölünk. Ezután kiszámítjuk a következő mennyiségeket:

$$R = g^r \bmod p,$$

$$S = r^{-1}(h(m) - a \cdot R) \bmod (p - 1).$$

Az m üzenet aláírása az (R, S) páros.

Az m üzenet (R, S) aláírásának ellenőrzéséhez, először megszerezzük az aláíró hiteles nyilvános kulcsát, (p, g, A) -t. Első lépésként ellenőrizzük, hogy $0 < R < p$ teljesül-e. Ha nem, akkor az aláírást nem fogadjuk el. Ha az ellenőrzés sikeres, akkor kiszámítjuk a következő mennyiségeket:

$$v_1 = A^R \cdot R^S \bmod p,$$

$$v_2 = g^{h(m)} \bmod p.$$

Az aláírást akkor fogadjuk el, ha $v_1 = v_2$.

Most megmutatjuk, hogy a fenti aláírás-ellenőrző módszer valóban működik. Ha (R, S) hiteles, akkor

$$S \equiv r^{-1}(h(m) - a \cdot R) \pmod{p-1}.$$

Ebből $h(m)$ -re a következő adódik:

$$h(m) \equiv r \cdot S + a \cdot R \pmod{p-1},$$

ahonnan

$$g^{h(m)} \equiv g^{r \cdot S + a \cdot R} \equiv A^R \cdot R^S \pmod{p}.$$

Tehát $v_1 = v_2$ valóban szükséges.

Az ElGamal aláírás séma egy módosított változata a Digital Signature Algorithm (DSA). Az ElGamal aláírás esetén, ha a p modulus 512 bit méretű, akkor az aláírás 1024 bit méretű lesz. Bizonyos alkalmazásokban azonban (pl. intelligens kártyák esetén) rövidebb aláírás kívánatos. A DSA algoritmus úgy módosítja az ElGamal algoritmust, hogy bár a p modulus továbbra is 512 bites, az előállított aláírás mérete csak 320 bit. Ennek módja, hogy a számításokat a Z_p^* egy 2^{160} méretű részcsoportháromban végezzük. Ennek az aláírás sémának a biztonsága azon a hiten alapszik, hogy ezen részcsoporthárombeli diszkrét logaritmus számítási feladat nehéz. A DSA és az ECDSA (elliptikus görbükre alapuló megfelelője) részletes leírása a függelékben található hivatkozáson keresztül érhető el.

7.4. Feladatok

7.1. Feladat. Két, egymás tevékenységében kevésbé bízó ország, A és B megállapodnak, hogy egymás területén elhelyezhetnek olyan érzékelőt, amely bizalmatlanságra okot adható tevékenységekről küld mérési adatokat. Az érzékelőt elhelyező ország (A) biztos akar abban lenni, hogy a mért adatokat adatátvitel során nem hamisítja meg az érzékelőt befogadó fél. B befogadó fél is biztos akar abban lenni, hogy csak olyan adatok kerülnek továbbításra, amelyek a megállapodásban szerepelnek. Milyen hitelesítési megoldást javasolna, ha

1. csak az első feltétel az elvárás,
2. ha mindkettő?

8.

Kulccsere protokollok

Szimmetrikus kulcsú rejtjelezés vagy üzenethitelesítő kódok használata esetén szükséges, hogy a kommunikáló felek rendelkezzenek egy titkos kulccsal, amit rajtuk kívül más nem ismer. Ebben a fejezetben azt vizsgáljuk, hogyan lehet ezt a közös, titkos kulcsot a kommunikáló felek birtokába juttatni.

Nagy vonalakban két követelményt kell kielégítenünk. Egyrészt szeretnénk, ha a kulcs valóban *titkos* maradna, ami alatt azt értjük, hogy a kommunikáló feleken (és megbízható harmadik feleken) kívül a kulcsot más nem ismerheti. Másrészt szeretnénk valamilyen garanciát nyújtani a kommunikáló feleknek a másik fél identitására vonatkozóan. Egészen pontosan azt szeretnénk elérni, hogy ha A azt hiszi, hogy a K kulcs egy közös titkos kulcs B -vel, akkor A valóban B -vel osztja meg K -t, és nem egy harmadik, feltehetően rossz szándékú féllel. Ilyen értelemben tehát, ez utóbbi követelmény a kulcs *hitelességére* vonatkozik.

A kulccsere probléma koncepcionálisan legegyszerűbb megoldása az, mikor A és B fizikailag (pl. személyesen) találkoznak, és megegyeznek egy közös kulcsban. Ez a megoldás – amit manuális kulccserének is hívnak – azonban csak korlátozott mértékben használható a gyakorlatban, mert drága és időigényes, továbbá sokszor az alkalmazás jellegénél fogva egyszerűen nem is használható (pl. A az Interneten keresztül szeretne vásárolni a B külföldi kereskedőtől).

A kulccsere probléma gyakorlatban is jól használható megoldását jelentik a kulccsere protokollok. Egy kulccsere protokoll lehetővé teszi két (vagy több) fél számára egy közös titok létrehozását anélkül, hogy a két fél

fizikailag találkozna. A kulccsere protokollok gyorsabbak és olcsóbbak a manuális kulccserénél, és így azt is lehetővé teszik, hogy a felek sűrűn váltogassák közös titkukat. A rövid élettartamú közös titkot kapcsolatkulcsnak is nevezük, ami arra utal, hogy a kulcsot tipikusan csak egy kapcsolat alatt használják a felek, a kapcsolat bontása után pedig eldobják azt. Ha egy újabb kapcsolatban ismét közös kulcsra van szükségük, akkor újra futtatják a kulccsere protokollt, és létrehoznak egy új kapcsolatkulcsot. A dinamikusan létrehozott kapcsolatkulcsok használatának számos előnye van, köztük a következők:

- A kapcsolatkulcs rövid élettartama miatt egy lehallgató támadónak csak korlátozott mennyiségi egyazon kulccsal rejtelezett üzenet áll rendelkezésére, ami nehezíti a kulcs megfejtését.
- Ha egy kapcsolatkulcs mégis kompromittálódik, akkor az csak korlátozott mennyiségi üzenet megfejtését teszi lehetővé. Továbbá, megfelelő tervezés esetén, a kapcsolatkulcs kompromittálódása nem befolyásolja a hosszú élettartamú mesterkulcsok biztonságát.
- Kapcsolatkulcsot csak akkor hoznak létre a felek, ha valóban szükségük van rá. Így a résztvevőknek kevesebb kulcsot kell biztonságosan tárolniuk és rövidebb ideig.
- A kapcsolatkulcsok alkalmazása függetlenné teszi egymástól a felek közötti különböző kapcsolatokat, ami önmagában kívánatos tulajdonság.

8.1. Kulccsere protokollok osztályozásának szempontjai

Kulcskontroll. A kulccsere protokolloknak alapvetően két fajtája létezik: kulcsszállító (key transport) és kulcsmeggyezés (key agreement) protokollok. Kulcsszállító protokollok esetében a kapcsolatkulcsot a protokoll valamelyik résztvevője (az egyik fél vagy egy megbízható harmadik fél) generálja, majd azt biztonságosan eljuttatja a többi résztvevőnek. A kapcsolatkulcs értéke tehát egy résztvevőtől függ. Ezzel szemben, kulcsmeggyezés protokollok esetében a kapcsolatkulcs értékéhez minden résztvevő hozzájárul. A résztvevők az általuk generált hozzájárulásokat kicserélik, majd minden résztvevő lokálisan generálja a közös kapcsolatkulcsot, a másik résztvevőtől kapott hozzájárulást is felhasználva.

A kulcsmeggyezés protokollok előnye, hogy a kapcsolatkulcs értékét egyik fél sem tudja befolyásolni, hiszen az a másik fél hozzájárulásától is függ. Kulcsszállító protokollok esetében az a résztvevő, amelyik a kulcsot

generálja, szándékosan választhat egy speciális tulajdonságokkal rendelkező (pl. valamelyen értelemben gyenge) kulcsot. A kulcsmegegyezés protokollok hátránya, hogy megvalósításuk általában nyilvános kulcsú kriptografiára épül, így végrehajtásuk nagyobb számítási kapacitást igényel a résztvevőktől. Ennek ellenére, a fent említett kedvező tulajdonságuk miatt, gyakorlati alkalmazásokban (pl. SSL, SSH, IPSec) gyakran használnak kulcsmegegyezés protokollokat.

Kulcsmegegyezés protokollokra a 8.4. szakaszban látunk majd néhány példát, míg a kulcsszállító protokollokkal a 8.3. szakaszban foglalkozunk részletesebben.

Szolgáltatások. A kulcscsere protokollok tovább osztályozhatók az általuk nyújtott szolgáltatások szerint. Ezen szolgáltatások a következők lehetnek:

Implicit kulcshtitelesítés. Egy kulcscsere protokoll akkor nyújt implicit kulcshtitelesítés szolgáltatást valamely A fél számára, ha a protokoll sikeres lefutása után A meg lehet győződve arról, hogy rajta kívül csak a feltételezett másik fél, mondjuk B, és esetleg egy megbízható harmadik fél (pl. a kulcsszerver) férhet hozzá a protokoll során létrehozott kapcsolatkulcschoz. Ez egy olyan alapvető szolgáltatás, amit minden kulcscsere protokolltól elvárunk. Fontos megjegyezni azt, hogy implicit kulcshtitelesítés esetén A nem feltétlenül biztos abban, hogy B ismeri a kapcsolatkulcsot. Csupán annyit követelünk meg, hogy A biztos legyen abban, hogy csak B-nek (és esetleg egy megbízható harmadik félnek) van meg a lehetősége arra, hogy a kapcsolatkulcschoz hozzáférjen.

Kulcskonfirmáció. Ez az a szolgáltatás, amelynek segítségével az egyik résztvevő, mondjuk A, meggyőződhet arról, hogy a másik résztvevő, mondjuk B, valóban birtokában van a protokoll futása során létrehozott kapcsolatkulcsnak. Ezen szolgáltatás megvalósítására több lehetőség is van: B elküldheti például A-nak a kapcsolatkulcs lenyomatát (hash értékét), vagy egy, a kapcsolatkulccsal rejtelezett publikus nyílt szöveget. Lenyomat küldése esetén, a hash függvény egyirányúsága miatt, egy támadó nem tudja megfejteni a kapcsolatkulcsot a megfigyelt hash értékből. Ismert nyílt szöveg rejtelezése esetén pedig a rejtelező függvény ismert nyílt szövegű támadás elleni ellenállóképessége biztosítja ugyanezt.

Explicit kulcshtitelesítés. Explicit kulcshtitelesítésről akkor beszélünk, ha a protokoll egyszerre biztosítja az implicit kulcshtitelesítés és a kulcskonfirmáció szolgáltatásokat ugyanazon fél számára.

Kulcsfrissesség. Ha a protokoll kulcsfrissesség szolgáltatást nyújt az A résztvevő számára, akkor a protokoll sikeres futása után A meg van győződve arról, hogy a létrehozott kapcsolatkulcs új, és nem egy korábban már használt és feltehetően azóta megfejtett kulcs.

Partnerhitelesítés. Kulccsere protokollok opcionálisan nyújthatnak partner-hitelesítés szolgáltatást is. Partnerhitelesítés alatt azt értjük, hogy a protokoll sikeres futása után, az egyik résztvevő meg van arról győződve, hogy a feltételezett másik résztvevő valóban jelen volt, és részt vett a protokoll végrehajtásában. Azok a protokollok, amelyek explicit kulcs-hitelesítés szolgáltatást nyújtanak, gyakran partner-hitelesítést is biztosítanak. A partner-hitelesítést mégis külön szolgáltatásként említjük, mert funkcionálisan különbözik az explicit kulcs-hitelesítéstől.

Szolgáltatásnyújtás iránya. Korábban már utaltunk rá, most azonban konkrétan is megemlíjtük, hogy a fent felsorolt szolgáltatásokat a protokoll nyújthatja, csak az egyik vagy minden fél számára. Ennek megfelelően beszélünk egyirányú vagy kölcsönös implicit kulcs-hitelesítésről, kulcskonfirmációról, kulcsfrissességről, és partner-hitelesítésről. Továbbá, a protokoll nem feltétlenül nyújtja ugyanazokat a szolgáltatásokat minden fél számára. Elképzelhető például (lásd lejjebb a Wide-Mouth-Frog-protokollt), hogy a protokoll az egyik résztvevő számára explicit kulcs-hitelesítést, míg a másik résztvevő számára csak implicit kulcs-hitelesítést biztosít.

Megbízható harmadik fél használata. A kulccsere protokollok további osztályozási szempontja a megbízható harmadik fél használata. Ez alapján megkülönböztetünk protokollokat, melyek egyáltalán nem használnak megbízható harmadik felet, és protokollokat, melyek használnak. Ez utóbbi csoporton belül a protokollokat tovább osztályozhatjuk aszerint, hogy a harmadik fél on-line vagy off-line módon vesz részt a protokollban. On-line harmadik fél esetén a megbízható harmadik fél minden protokollfutásban aktívan részt vesz. Erre példa a Kerberos-protokollban (a protokoll lefrását lásd később a 8.3. szakaszban) használt kulcsszerver. Ezzel szemben, off-line harmadik fél esetén, a protokoll résztvevői nem a protokoll végrehajtásával egyidőben veszik igénybe a megbízható harmadik fél szolgáltatásait. Erre példa a nyilvános kulcs infrastruktúrákban (leírást lásd később a 8.5. szakaszban) használt hitelesítés szolgáltató (Certification Authority – CA).

A részvétel módján kívül fontos az is, hogy pontosan milyen értelemben tekintik megbízhatónak a protokoll résztvevői a harmadik felet. A harma-

dik fél például lehet felelős kapcsolatkulcsok generálásáért, üzenetek hitelességének ellenőrzéséért, üzenetek frissességének ellenőrzéséért, a pontos idő megadásáért, felhasználók személyazonosságának ellenőrzéséért, titkok megőrzéséért stb. Különböző feladatok különböző követelményeket támasztanak a harmadik féllel szemben, és így különböző jellegű bizalmat igényelnek a protokoll résztvevői részéről.

Előzetesen szétosztott információk. A bizalommal kapcsolatban meg kell még említeni a protokoll által használt, de nem a protokoll keretein belül létrehozott vagy szétosztott információkkal kapcsolatos feltevéseket is. Egyetlen kulccsere protokoll sem teremt a semmiből titkos kapcsolatkulcsokat, hanem feltételezi, hogy a résztvevők már rendelkeznek bizonyos bizalmas és/vagy hiteles infomációkkal, és ezeket használva hozzák létre az új kapcsolatkulcsot. Ilyen előzetesen telepített információ lehet például egy hosszú élettartamú titkos kulcs (pl. mesterkulcs), vagy a gyökér hitelesítés szolgáltató (root CA) nyilvános kulcsa. Fontos tehát látni, hogy a protokoll milyen, a résztvevők által már ismert információkra épít, és hogyan történhet ezen információk előzetes szétosztása.

Hatékonyúság. Végül a protokollok osztályozásának fontos gyakorlati szempontja lehet a hatékonyúság. Itt olyan dolgokra kell gondolni, mint a protokoll által használt üzenetek száma, az üzenetek mérete, a protokoll végrehajtásának gyakorisága stb. Ezek a tényezők a protokoll által használt sávszélességet jellemzik, és így bizonyos alkalmazásokban (pl. vezeték nélküli rendszereknél) elsődleges fontosságúak. A szükséges sávszélességen kívül egy másik fontos, a protokoll hatékonyiséggel kapcsolatos szempont a protokoll végrehajtásához szükséges számítási- és tárkapacitás. Ráadásul, a protokollok számítás- és tárigényét résztvevőkre lebontva érdemes vizsgálni, hiszen bizonyos alkalmazásokban az egyes résztvevők kapacitása eltérő lehet (pl. egy mobil hálózatban olyan protokollra lehet szükségünk, mely egy kis számítási kapacitású, mobil eszköz és egy nagy kapacitású, fix szerver között fut). Fontos szempont lehet az is (pl. valós idejű alkalmazásokban), hogy a protokoll mennyire teszi lehetővé off-line előszámítások végzését, mellyel csökkenteni lehet a protokoll on-line végrehajtásához szükséges időt.

8.2. Támadó modell

A támadó képességei. Kulccsere protokollok vizsgálata esetén a támadó képességeivel kapcsolatban két nagyon fontos feltételezést teszünk. Az egyik feltételezésünk az, hogy a támadó teljes mértékben kontrollálni tudja a protokoll résztvevőinek kommunikációját. Ez alatt egyrészt azt értjük, hogy a támadó minden üzenetet képes lehallgatni, módosítani és törölni, valamint a támadó képes olyan hamis üzenetek generálására is, amit a rendelkezésére álló információk lehetővé tesznek (pl. ismert kulcsokkal tud rejtjelezni). Ezen képességeiből természetesen következik az is, hogy a támadó képes üzeneteket megfigyelni, megjegyezni és később visszajátszani. Lényegében tehát a résztvevők a támadón keresztül kommunikálnak egymással, azaz üzeneteiket a támadónak küldik, és a támadótól kapják. Másrészt, a támadó nemcsak az üzeneteket tudja manipulálni, hanem bármelyik résztvevőt be tudja vonni a protokoll végrehajtásába, sőt még arra is rá tud venni egy becsületes résztvevőt, hogy az kezdeményezze a protokoll végrehajtását a támadóval vagy egy másik becsületes résztvevővel.

A gyakorlatban ritkán van arra lehetősége egy támadónak, hogy a résztvevők teljes kommunikációját kontrollálni tudja. A fenti modell tehát óvatos (vagy pessimista) tervezői szemléletet tükröz, abban az értelemben, hogy a várhatónál nagyobb hatalmat biztosít a támadónak. Ezen modell használatától azt reméljük, hogy ha a protokoll ellenáll egy ilyen erekű (elméleti) támadónak, akkor ellen fog állni a gyakorlatban előforduló támadásoknak is.

Ezzel szemben a támadóval kapcsolatos másik fontos feltevésünk az, hogy a támadó nem tudja feltörni a protokoll által használt kriptográfiai építőelemeket (pl. rejtjelezőket, hash függvényeket, digitális aláírásokat stb.). Más szóval azt feltételezzük, hogy a támadó nem az építőelemek feltörésével próbálja elérni célját, hanem abban keresi a gyengeséget, ahogyan a protokoll ezeket az építőelemeket egymással kombinálja, használja. Az építőelemek fekete dobozként történő kezelése természetesen egy egyszerűsített támadó modellt eredményez, de mint látni fogjuk, még ez is számos érdekes észrevételekre vezet, és hasznos tanulságokkal szolgál a protokollok tervezői számára.

Támadás típusok. Fenti képességeit használva a támadó többfajta támadást is végrehajthat. Az egyszerűbb támadások közé tartozik a passzív lehallgatás, mely során a támadó csak megfigyeli a résztvevők üzeneteit, de nem módosítja azokat. Ilyen típusú támadások ellen a legtöbb protokoll megfelelően védekezik.

Egy fokkal bonyolultabbak azok a támadások, amikor a támadó nemcsak megfigyeli a protokoll futása során küldött üzeneteket, hanem aktívan be is avatkozik a protokollba üzenetek módosításával és törlésével. Ennek egy gyakori fajtája az ún. visszajátszásos támadás (replay attack), mely során a támadó a protokoll egyes üzeneteit a protokoll korábbi futásai során lehallgatott megfelelő üzeneteivel helyettesíti.

A legerősebb támadások, melyet a támadó fenti képességei lehetővé tesznek, azok a támadások, amikor a támadó maga is részt vesz a protokoll végrehajtásában. Ekkor természetesen feltételezzük azt, hogy a támadó rendelkezik a protokoll végrehajtásához szükséges információkkal (pl. a támadó egy legális felhasználó, akinek saját, a rendszer által elfogadott kulcsai vannak). A leggyakoribb ilyen jellegű támadás az ún. parallel session, vagy más néven interleaving támadás, mely során a támadó a protokoll több példányát is futtatja egyszerre, egy vagy több másik résztvevővel, és az egyes példányok futása során szerzett ismereteit más példányokban használja fel. Tipikusan a támadó az egyik protokoll példányban lehallgatott üzenetet vagy üzenetrészt egy másik konkurrens példányban játsza vissza.

Az interleaving jellegű támadásokkal szemben sok, nyilvános irodalomban és szabványban publikált protokoll védtelen. Ennek fő oka az, hogy az ilyen jellegű lehetséges támadások száma óriási (pl. ha nincs korlátozva, hogy a résztvevők hány protokoll példányt futtathatnak egyszerre, akkor a vizsgálandó esetek száma nem korlátos), és így a protokoll tervezője nem tudja egyszerűen ellenőrizni, hogy a protokoll minden lehetséges támadásnak ellenáll-e. Ezen probléma megoldására két hozzáállás vált elfogadottá: az empirikus és a formális módszer. Az empirikus módszer lényege, hogy a sok éves tervezői gyakorlat során nyert tapasztalatokat informális protokoll tervezési elvekben (ökölszabályokban) összegezzük, és ezen elvek követésével igyekszünk új protokollokat tervezni. Ezt a módszert a 8.6. szakaszban tárgyaljuk részletesen. A formális módszer alapgondolata, hogy a protokollt és a protokoll céljait precízen leírjuk valamilyen matematikai modellben, majd formális bizonyítást adunk arra, hogy a protokoll eléri a definiált célokot, és így az adott matematikai modellben korrektnek mondható. Erre a módszerre a 8.7. szakaszban mutatunk példát.

A támadó célja. A támadó célja tipikusan a protokoll által létrehozott kapcsolatkulcs megszerzése, vagy annak elhitetése egy becsületes *A* résztvevővel, hogy *A* a kapcsolatkulcsot egy másik becsületes *B* résztvevővel hozta létre, miközben valójában a támadóval osztja meg azt. Ha a fentieket nem

tudja elérni, a támadó célja lehet még az is, hogy egyszerűen megtévessze a becsületes résztvevőket a másik fél kilétét illetően (pl. elhiteti A-val, hogy B -vel hozta létre a kapcsolatkulcsot, miközben valójában C -vel futtatta a protokollt). Ez utóbbi esetben a támadó nem jut hozzá a kapcsolatkulcschoz, ennek ellenére a becsületes résztvevők ily módon történő megtévesztése (Denial-of-Service jellegű) támadásnak minősül, mert nem kívánatos következményekhez vezethet.

A támadó rendelkezésére álló információk. A támadó rendelkezésére álló információk tekintetében általában azt feltételezzük, hogy a nyilvánosan elérhető információkon kívül a támadó nem rendelkezik további információkkal. Ez tipikusan azt jelenti, hogy a támadó számára nem ismertek a protokoll által használt hosszú élettartamú kulcsok, a becsületes résztvevők privát kulcsai, a korábbi protokollfutások során létrehozott kapcsolatkulcsok stb.

Néha azonban érdekes megvizsgálni azt is, mi történik akkor, ha valamilyen feltételezett titok kopromittálódik, és mégis a támadó kezébe jut. Feltételezhetjük például, hogy a támadó valamilyen módon megszerzett egy (vagy több) korábbi kapcsolatkulcsot, és vizsgálhatjuk, hogy ez lehetővé teszi-e számára jövőbeli kapcsolatkulcsok megszerzését. Feltételezhetjük azt is, hogy a támadó megszerezte valamelyik becsületes résztvevő hosszú élettartamú vagy privát kulcsát. Ekkor a támadó birtokában van minden olyan információ, mely lehetővé teszi számára minden olyan jövőbeli kapcsolatkulcs megszerzését, melyet a kompromittálódott résztvevővel hoznak létre. Kérdés azonban, hogy a kompromittálódott résztvevő múltbeli kapcsolatkulcsait meg tudja-e fejteni a támadó. Erős protokolloktól természetesen azt várjuk, hogy a fenti kérdésekre a válasz nemleges.

8.3. Példák kulcsszállító protokollokra

Ebben a szakaszban néhány ismertebb kulcsszállító protokollt mutatunk be. Célunk egyrészt konkrét protokollok ismertetése, másrészt ezen protokollokon keresztül szeretnénk szemléltetni az eddig bevezetett fogalmakat. Küllönös figyelmet fordítunk a különböző támadás típusok bemutatására, ezért amelyik bemutatott protokollnak létezik ismert támadása, ott a protokoll ismertetése után a támadást is ismertetjük, és javaslatot teszünk a protokoll javítására. Meggyőződésünk, hogy ebből a gyakorlatból igen hasznos tapasztalatokat lehet szerezni.

A protokollok bemutatása során az irodalomban használatos jelöléseket használjuk. Ha a protokoll i -edik üzenete msg , melyet A küld B -nek, akkor azt írjuk, hogy

$$(i) \quad A \rightarrow B : \quad msg.$$

A becsületes résztvevőket általában A -val és B -vel, a megbízható harmadik felet S -sel vagy TTP -vel jelöljük. A támadó jelölésére általában az X -et használjuk, és X_A -t írnak, mikor ki szeretnénk hangsúlyozni, hogy a támadó az A becsületes fél nevében tesz valamit (pl. üzenetet küld vagy vesz). A résztvevők nevének üzeneteken belüli használata a résztvevők azonosítóját jelenti. A protokoll által létrehozott kapcsolatkulcsot k -val, a frissen generált véletlenszámokat N -nel, az időpecséteket pedig T -vel jelöljük, alsó indexbe téve a generáló résztvevő nevét (pl. N_A).

8.3.1. A Wide-Mouth-Frog-protokoll

Kezdjük mindenkor egy igen egyszerű protokollal, mely az irodalomban a Wide-Mouth-Frog-protokoll néven ismert. A protokollnak három résztvevője van, A , B és S , ahol A és B egyszerű résztvevők, akik egy kapcsolatkulcsot szeretnének létrehozni egymás között az S megbízható harmadik fél (szerver) segítségével. A protokoll minden össze két üzenetből áll, melyek a következők:

Wide-Mouth-Frog-protokoll

- | | | |
|-----|---------------------|---|
| (1) | $A \rightarrow S :$ | $A \mid \{B \mid k \mid T_A\}_{K_{AS}}$ |
| (2) | $S \rightarrow B :$ | $\{A \mid k \mid T_S\}_{K_{BS}}$ |
-

Mint látható, a Wide-Mouth-Frog-protokoll egy kulcsszállító protokoll. A k kapcsolatkulcsot A generálja, majd a szerver segítségével biztonságosan eljuttatja azt B -hez. A szerverre azért van szükség, mert A és B nem rendelkezik közös titokkal, mely a kapcsolatkulcs biztonságos átvitelét lehetővé tenné. Ezzel szemben a protokoll feltételezi, hogy A és S , valamint B és S rendelkezik ilyen K_{AS} illetve K_{BS} titokkal. Így A a K_{AS} kulcsot használva el tudja juttatni k -t S -hez, aki a K_{BS} kulcsot használva továbbítja azt B -nek.

Az üzenetek tartalmát a következőképpen magyarázhatjuk részletesebben. Az első üzenetben A elküldi S -nek B azonosítóját, a frissen generált k kapcsolatkulcsot, és egy friss T_A időpecsétet a K_{AS} kulccsal rejtjelezve. Az első üzenet részét képezi még A azonosítója, mely nyílt formában jelenik meg. Ennek az azonosítónak csak annyi a szerepe, hogy S számára tippet adjon, melyik kulccsal próbálja meg dekódolni a rejtjelezett részt, hiszen S felte-

hetően több résztvevőtől is kaphat ilyen formájú üzenetet. Ha egy támadó e nyílt azonosítót megváltoztatja, akkor S nem K_{AS} -sel dekódolja a rejtjelezett részt, és így eredményül egy pénzfeldobás sorozatból álló, használhatatlan nyílt üzenetet állít vissza, amit feltehetően felismer és egyszerűen eldob.

Mikor S megkapja az első üzenetet, akkor a benne található nyílt azonosító alapján előkeresi a K_{AS} kulcsot, és dekódolja a rejtjeles részt. S az üzenet formátumának megvizsgálásával ellenőrzi, hogy a dekódolás sikeres volt-e. Ha az üzenet egy azonosítóból, egy véletlen bitsorozatból és egy időpecsétből áll, akkor S nem dobja el az üzenetet, hanem folytatja annak feldolgozását. Természetesen a fenti mezők pontos felismerése problémát jelenthet (pl. ha az azonosítók változó hosszságú véletlen bitsorozatok, akkor S nem tudja, hogy hol végződik az azonosító, és hol kezdődik a kapcsolatkulcs), ezért a protokoll konkrét implementációja rögzítheti az egyes mezők hosszát vagy gondoskodhat megfelelő mezőszeparátorok típusindikátorok, és egyéb szükséges redundancia alkalmazásáról, mely lehetővé teszi a dekódolt üzenet egyértelmű értelmezését.

A dekódolás után S ellenőrzi a T_A időpecsétet. Ha az S lokális órája által jelzett időpont és T_A között túl nagy az eltérés, akkor S eldobja az üzenetet. Azt, hogy mi számít túl nagy eltérésnek, a protokoll nem specifikálja, hiszen ez az alkalmazás jellegétől is függhet. A protokoll konkrét implementációjának azonban nyilván rögzítenie kell az elfogadható maximális eltérést.

Ha az üzenet minden ellenőrzésen sikeresen átmegy, akkor S az üzenetben található B azonosító alapján előkeresi a K_{BS} kulcsot, majd azzal rejtjelez az A azonosítóját, a k kapcsolatkulcsot és egy frissen generált T_S időpecsétet, és az így nyert rejtjeles üzenetet elküldi B -nek. Figyeljük meg, hogy a második üzenet nem tartalmaz tippként használható nyílt azonosítót, hiszen S -sel ellentétben B -nek csak egyetlen kulcsa van, K_{BS} , amivel a protokoll során vett üzenet dekódolását megpróbálhatja.

A második üzenet vétele után B hasonló ellenőrzéseket hajt végre, mint S az első üzenet vételekor. Azaz B először megpróbálja dekódolni az üzenetet a K_{BS} kulccsal, majd ellenőrzi a visszaállított nyílt üzenet formátumát és a T_S időpecsétét. Ha a dekódolás sikeres volt, és az időpecsét nem túl régi, akkor B elfogadja az üzenetet, és ezzel együtt a k kulcsot. Más szavakkal, B meg van győződve arról, hogy k egy frissen létrehozott kapcsolatkulcs, melyet az A féllel oszt meg. Ez utóbbi tényre a második üzenetben található A azonosító enged következtetni.

Vizsgáljuk most meg a Wide-Mouth-Frog-protokoll tulajdonságait.

- **Kulcskontroll:** Mint korábban említettük, a protokoll a kulcsszállító protokollok osztályába tartozik. A kapcsolatkulcsot A generálja, és a szer- veren keresztül eljuttatja B-nek. A kulcs értékét tehát teljes egészében A kontrollálja. Ez nyilván azt is jelenti, hogy B-nek meg kell bízna abban, hogy A jó minőségű (megfelelően nem predikálható) kulcsokat tud gene- rálni.
- **Kulcshitelesítés:** A protokoll A számára implicit kulcshitelesítést bizto- sít. Ez azért van, mert a k kulcsot A K_{AS} -sel rejtjelezve küldi S-nek, S pedig K_{BS} -sel rejtjelezve küldi B-nek. Továbbá, A megbízik S-ben, hogy az a protokollnak megfelelően kizárolag az első üzenetben jelzett másik félnek, jelen esetben B-nek küldi tovább k -t. A k kulcsot tehát S-en és B- n kívül más nem ismerheti meg. Ugyanakkor A nem lehet biztos abban, hogy B valóban birtokában van a k kulcsnak, hiszen semmilyen vissza- jelzést nem kap erről a protokoll során. A protokoll tervezői valószínűleg úgy gondolkodtak, hogy a B által küldött első k -val kódolt üzenet bizto- sítja majd a kulcskonfirmációt, és ezért nem szükséges azt a protokollen belül megoldani. Ez elfogadható, ha az első k -val kódolt üzenet megfelelő mennyiségű redundanciát tartalmaz, mely lehetővé teszi A számára annak eldöntését, hogy valóban egy k -val kódolt üzenetet kapott. Mivel az üzenetek általában integritásvédelmet szolgáló üzenethitelesítő kódot is tartalmaznak, ezért a legtöbb esetben a redundancia biztosított, és a fenti tervezői hozzáállás elfogadható.

B számára viszont a protokoll explicit kulcshitelesítést biztosít. Ez azért van, mert A-hoz hasonlóan B is meg lehet arról győződve, hogy a k kulcsot A-n és S-en kívül más nem ismerheti, és mivel B tudja, hogy k -t maga A generálta, ezért abban is biztos lehet, hogy A birtokában van a k kulcsnak. A kulcskonfirmációt tehát jelen esetben a protokoll második üzenetének vétele biztosítja B számára.

- **Kulcs frissesség:** A számára nyilván biztosított a k kulcs frissessége, hiszen ő maga generálja k -t. B számára k frissességét időpecsétek alkalma- zásával próbálja biztosítani a protokoll. Első ránézésre úgy tűnik, hogy az időpecsétek elérik céljukat. Mikor B megkapja a második üzenetet, és T_S nem túl régi, akkor B biztos lehet abban, hogy S nem túl régen küldte k -t. Továbbá, B megbízik abban, hogy S becsületesen ellenőrizte a T_A időpe- csétet az első üzenetben, és hogy S nem késlelteti a protokoll futását. Ez azt jelenti, hogy B meg van arról győződve, hogy A nem túl régen küldte k -t, azaz k friss.

Sajnos azonban a protokoll mégsem biztosítja a k kulcs frissességét B számára. Alább részletesen tárgyaljuk a támadást, mely lehetővé teszi a támadó számára tetszőlegesen régi k elfogadatását B -vel. E támadás két fontos doogra világít rá. Egyszer azt szemlélteti, hogy a kulccsere protokollok elleni támadások milyen körmönfontak, és következésképpen nehezen felfedezhetők lehetnek. Másrészről azt is mutatja, hogy a protokoll helyességével kapcsolatos informális érvelés mennyire megbízhatatlan, és ilyen formán formális módszerek alkalmazását motiválja.

- **Megbízható harmadik fél:** Korábban említettük, hogy S egy megbízható harmadik fél, és az eddigi tárgyalásból az is világosan látszik, hogy S online módon vesz részt a protokollban. Többször utaltunk arra is, hogy S -et milyen szempontból tekintik megbízhatónak a protokoll többi résztvevői, de most még egyszer összefoglaljuk az S megbízhatóságával kapcsolatos két fontos feltevést:
 - Egyszer S -ről feltételezzük, hogy a k kulcsot nem fedi fel más számára, hanem kizártlag az első üzenetben megadott másik félnek küldi azt el. Ehhez kapcsolódóan megemlítjük még azt a feltevést is, hogy S helyesen kezeli az azonosítókat, azaz a második üzenetben az első üzenet küldőjének azonosítóját küldi el.
 - Másrészről feltételezzük, hogy megfelelően ellenőrzi az első üzenetben található időpecsétet, és helyes időpecséttel láta el a második üzenetet. Feltessük továbbá azt is, hogy S nem késlelteti a protokoll futását, azaz nem vár indokolatlanul sokat az első üzenet vétele és a második üzenet küldése között.

Bár A nem számít harmadik félnek, bizonyos szempontból neki is megbízhatónak kell lennie. Egészen pontosan arról van szó, hogy a protokoll feltételezi, hogy A megfelelő minőségű és friss kapcsolatkulcsokat generál.

- **Rendelkezésre álló információk:** A protokoll feltételezi, hogy A és S , illetve B és S rendelkezik egy hosszú élettartamú titkos K_{AS} , illetve K_{BS} kulccsal. Ezen kulcsok eljuttatása A -hoz és S -hez, illetve B -hez és S -hez nem a protokoll feladata.

Ezenkívül az időpecsétek ellenőrizhetőségének érdekében minden fél lokális órájáról feltételezzük, hogy egy globális órához van szinkronizálva. Az órák szinkronizálása szintén nem a protokoll feladata. A szinkronizáció szükséges pontosságát egyszer az alkalmazás jellege, másrészről a megkövetelt biztonság mértéke határozza meg. Az elérő pontosság termé-

szetesen hatással van az időpecsétek elfogadási intervallumára is (minél pontatlanabbak az órák, annál nagyobb intervallumot kell megengedni).

8.3.1.1. A Wide-Mouth-Frog-protokoll támadása

Az időpecsétek alkalmazása ellenére a Wide-Mouth-Frog-protokoll nem biztosít kulcsfrissességet B számára, ugyanis egy támadó tetszőlegesen régi k kulcsot el tud fogadtatni B -vel. A támadás menete a következő:

$$\begin{aligned}
 A &\rightarrow S: A \mid \{B \mid k \mid T_A\}_{K_{AS}} \\
 S &\rightarrow B: \{A \mid k \mid T_S\}_{K_{BS}} \\
 X_B &\rightarrow S: B \mid \{A \mid k \mid T_S\}_{K_{BS}} \\
 S &\rightarrow X_A: \{B \mid k \mid T_S^{(2)}\}_{K_{AS}} \\
 X_A &\rightarrow S: A \mid \{B \mid k \mid T_S^{(2)}\}_{K_{AS}} \\
 S &\rightarrow X_B: \{A \mid k \mid T_S^{(3)}\}_{K_{BS}} \\
 &\dots \\
 X_A &\rightarrow S: A \mid \{B \mid k \mid T_S^{(n-1)}\}_{K_{AS}} \\
 S &\rightarrow B: \{A \mid k \mid T_S^{(n)}\}_{K_{BS}}.
 \end{aligned}$$

Az X támadó először lehallgat egy A és B között futó protokollpéldányt. Utána a lehallgatott második üzenetet felhasználva generál egy hamis első üzenetet B nevében, és elküldi azt S -nek. Ha X elég gyors, akkor S még elfogadja a visszajátszott üzenetben található T_S időpecsétet, amit ő maga generált. S a kapott üzenetet úgy értelmezi, mint B próbálkozását egy kapcsolatkulcs felépítésére A -val, ezért a protokollnak megfelelő választ generálja, amit a $T_S^{(2)}$ időpecséttel lát el, ahol $T_S^{(2)} > T_S$. S a választ A -nak küldi, de mivel X kontrollálja a kommunikációt, ezért az üzenetet elkapja, és az soha nem jut el A -hoz. Ezután X az elkapott üzenetet felhasználva generál egy hamis első üzenetet A nevében, és elküldi azt S -nek. Ha X ismét elég gyors, akkor S még elfogadja az üzenetben található $T_S^{(2)}$ időpecsétet, amit ő maga generált. S a kapott üzenetet úgy értelmezi, mint A próbálkozását egy kapcsolatkulcs felépítésére B -vel, ezért a protokollnak megfelelő választ generálja, amit a $T_S^{(3)}$ időpecséttel lát el, ahol $T_S^{(3)} > T_S^{(2)}$. S választ X ismét elkapja, és így tovább. X a végtelenségig használhatja S -et az időpecsét frissítésére, és közben megpróbálhatja megfejteni vagy csalárd módon megszerezni k -t. Mikor k megvan, akkor X végül megengedi, hogy S B -nek szánt üzenete, mely egy friss $T_S^{(n)}$ időpecsétet tartalmaz, eljusson B -hez. B ellenőrzi a kapott üzenetet, és mivel az S -től származik és frissnek tűnik, ezért elfogadja k -t, mint egy A -től származó friss kapcsolatkulcsot.

8.3.1.2. A Wide-Mouth-Frog-protokoll javítása

A fenti támadás létezésének alapvetően két oka van: egyrészt az, hogy a Wide-Mouth-Frog-protokoll szimmetrikus kulcsú rejtjelezést használ, másrész pedig az, hogy a protokoll üzeneteinek rejtjelezett része teljesen azonos formátumú. Ezen két tényező szerencsétlen kombinációjából adódik az, hogy a protokoll első üzenetének rejtjelezett része visszajátszható második üzenetként, és fordítva. Általános tanulságként megállapíthatjuk tehát, hogy szimmetrikus kulcsú rejtjelezés (vagy üzenethitelesítés) használata esetén különös gondot kell fordítani arra, hogy a résznevők felismerjék a saját maguk által generált üzeneteket. Ez megoldható az üzenetek formátumának körültekintő megválasztásával, irányjelző bitek használatával, vagy irányonként különböző kulcsok használatával. Példaként, a Wide-Mouth-Frog-protokollt irányjelző bitek használatával javítjuk ki:

Javított Wide-Mouth-Frog-protokoll

- (1) $A \rightarrow S : A | \{0 | B | k | T_A\}_{K_{AS}}$
 - (2) $S \rightarrow B : \{1 | A | k | T_S\}_{K_{BS}}$
-

A javított protokollban minden rejtjelezett rész ki van egészítve egy bittel, mely az üzenet szándékolt irányát jelöli. Az első üzenetben található 0 bit például azt jelenti, hogy ez egy S -nek szánt üzenet, a második üzenetben található 1 bit pedig arra utal, hogy ez egy S -től származó üzenet. Természetesen S nem fogad el olyan üzenetet, melyben az irányjelző bit értéke 1. Az olvasóra bízzuk annak ellenőrzését, hogy a javított protokoll ellenáll-e a fenti támadásának.

8.3.2. A szimmetrikus kulcsú Needham–Schroeder-protokoll

Következő példánk a szimmetrikus kulcsú Needham–Schroeder-protokoll. A protokolلت 1978-ban publikálta Needham és Schroeder, és azóta folyamatosan jelen van az irodalomban mint a szimmetrikus kulcsú kulcsszállító protokollok jól ismert mintapéldája. Számos protokoll tervezési elv magyarázatánál és protokoll analízis módszer bemutatásánál használták szemléltető céllal. Továbbá, a Needham–Schroeder-protokoll egy módosított változata alkotja a széles körben használt Kerberos-rendszer alapját. A Wide-Mouth-Frog-protokolltól három fontos ponton különbözik: (a) a kapcsolatkulcsot nem A (vagy B), hanem egy megbízható harmadik fél generálja, (b) a kulcsfrissességet nem időpecsékkel igyekszik biztosítani, hanem frissen gene-

rált véletlen számokkal és (c) a protokollnak része a kulcskonfirmáció. A protokoll üzenetei a következők:

Szimmetrikus kulcsú Needham–Schroeder-protokoll

- (1) $A \rightarrow S : A \mid B \mid N_A$
 - (2) $S \rightarrow A : \{N_A \mid B \mid k \mid \{k \mid A\}_{K_{BS}}\}_{K_{AS}}$
 - (3) $A \rightarrow B : \{k \mid A\}_{K_{BS}}$
 - (4) $B \rightarrow A : \{N_B\}_k$
 - (5) $A \rightarrow B : \{N_B - 1\}_k$
-

A protokollnak három résztvevője van, A és B , akik egy új kapcsolatkulcsot szeretnének létrehozni egymás között, és S , aki megbízható harmadik félként segédkezik a kapcsolatkulcs létrehozásában. A protokoll végrehajtását A kezdeményezi, aki először generál egy friss N_A véletlenszámot, majd elküldi A és B azonosítóját, valamint N_A -t S -nek. Az első üzenet tehát egy kérés, amiből S tudomást szerez arról, hogy A akar egy új kapcsolatkulcsot létrehozni B -vel, és a kapcsolatkulcs frissességének bizonyítására az N_A számot kell használni.

Ekkor S generál egy k kapcsolatkulcsot, majd a második üzenettel válaszol A -nak. Ez az üzenet két részből áll. Az A -nak szóló rész tartalmazza az N_A számot, B azonosítóját és a k kulcsot. Az üzenet többi része B -nek szól, ez tartalmazza a k kapcsolatkulcsot és A azonosítóját, és rejtjelezve van a K_{BS} kulccsal, melyet csak B és S ismer. Továbbá, a teljes második üzenet (beleértve a B -nek szóló rejtjelezett részt is) rejtjelezve van a K_{AS} kulccsal, melyet csak A és S ismer.

A dekódolja a második üzenetet a K_{AS} kulccsal. A dekódolás helyességéről az üzenet formátumának ellenőrzésével, vagy implementációtól függően, további redundancia ellenőrzésével győződik meg. Ha a dekódolás helyes, akkor A biztos benne, hogy az üzenetet S generálta, hiszen A -n kívül más nem tud K_{AS} -sel rejtjelezni, annak pedig elenyészően kicsi a valószínűsége, hogy valaki képes legyen véletlen módon olyan rejtjeles üzenetet generálni, mely K_{AS} -sel dekódolva helyes formátumú, értelmes, nyílt üzenetet eredményez. Dekódolás után A ellenőrzi, hogy S válasza tartalmazza-e a korábban elküldött N_A számot. Mivel A frissen generálta N_A -t, ezért annak értékét S nem tudja predikálni. Más szóval, ha a második üzenet tartalmazza N_A -t, akkor A biztos lehet abban, hogy a második üzenet N_A felfedése, azaz az első üzenet elküldése után keletkezett. Ilyen értelemben tehát a második üzenet nem túl régi, és így A a k kulcsot frissnek fogadja el.

A továbbítja B -nek a neki szóló rejtjelezett részt. B a K_{BS} kulccsal dekódolja a harmadik üzenetet. A-hoz hasonlóan, ő is formátum-ellenőrzéssel, vagy egyéb redundancia ellenőrzésével győződik meg a dekódolás helyességéről. A visszaállított nyílt üzenetből B értesül arról, hogy A szeretne vele kapcsolatot teremteni, és S a k kulcsot generálta a kapcsolat védelmére. Az üzenet azonban nem tartalmaz semmit, ami meggyőzhetné B -t arról, hogy k friss. Ezt egy támadó ki tudja használni egy régi, kompromittálódott kapcsolatkulcs elfogadtatására B -vel. A támadást alább részletezzük, most azonban folytatjuk a további üzenetek értelmezését.

Ezen a ponton A is és B is rendelkezik az S által generált k kapcsolat-kulccsal. Azonban még nem tudják biztosan, hogy a másik fél megkapta-e k -t¹. Az utolsó két üzenet célja a kulcskonfirmáció biztosítása. Ehhez B generál egy friss N_B véletlenszámot, és rejtjelezzi azt k -val, majd az eredményt elküldi A -nak. A dekódolja a kapott üzenetet, az eredményül kapott számot eggyel csökkenti, és az eredményt k -val rejtjelezve visszaküldi B -nek. B dekódolja a kapott üzenetet, és ellenőrzi, hogy $(N_B - 1)$ -et kapta-e meg.

A számára a kulcskonfirmáció akkor biztosított, ha képes meggyőződni arról, hogy a jó kulcsot használta a negyedik üzenet dekódolásánál. Ehhez az kell, hogy a negyedik üzenet elegendő redundanciát tartalmazzon. Mivel N_B véletlenszám, ezért egy helyes implementáció nem pusztán N_B -t rejtjelez a negyedik üzenet előállításakor, hanem gondoskodik a megfelelő redundancia hozzáadásáról.

Bár B számára a kulcskonfirmáció már a harmadik üzenet vételekor biztosított (lásd korábbi lábjegyzet), az ötödik üzenet további konfirmációval szolgál. Ezenkívül, valaki azt is gondolhatná, hogy az utolsó üzenet B számára is biztosítja a k kulcs frissességét. A gondolatmenet a következő: Mivel N_B nem predikálható, ezért B meg lehet győződve arról, hogy az utolsó üzenet friss (azaz a negyedik üzenet elküldése után keletkezett), és mivel az utolsó üzenet közvetve tartalmazza k -t (azzal van rejtjelezve), ezért k -nak is frissnek kell lennie. Ez a gondolatmenet azonban hibás: attól, hogy egy kulcsot valaki nem túl régen használt, a kulcs maga még lehet nagyon régi. Erre a 8.6. szakaszban még vissza fogunk térní.

¹ Egészen pontosan A nem tudja, hogy B megkapta-e k -t. Vegyük észre azonban, hogy a harmadik üzenet vételekor B már tudja, hogy A ismeri k -t, hiszen a második üzenetben található egymásba ágyazott rejtjelezés miatt, A csak a második üzenet dekódolása után tudja elküldeni a harmadik üzenetet. A második üzenet dekódolásával azonban A nemcsak a B -nek szóló részhez jut hozzá, hanem a k kulcsot is látnia kell.

A fentieket összegezve, a Needham–Schroeder-protokoll a következő tulajdonságokkal rendelkezik:

- a kulcsszállító protokollok osztályába tartozik, ahol a kapcsolatkulcs értékét egy megbízható harmadik fél, az S kulcszerver kontrollálja;
- A és B számára explicit kulcs-hitelesítést (és partner-hitelesítést) biztosít;
- csak A számára biztosít kulcsfrissességet;
- S on-line módon vesz részt a protokollban;
- S -ről feltételezzük, hogy jó minőségű, friss kapcsolatkulcsot generál, és nem fedi azt fel, csak A -nak és B -nek;
- A és S , valamint B és S már rendelkeznek egy K_{AS} illetve egy K_{BS} közös titkos kulccsal.

8.3.2.1. A szimmetrikus kulcsú Needham–Schroeder-protokoll támadása

Mint korábban említettük, a protokoll hibája az, hogy B számára a kapcsolatkulcs frissessége nem biztosított. Egy támadó ezt a következőképpen tudja kihasználni. Tegyük fel, hogy az X támadó lehallgatta a protokoll egy példányát, melyben A és B a k kapcsolatkulcsot hozta létre. Később a támadó valamilyen módon hozzájut a k kulcschoz (pl. A vagy B rendszeréből k valamilyen hiba folytán kiszivárog). Ekkor X visszajátsza a lehallgatott protokoll példány harmadik üzenetét B -nek. B azt hiszi, A próbál kapcsolatot teremteni vele, és elfogadja a k kapcsolatkulcsot. B generál egy N'_B véletlenszámot, rejtjelezzi azt k -val, és az eredményt elküldi A -nak. X azonban elfogja az üzenetet, így A soha nem kapja azt meg. Helyette X válaszol B -nek. Meg tudja tenni, hiszen ismeri a k kulcsot. Végül B azt hiszi, sikeresen futatta a protokollt A -val, és bizalmas adatokat küld neki k -val rejtjelezve. A helyett azonban X olvassa B titkos üzeneteit.

8.3.2.2. A Kerberos-protokoll

A Needham–Schroeder-protokollt többféleképpen is ki lehet javítani. Egy lehetséges javítás Kerberos-protokoll néven vált ismertté, és a Kerberos hozzáférésvédelmi rendszer alapját képzi.

A Kerberos-protokoll a Needham–Schroeder-protokoll hibáját időpecsét alkalmazásával javítja ki. Figyeljük meg a második üzenet K_{BS} -sel kódolt, B -nek szóló részében, illetve a harmadik üzenetben az L paraméter alkalmazását. L egy időintervallumot jelöl, mely meghatározza, hogy a k kapcsolatkulcs mettől meddig érvényes. Így B csak ezen intervallumon belül fogadja

el a k -t tartalmazó harmadik üzenetet. A támadó még mindig próbálkozhat a harmadik üzenet visszajátszával B felé, de ha L -et megfelelően rövidre választjuk, akkor a k kulcsot gyakorlatilag nem tudja az üzenet érvényességi ideje alatt megszerezni, és így az üzenet második részét (a k -val kódolt T_A időpecsétet) nem tudja előállítani.

Kerberos-protokoll

- (1) $A \rightarrow S : A \mid B \mid N_A$
 - (2) $S \rightarrow A : \{k \mid N_A \mid L \mid B\}_{K_{AS}} \mid \{k \mid A \mid L\}_{K_{BS}}$
 - (3) $A \rightarrow B : \{k \mid A \mid L\}_{K_{BS}} \mid \{A \mid T_A\}_k$
 - (4) $B \rightarrow A : \{T_A\}_k$
-

Egyébként a Kerberos-protokollban a $\{k \mid A \mid L\}_{K_{BS}}$ üzenetrész visszajátszása legális dolog, ugyanis ezt az üzenetrészt az A felhasználó a B erőforráshoz történő hozzáférést engedélyező jegyként használja. Más szavakkal, az L időintervallumon belül A többször használhatja e jegyet a B -hez való hozzáférés engedélyeztetéséhez. Természetesen A minden alkalommal egy új T_A időpecsét rejtelezésével hitelesíti magát.

Vegyük észre még azt is, hogy a Kerberos-protokoll második üzenetében a rejtelezések nem egymásba ágyazottak, mint azt a Needham–Schroeder-protokollnál láttuk. Ez a változtatás a protokoll hatékonyságát javítja, ugyanis így kevesebb bitet kell a K_{AS} kulccsal rejtelezni. Ugyanakkor a protokoll tulajdonságai nem változnak. A kettős rejtelezés hatása a Needham–Schroeder-protokollban az volt, hogy B számára a harmadik üzenet vétele biztosította a kulcskonfirmációt (azaz B a harmadik üzenet vételekor tudta, hogy A már látta a k kulcsot). A Kerberos-protokollban ugyanezt a harmadik üzenet második felében található, k -val rejtelezett T_A időpecsét biztosítja.

8.3.3. A nyilvános kulcsú Needham–Schroeder-protokoll

Előző két példánkban a kulccsere protokoll alapvető építőeleme szimmetrikus kulcsú rejtelezés volt. Needham és Schroeder azonban 1978-as cikkében javasolt egy olyan protokollt is, ami nyilvános kulcsú rejtelezésre épül. Igaz, a protokollt eredetileg partner-hitelesítésre szánták, ám később kiderült, hogy az kapcsolatkulcs létrehozására is alkalmass lehet.

A protokollnak csak két (on-line) résztvevője van, A és B , akik szeretnék egymást hitelesíteni. Mindkét fél rendelkezik egy nyilvános – privát kulcs-párral, és minden fél ismeri a másik fél hiteles nyilvános kulcsát. A protokoll vérehajtását A kezdi. A először generál egy friss N_A véletlenszámot,

majd ezt és saját azonosítóját rejtjelez B nyilvános kulcsával, K_B -vel, és az eredményt elküldi B -nek. B saját privát kulcsával dekódolja az üzenetet. B is generál egy N_B véletlenszámot, majd N_A -t N_B -vel együtt rejtjelez A nyilvános kulcsával, K_A -val, és az eredményt visszaküldi A -nak. A saját privát kulcsával dekódolja az üzenetet, és ellenőrzi, hogy visszakapta-e az előzőleg elküldött N_A -t. Sikeres ellenőrzés esetén A tudja, hogy B jelen van, mert N_A nem predikálható, így a második üzenetet csak olyan valaki generálhatta, aki az első üzenet dekódolásával szerezte meg N_A -t. Erre azonban csak B lehetett képes. Az ellenőrzés után A rejtjelez N_B -t B nyilvános kulcsával, és az eredményt elküldi B -nek. B dekódolja az üzenetet saját privát kulcsával, és ellenőrzi, hogy visszakapta-e az előzőleg elküldött N_B -t. A-hoz hasonlóan, sikeres ellenőrzés esetén B meg lehet győződve arról, hogy A jelen van, mert csak ő lehetett képes a második üzenet dekódolására és N_B visszaküldésére. Így tehát minden két fél hitelesítette magát (igazolta jelenlétéit) a másik félnek.

Nyilvános kulcsú Needham–Schroeder-protokoll

- (1) $A \rightarrow B : \{N_A | A\}_{K_B}$
 - (2) $B \rightarrow A : \{N_A | N_B\}_{K_A}$
 - (3) $A \rightarrow B : \{N_B\}_{K_B}$
-

Vegyük észre továbbá, hogy N_A és N_B a protokoll során nem kerül nyíltan átvitelre. N_A és N_B értékét tehát csak A és B ismeri. Így a partner-hitelesítésen túl, A és B egy $f(N_A, N_B)$ közös titkos kulcsot is létrehozhat a protokoll segítségével.

A kulccserére használt nyilvános kulcsú Needham–Schroeder-protokoll azonban támadható a következő interleaving támadással:

$$\begin{aligned}
 A &\rightarrow X: \{N_A | A\}_{K_X} \\
 X_A &\rightarrow B: \{N_A | A\}_{K_B} \\
 B &\rightarrow X_A: \{N_A | N_B\}_{K_A} \\
 X &\rightarrow A: \{N_A | N_B\}_{K_A} \\
 A &\rightarrow X: \{N_B\}_{K_X} \\
 X_A &\rightarrow B: \{N_B\}_{K_B}.
 \end{aligned}$$

Tegyük fel, hogy az X támadó egy legális felhasználó, vagy hozzájutott egy legális felhasználó privát kulcsához. X nyilvános kulcsát K_X -szel jelöljük. Tegyük fel továbbá, hogy X rá tudja venni A -t, hogy kezdeményezze a protokoll futását vele. Ekkor X meg tudja személyesíteni A -t B felé, ami azt jelenti, hogy B azt hiszi, A -val hozott létre egy kapcsolatkulcsot, miközben valójában X -szel egyezett meg a kulcsban.

A támadás menete a következő. Mikor A elküldi a protokoll első üzenetét X-nek, X saját privát kulcsával dekódolja azt, majd az eredményt rejtjelezzi B nyilvános kulcsával, és elküldi B-nek. B tehát azt hiszi, hogy A kezdeményezte vele a protokoll végrehajtását. B válaszát X nem tudja dekódolni, felhasználhatja viszont A-t a dekódolás elvégzésére. Ehhez B válaszát elküldi A-nak, aki azt hiszi, hogy az X válasza az Ő első üzenetére. A protokollnak megfelelően A dekódolja az üzenetet, és rejtjelezzi N_B -t X nyilvános kulcsával, majd az eredményt elküldi X-nek. X dekódolja ezt az üzenetet, és ezzel hozzájut N_B -hez. Végül X befejezi a protokollt B-vel. Figyeljük meg, hogy a támadás során X hozzájutott mind N_A -hoz, mind N_B -hez, továbbá B azt hiszi, hogy A-val hajtotta végre a protokollt, hiszen pontosan az annak megfelelő üzeneteket kapta.

Érdemes megfigyelni, hogy a partner-hitelesítést nem kompromittálja a támadás, azaz B helyesen vonja le azt a következtetést, hogy A jelen van, és dekódolta B válaszát. A probléma az, hogy A nem B-vel futtatja a protokollt, hanem X-szel, és ez nem derül ki B számára (mint ahogy az sem derül ki A számára, hogy nem X üzenetére válaszol, hanem B-jére). Ez a hiba azonban csak akkor válik végzetessé, mikor a protokollt kulccserére szeretnénk használni.

Próbáljuk meg kijavítani a protokollt! B-t szeretnénk védeni. Mivel az első és a harmadik üzenet B nyilvános kulcsával van rejtjelezve, ezért azt bárki előállíthatja. B tehát semmilyen következtetést nem tud levonni ezen üzenetek eredetére vonatkozóan. Egyetlen dolog, amit tehet, hogy a saját maga által küldött második üzenetben elhelyezi saját azonosítóját, ezzel figyelmeztetve A-t, hogy az üzenetet Ő generálta (és nem X). Ez persze csak egy figyelmeztetés és nem bizonyíték A számára, hiszen A ugyanúgy semmilyen következtetést nem tud levonni a második üzenet eredetére vonatkozóan, mert az mindenki által ismert nyilvános kulcsával van rejtjelezve. Mindenestre a módosítás a fenti támadást megakadályozza, mert A olyan üzenetet vár, amiben X azonosítója található, és helyette olyat kap, ami B azonosítóját tartalmazza.

Javított nyilvános kulcsú Needham–Schroeder-protokoll

- (1) $A \rightarrow B : \{N_A | A\}_{K_B}$
- (2) $B \rightarrow A : \{N_A | N_B | B\}_{K_A}$
- (3) $A \rightarrow B : \{N_B\}_{K_B}$

Végül ismét kihangsúlyozzuk, hogy a protokoll feltételezi, hogy minden fél ismeri a másik fél nyilvános kulcsát. Előfordulhat, hogy a nyilvános kulcsok nem állnak a felek rendelkezésére. Ilyenkor a megfelelő nyilvános kulcs tanúsítványokat meg kell szerezni. A tanúsítványokat kiadó hitelesítés-szolgáltatót (CA) tekintetjük a protokoll által feltételezett off-line megbízható harmadik félnek is.

8.3.4. Digitális aláírás és nyilvános kulcsú rejtjelezés

Aláírt kapcsolatkulcs rejtjelezése. A nyilvános kulcsú Needham–Schroeder-protokoll érdekes tulajdonsága, hogy csak nyilvános kulcsú rejtjelezést használ építőelemként. Ugyanakkor a kulcs-hitelesítés egyszerűbben megoldható a digitális aláírás és a rejtjelezés együttes alkalmazásával. Erre mutatunk példákat ebben a szakaszban.

Tekintsük az alábbi egy üzenetből álló protokolلت:

Aláírt kapcsolatkulcs rejtjelezése

$$(1) \quad A \rightarrow B : \{A \mid k \mid T_A \mid S_A(B \mid k \mid T_A)\}_{K_B}$$

A protokollnak két (on-line) résztvevője van, A és B , akik egy kapcsolatkulcsot szeretnének létrehozni egymás között. A generál egy k kulcsot, és aláírja azt egy T_A időpecséttel és B azonosítójával együtt. Ezután saját azonosítóját, a k kulcsot, a T_A időpecsétet és az aláírást rejtjelez B nyilvános kulcsával, és az eredményt elküldi B -nek. B először dekódolja az üzenetet a privát kulcsával, majd ellenőrzi a T_A időpecsétet és A aláírását. Ha az ellenőrzések sikeresek, akkor B elfogadja a k kulcsot.

A protokoll feltételezi, hogy minden fél ismeri a másik fél nyilvános kulcsát. A nyilvános kulcsú Needham–Schroeder-protokollhoz hasonlóan, a nyilvános kulcsokat vagy a kulcs tanúsítványokat szolgáltató hitelesítés-szolgáltatót tekintetjük a protokoll által feltételezett off-line megbízható harmadik félnek is.

A protokoll üzenetének vétele és sikeres feldolgozása után B biztos benne, hogy a k kulcsot A generálta, és tudja, hogy k friss. Előbbit A digitális aláírása garantálja, utóbbiról pedig a T_A időpecsét ellenőrzésével győződhet meg B . Az aláírásban szerepel B azonosítója is. Ez azt igazolja, hogy A a k kulcsot valóban B -nek szánta. Ha B azonosítóját nem tartalmazná az aláírás, akkor bárki felhasználhatná az aláírást arra, hogy A nevében kapcsolatkulcsot küldjön B -nek (lásd később a 8.6. szakaszban a 3. protokoll tervezési elv tárgyalását).

Az üzenetben szerepel még az A azonosítója, a k kulcs és a T_A időpecsét. A azonosítója csak egy tipp, ami segít B -nek kitalálni, hogy kinek a kulcsával próbálja meg ellenőrizni az üzenetben található aláírást. A k kulcs és a T_A időpecsét azért szerepel az üzenetben, mert az aláírásból azok nem feltétlen nyerhetők ki (pl. hash-and-sign módszer alkalmazása esetén az aláírást a hash értéken képezzük, így abból az aláírt üzenet nem állítható vissza).

Rejtjelezett kapcsolatkulcs aláírása. Az előző protokoll egy hátrányos tulajdonsága, hogy a teljes üzenetet rejtjelezzi, pedig igazából csak a k kulcs az ami titkos. A következő protokollban ezért először k -t rejtjelezzi A , majd a rejtjelezett kulcsot írja alá a hitelesség biztosítása érdekében.

Rejtjelezett kapcsolatkulcs aláírása

$$(1) \quad A \rightarrow B : \quad A \mid \{k\}_{K_B} \mid T_A \mid S_A(B \mid \{k\}_{K_B} \mid T_A)$$

A protokoll üzenetének magyarázata hasonló mint az előző protokoll esetében, ezért azt most nem részletezzük. Annyit azonban megjegyzünk, hogy a hatékonyságon kívül van még különbség ezen protokoll és az előző szakaszban tárgyalt protokoll között. Nevezetesen arról van szó, hogy az előző protokollban az aláírás tartalmazza a k kulcsot, és így B tudja, hogy A ismeri k -t. Azaz a protokoll biztosítja a kulcskonfirmációt B számára. Ezzel szemben, jelen protokoll esetén, B nem lehet biztos abban, hogy A ismeri k -t. Lehet, hogy A lehallgatott egy protokoll példányt, amit valamikor C futtatott B -vel, és szert tett $\{k\}_{K_B}$ -re, de nem ismeri k -t. A később bármikor aláírhatja $\{k\}_{K_B}$ -t egy friss időpecséttel együtt.

Rejtjelezett és aláírt kapcsolatkulcs. A két korábbi protokollt ötvözve mindkettő hiányosságait ki tudjuk küszöbölni:

Rejtjelezett és aláírt kapcsolatkulcs

$$(1) \quad A \rightarrow B : \quad A \mid \{k\}_{K_B} \mid T_A \mid S_A(B \mid k \mid T_A)$$

Ezen protokoll esetében természetesen feltételezzük, hogy az S_A aláírásból a k kulcs nem fejthető meg. Ez praktikusan azt jelenti, hogy a hash-and-sign módszert alkalmazzuk. A protokoll tulajdonságainak vizsgálatát az olvasóra bízzuk.

8.4. Példák kulcsmegegyezés protokollokra

Ebben a szakaszban két kulcsmegegyezés protokollt mutatunk be. Az első a történelmi jelentőségű Diffie–Hellman-protokoll, amit Diffie és Hellman publikált híres, 1976-os cikkében, melyben a nyilvános kulcsú kriptográfia koncepcióját is kidolgozták. Mint látni fogjuk azonban, a Diffie–Hellman-protokollnak van egy súlyos hiányossága, ezért az eredeti változatot ritkán használják. A protokollnak számos kibővített változata létezik, melyek igyekeznek az eredeti protokoll hiányosságát megszüntetni. Az eredetivel ellenetben, a különböző változatok igen széles körben használatosak a gyakorlatban. Egy ilyen változatot, a Station-to-Station-protokollt mutatjuk be második példaként.

8.4.1. A Diffie–Hellman-protokoll

A protokollnak két résztvevője van, akiket A -val és B -vel jelölünk. A korábban leírt protokollokkal ellentétben, itt A és B szerepe teljesen szimmetrikus. Feltessük, hogy minden fél számára ismert egy nagy p prímszám, és a Z_p^* véges csoport egy g generátor eleme, ahol Z_p^* a p által definiált véges multiplikatív csoportot jelöli, azaz az $\{1, 2, \dots, p-1\}$ halmazt a mod p szorzás-sal, mint művelettel. A generál egy x véletlen számot, melyre $1 \leq x \leq p-2$, majd kiszámolja $g^x \bmod p$ értékét, és az eredményt elküldi B -nek. Hasonlóan, B generál egy y véletlen számot, melyre $1 \leq y \leq p-2$, majd kiszámolja $g^y \bmod p$ értékét, és az eredményt elküldi A -nak. Ezek után a B -től kapott $g^y \bmod p$ értéket és saját titkos x számát felhasználva, A kiszámolja $(g^y)^x \bmod p$ értékét. B hasonlóan kiszámolja $(g^x)^y \bmod p$ értékét. A hatványozás kommutativitása miatt azonban minden fél ugyanazt a $k = g^{xy} \bmod p$ számot kapja, és ez lesz (vagy további számítással ebből kapható) a közös kapcsolatkulcs.

Diffie–Hellman-protokoll

- (1) $A \rightarrow B : g^x \bmod p$
 - (2) $B \rightarrow A : g^y \bmod p$
 - (3a) $A : k = (g^y)^x \bmod p = g^{xy} \bmod p$
 - (3b) $B : k = (g^x)^y \bmod p = g^{xy} \bmod p$
-

A Diffie–Hellman-protokoll biztonsága a Diffie–Hellman-probléma és a diszkrét logaritmus probléma nehézségén alapszik. A Diffie–Hellman-probléma a következő: adott egy p prím, a Z_p^* csoport egy g generátor eleme, valamint a $g^x \bmod p$ és $g^y \bmod p$ számok; számoljuk ki $g^{xy} \bmod p$ értékét. Ehhez

szorosan kapcsolódik a diszkrét logaritmus probléma, mely a következőképpen szól: adott egy p prím, a Z_p^* csoport egy g generátor eleme, és egy $X \in Z_p^*$ szám; adjuk meg azt az $x \in Z_p^*$ számot, melyre $g^x \bmod p = X$. A két probléma közötti kapcsolatról annyit tudunk, hogy ha a diszkrét logaritmust meg tudnánk oldani, akkor a Diffie–Hellman-problémára is megoldást kapnánk: először $g^x \bmod p$ -ből meghatározzuk x -et, majd kiszámítjuk $(g^y)^x \bmod p$ értékét. Mivel a fordított irány nem bizonyított, ezért nem tudjuk, hogy a két probléma ekvivalens-e. Mindenesetre, úgy sejtik, hogy minden két probléma nehéz, abban az értelemben, hogy nem lehet polinom időben megoldani őket. A pontos komplexitása azonban egyik problémának sem ismert.

A Diffie–Hellman-protokoll egyszerű és elegáns. Sajnos azonban a protokoll nem biztosít kulcsbiztonságot, ezért csak passzív támadóval szemben biztonságos. Ha az aktív támadás lehetőségét nem lehet kizární, akkor A és B nem lehetnek biztosak abban, hogy egymással hozták létre a kapcsolatkulcsot, mert a támadó kettőjük közé ékelődve, minden két féllel futtatni tudja a protokollt. Ezt a problémát igyekszik megoldani a következő protokoll.

8.4.2. A Station-to-Station-protokoll

A Station-to-Station-protokoll a Diffie–Hellman-protokoll egy hitelesítéssel kiegészített változata. A Diffie–Hellman-protokolltól annyiban különbözik, hogy a felek aláírják a protokoll futása során használt $g^x \bmod p$ és $g^y \bmod p$ publikus Diffie–Hellman-értékeket, és ezzel hitelesítik azokat. Ezen kívül, a protokoll kulcskonfirmációt is biztosít, azaz minden két fél bizonyítja, hogy ismeri a létrehozott k kapcsolatkulcsot. A protokoll üzenetei a következők (ahol az egyszerűség kedvéért mindenhol elhagytuk a $\bmod p$ jelölést):

Station-to-Station-protokoll

- (1) $A \rightarrow B : g^x$
 - (2) $B \rightarrow A : g^y | \{S_B(g^y | g^x)\}_k$
 - (3) $A \rightarrow B : \{S_A(g^x | g^y)\}_k$
-

A Station-to-Station-protokoll tehát minden két fél számára explicit kulcsbiztosítést biztosít. Ezen kívül minden két fél számára biztosítva van a kulcsfrissesség is. Ezt azonban nem explicit időpontok vagy friss véletlenszámok kihívásként történő alkalmazásával oldja meg a protokoll. A kulcs frissességeit itt az biztosítja, hogy minden két fél egy friss, nem predikálható elemmel járul hozzá a kulcshoz ($g^x \bmod p$ nem predikálható x nem predikálható volta miatt) és minden két fél sem tudja egyedül befolyásolni a kulcs értékét.

Míg az eredeti Diffie–Hellman-protokollnak csak két résztvevője van, addig a Station-to-Station-protokoll résztvevőinek esetleg egy off-line megbízható harmadik fél szolgáltatásait is igénybe kell venniük egymás nyilvános kulcsának megszerzéséhez. Erre a digitális aláírások ellenőrzése miatt van szükségük.

8.5. Nyilvános kulcs infrastruktúra alapjai

Nyilvános kulcsú kriptografiára épülő protokoll példáinkban eddig feltételeztük, hogy a felek ismerik egymás hiteles nyilvános kulcsát. Most azt vizsgáljuk, hogy hogyan lehet a hiteles nyilvános kulcsokhoz hozzájutni. Vegyük észre, hogy ha A szeretné megszerezni B nyilvános kulcsát, K_B -t, akkor nem jó megoldás az, ha B mindenféle védelem nélkül elküldi K_B -t A -nak. Igaz ugyan, hogy K_B nem titkos, ezért egy hallgatózó támadótól nem kell tartani. Azonban ha az X támadó aktív támadást is képes végrehajtani, akkor K_B -t az átvitel során saját nyilvános kulcsával, K_X -szel helyettesítheti. Ekkor A B -nek szánt üzeneteit K_X -szel fogja rejtjelezni, és így azokat X meg tudja fejteni. Sőt, a megfejtett üzeneteket K_B -vel rejtjelezve és továbbítva B felé, X a támadást észrevétenlül tudja végrehajtani. Ezért nagyon fontos, hogy A meg tudjon győződni a kulcs hitelességéről, azaz arról, hogy a kulcs amit B nyilvános kulcsának hisz, valóban B nyilvános kulcsa.

Ezen probléma megoldására a gyakorlatban a *nyilvános kulcs tanúsítványok* használata terjedt el. A nyilvános kulcs tanúsítvány egy adatstruktúra, mely minimálisan a következő adatokat tartalmazza:

- a nyilvános kulcsot,
- a nyilvános kulcs tulajdonosának azonosítóját, és
- egy aláírást, mely az előző két mezőt elválaszthatatlanul összeköti.

Az aláírást általában egy megbízható fél, a *hitelesítés-szolgáltató* (Certification Authority – CA) generálja, aki kompetens annak meghatározásában, hogy a tanúsítvány által tartalmazott kulcs és azonosító valóban összetartoznak-e. A tanúsítvány lényegében ezen megbízható fél állítása, miszerint az adott kulcs az adott tulajdonoshoz tartozik.

A gyakorlatban a tanúsítvány további adatokat is tartalmaz, mint például a tanúsítvány aláírójának, a kibocsájtónak az azonosítóját, a kibocsájtás dátumát, a tanúsítvány érvényességi idejét (lejáratú dátumát) stb.

A hitelesítés-szolgáltató általában off-line. Ez azért előnyös, mert így könnyebb garantálni a biztonságát. Ugyanakkor a tanúsítványok megszerzé-

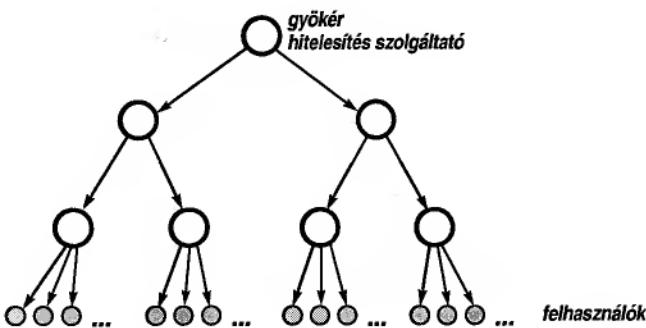
sének megkönnyítésére a rendszerben lehetnek on-line szerverek (adatbázisok), ahol a tanúsítványokat el lehet helyezni, illetve ahonnan azokat szükség esetén le lehet tölteni. Ezen szervereknek azonban nem kell megbízhatónak lenniük: a kulcs hitelességét a tanúsítványban található aláírás garantálja, és nem a szerver, ahonnan a tanúsítványt letöltöttük. A megbízható és nem feltétlen megbízható elemek ilyetén szérválasztása a tanúsítvány-alapú nyilvános kulcs infrastruktúra egyik nagy előnye.

Tegyük fel, hogy a rendszerben csak egy hitelesítés-szolgáltató van, és az bocsátja ki minden felhasználó tanúsítványát. Ekkor minden felhasználó minden tanúsítványt ellenőrizni tud, feltéve hogy ismeri a hitelesítés-szolgáltató nyilvános kulcsát. Ehhez tipikusan akkor juthat hozzá biztonságosan, mikor a saját tanúsítványát igényli a hitelesítés-szolgáltatónál, és személyazonosságának igazolása érdekében személyesen megjelenik a szolgáltatónál. Az egyetlen hitelesítés szolgáltatóra épülő rendszer előnye, hogy egyszerű. Hátránya viszont, hogy nem skálázható, azaz a felhasználók számának növekedésével a rendszer használata egyre nehézkesebbé válik, a megbízható működés esetleg nem is garantálható.

Nagy rendszerekben több hitelesítés-szolgáltató található, melyek általában valamilyen struktúra szerint szerveződnek. A skálázhatósággal kapcsolatos problémák megoldására általában célszerű a hitelesítés-szolgáltatókat hierarchiába szervezni. A hierarchia tetején általában egyetlen szolgáltató áll, amit gyökérszolgáltatónak (root CA) hívunk. Ez alatt helyezkednek el az első szintű szolgáltatók, majd alattuk a második szintűek, és így tovább. Tipikusan minden szolgáltató a közvetlenül alatta elhelyezkedő szolgáltatók számára bocsát ki tanúsítványokat, azaz minden alatta elhelyezkedő szolgáltató azonosítóját és nyilvános kulcsát aláírja. A felhasználók tanúsítványát a legalsó szinten elhelyezkedő szolgáltatók bocsátják ki. Ezt a fajta szerveződést a 8.1. ábra szemlélteti.

Csakúgy, mint az egyetlen hitelesítés-szolgáltatót tartalmazó rendszerben, itt is feltesszük, hogy minden felhasználó ismeri a gyökérszolgáltató hiteles nyilvános kulcsát. Ez azonban még nem elég egy adott felhasználó tanúsítványának ellenőrzéséhez, hiszen a felhasználók tanúsítványait nem a gyökérszolgáltató írja alá. Amire szükség van, az egy *tanúsítvánnylánc*, mely a következő tulajdonságokkal rendelkezik:

- A lánc első eleme egy olyan tanúsítvány, amit a gyökérszolgáltató adott ki, és így a benne található nyilvános kulcs hitelessége bárki által ellenőrizhető.



8.1. ábra. Hitelesítés-szolgáltatók tiszta hierarchikus szerveződésének illusztrációja. minden nyíl egy kibocsátott tanúsítványt reprezentál, ahol a nyíl iránya a kibocsátás irányára utal

- A lánc minden további tanúsítványára igaz, hogy ellenőrizhető a láncban őt közvetlenül megelőző tanúsítványban található nyilvános kulccsal.
- A lánc utolsó tanúsítvanya tartalmazza a kérdéssel, hitelesíteni kívánt felhasználói nyilvános kulcsot.

Könnyen látható, hogy egy ilyen tanúsítvánnyalánc birtokában, és a gyökér-szolgáltató nyilvános kulcsának ismeretében bármely felhasználó bármely másik felhasználó tanúsítványát ellenőrizni tudja. Ehhez a lánc tanúsítványait kell sorrendben ellenőrizni.

Fontos megjegyezni, hogy tanúsítvánlyláncok használata esetén, a lánc minden egyes tanúsítványának kibocsájtójában meg kell bíznunk. Ha egy hitelesítés szolgáltató nem feltétlenül megbízható, akkor a lánc ellenőrzése az ezen szolgáltató által kibocsátott tanúsítványnál megakad, hiszen az abban található nyilvános kulcs hitelességében nem lehetünk biztosak.

A gyakorlatban a hitelesítés szolgáltatók nem feltétlenül szerveződnek tiszta hierarchiába. Előfordulhat például, hogy több szervezet szeretné egyesíteni már létező hierarchikus infrastruktúráját. Ezt megtehetik például úgy, hogy gyökér szolgáltatóik *keresztbe tanúsítják* egymást, azaz minden gyökér minden gyökér számára kibocsát egy tanúsítványt. Ez lehetővé teszi, hogy a különböző szervezetekhez tartozó felhasználók is ellenőrizni tudják egymás tanúsítványát. Az is előfordulhat, hogy nemcsak a felsőbb szintű szolgáltatók adnak ki tanúsítványokat az alattuk elhelyezkedő szolgáltatóknak, hanem fordítva is. Akárhogyan is szerveződnek azonban a hitelesítés-szolgáltatók,

a tanúsítvánnyaláncon alapuló hitelesítés fentebb bemutatott elve és feltételei érvényesek maradnak.

8.5.1. A nyilvános kulcs infrastruktúra további szereplői és funkciói

A nyilvános kulcs infrastruktúra két alapvető résztvevője a fentebb bevezetett hitelesítés-szolgáltató és az általa kibocsátott tanúsítvány tulajdonosa. E két főszereplőn túl, általában további szereplők is vannak a rendszerben, úgy mint

- **az akkreditáló hatóság:** Az adott ország törvényei szabályozzák a hitelesítés-szolgáltatóként való működés feltételeit. Ezen törvényeknek és az elfogadott biztonságpolitikai elveknek való megfelelést az akkreditáló hatóság független auditor szervezetekkel ellenőrizteti.
- **a regisztrációs-szolgáltató (Registration Authority – RA):** A regisztrációs-szolgáltató – amennyiben különválik a hitelesítés-szolgáltatotól – a szolgáltatást a hitelesítés-szolgáltatóval kötött szerződés alapján végezheti, amely magába foglalhatja:
 - a tanúsítványért folyamodók azonosítását,
 - azon információk gyűjtését és rendszerbe vitelét, amelyek a tanúsítvány tartalmát képezik, s amelyre a hitelesítés-szolgáltató az aláírását fogja adni,
 - a biztonságos kommunikációt a hitelesítés-szolgáltatóval,
 - a hitelesítés-szolgáltatotól érkező friss tanúsítványok fogadását, majd „kézbesítését” a tulajdonosnak.

A regisztrációs-szolgáltató funkciói – rendszertől függően – magukba foglalhatják továbbá a felhasználók titkos kulcsának tárolására használt hardver eszközök (pl. chip-kártya) kiadását a felhasználók számára, a tanúsítvány-visszavonás eseménnyel kapcsolatos teendőket (pl. riport készítés), a kulcsgenerálás funkcióit, és a kulcsok archiválását is.

- **auditor szervezetek:** Az auditálás feladata nemcsak a hitelesítés-szolgáltatóként való működés engedélyezése kapcsán jelenik meg, hanem periodikusan, a hitelesítés-szolgáltató működése során is szükség van auditálásra egészen a szolgáltatás megszüntetésig. Az auditálás során ellenőrzésre kerül, hogy a szolgáltató valóban a meghirdetett biztonságpolitikájának (TBSz) megfelelően telepítette és működteti a szolgáltatás funkcióit. Az auditor szervezetileg elkülönült kell legyen a hitelesítés-szolgáltatotól, és

– természetesen – szakmailag kompetensnek kell lennie a megfelelőségi auditálás (compliance audit) feladatkörében.

Amennyiben az auditor hiányosságot talál a hitelesítés-szolgáltató működésében, arról feljegyzést készít, s azt megküldi mind az auditált, mind a felettes szolgáltatónak. Az auditált szolgáltató javaslatot kell tegyen a hiányosság orvoslására, megadva annak teljesítmény határidejét is.

- *a tanúsítvány-felhasználó fél:* Van egy további természetes szereplő is a rendszerben: a tanúsítványt felhasználó fél (pl. egy Web alapú házi-bank szolgáltatás esetén, a bank tanúsítványát a böngészője felhasználásával ellenőrző fél). Ez a szereplő, a tanúsítvány-felhasználó nem feltétlen olyan szereplő, akinek saját tanúsítványa van (a tipikus jelenleg az, hogy a tanúsítvány felhasználók száma sokkal nagyobb, mint a tanúsítvány-tulajdonosok száma). A tanúsítvány-felhasználó kötelessége, hogy ellenőrizze a tanúsítvány érvényességét, mielőtt arra alapozva elfogadna egy tranzakciót. Kötélessége továbbá, hogy ellenőrizze a visszavonási és felfüggesztési listát. Tudnia kell azt is, hogy az adott tanúsítvány milyen alkalmazásokkal kapcsolatosan használható.

A nyilvános kulcs infrastruktúra főbb funkciói a következők:

- *Kezdeti regisztráció és tanúsítvány-kibocsátás:* Miután a regisztrációs-szolgáltató azonosította a tanúsítványért folyamodót (továbbiakban előfizető), a hitelesítés szolgáltató elkészíti a tanúsítványt az előfizető nyilvános kulcsára. Az elkészült tanúsítványt eljuttatja az előfizetőnek, valamint elhelyezi a tanúsítvány-tárban is. A tanúsítvány publikálását – rendszertől függően – végezheti a regisztrációs-szolgáltató is. Ezen kezdeti lépések eleme az előfizetői hardver eszköz inicializálása, amelynek során meg-történik a hitelesítés-szolgáltató hiteles nyilvános kulcsának betöltése, valamint az előfizetői kulcspár telepítése.
- *Előfizetői kulcspár frissítés:* Az előfizetői kulcspárt rendszeresen frissíteni kell. A frissítés során a régi kulcspár újra cserélődik, s egyúttal új tanúsítvány is kibocsátásra kerül.
- *Tanúsítvány frissítés:* Amikor a tanúsítvány érvényességi ideje lejár, vagy az előfizetői adatokban változás áll be, a tanúsítványt frissíteni kell, ami gyakorlatilag egy új tanúsítvány kibocsátását jelenti.
- *Publikálás:* A kibocsátott tanúsítványt vagy egy tanúsítvány visszavonását publikálni kell, amely alapvetően annak elhelyezését jelenti a tanúsítvány-tárban illetve a visszavonási listán (Certificate Revocation List).

- CRL). A szolgáltató feladata a tanúsítvány-tár és a visszavonási lista hozzáférésvédelmének garantálása. Ezen kívül a hitelesítés-szolgáltatónak publikálnia kell saját tanúsítvány-aláíró kulcsának tanúsítványát, továbbá a biztonságpolitika dokumentumának egy példányát is. A publikálás történet rendszeren kívüli csatornán is.
- **Visszavonás:** Előfordul, hogy egy tanúsítványt vissza kell vonni. Ennek tipikus okai lehetnek a következők:
 - a tanúsítvány tulajdonosának azonosító adataiban változás állt elő, azaz a tanúsítványban feltüntetett adatok már nem érvényesek,
 - a tanúsítvány felhasználhatósági köre szűkült,
 - a tanúsítvány tulajdonosa megsértette a biztonsági rendszabályokat,
 - a tanúsítványban szereplő nyilvános kulcshoz tartozó titkos kulcs kompromittálódott,
 - a tanúsítvány tulajdonosa kéri annak visszavonását.

A visszavonást a szolgáltató publikálja a visszavont tanúsítvány visszavonási listán történő elhelyezésével. A visszavonást a lehető leggyorsabban meg kell tenni. Amennyiben a tulajdonostól érkezik a visszavonási kérelem, a kérelem hitelességét ellenőrizni kell, hogy ne fordulhasson elő, hogy egy támadó visszavonathassa mások tanúsítványát. Ezért a visszavonási kérelmet aláírással (manuális vagy digitális) kell ellátni.

A visszavonási listát periódikusan frissíteni kell, például 1 nap frissítési periódusidővel. A titkos kulcs kompromittálódása esetén azonban nem a periódus időt kell figyelembe venni, hanem amilyen gyorsan csak lehetséges a megfelelő tanúsítványt el kell helyezni a visszavonási listán.

8.5.2. Kulcsgondozás

Egy nyilvános kulcs infrstruktúrában a következő kulcsgondozással kapcsolatos feladatokat kell ellátni:

- **Generálás:** Az előfizetői kulcspár generálása – rendszertől függően – történhet az előfizetőnél, a hitelesítés-szolgáltatónál vagy a regisztrációs-szolgáltatónál. A szükséges véletlen elemek megfelelő generálása kritikus biztonsági kérdés. A kulcsgeneráláshoz felhasznált véletlen elemek generálása – az alkalmazás biztonsági besorolásától függően – hardver véletlen generátorral (valódi, fizikai véletlen) vagy álvéletlen generátorral (valódi véletlennel inicializált algoritmus) történik.

- Tárolás:** A titkos kulcs nyílt formában nem hagyhatja el azt az eszközt (modult), amelyben generálódott. Rejtjelezett formában a modulon kívül is megjelenhet, egy másik modulba töltés vagy biztonságos háttértárolás (későbbi kulcs helyreállítás) okán. Nagy előfizetői populációt gondozó szolgáltató esetén felmerülhet a szolgáltató titkos kulcsát tartalmazó modulhoz történő hozzáférés olyan jellegű korlátozása, hogy azzal kapcsolatos bizonyos műveleteket csak két személy együttes közreműködésével lehessen végrehajtani.

Ha a kulcsot nem az előfizető generálja, hozzá hardver token vagy rejtjelezett szoftver modul (fájl) formájában jut el.

Az előfizetői eszköz része egy helyi biztonságos tár (HBT), amely lehet szoftver (védett fájl) vagy (cél)hardver alapú (token). A HBT tartalmához történő hozzáférést valamely aktivációs kulcs engedélyezi. A titkos kulcs aktiválása történhet például jelszóval, jelmondattal, vagy biometrikus adatok alapján. Tipikus, hogy az aktivációs adatot (például jelszó) az előfizető választja. Ha a rendszer magas biztonsági osztályt is megkülönböztet, akkor ott tipikusan tilos a személyes választás, véletlenül generált aktivációs adatot kell használni. Az aktivált modul inaktiválásának módját, teendőit definiálni kell.

Az aktivációs adatokat a kulcsot hordozó modultól független csatornán (közvetlenül kézbe adás, postai küldemény) kell eljutatni az előfizetőhöz. Amennyiben nem közvetlenül kézbe adódik az aktivációs adat, akkor az előfizetőt a kézbesítés várható körülményeirol (módja, várható időpontja) tájékoztatni kell, s a sikeres kézbesítést a címzettnek aláírásával igazolnia kell. Az aktivációs adat biztonságos tárolásáról is intézkedni kell. Legbiztonságosabb fejben megjegyezni az aktivációs adatot, nyíltan feljegyezni nem szabad. Biztonsági szinttől függően az aktivációs adat periodikus cseréje lehet szükséges.

- Másolatkészítés:** Előfizetői szinten a kulcsgondozás szabályozási lehetőségei rendszertől függőek. Ha a szolgáltató a titkos kulcsot hardver tokenben adta át, amelyet az előfizető másolni nem képes, akkor előfizetői szinten is kontrollálható a kulcsmásolás. A szolgáltatónak lehetősége van arra, hogy biztonsági másolatot készítsen előfizetői kérés alapján. Fájlokat ugyanakkor az előfizető is képes másolni.
- Nyilvános kulcs továbbítás:** A fentiekben a titkos kulcs biztonsági szempontjait hangsúlyoztuk. Természetesen az előfizetői nyilvános kulcs továbbításáról is gondoskodni kell. Két esetet különböztetünk meg: (a) a

szolgáltató generálja a kulcspárt az előfizető számára, (b) az előfizető állítja elő saját kulcsait.

Az első esetben a szolgáltató a tanúsítvánnyal együtt küldi a kulcspárt. Ekkor kiegészítő biztonsági megoldásként a token csak azután kerülhet aktiválásra (az aktiválási adatok elküldése útján), mihelyt az előfizető vissza jelezte a token átvételét. Amennyiben az előfizető generálja a kulcspárját, a nyilvános kulcsát tanúsítványozásra el kell juttatnia a szolgáltatóhoz. Ennek tipikus módja lehet a mágneslemezen (floppy) postai, futár útján történő vagy a személyes továbbítás.

A szolgáltató nyilvános kulcsának előfizetőhöz történő továbbítása a következő módszerekkel történhet:

- továbbítás hardver token tartalmaként biztonságos, rendszeren kívüli úton,
- a nyilvános kulcs lenyomatának (hash értékének) rendszeren kívüli csatornán történő továbbítása.
- a szolgáltató tanúsítványának letöltése a szolgáltató web oldaláról abban az esetben, ha az előfizető böngészője megfelelő helyben rendelkezésre álló megbízható tanúsítványokkal ellenőrizni képes a letöltött tanúsítvány hitelességét.
- *Megsemmisítés, archiválás:* A titkos kulcs életciklusának végállomása a megsemmisítés. Például szoftver modul esetén ez a fájl biztonságos felülírását, célhardver modul esetén a kulcs elektronikus törlését jelentheti. A nyilvános kulcsok azonban tovább élnek, mint a titkos kulcsok, s a tanúsítványokkal archiválásra kerülnek (erre azért van szükség, hogy az előfizető által generált aláírásokat később is ellenőrizni lehessen).
- *Kulcs-escrow:* Léteznek megoldások arra, hogy egy megbízható harmadik fél kulcs-escrow szolgáltatást nyújtson, ami azt jelenti, hogy egy harmadik fél képes a titkos kulcsot algoritmikusan rekonstruálni. Rendszertől, jogi szabályozástól függő, hogy ilyen technika alkalmazására sor kerül, egyáltalán sor kerülhet-e.

8.6. Informális protokoll-tervezési elvek

Ebben a szakaszban 11 protokoll-tervezési elvben foglaljuk össze a kulcscsere protokollokkal kapcsolatos eddigi ismereteinket, illetve a kulcscsere protokollok tervezése során elkövetett gyakori hibákkal kapcsolatos észrevételeinket. Ezen protokoll-tervezési elveket tekinthetjük informális ökoliszabályoknak is, melyek követésével számos gyakori protokoll-tervezési hibát elkerülhetünk. Ki szeretnénk hangsúlyozni azonban, hogy ezen tervezési elvek követése se nem szükséges, se nem elégsges feltétele a hibátlan protokollok tervezésének. Azaz létezhet protokoll, mely valamelyik itt tárgyalt elvnek ellentmond, és mégsem hibás. És sajnos fordítva is: elköpzelhető, hogy egy ezen elvek követésével tervezett protokoll mégis hibás lesz. Az utóbbinak azonban kicsi a valószínűsége, ezért az itt tárgyalt protokolltervezési elveknek nagy a gyakorlati jelentősége.

Valamilyen formában minden tervezési elv azt a követelményt fogalmazza meg, hogy a tervező legyen *explicit*. Az első elv általánosan a protokoll üzeneteinek tartalmára nézve mondja ki ezt a követelményt, a második pedig általánosan a protokoll által használt feltevésekre. A további kilenc elv az első kettő finomítása, konkrét protokolltervezési problémákra történő alkalmazása.

- 1. A protokoll üzeneteinek jelentése legyen egyértelmű; minden üzenet értelmezhető legyen pusztán a tartalma alapján.*

Veszélyes lehet az, ha egy üzenet jelentése nem pusztán az üzenet tartalmától, hanem az üzenet feldolgozásának kontextusától is függ, hiszen ekkor az üzenetet más kontextusba helyezve (pl. a protokoll egy másik példányában visszajátszva) az üzenet más, nagy valószínűséggel hibás értelmezést kaphat. Érdemes tehát a protokoll üzeneteit explicitté tenni, azaz minden olyan információt belehelyezni az üzenetbe, ami az üzenet helyes értelmezéséhez nélkülözhetetlen. Ehhez természetesen pontosan érteni kell az üzenet célját a protokollen belül.

- 2. Az üzenetek feldolgozása során használt feltevések legyenek átgondolva, és képezzék a protokoll leírásának részét.*

Mivel a protokoll maga nem oldhat meg minden feladatot, ezért ellengedhetetlen az, hogy ne használjon különböző feltevéseket. Sok protokoll azonban a használt feltevésektől függően tekinthető helyesnek vagy hibásnak (azaz bizonyos feltevések mellett helyes, míg mások mellett hibás). Természetesen szeretnénk elkerülni azt, hogy egy adott feltételek mellett helyes protokollt olyan környezetben alkalmazzon valaki, mely-

ben a protokoll feltevései nem teljesülnek, hiszen így a protokoll helyessége az adott környezetben nem garantált. Ezt úgy segíthetjük, ha a protokoll által használt feltevéseket explicitté tesszük, és ezzel megkönnítjük annak ellenőrzését, hogy az adott alkalmazási környezetben a feltevések teljesülnek-e. A protokollt feltehetően úgyis tervezője érti a legjobban, így elvárható, hogy ő adja meg a protokoll által használt feltevések pontos leírását is.

3. *Ha egy résztvevő azonosítója szükséges egy üzenet értelmezéséhez, akkor az üzenetnek tartalmaznia kell az azonosítót.*

Ez az elv az első elv finomítása. Gyakori hiba, hogy a protokoll tervezője úgy próbálja az üzenetek méretét csökkenteni, hogy kihagyja azokból a résztvevők azonosítóját, arra gondolva, hogy azok úgyis kitalálhatók abból, hogy kik futtatták a protokollt. Ezáltal tehát az üzenetek értelmezése kontextusról függővé válik, és a legtöbb esetben a protokoll támadhatóságát vonja maga után. Erre a hibára számtalan példa található az irodalomban. Mi is találkoztunk vele a nyilvános kulcsú Needham–Schroeder-protokollnál. Továbbá, a 8.3.4. szakaszban, az aláírt kapcsolatkulcs rejtelezésére épülő protokollnál említettük, hogy fontos, hogy B azonosítóját tartalmazza az aláírás. Példaként vizsgáljuk most meg, hogy mi történne, ha ez nem így lenne:

$$(1) \quad A \rightarrow B : \{A \mid k \mid T_A \mid S_A(k \mid T_A)\}_{K_B}$$

Ekkor a következő támadás lehetséges:

$$\begin{aligned} A \rightarrow X : & \{A \mid k \mid T_A \mid S_A(k \mid T_A)\}_{K_X} \\ X_A \rightarrow B : & \{A \mid k \mid T_A \mid S_A(k \mid T_A)\}_{K_B}. \end{aligned}$$

A támadás menete a következő: Először az X rosszindulatú résztvevő ráveszi A -t, hogy küldjön neki egy k kapcsolatkulcsot, majd ezt a k kulcsot továbbküldi B -nek A aláírásával együtt (az időpont érvényességi idején belül). Az eredmény az, hogy B azt hiszi, hogy a k kulcs közte és A között jött létre.

B azonosítójának kihagyása az aláírásból tehát súlyos hibához vezet. A helyes protokoll a 8.3.4. szakaszban található.

4. *A rejtelezés nem a biztonság szinonimája, sőt helytelen használata súlyos hibákhoz vezethet. Ezért legyen átgondolva, hogy ki, mit és miért rejtelez.*
- A rejtelezés egy sokoldalúan használható kriptográfiai primitív, melyet többféle céllal is alkalmazhatunk egy protokollban. Ilyen célok lehetnek például a következők:

- egy adatelem (pl. kulcs) titkosságának biztosítása,
- egy üzenet hitelességének biztosítása²,
- egy üzenet mezőinek összeláncolása, egymáshoz kötése³,
- véletlenszámok generálása stb.

Fontos, hogy a tervező tisztában legyen azzal, hogy hol és miért alkalmaz rejtjelezést. Tekintsük például a szimmetrikus kulcsú Needham–Schroeder-protokollt. A második és harmadik üzenetben alkalmazott rejtjelezéseknek több céljuk is van. Egyrészt biztosítják a kapcsolatkulcs titkosságát, másrészt egymáshoz kötik az üzenetek mezőit. Az első cél nyilvánvalóan fontos, de a második cél is lényeges, hiszen A csak akkor tudja azt a következtetést levonni, hogy a k kulcs friss, ha az valamilyen módon az N_A számhoz van kötve. Ha A a $\{k\}_{K_{AS}} | \{N_A\}_{K_{AS}}$ üzenetet kapná a szervertől, akkor semmit nem tudna k frissességével kapcsolatban megállapítani.

Figyeljük meg, hogy a Needham–Schroeder-protokoll negyedik és ötödik üzenetében a rejtjelezés egészben más szerepet tölt be. Itt a cél nem N_B rejtése, és nincsenek mezők sem, amiket össze kellene láncolni. A rejtjelezés célja annak bizonyítása, hogy a felek ismerik a k kulcsot. Említettük azonban, hogy B már akkor tudja, hogy A ismeri k -t, mikor a harmadik üzenetet megkapja. Ezért az ötödik üzenet valójában felesleges. A rejtjelezés céljának körökintőbb átgondolásával tehát a tervezők egyszerűbb protokollra jutottak volna. Egy másik lehetséges egyszerűsítést a Kerberos-protokoll mintájára tehetünk. Ez abból áll, hogy a második üzenetben nem alkalmazunk egymásba ágyazott rejtjelezést. Ekkor azonban szükség van az ötödik üzenetre.

5. Rejtjelezett szöveg aláírása nem jelenti azt, hogy az aláíró ismeri a nyílt szöveget.

Erre a 8.3.4. szakaszban már utaltunk, mikor a rejtjelezett kapcsolatkulcs aláírására épülő protokollt tárgyalunk. Egy másik példaként tekintsünk egy olyan hozzáférés-védelmi rendszert, ahol egy erőforráshoz való hozzáférés csak akkor biztosított, ha a felhasználó ismeri az erőforráshoz rendelt titkos jelszót. A jelszó védelme érdekében azt a felhasználó nem nyíltan küldi el az erőforrásnak, hanem annak nyilvános kulcsával rejtjelezí-

² Ha az üzenet megfelelő redundanciát tartalmaz, akkor a szimmetrikus kulcsú rejtjelezés üzenet hitelesítést is biztosíthat, mert egy támadó a kules ismerete nélkül nem tudja úgy manipulálni a rejtjelezett üzenetet, hogy az helyes redundanciát tartalmazó nyílt üzenetet eredményezzen a dekódolás után.

³ Ekkor feltételezzük, hogy a rejtjelezőt CBC, CFB, OFB vagy CTR módban használjuk.

azt, majd az üzenetet aláírásával hitelesíti. Mikor a szerver megkapja ezt az üzenetet, nem lehet biztos abban, hogy az aláíró ismeri a jelszót, és így jogosan szeretne hozzáérni az erőforráshoz, mert elképzelhető, hogy csak megszerezte az erőforrás nyilvános kulcsával rejtelezett jelszót, és anélkül írta azt alá, hogy magát a jelszót ismerné.

6. Legyen egyértelmű a protokoll által használt véletlenszámok szerepe.

A rejtelezéshez hasonlóan a véletlenszámok alkalmazása is többféle cél-lal történhet a protokollokban. Tipikus célok például a következők:

- nem predikálható véletlenszám alkalmazása kihívásként a válasz frissességeinek bizonyítására,
- véletlenszám használata tranzakció azonosítóként,
- véletlenszám használata a protokoll résztvevőinek ideiglenes azonosítására stb.

Tekintsük például az Otway–Rees-protokolلت:

Otway–Rees-protokoll

- (1) $A \rightarrow B : M | A | B | \{N_A | M | A | B\}_{K_{AS}}$
 - (2) $B \rightarrow S : M | A | B | \{N_A | M | A | B\}_{K_{AS}} | \{N_B | M | A | B\}_{K_{BS}}$
 - (3) $S \rightarrow B : M | \{N_A | k\}_{K_{AS}} | \{N_B | k\}_{K_{BS}}$
 - (4) $B \rightarrow A : M | \{N_A | k\}_{K_{AS}}$
-

A protokoll részleteit most nem ismertetjük, megvizsgáljuk azonban a protokoll által használt véletlenszámok szerepét. A protokollban három véletlenszám van: M , N_A , és N_B . M szerepe az aktuális protokollfutás azonosítása, tehát tranzakció-azonosítóként működik, ezért is jelenik meg minden üzenetben. Érdekes módon azonban a harmadik és a negyedik üzenetben M csak nyíltan kerül átvitelre, így azt gondolhatnánk, hogy a k kapcsolatkulcsot nem lehet biztosan az M által azonosított tranzakcióhoz kötni. Ez nem így van, mert az első és a második üzenetben található rejtelezések M -hez kötik N_A -t és N_B -t, a harmadik és a negyedik üzenetben található rejtelezések pedig N_A -hoz és N_B -hez kötik k -t. Hasonló okokból, A és B azonosítója is közvetett módon van a k kulcschoz kötve. N_A és N_B tehát egyfelől hivatkozásnéként szolgál az M tranzakció-azonosítóra, valamint az A és B résztvevők azonosítójára, másfelől természetesen a k kulcs frissességet is hivatott bizonyítani. Ez a „túlterhelés” nem igazán szerencsés megoldás, mert nehezen átláthatóvá teszi a protokolلت. Az alábbi protokoll egyszerűbb, tisztább megoldás, ahol minden véletlenszám célja azonnal

egyértelmű, a résztvevők azonosítója expliciten szerepel az üzenetekben (lásd 3. elv), ráadásul kevesebb rejtjelezésre van szükség:

Módosított Otway–Rees-protokoll

- (1) $A \rightarrow B : A | N_A$
 - (2) $B \rightarrow S : A | B | N_A | N_B$
 - (3) $S \rightarrow B : \{N_A | A | B | k\}_{K_{AS}} | \{N_B | A | B | k\}_{K_{BS}}$
 - (4) $B \rightarrow A : \{N_A | A | B | k\}_{K_{AS}}$
-

7. Legyünk óvatosak a predikálható elemek frissesség bizonyítására történő használatával.

Egy régi üzenet visszajátszását egyszerű sorozatszámok (és kriptográfiai védelem, pl. üzenethitelesítés) alkalmazásával is detektálni lehet. Ez azt sugallja, hogy predikálható mennyiségek is használhatók üzenetek frissességének bizonyítására. Óvatosan kell azonban bánni a predikálható elemek kihívásként történő használatával. Példaként tekintsük a következő óraszinkronizáló protokollt, melyben egy A résztvevő egy S időszervertől tölti le a pontos időt:

-
- (1) $A \rightarrow S : A | N_A$
 - (2) $S \rightarrow A : \{N_A | T\}_{K_{AS}}$
-

Ha N_A nem predikálható, akkor a protokoll helyesen működik: N_A ellenőrzésével A ellenőrizni tudja S válaszának frissességét, sőt az első üzenet elküldése és a második üzenet vétele között eltelt idő mérésével felső korlátot is kap T pontatlanságára vonatkozóan.

Ezzel szemben, ha N_A predikálható, akkor a protokoll nem biztonságos. Tegyük fel ugyanis, hogy az X támadó a T időpontban kitalálja A következő N_A értékét. Ennek ismeretében küld egy kérést S -nek, aki az $\{N_A | T\}_{K_{AS}}$ üzenettel válaszol. Később, a T' időpontban, mikor A elküldi N_A -t tartalmazó kérését S -nek, akkor X elfogja a kérést, és S helyett ő válaszol S korábbi válaszának visszajátszásával. Az eredmény az lesz, hogy A helytelenül azt hiszi, hogy az idő T , miközben az már $T' > T$.

8. Ha a protokoll időpecséteket használ, akkor gondoskodni kell az órák szinkronizálásáról.

Több protokoll is időpecséteket használ üzenetei frissességének bizonyítására, illetve a visszajátszásos támadások detektálására. Ne feledkezzünk meg azonban arról, hogy ekkor a résztvevők óráit egy globális órához

kell szinkronizálni. Továbbá, az óraszinkronizáló protokoll nem használhat időpecséteket, hiszen azok ellenőrzése már szinkronizált órát igényel, és pont ezt a feladatot szeretnénk magával a protokollal megoldani. Így marad a friss véletlenszámok használata, mint például az előző elvnél tárgyalt protokollnál.

Megjegyezzük, hogy biztonsági szempontból mind a késő, mind a siető óra veszélyes lehet. Ha egy résztvevő órája késik, akkor például hibában elfogadhat már lejárt kulcsának útványokat, vagy Kerberos-jegyeket. Ha egy résztvevő órája siet, akkor a T időpontban aláírhat egy üzenetet $T' > T$ időpecséttel, amit egy támadó a később a T' időpontban sikeresen felhasználhat.

9. Egy mostanában használt kulcs nem feltétlenül friss.

Ezen elv fontosságára a szimmetrikus kulcsú Needham–Schroeder-protokoll hibája hívja fel a figyelmet. A kapcsolódó támadást a protokoll tárnyalásánál bemutattuk, így azzal itt most nem foglalkozunk.

10. minden üzenetről legyen megállapítható, hogy az melyik protokoll, melyik példányának, melyik üzenete.

Ez ismét az 1. elv explicit üzenetekre vonatkozó követelményének finomítása. Ezen elv fontosságának megértésére a Wide-Mouth-Frog-protokoll elleni támadás szolgáltat példát. A Wide-Mouth-Frog-protokoll azért támadható, mert a protokoll első üzenete felhasználható második üzenetként egy másik protokoll-példányban, és fordítva. Más szavakkal, az üzenetek nem tartalmaznak explicit információt, ami alapján meg lehet állapítani, hogy a protokoll első vagy a második üzenetéről van szó. A javításnál használt irányjelző bitek explicitté teszik az üzenet szándékolt irányát, és a támadás meghiúsul.

Kiegészített Wide-Mouth-Frog-protokoll

- (1) $A \rightarrow S : A \mid \{\text{"WideMouthFrog\#1"} \mid B \mid k \mid T_A\}_{K_{AS}}$
 - (2) $S \rightarrow B : \{\text{"WideMouthFrog\#2"} \mid A \mid k \mid T_S\}_{K_{BS}}$
-

Fontos még megjegyezni azt is, hogy nemcsak egyazon protokoll különböző üzenet(rész)ei lehetnek egymással felcserélhetők. A támadó próbálkozhat különböző protokollok üzeneteinek felcserélésével is. Ezért hasznos lehet valamilyen protokoll-azonosítót elhelyezni a protokoll üzeneteiben. Az azonosító globális egyediségének biztosítása persze szabványsítási kérdéseket vet fel, de erre praktikusan úgysincs szükség. Már egy

egyszerű, a protokoll nevére utaló ASCII sztring is minimálisra csökkenti a visszaélések lehetőségét.

11. A megbízhatósággal kapcsolatos feltevések legyenek átgondolva és explicitté téve.

Ez az elv a 2. elv finomítása, és arra hívja fel a figyelmet, hogy a résztvevők megbízhatóságával kapcsolatos feltevések tisztázására különös figyelmet kell fordítani. Egy adott protokoll látszólag minden tekintetben megfelelhet egy adott alkalmazási környezetben, de ha a résztvevők megbízhatóságával kapcsolatos feltevések nem teljesülnek, akkor a protokoll alkalmazása súlyos hibákhoz vezethet. Például tekintsük a Wide-Mouth-Frog-protokollt, ahol A generálja a kapcsolatkulcsot. Emiatt a protokollt valószínűleg nem tanácsos egy mobil-telefon alkalmazásban használni úgy, hogy a mobil játsza A szerepét, mert kérdéses, hogy a mobil készülék megfelelő minőségű kriptográfiai kulcsot tud-e generálni.

8.7. Kulccsere protokollok formális ellenőrzése és a BAN-logika

Az eddigiekből kiderült, hogy a kulccsere protokollok hibái nem minden nyilvánvalóak, és néha csak körmönfont támadásokkal lehet őket kihasználni. Ebből azonban az is következik, hogy a hibák felfedezése általában nem egyszerű feladat. Ezen problémára megoldást jelenthet valamelyen szisztematikus protokoll-ellenőrzési módszer használata. Ebben a szakaszban egy ilyen módszert mutatunk be.

Burrows, Abadi, és Needham 1989-ben publikált egy formális módszert kulccsere és partner-hitelesítési protokollok ellenőrzésére, mely BAN-logika néven vált ismertté (a szerzők kezdőbetűi után). A BAN-logika viszonylag egyszerű és intuitív, ezért hamar kedveltté vált a protokoll-tervezők körében. A BAN-logika több szempontból is jelentős lépés volt a biztonságos protokollok tervezésében. Egyrészt a protokollokban használt alapelvek formalizálásával segítette azok jobb megértését és ezen keresztül a tervezést. Másrészt lehetővé tette konkrét protokollok szisztematikus elemzését. Ezt kihasználva a BAN-logikát többek sikerrel alkalmazták ismert protokollok hibáinak felfedezésére. Végül elindított egy új kutatási területet⁴, nevezetesen a formális módszerek alkalmazását kriptográfiai protokollok ellenőrzé-

⁴ Bár a BAN-logika előtt léteztek már kezdeti próbálkozások kriptográfiai protokollok formális ellenőrzésére, azok nem voltak olyan egyszerűek és intuitívak, mint a BAN-logika, így nem terjedtek el. A BAN-logika volt az első, ami valóban széles körben elterjedt, és számos új formális módszer kidolgozását inspirálta. Ezért joggal tekinthetjük úgy, hogy gyakorlati-

sére. Ez a terület azóta számos új módszerrel gyarapodott, és mára éretté vált.

A BAN-logika alapötlete a következő. Egy kulcscsere protokoll célja lényegében az, hogy egymástól távoli legális felek meggyőzzék egymást a kapcsolatkulcs hitelességéről. Mivel a felek fizikailag erről nem tudnak meggyőződni, ezért támaszkodniuk kell a távoli fél által közölt információkra, s ezek alapján kell az adott hitelességbe vetett bizalmukat olyan szintre emelni, amely már elfogadható számukra, azaz már hisznek benne. Ezen felfogásban tehát a protokoll nem más, mint egy bizalom (vagy hit) evolúciója a kommunikáció során, az induló feltevésekkel kezdődően, a protokoll lépésein keresztül, a protokoll által elérő célokig. A BAN-logika tehát azt vizsgálja, hogy a protokoll tervezett használata során, azaz amikor a protokollt legális és bocsátó felek hajtják végre, a protokoll eléri-e a kívánt (specifikált) célokat a felekben kialalkult hit tekintetében. Ez elég temésztes hozzáállás, hiszen ha a protokoll ez esetben sem valósítja meg a célját, akkor alapvetően nem alkalmas a feladatára. Ha a célokat reprezentáló, a felek hitére vonatkozó állítások a BAN-logikában levezethetőek a protokollból, akkor a protokollt helyesnek fogadjuk el (lásd azonban a 8.7.4. szakaszt).

A BAN-logika egy nyelvből és következtetési szabályok egy halmozából áll. A nyelv segítségével logikai állítások formájábán írhatók le a protokoll által használt feltevések, a protokoll kitűzött céljai és a protokoll lépései. A következtetési szabályok segítségével rendelkezésre álló logikai formulákból újabb formulákat vezethetünk le. Az analízis célja, hogy megmutassuk, hogy a feltevéseket és a protokoll lépéseit reprezentáló formulákból lehetők a protokoll céljait leíró formulák. A továbbiakban részletesen bemutatjuk a BAN-logika nyelvét és következtetési szabályait, majd ismertetjük a BAN logikai használatának metodikáját. Ezután egy példán keresztül illusztráljuk a BAN-logika használatát. Végül röviden kitérünk a BAN-logika korlátaira.

8.7.1. A BAN-logika nyelve és következtetési szabályai

Nyelv. Ebben a szakaszban a résztvevőket P -vel és Q -val, a szimmetrikus kulcsokat K -val, a nyilvános és a privát kulcsokat $K^{(pub)}$ -bal és $K^{(sec)}$ -kel (a tulajdonosra vonatkozó indexsel kiegészítve), az üzeneteket M -mel, a logikai formulákat pedig Φ -vel fogjuk jelölni. Megengedjük továbbá, hogy

lag a BAN-logika indította el a kriptográfiai protokollok formális ellenőrzésével foglalkozó kutatási területet.

az üzenetek logikai formulákat is tartalmazzanak. A $\{.\}_k$ jelölés továbbra is rejtjelezést jelent, ha k egy szimmetrikus kulcs vagy egy nyilvános kulcs, és aláírást fog jelenteni, ha k egy privát kulcs.

A BAN-logika nyelve a következő alapelemekből felépülő nyelv:

- $P \triangleleft M$ (P látja M -et)

Ez a formula akkor igaz, ha P kapott valamilyen üzenetet, ami P által hozzáérhető formában tartalmazta M -et. Ezt a konstrukciót fogjuk használni annak az eseménynek a leírására, mikor egy protokollrészttel vesz egy protokoll üzenetet.

- $P \mid\sim M$ (az M üzenet(rész) P -től származik)

Ez a formula azt jelenti, hogy P valamikor küldött egy M -et tartalmazó üzenetet. Fontos megjegyezni, hogy ezen formula érvényessége nem garantálja M frissességét, azaz nem tudjuk pontosan, hogy P mikor mondta M -et.

- $\#(M)$ (az M üzenet friss)

Ez a formula akkor igaz, ha M friss. A BAN-logikában az idő két részre van osztva: jelenre és múltra. minden, ami az aktuálisan elemzett protokoll futás kezdete előtt történt, a múlthoz tartozik, és minden, ami a protokoll kezdete óta történt, a jelen része. Az M üzenet(rész) akkor friss, ha a múltban még soha nem használták. Tipikusan a frissen generált véletlenszámokat, kulcsokat és időpecséteket szoktuk frissnek tekinteni az analízis során.

- $P \models \Phi$ (P elhiszi a Φ állítást)

Ezt a nyelvi konstrukciót használjuk annak a ténynek a leírására, hogy a P résztvevő elhiszi a Φ logikai formula által reprezentált állítást. $P \models \Phi$ igaz voltából még nem következik, hogy a Φ állítás igaz. $P \models \Phi$ csupán – annyit jelent, hogy P – eddigi ismeretei (azaz aktuális állapota) alapján – igaznak fogadja el Φ -t, és ennek megfelelően cselekszik a protokollban.

- $P \Rightarrow \Phi$ (P kompetens a Φ állítás igazságának előtöntésében)

Ezen típusú formuláknak elsődleges szerepe van a különböző bizalommal kapcsolatos feltevések leírásában. Például, ha Φ egy a kapcsolatkulcs frissességére vonatkozó állítás, és S a kapcsolatkulcsot generáló szerver, akkor az $A \models (S \Rightarrow \Phi)$ állítás azt fejezi ki, hogy A megbízik abban, hogy S friss kulcsokat generál.

- $P \xrightarrow{K} Q$ (K egy titkos szimmetrikus kulcs P és Q között)

Ez a formula a kulcs-hitelesség leírására szolgál, és azt fejezi ki, hogy a K kulcshoz P -n és Q -n (valamint esetleges megbízható harmadik feleken) kívül más nem férhet hozzá. A K kulcs lehet egy hosszú élettartamú szimmetrikus kulcs, vagy a protokoll által létrehozott kapcsolatkulcs.

- $\xrightarrow{K^{(pub)}} P$ (P nyilvános kulcsa $K^{(pub)}$)

Ez a formula azt a tényt írja le, hogy P nyilvános kulcsa $K^{(pub)}$. Ez egyúttal azt is jelenti, hogy a $K^{(pub)}$ -hoz tartozó $K^{(sec)}$ privát kulcsot csak P ismeri.

Következtetési szabályok. A BAN-logika szabályai lényegében a kriptográfiai primitívek (elsősorban a rejtjelezés és a digitális aláírás), és a protokoll üzenetek vétele során alkalmazott egyszerű feldolgozási szabályok működését, tulajdonságait írják le absztrakt módon. A főbb következtetési szabályok a következők:

- Ha P kap egy M üzenetet, mely a K kulccsal van rejtjelezve, és P megbízik abban, hogy K -t rajta kívül csak Q ismeri, akkor P biztos lehet abban, hogy M -et Q küldte⁵:

$$\frac{P \models (P \xrightarrow{K} Q), P \triangleleft \{M\}_K}{P \models (Q \vdash M)}.$$

- Hasonlóképpen, ha P kap egy digitálisan aláírt M üzenetet, ahol a digitális aláírás Q nyilvános kulcsával, $K^{(pub)}$ -bal sikeresen ellenőrizhető, akkor P biztos lehet abban, hogy M Q -tól származik:

$$\frac{P \models (\xrightarrow{K^{(pub)}} Q), P \triangleleft \{M\}_{K^{(sec)}}}{P \models (Q \vdash M)}.$$

- minden résztvevő megbízik abban, hogy a másik résztvevő becsületes, abban az értelemben, hogy csak olyan dolgokat állít, amiben hisz. Azaz, ha P kap egy friss Φ formulát tartalmazó üzenetet Q -tól, akkor P biztos benne, hogy Q a Φ állítást igaznak tartja:

$$\frac{P \models (Q \vdash \Phi), P \models \#(\Phi)}{P \models (Q \equiv \Phi)}.$$

⁵ Feltételezzük, hogy M elegendő redundanciát tartalmaz ahhoz, hogy P meg tudja állapítani, hogy jó kulcsot használt a dekódoláshoz, továbbá P felismeri és eldobja a saját maga által generált üzeneteket.

- Ha P azt hiszi, hogy Q igaznak tartja Φ -t, és P megbízik abban, hogy Q kompetens Φ igazságának eldöntésében, akkor nyilván P is el kell hogy higgye Φ -t:

$$\frac{P \models (Q \models \Phi), P \models (Q \Rightarrow \Phi)}{P \models \Phi}.$$

- A következő két szabály a több részből álló üzenetek feldolgozására vonatkozik. Az első azt mondja ki, hogy ha P vesz egy összetett üzenetet, akkor látja annak minden részét. A második szabály pedig azt mondja ki, hogy ha P azt hiszi, hogy egy összetett üzenet Q -tól származik, akkor úgy tekinti, hogy az üzenet minden része Q -tól származik:

$$\frac{P \triangleleft M|M'}{P \triangleleft M, P \triangleleft M'},$$

$$\frac{P \models (Q \succ M|M')}{P \models (Q \succ M), P \models (Q \succ M')}.$$

- Egy üzenetet akkor tekinthetünk frissnek, ha tartalmaz friss elemet:

$$\frac{P \models \#(M)}{P \models \#(M'|M|M'')}.$$

- Ha P olyan üzenetet kap, ami számára ismert szimmetrikus kulccsal vagy saját publikus kulcsával van rejtelezve, akkor P dekódolni tudja az üzenetet és látja a nyílt üzenet tartalmát. Továbbá, az aláírt üzenetek tartalmához bárki hozzáférhet:

$$\frac{P \triangleleft \{M\}_K, P \models (P \xrightarrow{K} Q)}{P \triangleleft M},$$

$$\frac{P \triangleleft \{M\}_{K^{(pub)}}, P \models ({}^{K^{(pub)}} \rightarrow P)}{P \triangleleft M},$$

$$\frac{P \triangleleft \{M\}_{K^{(sec)}}}{P \triangleleft M}.$$

- Végül, a következő szabályok a \models operátor tulajdonságait írják le:

$$\frac{P \models \Phi, P \models \Phi'}{P \models (\Phi \wedge \Phi')},$$

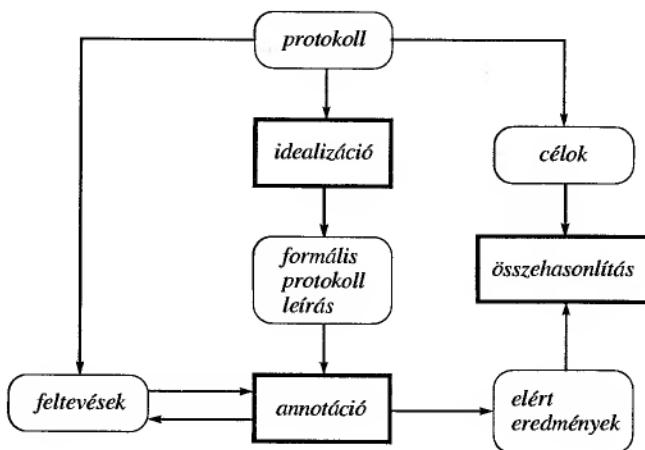
$$\frac{P \models (\Phi \wedge \Phi')}{P \models \Phi, P \models \Phi'},$$

$$\frac{P \models (Q \models \Phi), P \models (Q \models \Phi')}{P \models (Q \models (\Phi \wedge \Phi'))},$$

$$\frac{P \models (Q \models (\Phi \wedge \Phi'))}{P \models (Q \models \Phi), P \models (Q \models \Phi')}.$$

8.7.2. A BAN-logika használata

A BAN-logika használatának sémáját a 8.2. ábrán láthatjuk. A protokoll megszokott formában megadott informális leírásából indulunk ki, amit egy formális leírássá transzformálunk. Ez azt jelenti, hogy a protokoll üzeneteit a jelentésüket tükröző logikai formulákkal egészítjük ki. Ezt a lépést *idealizáció*nak is nevezzük, mert az így nyert formális leírás a protokoll ideális leírása abban az értelemben, hogy minden üzenet expliciten tartalmazza az üzenet jelentését, és nem az üzenet adatmezőiből kell arra következtetni.



8.2. ábra. A BAN-logika használatának sémája

Az idealizációknak nincsenek formális szabályai. Ahhoz, hogy az idealizációt végre tudjuk hajtani, valamilyen szinten már érteni kell a protokollt, hiszen csak így tudjuk kiegészíteni azt az üzenetek vélét jelentését tükröző formulákkal. Ökol szabályként azt mondhatjuk, hogy egy M üzenet(rész) egy

$\Phi(M)$ formulával egészíthető ki, ha M vételekor a vevő arra a következetesre juthat, hogy a küldő $\Phi(M)$ -et igaznak tartotta, mikor M -et elküldte. Ezenkívül az idealizált üzenetekből kihagyhatunk minden olyan részt, ami nem járul hozzá a vevő hitének fejlődéséhez, különös tekintettel a nyílt adatelemekre. Miután egy adott M üzenetet a megfelelő $\Phi(M)$ formulával kiegészítettük, a protokoll

$$(i) \quad P \rightarrow Q: \quad M$$

lépését a $Q \triangleleft M | \Phi(M)$ formulával helyettesítjük.

A protokoll üzenetein kívül, a protokoll által használt feltevéseket és a protokoll céljait is logikai állítások formájában kell megfogalmaznunk. Ez általában egyszerűbb feladat, mint a protokoll idealizációja, mert a feltevések és a célok sok protokollnál megegyeznek, így ismert séma szerint felírhatók. Tipikus feltevések például a következők:

- Már létezik egy hosszú élettartamú K_{AS} kulcs A és S között: $A \models (A \xleftrightarrow{K_{AS}} S)$ és $S \models (A \xleftrightarrow{K_{AS}} S)$.
- A ismeri B nyilvános kulcsát, $K_B^{(pub)}$ -t: $A \models (K_B^{(pub)} \rightarrow B)$.
- A megbízik S -ben, hogy az friss kapcsolatkulcsokat generál A és B számára, és nem fedi azt fel senkinek, csak A -nak és B -nek: $A \models (S \Rightarrow \#(K))$ és $A \models (S \Rightarrow (A \xleftrightarrow{K} B))$.
- A frissnek tartja a saját maga által generált N_A friss véletlenszámot: $A \models \#(N_A)$.
- A ellenőrizni tud bármilyen T időpecsétet: $A \models \#(T)$.

A protokoll céljai közé tartozik általában

- az implicit kulcshtitelesítés: $A \models (A \xleftrightarrow{K} B)$,
- a kulcskonfirmáció: $A \models (B \models (A \xleftrightarrow{K} B))$,
- és a kulcsfrissesség biztosítása: $A \models \#(K)$.

Miután a protokolلت idealizáltuk, és a feltevéseket és célokat formálisan leírtuk, elkezdjük a protokoll annotálását. Legyen az idealizált protokoll a következő:

$$B \triangleleft M_1 | \Phi_1(M_1)$$

$$A \triangleleft M_2 | \Phi_2(M_2)$$

...

$$A \triangleleft M_n | \Phi_n(M_n).$$

A protokoll minden sora után leírjuk azokat a formulákat, melyek a protokoll adott pontján érvényesek (igazak). Az első sor elő kerül a protokoll feltevéseinek halmaza, melyet S_0 -lal jelölünk. Az első sor után azon formulák S_1 halmazát írjuk, melyek a következetési szabályok segítségével levezethetők az $S_0 \cup \{B \triangleleft M_1 | \Phi_1(M_1)\}$ halmaz formuláiból. Hasonlóképpen, a második sor után azon formulák S_2 halmaza kerül, melyek levezethetők az $S_1 \cup \{A \triangleleft M_2 | \Phi_2(M_2)\}$ halmaz formuláiból, és így tovább:

$$\begin{aligned} & S_0 \\ & B \triangleleft M_1 | \Phi_1(M_1) \\ & S_1 \\ & A \triangleleft M_2 | \Phi_2(M_2) \\ & S_2 \\ & \cdots \\ & A \triangleleft M_n | \Phi_n(M_n) \\ & S_n \end{aligned}$$

Ha S_n tartalmazza az összes cél formuláját, akkor az ellenőrzés sikeres. Ha S_n nem tartalmazza valamelyik célt, akkor a protokoll hibás, és az analízis gyakran rámutat arra is, hogy hol a hiba, és hogyan lehetne azt kijavítani.

8.7.3. A Needham–Schroeder-protokoll BAN-analízise

Példaként most bemutatjuk a szimmetrikus kulcsú Needham–Schroeder-protokoll (leírást lásd a 8.3.2. szakaszban) BAN-analízisét. Kezdjük mindenjárt a protokoll idealizációjával:

- (M2) $A \triangleleft \{N_A \mid B \mid k \mid A \xleftarrow{k} B \mid \#(k) \mid \{k \mid A \mid A \xleftarrow{k} B \mid \#(k)\}_{K_{BS}}\}_{K_{AS}}$
- (M3) $B \triangleleft \{k \mid A \mid A \xleftarrow{k} B \mid \#(k)\}_{K_{BS}}$
- (M4) $A \triangleleft \{N_B \mid A \xleftarrow{k} B\}_k$
- (M5) $A \triangleleft \{N_B - 1 \mid A \xleftarrow{k} B\}_k$.

Az első üzenet nem jelenik meg az idealizációban, mert az csak nyílt elemeket tartalmaz, így az üzenet vevője semmiféle következetést nem tud levonni az üzenet küldőjére vonatkozóan. A második és harmadik üzenet idealizációja magáért beszél. Ezeket az üzeneteket azokkal a formulákkal egészítettük ki, melyek leírják, hogy az S szerver mit is akart A és B tudtára adni ezen üzenetekkel. Érdekes a negyedik és ötödik üzenet idealizáltja, mert ezen üzenetek olyan formulával egészültek ki, melynek semmi köze az üzenet N_B tartalmához. Van viszont köze ahhoz a k kulcsnak, mellyel ezen üzenetek rejtjelezve vannak. Ezen üzenetek célja lényegében az, hogy a felek egymás

tudtára adják, hogy elfogadták a k kapcsolatkulcsot mint a köztük letrejött közös titkot. Ezt a jelentést hordozzák a kiegészítésként használt formulák.

A Needham–Schroeder-protokoll feltevései a következők:

- (A1) $A \equiv (A \xleftrightarrow{K_{AS}} S)$,
- (A2) $S \equiv (A \xleftrightarrow{K_{AS}} S)$,
- (A3) $B \equiv (B \xleftrightarrow{K_{BS}} S)$,
- (A4) $S \equiv (B \xleftrightarrow{K_{BS}} S)$,
- (A5) $A \equiv (S \Rightarrow (A \xleftarrow{k} B))$,
- (A6) $B \equiv (S \Rightarrow (A \xleftarrow{k} B))$,
- (A7) $A \equiv (S \Rightarrow \#(k))$,
- (A8) $B \equiv (S \Rightarrow \#(k))$,
- (A9) $A \equiv \#(N_A)$,
- (A10) $B \equiv \#(N_B)$,
- (A11) $S \equiv (A \xleftarrow{k} B)$,
- (A12) $S \equiv \#(k)$.

Az (A1)–(A4) feltevések a már létező, hosszú élettartamú kulcsokra vonatkoznak. Az (A5)–(A8) feltevések A és B bizalmát írják le S -ben, a tekintetben, hogy S megfelelő friss kulcsot generál, és nem fedi azt fel másnak A -n és B -n kívül. Az (A9) és (A10) feltevések a protokollban használt véletlenszámok frissességére vonatkoznak. Vegyük észre, hogy minden fél csak a saját maga által generált véletlenszám frissességében bízik meg. Végül az (A11) és (A12) feltevések azt fejezik ki, hogy maga S hisz a generált k kapcsolatkulcs jóságában és frissességében. Figyeljük meg azonban, hogy sem A -ra, sem B -re vonatkozóan nincs hasonló feltevés, hiszen ez a protokoll elérni kívánt céljai közé tartozik.

A Needham–Schroeder-protokoll tárgyalásánál említettük, hogy a protokoll célja az explicit kulcsfeleltesítés és a kulcsfrissesség biztosítása. Formálisan:

- (G1) $A \equiv (A \xleftrightarrow{k} B)$,
- (G2) $B \equiv (A \xleftrightarrow{k} B)$,
- (G3) $A \equiv (B \equiv (A \xleftrightarrow{k} B))$,
- (G4) $B \equiv (A \equiv (A \xleftrightarrow{k} B))$,
- (G5) $A \equiv \#(k)$,
- (G6) $B \equiv \#(k)$.

A (G1) és (G2) célok az implicit kulcsfeleltesítést írják le, míg a (G3) és (G4) célok a kulcskonfirmációra vonatkoznak. Együttesen tehát a (G1)–(G4)

célok elérése kölcsönös explicit kulcshitelesítést jelent. A (G5) és a (G6) célok a kulcsfrissességet írják le A , illetve B számára.

Most próbáljuk meg a protokollt analizálni a fent leírt módszerrel! Az (M2) formulából és a feltevések ből a következőket lehet levezetni:

$$(M2) \quad A \triangleleft \{N_A \mid B \mid k \mid A \xleftrightarrow{k} B \mid \#(k) \mid \{k \mid A \mid A \xleftrightarrow{k} B \mid \#(k)\}_{K_{BS}}\}_{K_{AS}}$$

$$(A1) \quad A \equiv (A \xleftrightarrow{K_{AS}} S)$$

$$(D1) \quad A \equiv (S \succ [N_A \mid B \mid k \mid A \xleftrightarrow{k} B \mid \#(k) \mid \{k \mid A \mid A \xleftrightarrow{k} B \mid \#(k)\}_{K_{BS}}]);$$

$$(A9) \quad A \equiv \#(N_A)$$

$$(D2) \quad A \equiv \#(N_A \mid B \mid k \mid A \xleftrightarrow{k} B \mid \#(k) \mid \{k \mid A \mid A \xleftrightarrow{k} B \mid \#(k)\}_{K_{BS}});$$

$$(D1) \quad A \equiv (S \succ [N_A \mid B \mid k \mid A \xleftrightarrow{k} B \mid \#(k) \mid \{k \mid A \mid A \xleftrightarrow{k} B \mid \#(k)\}_{K_{BS}}])$$

$$(D2) \quad A \equiv \#(N_A \mid B \mid k \mid A \xleftrightarrow{k} B \mid \#(k) \mid \{k \mid A \mid A \xleftrightarrow{k} B \mid \#(k)\}_{K_{BS}})$$

$$(D3) \quad A \equiv (S \models (A \xleftrightarrow{k} B))$$

$$(D4) \quad A \equiv (S \models \#(k));$$

$$(D3) \quad A \equiv (S \models (A \xleftrightarrow{k} B))$$

$$(A5) \quad A \equiv (S \models (A \xleftrightarrow{k} B))$$

$$(D5) \quad A \equiv (A \xleftrightarrow{k} B);$$

$$(D4) \quad A \equiv (S \models \#(k))$$

$$(A7) \quad A \equiv (S \models \#(k))$$

$$(D6) \quad A \equiv \#(k).$$

Vegyük észre, hogy (D5) és (D6) megegyezik (G1)-gyel és (G5)-tel, azaz A számára teljesül az implicit kulcshitelesítés, és biztosítva van a kulcsfrissesség. Folytassuk az analízist az (M3) formula feldolgozásával:

$$(M3) \quad B \triangleleft \{k \mid A \mid A \xleftrightarrow{k} B \mid \#(k)\}_{K_{BS}}$$

$$(A3) \quad B \equiv (B \xleftrightarrow{K_{BS}} S)$$

$$(D7) \quad B \equiv (S \succ [k \mid A \mid A \xleftrightarrow{k} B \mid \#(k)]).$$

Sajnos az analízis itt elakad. Nincs olyan szabály, melynek segítségével a rendelkezésre álló formulákból le tudnánk vezetni a kívánt

$$B \equiv \#(k \mid A \mid A \xleftrightarrow{k} B \mid \#(k))$$

formulát. Azaz detektáltuk a Needham-Schroeder-protokoll ismert hibáját, nevezetesen azt, hogy B nem kap semmilyen friss elemet, mely alapján S üzenetének frissességéről meggyőződhetne.

Nézzük most meg, hogy mi történne, ha a fenti formulát mégis le tudnánk vezetni:

$$(D7) \quad B \equiv (S \succ [k \mid A \mid A \xleftrightarrow{k} B \mid \#(k)])$$

$$(H) \quad B \equiv \#(k \mid A \mid A \xleftrightarrow{k} B \mid \#(k))$$

$$(D8) \quad B \equiv (S \equiv (A \xleftrightarrow{k} B))$$

$$(D9) \quad B \equiv (S \equiv \#(k));$$

$$(D8) \quad B \equiv (S \equiv (A \xleftrightarrow{k} B))$$

$$(A6) \quad B \equiv (S \Rightarrow (A \xleftrightarrow{k} B))$$

$$(D10) \quad B \equiv (A \xleftrightarrow{k} B);$$

$$(D9) \quad B \equiv (S \equiv \#(k))$$

$$(A8) \quad B \equiv (S \Rightarrow \#(k))$$

$$(D11) \quad B \equiv \#(k).$$

(D10) és (D11) megegyezik (G2)-vel és (G6)-tal, azaz ha (H)-t le tudnánk vezetni, akkor a protokoll B számára is biztosítaná az implicit kulcsitelesítést és a kulcsfrissességet. Folytassuk az (M4) formulával:

$$(M4) \quad A \triangleleft \{N_B \mid A \xleftrightarrow{k} B\}_k$$

$$(D5) \quad A \equiv (A \xleftrightarrow{k} B)$$

$$(D12) \quad A \equiv (B \succ [N_B \mid A \xleftrightarrow{k} B]);$$

$$(D6) \quad A \equiv \#(k)$$

$$(D13) \quad A \equiv \#(N_B \mid A \xleftrightarrow{k} B);$$

$$(D12) \quad A \equiv (B \succ [N_B \mid A \xleftrightarrow{k} B])$$

$$(D13) \quad A \equiv \#(N_B \mid A \xleftrightarrow{k} B)$$

$$(D14) \quad A \equiv (B \equiv (A \xleftrightarrow{k} B)).$$

Az utolsó üzenetet hatását hasonló módon elemezhetjük:

$$(M5) \quad A \triangleleft \{N_B - 1 \mid A \xleftrightarrow{k} B\}_k$$

$$(D10) \quad B \equiv (A \xleftrightarrow{k} B)$$

$$(D11) \quad B \equiv \#(k)$$

$$(D15) \quad B \equiv (A \equiv (A \xleftrightarrow{k} B)).$$

(D14) és (D15) megegyezik (G3)-mal és (G4)-gel, vagyis a protokoll minden két fél számára biztosítja a kulcskonfirmációt (feltéve, hogy (H) teljesül). Érdekes megfigyelni, hogy a levezetés során nem használtuk fel az (A10) feltevést, azaz nem szükséges, hogy B egy friss véletlenszámot küldjön A -nak a protokoll negyedik üzenetében. A kapcsolatkulcs frissessége garantálja a negyedik és az ötödik üzenet frissességét, így egy ismert publikus üzenetet is használhatnánk N_B helyett.

Azt is észrevehetjük, hogy a második üzenet egymásba ágyazott rejtelezésének nincs különösebb hatása a levezetésre. Ez egyrészt alátámasztja azt, hogy nincs szükség az egymásba ágyazott rejtelezésre, másrészt viszont meglepő, hogy az (M3) formula felhasználásánál még nem tudjuk levezetni a kulcskonfirmációt jelentő (G4) célt, pedig korábban úgy érveltünk, hogy ezen a ponton B tudja, hogy A már ismeri a k kulcsot, és elfogadta azt, különben nem küldte volna el a protokoll harmadik üzenetét. Ez a BAN-logika egy hiányosságára utal, nevezetesen arra, hogy az idealizálás során csak az üzenetek tartalmát egészíthetjük ki logikai formulákkal, az üzenetküldés pusztán tényéhez azonban nem csatolhatunk jelentést. Ez a gondolat átvezet minket a következő szakaszba, ahol a BAN logikai korlátait tárgyaljuk.

8.7.4. A BAN-logika korlátai

A BAN-logika lényegében egy absztrakt modell, és mint minden modell, a BAN-logika is egyszerűsítő. A valóság leegyszerűsítése azonban a BAN-logika esetében az analízis-képességekkel kapcsolatos korlátokhoz vezet. A BAN-logikát (és bármely más formális modellt is) csak az tudja helyesen használni, azaz az analízisből helyes következtetéseket levonni, aki tisztában van a korlátaival. Ezért fontos ezen korlátok áttekintése.

Az első doleg, amit minden szem előtt kell tartani, a BAN-logika által használt feltevések halmaza. A BAN-logika feltételezi például, hogy a kriptográfiai primitívek tökéletesen biztonságosak. Ez összhangban van ugyan a protokollok vizsgálata során gyakran alkalmazott támadó modellel, de érdekes emlékezni rá. Egy másik feltevés, hogy minden rejtjelezett üzenet elegettedő redundanciát tartalmaz ahhoz, hogy a vevő meg tudja állapítani, hogy a jó kulcsot használta-e a dekódolásnál. A Needham-Schroeder-protokoll negyedik üzenetében például B az $\{N_B\}_k$ rejtett szöveget küldi A -nak, ahol N_B egy véletlen szám. A BAN-logika szintjén ez nem jelent problémát, azaz mikor A dekódolja az üzenetet, akkor meg tudja állapítani, hogy a dekódolás sikeres volt-e vagy sem. Egy konkrét implementációban azonban nagy hiba volna pusztán egy véletlenszámot rejtjelezni.

Egy még ennél is erősebb feltevés, hogy minden résztvevő felismeri a saját maga által generált üzeneteket és azokat nem fogadja el. Ezzel a BAN-logika lényegében kizáraja a visszajátszásos támadások nagy részének lehetőségét. Más szavakkal a BAN-logika nem minden detektál olyan támadásokat, ahol egy becsületes résztvevő a saját üzenetét kapja vissza a támadótól. Könnyen ellenőrizhető például, hogy a Wide-Mouth-Frog-protokoll BAN-analízise nem detektálja a protokollnál tárgyalt támadást. Ezért az ellenőrzést végzők az ilyen jellegű támadások elleni ellenállóképességet külön kell ellenőriznie.

A BAN-logika egy másik korlátja az idő egyszerű reprezentálásából fakad. A BAN-logikában minden, ami az aktuális protokollfutás alatt történt friss eseménynek számít. Ezért, ha a támadó egy protokollfutáson belül ismét meg üzeneteket, akkor azt a BAN-logika nem minden detektálja. Az idő leegyszerűsített kezelése ugyanakkor a BAN-logika egyszerűségének talán egyik legfőbb oka, ezért a logika használhatóságának szemszögéből nézve előnyös tulajdonság.

A BAN-logika feltételezi, hogy a protokoll résztvevői becsületesek. Ez azt jelenti, hogy a résztvevők csak olyan állításokat tesznek, melyek igazában maguk is hisznek. Tekintsük például a következő protokollt:

Nessett-protokoll

- (1) $A \rightarrow B : \{B \mid k \mid T_A\}_{K_A^{(sec)}}$
 - (2) $B \rightarrow A : \{T_A\}_k$
-

Ha az első üzenetet a korábbi mintákat követve idealizáljuk, akkor a következő formulára jutunk:

$$B \triangleleft \{B \mid k \mid T_A \mid A \xleftarrow{k} B \mid \#(k)\}_{K_A^{(sec)}}.$$

Ezt és a szokásos feltevéseket felhasználva, a következtetési szabályok alkalmazásával le tudjuk vezetni a protokoll helyességét bizonyító $B \models (A \xleftarrow{k} B)$ állítást. A BAN-analízis alapján tehát a protokollt helyesnek tekinthetjük. Ugyanakkor a protokoll egyértelműen hibás, hiszen A nyilvános kulcsának ismeretében bárki hozzájuthat a k kulcschoz. A hibát az analízis során követtük el, mégpedig akkor, amikor a fenti formulával idealizáltuk a protokoll első üzenetét. Ez ugyanis ellentmond a BAN-logika azon feltevéénének, hogy minden résztvevő igaznak tartja amit állít: A nem tarthatja igaznak az $A \xleftarrow{k} B$ állítást, miközben ő maga publikálja k -t az első üzenetben. E példa tanulsága kettős. Egyrészt arra hívja fel a figyelmet, hogy az idealizációt körültekintően kell eljárni, és meg kell győződni arról, hogy az üzenetek kiegészítéseként alkalmazott formulákat az azokat küldő résztvevők valóban igaznak tarthatják-e az üzenet elküldésének pillanatában. Másrészt, az általánosabb tanulság az, hogy a BAN-logika nem képes olyan támadásokat detektálni, melyet egy legális, de rosszindulatú résztvevő hajt végre, akiről a többi résztvevő helytelenül azt feltételezi, hogy becsületes. Utóbbita további példa a nyilvános kulcsú Needham–Schroeder-protokoll elleni támadás, melyet szintén nem detektál a BAN-logika. Ez azonban olyan körmönfont támadás, hogy még az idealizáció hibás volta sem olyan szembetűnő, mint a fenti Nessett-protokoll esetében.

Megemlítfük még, hogy a BAN-logikában a hitek stabilak, ami azt jelenti, hogy ha egy résztvevő egyszer elhitt valamit, akkor később már nem gondolhatja meg magát. Más szavakkal, ha egyszer levezettük a $P \models \Phi$ formulát, akkor az véglegesen bentmarad a rendelkezésre álló formulák halmazában, tehát sosem veszti érvényét. Ez ismét a BAN-logika egy egyszerűsítő feltevése, mely magát a logikát könnyen használhatóvá teszi, ugyanakkor bizonyos támadásokat emiatt nem detektál a logika.

Végül kitérünk az idealizációra mint a BAN-logika egyik legnagyobb gyakorlati korlátjára. minden formális módszer alkalmazása megköveteli, hogy a problémát először az adott módszer nyelvén írjuk le. A valós problema formális leírása azonban nem minden módszer esetében egyforma nehézségű feladat. Ebből a szempontból a BAN-logika egy nehezen használható módszer, mert az idealizáció szubjektív, kreativitást igénylő folyamat, és ezért sok hibalehetőséget rejt magában. Ha pedig egyszer az idealizációt elrontottuk, akkor az analízis eredménye már haszontalan. Sajnos azonban

nincsen módszer arra, hogy magának az idealizációnak a helyességét ellenőrizzük. Így marad a bizonytalanság.

Korlátai ellenére a BAN-logikát még sikerrel alkalmazták több ismert protokoll analízisére is, és nemegyszer hibát is sikerült találni vele helyesnek hitt protokollokban. Ezért a BAN-logika végül is nagyon hasznos eszköz. Nem szabad azonban elfeledkezni a BAN-logika korlátairól: egy BAN-logika által helyesnek ítélt protokoll még lehet támadható egy olyan támadó által, vagy egy olyan módon, melyet a BAN alapfeltevései kizárnak.

8.8. Feladatok

8.1. Feladat. Anna és Béla egy olyan kommunikációs csatornát használnak, amelyen áthaladó rejtett üzeneteket Cili hallgatja, s tárolja. A rejtjelezés nyilvános kulcsú módszerrel történik. Cili egy napon betör Anna és Béla számítógépébe, s az ott található titkos kulcsot megszerzi, amivel dekódolja a tárolt rejtett üzeneteket. Javasoljon egy nyilvános kulcsú módszerre támaszkodó protokollt, amely lehetetlenné teszi, hogy Cili utólag rekonstruálja a nyílt üzeneteket.

8.2. Feladat. Tekintse a következő egy üzenetből álló kulcsszállító protokollt:

$$(1) \quad A \rightarrow B: \quad \{A|k|T_A|\{B|k|T_A\}_{PrA}\}_{PuB},$$

ahol k a kapcsolatkulcs, T_A egy A által generált időpecsét, PrA az A privát kulcsa, és PuB a B nyilvános kulcsa.

1. Adja meg a protokoll feltevéseinek listáját a BAN-logika nyelvén!
2. Adja meg a protokoll által elérte célokat a BAN-logika nyelvén!
3. Vezesse le a 2. pontban megadott célokat a BAN-logika következtetési szabályait használva, ha a protokoll idealizációja a következő:

$$B \triangleleft \{A|k|T_A|\{A \xrightarrow{k} B|\#(k)|T_A\}_{PrA}\}_{PuB}.$$

9.

Partner-hitelesítés

A partner-hitelesítés protokollok feladata, hogy lehetővé tegyék két kommunikáló fél számára egymás identitásának megbízható ellenőrzését. A megbízható jelző itt arra utal, hogy a felek valamilyen módon *bebizonítják* állított identitásukat. Az identitás bizonyítása többféle módszerrel is történhet, melyek alapvetően három osztályba sorolhatók:

- *biometriai módszerek*: Ezen módszerek elsősorban személyek identitásának bizonyítására használatosak. A bizonyítás alapját valamelyen személyes biológiai jellemző (pl. hang, ujjlenyomat, írisz mintázat stb.) képezi.
- *hardver alapú módszerek*: Ezen módszereknél a bizonyítás alapja valamelyen hardver token (badge, smart kártya stb.) birtoklása.
- *algoritmusos módszerek*: A bizonyítás alapja valamelyen számítás elvégzésének képessége.

Jelen fejezetben az algoritmusos módszerekkel foglalkozunk. Ezen módszerek közül a következő hármat tárgyaljuk:

- Gyakran használt, tipikus algoritmikus módszer valamelyen titok ismertetének bizonyítása magának a titoknak a felfedésével. Ezt jelszó alapú partner-hitelesítésnek nevezzük, és a 9.1. szakaszban tárgyaljuk.
- Ha a felek képesek bonyolultabb számítások elvégzésére, akkor kriptográfiai alapú módszereket is használhatnak. Ezekről a 9.2. szakaszban szóunk. Ebben az esetben egy titkos kulcs használata (pl. egy ismert szöveg digitális aláírása) képezi az identitás bizonyításának alapját, azzal a felté-

telzéssel, hogy a titkos kulcs valamilyen módon a bizonyítást végző félhez kötődik.

- Végül a 9.3. szakaszban egy olyan kriptográfiai technikát mutatunk be, mely garantálja, hogy az identitás bizonyítása során a bizonyítás alapját képező titokról semmilyen információ nem szivárog ki azon kívül, hogy a bizonyítást végző fél ismeri a titkot.

9.1. Jelszó alapú partner-hitelesítés

A jelszavas partner-hitelesítés régóta ismert azonosítási eljárás, amit ma is nagyon elterjedten használnak. A jelszó lényegében egy titok, amit a felhasználó megoszt azzal az erőforrással, amihez alkalmanként hozzá szeretne férfi. A felhasználót a jelszó ismerete hitelesíti. A jelszó ismeretét többféleképpen is bizonyíthatja az erőforrás számára (pl. felfedheti a jelszót, elküldheti a jelszó lenyomatát stb.). A jelszavas rendszerek szokásos problémái a következők:

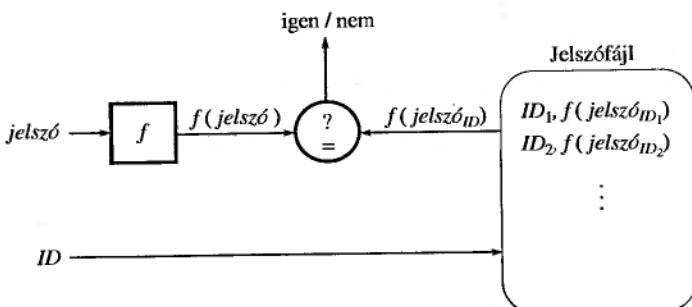
- nem megfelelő erősségű, könnyen kitalálható jelszó választása,
- a jelszó nyílt alakban történő továbbítása a rendszerbe jutás pontjától (pl. terminál klaviatúra) az ellenőrzés pontjáig (pl. gazdagép),
- a jelszavak nem elégé védett tárolása (mind a felhasználó oldalán, mind pedig a jelszófájl tekintetében).

Az alábbiakban több különböző protokollon keresztül vizsgáljuk ezen problémák megoldásának vagy enyhítésének módjait.

9.1.1. Egyirányú függvényes leképezés

A jelszavak gazdagépoldali védelmén javít azok egyirányú függvényel történő leképezése. Ekkor a gazdagép a jelszófájlban nem magukat a jelszavakat, hanem csak azok egyirányú leképezettjét, lenyomatát tárolja az egyes felhasználókra, a felhasználói azonosítókkal együtt. A jelszó továbbra is nyílt alakban érkezik az ellenőrzés helyére, ahol előbb kiszámításra kerül annak egyirányú függvényes leképezése, majd ennek eredménye kerül összevetésre a jelszófájl bejegyzésével. Ezt a folyamatot szemlélteti a 9.1. ábra.

Úgy tűnhet, hogy ezzel a megoldással a jelszófájl illetéktelen olvasása önmagában nem jelent veszélyt, hiszen az egyirányú leképezés gyakorlatilag invertálhatatlan tulajdonsága biztosítja, hogy a jelszavakhoz a támadó nem férhet hozzá. Ez azonban csak feltételek mellett igaz: szükséges, hogy a jelszóméret elég nagy legyen a teljes kipróbálás megakadályozásához, és



9.1. ábra. A jelszó ellenőrzésének tipikus folyamata

a jelszavak a teljes jelszótérből kerüljenek kiválasztásra. Ha ezen feltételek nem teljesülnek (pl. a felhasználók nem véletlenszerűen választanak az adott hosszúságú alfanumerikus karaktersorozatok közül, hanem értelmes szavakat, jellemző dátumokat, ezek triviális kombinációit használják), akkor a fent leírt rendszer ki van téve a szótár alapú támadás veszélyének.

Szótár alapú támadás. A szótár alapú támadások a rendszer jelszófájljában tárolt információkat és a támadó által generált vagy szintén a rendszerben található szótárfájlokat (pl. /usr/dict/words) használják a felhasználók jelszavainak *off-line* megfejtésére. A fent említett forrásokból összeállítanak egy nagy méretű szótárat, melynek szavait különböző transzformációknak vetik alá (pl. csupa kisbetűre/nagybetűre konvertálják, megismétlik, tükrözik, számokkal egészítik ki stb.), majd a jelszavak leképezéséhez használt egirányú függvényeket transzformálják őket, és az eredményt összehasonlítták a jelszófájlban tárolt értékekkel. Egyes jelszótörő programok a jelszófájlban talált egyéb, a felhasználókra vonatkozó információkat is felhasználnak (pl. a felhasználó teljes nevének kezdőbetűiből különböző kombinációk szerint karaktersorozatokat állítanak elő, mert sok felhasználó választ így jelszót, bízva abban, hogy az így nyert többnyire értelmetlen szó elég erős jelszó lesz). Nagyon fontos kihangsúlyozni, hogy a szótár alapú támadások többnyire off-line módon kerülnek kivitelezésre, azaz a támadónak „korlátlan” mennyiségi idő áll rendelkezésre a próbálgatáshoz.

A szótár alapú támadások ellen a jelszavak körültekintő megválasztásával védekezhetünk a leghatékonyabban. Tanácsos olyan jelszót választani, ami nem értelmes szó, egyszerű transzformációkkal (pl. ismétlés, tükrözés) nem

nyerhető értelmes szóból, és nem kapcsolódik a rendszerben nyilvánosan elérhető információkhöz. Jó választás lehet például egy, a kedvenc versünk első pár sorának kezdőbetűiből alkotott karakterSORozat, néhány numerikus karakterrel megtűzelve.

Salting. A szótáralapú támadások másik ellenszere az ún. salting. A salting azt jelenti, hogy a jelszavakat egy adott hosszúságú, véletlen bitsorozattal egészítjük ki, és ezután alkalmazzuk az egyirányú leképezést. Ezen véletlen bitsorozat (salt) nyíltan kerül tárolásra a jelszófájlban.

Ha a salt mérete r bit, akkor a szótáras támadónak minden jelszójelölt 2^r számú különböző variációját kell ellenőriznie, azaz a szótár mérete irreálisan nagyá dagadhat (r méretétől függően). A salting további előnye, hogy ha két felhasználó azonos jelszót választ, a felhasználókhoz tartozó, a jelszófájlban tárolt jelszólenyomatok akkor is különbözni fognak.

9.1.2. Kétirányú jelszóellenőrzés

Az egyirányú jelszóellenőrzés folyamatában az „Enter your password:” felszólításra a felhasználó nyíltan átad egy titkot. Ezzel kockázatot vállal, hiszen a másik fél nem azonosította magát. Ezt a problémát igyekszik enyhíteni a következő protokoll:

Kétirányú jelszóellenőrzés

- (1) $A \rightarrow B : N_A$
 - (2) $B \rightarrow A : f(P_B | N_A) | N_B$
 - (3) $A \rightarrow B : f(P_A | N_B)$
-

Feltessük, hogy minden fél birtokában van saját és partnere jelszavának. A jelszavak nem kerülnek nyíltan átvitelre az azonosítási folyamat során. Ehelyett a „kihívás-válasz” módszerét alkalmazzuk. A egy frissen generált N_A véletlen elemet küld át B -nek, aki ezt hozzáfüzi P_B jelszavához, és ezt a bővíttet jelszót egy f egyirányú függvénytel leképezi. B a leképezés eredményét visszaküldi A -nak, aki ugyanazon számítást maga is elvégzi, s a számítás eredményét egybeveti a kapott értékkel. Hasonlóan B is küld egy N_B kihívást A -nak, aki azt P_A jelszavához fűzi, a kiegészített jelszót az f függvénytel leképezi, majd az eredményt visszaküldi B -nek. B maga is elvégzi a számítást, és az eredményt összeveti az A -tól kapott értékkal.

A kapcsolat-felvételenként frissített véletlen elemek (N_A és N_B) alkalmazásának célja nyivánvaló, hiszen enélküli nem lenne értelme az egyirányú le-

képezéses védelemnek, mivel ekkor lényegében a leképezés eredménye játszaná a nyílt jelszó szerepét. A szótár alapú támadás veszélyét ez a megoldás nem szünteti meg teljesen, hiszen az N_A és N_B véletlen elemeket a támadó is láthatja (esetleg a támadó az egyik fél), így a válaszokban csak a jelszavak az ismeretlenek. A támadást azonban praktikusan lehetetlen off-line módon kivitelezni (a véletlen kihívás nem más, mint egy egyszer használatos salt).

9.1.3. Egyszer használatos jelszavak

Ha a jelszót kapcsolat-felvételenként változtatnánk, nyilván semmi értelme nem lenne a jelszó megismerésére irányuló támadásnak, feltéve, hogy a korábbi jelszavakból nem kiszámítható a következő jelszó. Ekkor tehát az aktuális jelszót nyíltan is átküldhetjük. Erre az ötletre épül az S/KEY nevű, egyszer használatos (one-time) jelszórendszer.

S/KEY. Tegyük fel, hogy A akarja hitelesíteni magát B -nek. A protokoll inicializálásakor A választ egy R titkos véletlensemet, majd az f egyirányú függvény iteratív alkalmazásával generál n darab egyszer használatos jelszót, melyeket P_i -vel jelölünk ($1 \leq i \leq n$):

$$P_i = \begin{cases} R & \text{ha } i = n \\ f(P_{i+1}) = f^{(n-i)}(R) & \text{ha } 1 \leq i < n. \end{cases}$$

Ezután A hitelesen eljuttatja $P_0 = f(P_1)$ -et B -nek. A protokoll inicializálása ezzel véget ér. Az i -edik bejelentkezéskor, A a P_i jelszóval hitelesíti magát. Ezt B úgy ellenőrzi, hogy $f(P_i)$ -t P_{i-1} -hez hasonlítja, ahol P_{i-1} az előzőleg elfogadott jelszó vagy az inicializálnál kicserélt P_0 . Ha $f(P_i)$ megegyezik P_{i-1} -gyel, akkor a hitelesítés sikeres. Mivel f egyirányú, ezért $P_i = f(P_{i+1})$ megfigyeléséből nem lehet kiszámítani P_{i+1} -et.

A -nak vagy tárolnia kell a (P_i) jelszósorozat még fel nem használt elemeit, vagy csak R -et tárolja, és abból az f függvény alkalmazásával minden kiszámolja az aktuális jelszót $P_i = f^{(n-i)}(R)$ alapján.

A két félnek szinkront kell tartania a jelszósorszám vonatkozásában. Ha azonban valamely hiba következtében az átvitel során elveszne egy jelszó, akkor A eggyel továbblép a jelszósorszámban, és új jelszót küld, mellette jelezve az egylépéses szinkronhibát.

A módszer biztonságos, hátránya azonban, hogy egy biztonságos eszközt is igényel a számítások végrehajtására. Ha kizárolagosan saját használatú számítógépről jelentkezünk be, akkor ez a probléma megoldható szoftverrel.

A másik véglet az, mikor csak egy bárki által használható terminál billentyűzetéhez férünk hozzá. Ekkor hozzáhozható (kézi) jelszógenerátor vagy jelszótároló eszköz szükséges. Utóbbi lehet akár egy, a jelszólistát tartalmazó papírlap is, amin minden kihúzzuk a már felhasznált jelszavakat. Ebben az esetben azonban gondoskodni kell arról, hogy a papírlaphoz illetéktelenek ne férjenek hozzá (ne lássák).

9.2. Kihívás-válasz protokollok

A jelszó alapú partner-hitelesítést elsősorban humán felhasználók hitelesítésére használjuk. Ha a partnerek számítást végezni képes eszközök (azaz számítógépek, ideértve pl. a smart kártyákat is), akkor a partner-hitelesítés épülhet erősebb, kriptográfiai protokolloakra is. Ekkor A úgy hitelesíti magát B-nek, hogy bebizonyítja, hogy birtokában van egy olyan kriptográfiai kulcsnak, ami valamilyen módon A-hoz van rendelve (azaz B tudja, hogy csak A ismerheti). A bizonyításhoz A használja a kulcsot, azaz rejtjelez, dekódol vagy aláír valamit. A visszajátszásos támadások megakadályozását az ún. kihívás-válasz módszer alkalmazásával érhetjük el. Ekkor A egy B által frissen generált elemen (kihíváson) alkalmazza a kulcsot, így A válasza nemcsak a kulcstól függ, hanem a kihívástól is.

Tekintsük a következő, építőelemként szolgáló protokollrészleteket:

- *Hitelesítés szimmetrikus kulcsú rejtjelezéssel:*

(1)	$B \rightarrow A: N_B$
(2)	$A \rightarrow B: \{N_B\}_{K_{AB}}$

B generál egy N_B friss véletlenszámot, és nyíltan elküldi azt A -nak. A a K_{AB} kulccsal rejtjelezzi N_B -t, ahol K_{AB} egy csak A és B számára ismert szimmetrikus kulcs. B dekódolja A válaszát, és ha visszakapta a korábban elküldött N_B számot, akkor A hitelesítette magát. A hitelesítés alapja, hogy (B -n kívül) csak A tud rejtjelezni K_{AB} -vel.

- *Hitelesítés szimmetrikus kulcsú dekódolással:*

(1)	$B \rightarrow A: \{N_B\}_{K_{AB}}$
(2)	$A \rightarrow B: N_B$

B generál egy N_B friss véletlenszámot, rejtjelezzi azt a K_{AB} kulccsal, ahol K_{AB} egy csak A és B számára ismert szimmetrikus kulcs, és az eredményt elküldi A -nak. A dekódolja B üzenetét, majd visszaküldi a dekódolás eredményét B -nek. B ellenőrzi A válaszát, és ha az megegyezik az N_B szám-

mal, akkor A hitelesítette magát. A hitelesítés alapja, hogy (B -n kívül) csak A tud dekódolni K_{AB} -vel.

■ *Hitelesítés digitális aláírással:*

$$(1) \quad B \rightarrow A: \quad N_B$$

$$(2) \quad A \rightarrow B: \quad \{N_B\}_{K_A^{(sec)}}$$

B generál egy N_B friss véletlenszámot, és nyíltan elküldi azt A -nak. A saját privát kulcsával, $K_A^{(sec)}$ -val aláírja N_B -t. B ellenőrzi A aláírását A nyilvános kulcsának, $K_A^{(pub)}$ -nak a felhasználásával, és ha az aláírás hiteles, akkor A hitelesítette magát. A hitelesítés alapja, hogy csak A tud aláírni $K_A^{(sec)}$ -val.

■ *Hitelesítés nyilvános kulcsú rejtjelezéssel:*

$$(1) \quad B \rightarrow A: \quad \{N_B\}_{K_A^{(pub)}}$$

$$(2) \quad A \rightarrow B: \quad N_B$$

B generál egy N_B friss véletlenszámot, rejtjelezzi azt A nyilvános kulcsával, $K_A^{(pub)}$ -val, és az eredményt elküldi A -nak. A dekódolja B üzenetét saját privát kulcsával, $K_A^{(sec)}$ -val, majd visszaküldi a dekódolás eredményét B -nek. B ellenőrzi A válaszát, és ha az megegyezik az N_B számmal, akkor A hitelesítette magát. A hitelesítés alapja, hogy csak A tud dekódolni $K_A^{(sec)}$ -val.

Az alkalmazás jellegétől függ, hogy a fenti építőelemeket hogyan lehet felhasználni partner-hitelesítésre. Tegyük fel például, hogy egy bankkártya-alkalmazásban az ügyfelek bankkártyáinak kell hitelesíteniük magukat a bank ATM automatái felé. Más szavakkal, a kártya minden bizonyító szerepet játszik, az automata pedig minden ellenőrzőt, és sosem fordítva. Ekkor a fenti protokollok önmagukban is megállják a helyüket. Ezzel ellentétben, egy olyan alkalmazásban, ahol kölcsönös partner-hitelesítésre van szükség, azaz minden fél játszhat bizonyító és ellenőrző szerepet is, a fenti protokollokat esetleg további elemekkel kell kiegészíteni és körültekintően kell kombinálni. Példaként próbálunk meg a fenti, szimmetrikus kulcsú rejtjelezésre épülő egyirányú protokollrészletből kölcsönös hitelesítést végző protokollt konstruálni. Naíván úgy gondolhatnánk, hogy a protokollrészletet minden két irányba lejátszva a feladatot megoldottuk:

$$(1) \quad A \rightarrow B: \quad N_A$$

$$(2) \quad B \rightarrow A: \quad \{N_A\}_{K_{AB}} \mid N_B$$

$$(3) \quad A \rightarrow B: \quad \{N_B\}_{K_{AB}}$$

Vegyük észre azonban, hogy a fenti protokoll hibás. Egy támadó A saját üzeneteit visszajátszva, meg tudja személyesíteni B -t (hasonló módon A is megszemélyesíthető). A támadás menete a következő:

$$\begin{aligned} A \rightarrow X_B: & N_A \\ X_B \rightarrow A: & N_A \\ A \rightarrow X_B: & \{N_A\}_{K_{AB}} \mid N'_A \\ X_B \rightarrow A: & \{N_A\}_{K_{AB}} \mid N_X \\ A \rightarrow X_B: & \{N_X\}_{K_{AB}}. \end{aligned}$$

X visszajátsza A N_A kihívását A -nak B nevében. Ekkor A előállítja $\{N_A\}_{K_{AB}}$ -t. X -nek pontosan erre van szüksége az első protokoll példány besejtéséhez.

A protokollt, a Wide-Mouth-Frog-protokollhoz hasonlóan, irányjelző bitek alkalmazásával javíthatjuk ki, melyek lehetővé teszik a részvevők számára saját üzeneteik felismerését.

További példaként tekintsük a 8.3.3. szakaszban tárgyalt nyilvános kulcsú Needham–Schroeder-protokollt, amely lényegében a fenti, nyilvános kulcsú rejtelezésre épülő protokollrészlet kétirányú alkalmazásából származtatott. Láttuk azonban, hogy a nyilvános kulcsú Needham–Schroeder-protokoll támadható. Ez ismét arra hívja fel a figyelmet, hogy a fenti protokollrészleteket körültekintően kell kombinálni. Jó hasznát vehetjük itt a 8.6. szakaszban tárgyalt protokoll tervezési elveknek és a 8.7. szakaszban ismertetett formális protokoll-ellenőrzési technikának.

9.3. Zero-knowledge-protokollok partner-hitelesítésre

Partner-hitelesítés során A szeretné bizonyítani B -nek, hogy ő valóban A . Ha A a jelszavát mutatja fel B -nek, akkor B olyan információhoz jut, amellyel egy harmadik fél felé képes A -t megszemélyesíteni. Lehetséges-e olyan azonosítási protokollt konstruálni, amelyben B nem jut olyan információhoz, amelyet később A megszemélyesítésére használhat? Másképpen megfogalmazva, képes-e A azonosítani magát B felé anélkül, hogy ilyen ismeret-hez (knowledge) hozzájuttatná B -t. Az alábbiakban bemutatásra kerülő Fiat–Shamir-protokoll egy ilyen, zero-knowledge protokoll. A protokoll a modulo összetett szám szerinti négyzetgyökvonás nehézségén alapul, ezért először tömörön összefoglaljuk a négyzetgyökvonás feladatot.

Négyzetgyökvonás modulo $n = pq$. Ha létezik oyan b természetes szám, ahol $0 < b < n$, amely az

$$x^2 = c \pmod{n} \quad (9.1)$$

egyenlet megoldása, akkor a c számot kvadratikus maradéknak, a b megoldást pedig a c négyzetgyökének nevezzük modulo n . Több kérdés merül fel a (9.1) egyenlettel kapcsolatban: Mikor van megoldása, azaz mely c számokból lehet gyököt vonni? Ha van megoldása, hány különböző megoldása van? Van-e praktikus algoritmus a gyökök kiszámítására? A válaszok nagyban függnek az n modulustól. Ennek megfelelően két esetet tekintünk: az egyik, amikor n prímszám, a másik, amikor n két különböző prímszám szorzata.

Ha $n = p$, ahol p prímszám, akkor ha egy c számra van megoldása a (9.1) egyenletnek, s ez a megoldás b , akkor a másik megoldás $p - b$. Miután ekkor modulo p test feletti egyenletről van szó, legfeljebb két különböző megoldás lehetséges. Ha a (9.1) egyenlet megoldható, akkor a megoldásra ismertes praktikus algoritmus, amely polinomiális futási idejű ($O(\log^4 p)$ futási idővel). Azt is egyszerűen beláthatjuk, hogy a (9.1) egyenlet a szóbajövő c értékeknek pontosan felére oldható meg.

9.1. Tétel. *Ha $n = p$ prímszám, akkor a (9.1) egyenlet pontosan $s = \frac{p-1}{2}$ különböző c értékre oldható meg.*

Bizonyítás: Ha b megoldása a (9.1) egyenletnek, akkor $p - b$ is az, ahol $0 < b < p$. Tehát a kvadratikus maradékok száma legfeljebb s . Ha belátjuk, hogy tetszőleges $b_1 \neq b_2$, $0 < b_1 < b_2 \leq s$ pár esetén $b_1^2 \neq b_2^2 \pmod{p}$, akkor bebizonyítottuk az állítást. Tegyük fel, hogy $b_1^2 = b_2^2 \pmod{p}$ fennáll valamely b_1, b_2 párra, azaz $p|(b_2^2 - b_1^2)$. Innen vagy $p|(b_2 - b_1)$ vagy $p|(b_1 + b_2)$ következik, ami nem lehetséges, mivel $0 < b_2 - b_1 < p$ és $0 < b_2 + b_1 < p$. \square

Ha $n = pq$, ahol p és q prímszámok, akkor ha egy c számra van megoldása a (9.1) egyenletnek, akkor négy megoldása van, s a megoldások $b_1, n - b_1, b_2, n - b_2$, ahol $0 < b_1, b_2 < n$. A négy megoldást az

$$\begin{aligned}x^2 &= c \pmod{p}, \\x^2 &= c \pmod{q}\end{aligned}$$

egyenletek megoldásait felhasználva a kínai maradékok tétele segítségével kaphatjuk meg. Mivel a modulo prímszám egyenletek polinomiális idejű megoldási algoritmusa ismert, ezért ha ismerjük n két faktorját, akkor a (9.1) egyenlet megoldását is kiszámíthatjuk polinomiális időben. Ha viszont ezen faktorokat nem ismerjük, egy nehéz feladattal állunk szemben. Ezen állításunkat egyszerűen beláthatjuk. Válasszunk véletlenszerűen egy d , $0 < d < n$ számot, és számítsuk ki a négyzetet modulo n . Legyen $c = d^2 \pmod{n}$. Számítsuk ki a (9.1) egyenlet egy d' megoldását – ez feltevésünk szerint nem nehéz feladat. Ekkor $d^2 - d'^2 = 0 \pmod{n}$ áll fenn, azaz

$$n \mid (d - d')(d + d'). \quad (9.2)$$

Mivel a d számot véletlenszerűen választottuk, ezért $\frac{1}{2}$ a valószínűsége annak az eseménynek, hogy $d = d'$ vagy $d = n - d'$, azaz $d - d' = 0$ vagy $d + d' = n$. Ekkor a (9.2) oszthatóság triviális. Azonban ugyancsak $\frac{1}{2}$ a valószínűsége annak is, hogy $d - d' \neq 0 \pmod{n}$, amely esetben $|d - d'| < n$ és $d + d' < 2n$ miatt akár az $l.n.k.o.(n, |d - d'|)$, akár $l.n.k.o.(n, d + d')$ legnagyobb közös osztó számítással n faktORIZÁLÁSHOZ JUTUNK, AMIRŐL VISZONT TUdjUK, HOGY NEHÉZ FELADAT.

Az alábbiakban bemutatásra kerülő protokollok arra épülnek, hogy a (9.1) egyenlet megoldása könnyű feladat, ha ismerjük n faktorjait, ellenkező esetben azonban nehéz.

Fiat-Shamir partner-hitelesítési protokoll. A protokoll egy kulcskiosztó központot használ, amely véletlenszerűen választ két prímszámot, és azok szorzatából előállít egy n modulust. Amikor A be kíván lépni a hitelesítő rendszerbe, a központtól kap egy nyilvános és egy titkos kulcsot. A titkos kulcs, amit u -val jelölünk, egy véletlenszerűen választott szám, a nyilvános kulcs, v , pedig ennek négyzete modulo n , azaz $v = u^2 \pmod{n}$.

A protokoll üzenetei a következők:

Fiat-Shamir-protokoll

- (1) $A \rightarrow B: z = R^2 \pmod{n}$
 - (2) $B \rightarrow A: b$
ha $b = 0$, akkor
 - (3) $A \rightarrow B: R$
ha $b = 1$, akkor
 - (3) $A \rightarrow B: w = R \cdot u \pmod{n}$
-

ahol R egy véletlen szám ($0 < R < n$), amelyet A az első lépést megelőzően generál, míg b egy véletlen bit, amelyet B a második lépést megelőzően állít elő. A harmadik üzenet vétele után B a következő ellenőrzést végzi: Ha $b = 0$ volt, akkor B ellenőrzi, hogy $R^2 \pmod{n}$ egyenlő-e z -vel. Ha $b = 1$ volt, akkor viszont azt ellenőrzi, hogy fennáll-e a $z \cdot v = w^2 \pmod{n}$ egyenlőség.

Hogyan próbálhatja egy X támadó megszemélyesíteni az A felet?

1. X végrehajtja az első lépést, és megfigyeli a második üzenetben a b bitet. Ismerve A nyilvános v kulcsát, X feladata b értékétől függően vagy R küldése, amit könnyedén megtehet, hiszen ő maga generálta azt az első lépés előtt, vagy kényetlen megpróbálkozni azzal, hogy gyököt vonjon a $z \cdot v$

szorzatból modulo n , ami gyakorlatilag kivitelezhetetlen számára. Azaz $\frac{1}{2}$ valószínűséggel tud sikeresen támadni.

2. X megpróbálja predikálni a második lépéssben átküldésre kerülő b bitet. Ha sejtése $b = 0$, akkor az első üzenetben $z = R^2 \bmod n$ -et küldi, majd a harmadik üzenetben $R \bmod n$ -et. Ha sejtése $b = 1$, akkor az első üzenetben $z = R^2 \cdot v^{-1} \bmod n$ -et küldi, a harmadik üzenetben pedig ismét $R \bmod n$ -et. Így X sikeres lehet, de mivel a b bitet valódi véletlen módon sorsoltuk, ezért C nem képes azt $\frac{1}{2}$ -nél nagyobb valószínűsséggel helyesen predikálni. Következőképpen: X ezzel a módszerrel is csak $\frac{1}{2}$ valószínűsséggel tudja A -t sikeresen megszemélyesíteni.

Ha tehát t számszor megismételjük a fenti protokollt, akkor már csak 2^{-t} annak valószínűsége, hogy X sikeresen tudja megszemélyesíteni A -t.

A módszer egy finomítása az, mikor k számú titkos kulcs – nyilvános kulcs párt ad a központ egy újonnan a rendszerbe lépő félnek, azaz egy $(v_1, v_2, \dots, v_k; u_1, u_2, \dots, u_k)$ $2k$ elemű vektort, ahol $v_i = u_i^2 \bmod n$. Továbbá, a protokoll második lépéssében, B nem egy véletlen bitet küld át A -nak, hanem k bitet, azaz egy b_1, b_2, \dots, b_k bináris sorozatot. A harmadik lépéssben

$$w = R \cdot u_1^{b_1} \cdot u_2^{b_2} \cdots \cdot u_k^{b_k} \bmod n$$

kerül átküldésre. Ennek megfelelően B azt ellenőrzi, hogy fennáll-e a

$$w^2 = z \cdot v_1^{b_1} \cdot v_2^{b_2} \cdots \cdot v_k^{b_k} \pmod{n}$$

egyenlőség. Ezen módosítással a protokoll egyszeri végrehajtása után is már 2^{-k} annak a valószínűsége, hogy X sikeresen tudja A -t megszemélyesíteni.

A Fiat–Shamir-protokoll jól alkalmazható intelligens, kártyás azonosító-rendszerekben, ahol a kártya és a tulajdonosa az A fél, akik egy azonosító terminállal mint B féllel állnak szemben. Tegyük fel, hogy az azonosító terminálok üzemeltetője és a kártya kiállítója nem ugyanaz a szervezet, azaz például nem egy pénzintézet. A terminálok bárki tulajdonában lehetnek, így a kártyának fel kell készülnie arra is, hogy a terminál a partner-hitelesítés kapcsán megpróbál jogtalanul titkos ismeretekhez jutni, amelyeket a terminál tulajdonosa esetleg felhasználhat a kártya tulajdonosának megszemélyesítésére. Gondolunk például arra, hogy a kártya pénzhelyettesítőként funkcionál egy POS- (Point Of Sale)-terminál felé. A cél tehát az, hogy a rendeltetésüknek megfelelően használt terminálok biztonságosan azonosíthassák a kártyát (tulajdonosát), míg csaló szándékú terminálok semmilyen használható titokhoz ne jussanak.

Ezen alkalmazásnál a nyilvános kulcsot $v = f(I, c)$ módon számoljuk, ahol f egy hash függvény, I az ügyfél nyilvános azonosítója (például neve, számlaszáma, kártyaszáma), továbbá c valamely kis (nyilvános) érték, melynek egyetlen megválasztási szempontja az, hogy v kvadratikus maradék legyen. A hash függvény alkalmazásának célja a v méretének használható méretűvé szűkítése, és emelett annak biztosítása, hogy gyakorlatilag ne fordulhasson elő, hogy két különböző felhasználó azonos v nyilvános kulcsot kap. Az u titkos kulcs a kártyában ki nem olvasható módon kerül elhelyezésre. A kártya lehetővé teszi az I és c adatelemek ellenőrzését. A kártya tehát azonosítja a terminál felé a (jogos) tulajdonosát anélkül, hogy az bármit is megtudna a titkos elemmel, azaz a titkos kulccsal kapcsolatban. A kártya helyettesíti jogos tulajdonosát, annak közreműködése nélkül működik, következésképpen elvesztése egy csaló megtalálót egyszerű helyzetbe hoz.

A protokoll, az RSA hatványozásával szemben, csak szorzást használ, ami jóval kisebb számításigényt jelent.

9.4. Feladatok

9.1. Feladat. Tegyük fel, hogy egy A űrjármű leszálláshoz készülődik egy távoli bolygó B űrállomásán, s ehhez először azonosítania kell magát. A feltételek a következők:

1. B ismeri A jelszavát, ezen kívül más közös titkuk nincs.
2. A mod 2 összeadásnál bonyolultabb műveletet nem tud végezni.
3. A lesugárzott jeleket egy, az űrállomás környéki C támadó is lehallgathatja, mivel nem lehet jól koncentrálni a sugárzást. Ugyanakkor a bolygon levő űrállomás képes úgy jeleket továbbítani, hogy azok a bolygó felszínén nem vehetők.

Javasoljon egy kétlépéses protokolلت az azonosításra!

9.2. Feladat. A jelszavas rendszerek ismert problémája a redundáns, megjegyezhető jelszavak választása, ami által a jelszótér mérete lényegesen leszűkül, s így könnyebben támadható. Az ember nem szívesen memorizál véletlenül generált alfanumerikus füzereket. Lehetséges-e ennek ellenére egy olyan megoldás a problémára, amely hosszabb értelmes szövegekből indul ki jelszóválasztáskor?

9.3. Feladat. Attól tartunk, hogy egy aktív támadó képes az átküldött jelszót „elfogni”, s kicsit később saját céljaira felhasználni. Ez ellen még az egyszer használatos jelszó sem véd. Szeretnénk tehát, hogy az átküldött „azo-

nosító” ne legyen felhasználható a támadó által még ez esetben sem. Nem támászkodhatunk rejtjelező kódolóra, vagy véletlen forrásra. Hash függvényünk azonban van. Mit tehet két, egyébként egymásban megbízó fél?

9.4. Feladat. A Woo–Lam-protokollban A hitelesíti magát B -nek az S megbízható harmadik fél segítségével. S -re azért van szükség, mert feltételezzük, hogy A és B nem rendelkezik közös kulccsal. Ezzel szemben feltesszük, hogy A és S , valamint B és S rendelkezik egy közös K_{AS} , illetve K_{BS} kulccsal. A protokoll lépései a következők:

Woo–Lam-protokoll

- (1) $A \rightarrow B: A$
 - (2) $B \rightarrow A: N_B$
 - (3) $A \rightarrow B: \{N_B\}_{K_{AS}}$
 - (4) $B \rightarrow S: \{A | \{N_B\}_{K_{AS}}\}_{K_{BS}}$
 - (5) $S \rightarrow B: \{N_B\}_{K_{BS}}$
-

Az első üzenetben A elküldi saját azonosítóját B -nek, ami egy frissen generált N_B kihívással válaszol. B rejtjelezí N_B -t a K_{AS} kulccsal, és az eredményt visszaküldi B -nek. B nem tudja dekódolni A válaszát, ezért továbbküldi azt S -nek, melléteve A azonosítóját, és az egész üzenetet a K_{BS} kulccsal rejtjelezve. S dekódolja az üzenetet sorban minden kulccsal, majd N_B -t rejtjelezí K_{BS} -sel, és az eredményt visszaküldi B -nek. Most már B is dekódolni tudja az üzenetet, és ellenőrzi, hogy visszakapta-e N_B -t. Ha igen, akkor B úgy gondolja, hogy A hitelesítette magát.

1. Mutassuk meg, hogy egy legalis, de rosszindulatú X résznevő meg tudja személyesíteni A -t! (Segítség: Mivel X legalis résznevő, ezért rendelkezik egy K_{XS} kulccsal, és kezdeményezni tudja a protokolلت B -vel.)
2. Javítsuk ki a protokolلت!

9.5. Feladat. Anna be kívánja bizonyítani Bélának, hogy tudja $y = g^x$ mod p diszkrét hatvány g szerinti diszkrét logaritmusát, azaz x -et, ahol p egy prímszám, és g egy primitív elem mod p . Mindezt úgy szeretné tenni, hogy eközben ne kelljen Bélának felmutatnia x -et, azaz Béla úgy nyerjen bizonyosságot, hogy végül ne ismerje meg x értékét.

1. Javasoljon protokolلت!
2. Hogyan használná ezt a módszert egy tetszőleges, titokkal kapcsolatos ismeret bizonyítására?

III.

Alkalmazások

10.

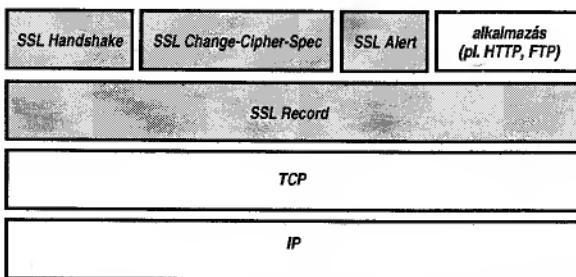
Internet biztonsági protokollok

Ebben a fejezetben három internet biztonsági protokollt mutatunk be. Ezek az SSL, az IPSec és a PGP protokollok. E három protokoll több szempontból is érdekes példának tekinthető. Egyrészt azért, mert e három protokoll a hálózati architektúra három különböző rétegében helyezkedik el: a PGP az alkalmazási, az SSL a szállítási, az IPSec pedig a hálózati réteghez tartozik. Ezért ezen protokollok azt példázzák, hogy a biztonsági szolgáltatásokat minden rétegen meg lehet valósítani, természetesen figyelembe véve az adott réteg sajátosságait.

Másrészt, ez a három protokoll három különböző típusú kommunikációs szolgáltatás biztonságát igyekszik megoldani. Az SSL feladata egy megbízható, kapcsolatorientált szolgáltatás (TCP) biztonságossá tétele, az IPSec feladata pedig egy megbízhatatlan, datagram szolgáltatás (IP) védelme. A PGP egyik fő alkalmazási területe az elektronikus levelezés biztonságossá tétele. Az e-mail sajátossága, hogy a kommunikáció aszinkron jellegű, azaz a küldő és a vevő nincs egy időben jelen. Látni fogjuk majd, hogy a kommunikációs szolgáltatás típusa nagy mértékben befolyásolja a védelemre szánt biztonsági protokoll tulajdonságait.

10.1. SSL (Secure Socket Layer)

A Netscape még a web korszak elején, a 90-es évek közepén felismerte, hogy a web technológia minden napjai életünk részévé fog válni, és olyan feladatokra is használni fogjuk majd, melyek biztonsági követelményeket is támasztanak. A Netscape célja tehát az volt, hogy a webböngésző és a web-



10.1. ábra. Az SSL illeszkedése az internet protokoll-architektúrájába

szerver közötti kapcsolatot biztonságossá tegye. Ezt a feladatot meg lehetett volna oldani a web protokolla, a HTTP szintjén is (voltak ilyen próbálkozások is, például Secure-HTTP). A Netscape azonban egy olyan általános célú protokolloval fejlesztett ki, ami lehetővé teszi biztonságos TCP kapcsolatok kiépítését tetszőleges, amúgy TCP-t használó alkalmazások (például HTTP, FTP, SMTP stb.) között. Mivel a TCP programozói interfésze az ún. *socket* absztrakcióra épül, ezért az új protokolloval Secure Socket Layer-nek, röviden SSL-nek nevezték el. Mára az SSL de facto szabvánnyá vált, szinte minden webböngésző és webszerver implementáció támogatja. Sőt, TLS néven (Transport Layer Security) és kisebb javításokkal, az SSL megindult az internet szabvánnyá válás útján is (lásd RFC 2246).

Az SSL protokoll tehát a TCP réteg felett és az alkalmazások alatt helyezkedik el az internet protokoll-hierarchiájában. Az SSL-lel kiegészített protokoll architektúrát a 10.1. ábra szemlélteti. Mint az az ábrán is látható, az SSL protokoll négy alprotokollból áll, melyek a következők:

- **Record protokoll:** A Record protokoll feladata a kliens és a szerver (például egy webböngésző és egy webszerver), valamint a felsőbb SSL protokoll entitások (Handshake, Change-Cipher-Spec és Alert) közötti kommunikáció védelme, mely titkosítást, integritásvédelmet és üzenetvisszajátszás elleni védelemet jelent.
- **Handshake protokoll:** A kliens és a szerver a Handshake protokoll segítségével egyezteti a Record protokollban használt kriptográfiai algoritmusokat és az algoritmusok paramétereit, beleértve a kapcsolatkulcsokat is. A Handshake protokoll további feladata a felek hitelesítése. Tipikusan a szerver minden hitelesítő magát, a kliens hitelesítés azonban opcionális.

- *Change-Cipher-Spec protokoll:* A Change-Cipher-Spec protokoll egyetlen üzenetből áll, mely a Handshake protokoll kulcscsere részének végét jelzi. Ezen üzenet elküldése után az adott fél az új algoritmusokat és kulcsokat kezdi használni küldésre, a vétel azonban még mindig a Handshake előtti állapot szerint történik. Mikor az adott fél megkapja a másik fél Change-Cipher-Spec üzenetét, akkor a vételi állapotát megváltoztatja, azaz vételre is az új algoritmusokat és kulcsokat kezdi használni.
- *Alert protokoll:* Az Alert protokoll feladata a figyelmeztető- és hibaüzenetek továbbítása.
A továbbiakban a Record és a Handshake protokollokat tárgyaljuk részletesen. A Change-Cipher-Spec protokollra a Handshake protokoll leírásánál még utalni fogunk. Az Alert protokollal nem foglalkozunk.

10.1.1. Az SSL Record protokoll

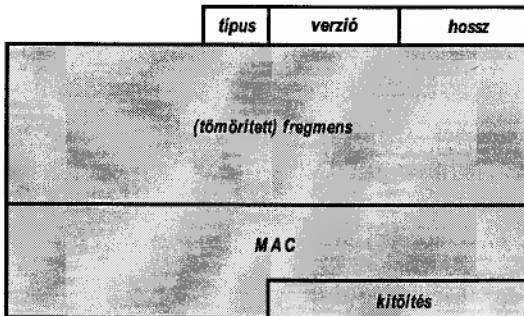
Az SSL Record protokoll a felsőbb protokoll rétegektől érkező üzeneteket a következő lépésekben keresztül dolgozza fel:

1. a hosszú üzeneteket fragmentálja,
2. a fragmenseket tömöríti,
3. minden tömörített fragmenst fejléccel lát el,
4. a fejléccel ellátott, tömörített fragmensre üzenethitelesítő kódot (MAC) számol, és azt a fragmenshez csatolja,
5. majd az üzenethitelesítő kóddal ellátott fragmenst rejtjelezí.

A vevő a feldolgozást értelemszerűen ellentétes sorrendben végzi. A feldolgozás során használt tömörítő, üzenethitelesítő és rejtjelezési algoritmusokban, paraméterekekben, illetve kulcsokban a Handshake protokoll végrehajtása során egyeznek meg a felek.

A fenti lépések eredményeként előálló Record üzenet formátumát a 10.2. ábra mutatja. A fejléc három mezőt tartalmaz, melyek a következők:

- *típus (type):* A típusmező értéke utal arra, hogy a Record üzenet belséjében melyik felsőbb protokoll üzenete (vagy üzenetfragmense) található. Ennek megfelelően a típus mező négyféle értéket tartalmazhat: SSL Handshake, SSL Change-Cipher-Spec, SSL Alert és alkalmazás.
- *verzió (version):* A verziómező az éppen használt SSL verziót tartalmazza. A jelenlegi legmagasabb verzió a 3.0, mi is ezt tárgyaljuk. Néhány alkalmazásban a verziómezőt a 1.0.3. értékkel jelölik.



10.2. ábra. Az SSL Record protokoll üzenet formátuma

mazás még mindig támogatja az SSL 2.0 verzióját is. Ez azonban komoly biztonsági hiányosságokkal rendelkezik, ezért használata kerülendő.

- *hossz (length)*: A hossz mező a Record üzenet belsejében található (tömörített) fragmens hosszát tartalmazza bájtban mérve.

Az üzenethitelesítő kód generálása a HMAC egy korai verziójával történik az alábbiak szerint:

$$MAC = H(K_{\text{write}}^{\text{MAC}} | pad_2 | H(K_{\text{write}}^{\text{MAC}} | pad_1 | seqnum | type | length | payload)),$$

ahol:

- *H* az MD5 vagy a SHA-1 hash függvény, attól függően, hogy a Handshake során miben egyeztek meg a felek.
- $K_{\text{write}}^{\text{MAC}}$ a MAC érték generálásához használt titkos kulcs, melyet csak a kliens és a szerver ismer. Az SSL különböző irányokban különböző kulcsot használ, azaz a klienstől a szerver felé tartó üzenetek integritását egy $K_{C \rightarrow S}^{\text{MAC}}$ kulccsal, a szervtől a kliens felé tartó üzenetek integritását pedig egy $K_{S \rightarrow C}^{\text{MAC}}$ kulccsal védi. A kliens a $K_{C \rightarrow S}^{\text{MAC}}$ kulcsra $K_{\text{write}}^{\text{MAC}}$ néven hivatkozik (küldésre használt MAC kulcs), a $K_{S \rightarrow C}^{\text{MAC}}$ kulcsra pedig a $K_{\text{read}}^{\text{MAC}}$ néven (vételre használt MAC kulcs). A szerver oldalon az egyes kulcsok szerepe nyilván fordított.
- pad_1 és pad_2 két konstans bájtsorozat.
- $seqnum$ az üzenet sorszáma. Az SSL implicit üzenetsorszámot használ, ami azt jelenti, hogy a sorszám nem jelenik meg explicit mezőként az üzenetben (lásd 10.2. ábra), de a MAC számításban felhasználják aktuális

értékét, melyet minden fél lokálisan számon tart. Ezt azért lehet így megtenni, mert az SSL a TCP protkoll felett helyezkedik el, és a TCP normális körülmények között biztosítja az üzenetek sorrendhelyes vételét. Ezért általában igaz az, hogy mindenkor a várt sorszámu üzenetet veszik a felek. Ha valamelyen támadás vagy hiba folytán nem a várt sorszámu üzenet érkezik meg (ami tehát abnormális jelenség), akkor a MAC ellenőrzés sikertelen lesz, és a vevő bontja a kapcsolatot.

- *type* és *length* az üzenet fejlécében található típus és hossz mezők értéke.
- *payload* az üzenetben található (tömörített) fragmens.

A 10.2. ábrán a Record üzenet rejtjelezett részét szürke színnel jelöltük. Mint látható, a fejléc kivételével az egész üzenet rejtjelezve van. Az SSL alapértlemezett (default) rejtjelező algoritmus az RC4 kulcsfolyamatos rejtjelező, de a Handshake protokoll végrehajtása során a felek más algoritmusban is megegyezhetnek. Az SSL támogatja még az RC2, a DES, a három kulcsos 3DES, az IDEA és a Fortezza blokkrejtjelezők CBC módban történő használatát. Amennyiben a felek valamelyik blokkrejtjelező használatában egyeznek meg, úgy a rejtjelezés előtt minden Record üzenetet ki kell tölteni, hogy hossza a rejtjelező blokkméretének egész számú többszöröse legyen. Az SSL az 5.1. algoritmus szerinti kitöltési sémát használja. A kitöltés bájtjai a MAC után kerülnek az üzenetbe. Kulcsfolyamatos rejtjelező használata esetén nincs kitöltés.

Blokkrejtjelező használata esetén, a CBC mód miatt, szükség van *IV*-re is. Az első üzenet rejtjelezéséhez használt *IV*-t a Handshake során generálják a felek (a kulcsokkal együtt). minden további üzenetnél az előző üzenet utolsó rejtjeles blokkját használják *IV*-nek. Ez gyakorlatilag azt jelenti, mintha a kapcsolat során elküldött összes üzenetet egyetlen nagy üzenetként CBC módban rejtjeleznék (az üzenetenkénti kitöltéstől eltekintve).

10.1.2. Az SSL viszony és az SSL kapcsolat

Mielőtt a Handshake protokoll részletes ismertetésébe kezdenénk, be kell vezetnünk az SSL viszony (session) és az SSL kapcsolat (connection) fogalmát és a közöttük fennálló kapcsolatot. Az SSL két fél között több párhuzamos viszony kialakítását teszi lehetővé, és minden viszonyon belül több kapcsolat lehetséges. Ezen lehetőségek közül azonban a legtöbb implementáció csak az egy viszonyon belüli több kapcsolatot támogatja, a több párhuzamos viszony lehetőségét nem. Tehát mi is azt fogjuk most feltételezni, hogy a felek között egy viszony van, és a viszonyon belül több kapcsolat lehetséges.

Mind a viszony, mind a kapcsolatok a felek közös állapotának egy-egy részletét írják le. A viszony tartalmazza az állapot azon részleteit, amit a viszonyon belüli kapcsolatok megosztanak, közösen használnak. A viszony része például a viszony azonosító, a felek nyilvános kulcs tanúsítványa (ha rendelkeznek ilyennel), a tömörítő algoritmus, a rejtjelező és MAC algoritmus, az ezen algoritmusok által használt méretek (például a MAC kulcs hossza) és egy mestertitok (master secret). A viszony része még az ún. „is resumable” jelzőbit, melynek 1 értéke azt jelenti, hogy az adott viszonyon belül még létrehozható új kapcsolat, 0 értéke pedig azt, hogy új kapcsolat létrehozása nem lehetséges. Ez a jelzőbit akkor válik 0 értékűvé, ha valamelyik, a viszonyhoz tartozó kapcsolatban egy fatális hiba keletkezik (például egy üzenet MAC értékének ellenőrzése sikertelen). Ekkor a még folyamatban levő kapcsolatok folytatódnak, de a jelzőbit nullázása azt eredményezi, hogy új kapcsolatot a viszonyon belül már nem lehet létrehozni.

A kapcsolat részét képezi például a kapcsolatban használt két rejtjelező kulcs, a két MAC kulcs (különböző irányokban különböző kulcsokat használ a protokoll mind a rejtjelezéshez, mind a MAC számításhoz), a különböző irányokban használt üzenetsorszámok, és blokkrejtjelező használata esetén az IV-k. Fontos tehát látni, hogy minden kapcsolatnak saját kulcsai vannak, ezért a kulcsok a kapcsolat részét képezik, ugyanakkor az egy viszonyhoz tartozó kapcsolatok ugyanazokat az algoritmusokat használják, és a mestertitok is közös, amiből a kapcsolatkulcsokat generálják. Ezért az algoritmusok és a mester titok a viszonyhoz tartoznak.

10.1.3. Az SSL Handshake protokoll

A Handshake protokoll feladata a két fél közötti viszony felépítése, vagy ha az már létezik, és annak „is resumable” jelzőbitje 1 értékű, akkor a viszonyon belül egy új kapcsolat létrehozása. Az SSL Handshake protokoll egy komplex protokoll. A komplexitás legfőbb oka az, hogy a Handshake protokoll többféle kulccsere módszer használatát is támogatja, és a Handshake üzenetek értelmezése az egyes módszerek esetén más és más. A Handshake protokoll általános váza a 10.3. ábrán látható, ahol C jelöli a klienst, S pedig a szervert.

A Handshake protokoll négy fázisra tagolódik. Az első fázisban (1. és 2. üzenet) történik meg az algoritmusok egyeztetése, beleérve a Handshake hátralevő részében használt kulccsere módszert is. Ezen kívül a felek az első fázisban kicsérélnek két frissen generált véletlen számot is, melyet a további számításoknál használni fognak majd. A második fázisban (3–6. üzenetek) a

SSL Handshake protokoll

- (1) $C \rightarrow S$: **client-hello**
 - (2) $S \rightarrow C$: **server-hello**
 - (3) $S \rightarrow C$: certificate
 - (4) $S \rightarrow C$: server-key-exchange
 - (5) $S \rightarrow C$: certificate-request
 - (6) $S \rightarrow C$: **server-hello-done**
 - (7) $C \rightarrow S$: certificate
 - (8) $C \rightarrow S$: **client-key-exchange**
 - (9) $C \rightarrow S$: certificate-verify
 - (10) $C \rightarrow S$: **change-cipher-spec**
 - (10) $C \rightarrow S$: **client-finished**
 - $S \rightarrow C$: **change-cipher-spec**
 - (11) $S \rightarrow C$: **server-finished**
-

10.3. ábra. Az SSL Handshake protokoll váza. A vastagon szedett üzenetek kötelezőek, a többi opcionális (a korábbi üzenetek tartalmától függ, hogy kell-e őket küldeni vagy sem).

szerver a kiválasztott módszernek megfelelően végrehajtja a kulcscsere ráső részét, míg a kliens a harmadik fázisban (7–9. üzenetek) teszi meg ugyanezt. A harmadik fázis után, az addig kicsérélte információkat felhasználva, minden két fél előállítja az új kapcsolatkulcsokat. A negyedik fázisban (10. és 11. üzenet) a felek áttérnek az új algoritmusok és kulcsok használatára. Mindkét fél egy **finished** üzenet küldésével fejezi be a protokolلت, mely az első olyan üzenet, ami már az új algoritmusokat használva, az új kulcsokkal van kódolva.

Vegyük észre, hogy a **change-cipher-spec** üzeneteket a fenti leírásban nem számoztuk meg. Ez azért van, mert ezek az üzenetek a Change-Cipher-Spec protokollhoz tartoznak és nem a Handshake protokoll részei. Erre az SSL analízisénél még vissza fogunk térni.

A **client-hello** üzenet a következő információkat tartalmazza:

- **kliens verzió:** A kliens tájékoztatja a szervert az általa támogatott legmagasabb SSL verzió számáról.
- **véletlenszám:** A kliens generál egy friss véletlenszámot, és elküldi azt a szervernek.

- *viszony azonosító:* Ha a kliens azt szeretné, hogy az új kapcsolat egy már létező viszonyon belül jöjjön létre, akkor elküldi ezen viszony azonosítóját a szervernek. Ha a kliens új viszonyt akar létrehozni, akkor ez a mező üres.
- *biztonsági algoritmusok:* A kliens tájékoztatja a szervert az általa támogatott biztonsági algoritmusokról. Ez a mező egy, a kliens preferenciái szerint rendezett listát tartalmaz, ahol a lista elemei a támogatott algoritmus-kombinációk azonosítói. Egy ilyen listaelem lehet például az SSL-RSA-with-3DES-EDE-CBC-SHA azonosító, mely a következő algoritmusok kombinációját jelöli:
 - RSA alapú kulccscsere módszer,
 - 3DES rejtjelezés, EDE konfigurációban, CBC módban,
 - SHA hash függvény és arra épülő MAC.

A megjelölt algoritmusok közvetve definiálják a paramétereket is (például 3DES esetén az IV méret 64 bit, SHA esetén a MAC mérete 160 bit stb.).

- *tömörítő algoritmusok:* A kliens tájékoztatja a szervert az általa támogatott tömörítő algoritmusokról. Az előző mezőhöz hasonlóan ez a mező is egy preferencia szerint rendezett listát tartalmaz, ahol azonban a lista elemeit tömörítő algoritmusok azonosítói alkotják.

A server-hello üzenet ugyanazokból a mezőkből áll, mint a client-hello üzenet, csak a mezők értelmezése kicsit más:

- *szerver verzió:* A szerver azt a legmagasabb verziót választja, melyet mind a kliens, mind a szerver támogat, és erről tájékoztatja a klienst.
- *véletlenszám:* A szerver is generál egy (a kliensétől független) friss véletlenszámot, és elküldi azt a kliensnek.
- *viszonyazonosító:* Ha a kliens javasolt egy viszonyazonosítót, akkor a szerver ellenőrzi, hogy az adott viszonyon belül lehet-e még új kapcsolatot létrehozni („is resumable” bit értéke 1). Ha igen, akkor a szerver az adott viszony azonosítójával válaszol. Ha a viszonyon belül már nem lehet új kapcsolatot létrehozni, vagy a kliens nem javasolt viszonyazonosítót, akkor a szerver generál egy új viszonyazonosítót, és azt küldi el a kliensnek.
- *biztonsági algoritmusok:* A szerver választ egy algoritmus-kombinációt a kliens listájáról, és csak a kiválasztott kombináció azonosítóját küldi vissza a kliensnek.

- *tömörítő algoritmusok*: A szerver választ egy algoritmust a kliens listájáról, és annak azonosítóját küldi vissza a kliensnek.

A további üzenetek értelmezése attól függ, hogy milyen kulcscseremódszerben egyeztek meg a felek. Az SSL öt kulcscseremódszert támogat. Ezek a következők:

- *RSA alapú*: RSA alapú kulccsere esetén a kliens generál egy 48 bajt méretű véletlen blokkot, amit a szerver nyilvános RSA kulcsával kódolva elküld a szervernek. Ebből a véletlen blokkból aztán mind a kliens, mind a szerver előállítja a közös mestertitkot.
- *fix Diffie–Hellman*: Fix Diffie–Hellman-kulccsere esetén a felek a Diffie–Hellman-algoritmust használják, és az ennek eredményeként kialakult közös értékből generálják a mestertitkot. A fix jelző arra utal, hogy a szerver Diffie–Hellman-paraméterei (p , g , $g^x \bmod p$) fixek, azokat egy hitelesítés-szolgáltató aláírta, és az erről szóló tanúsítványt a kliens ellenőrizni tudja.
- *egyszer használatos Diffie–Hellman*: Az előző módszerhez hasonlóan Diffie–Hellman-algoritmust használnak a felek, de a szervernek nincsenek fix, aláírt Diffie–Hellman-paraméterei. Helyette a szerver egyszer használatos paramétereket generál, és azokat RSA vagy DSS aláíró kulcsával aláírva juttatja el a kliensnek. A kliens minden a fix, minden az egyszer használatos Diffie–Hellman-kulccsere esetén egyszer használatos Diffie–Hellman nyilvános értéket generál a szerver p és g paramétereit használva.
- *anonim Diffie–Hellman*: Ezen módszer használata esetén a felek az eredeti, hitelesítés (aláírás) nélküli Diffie–Hellman-algoritmust hajtják végre. Ezen módszer használata nem tanácsos, ha az aktív támadások veszélyét nem lehet kizárni.
- *Fortezza*: A Fortezza kulccsere protokoll a Netscape saját fejlesztésű módszere, amivel itt most nem foglalkozunk.

A Handshake protokoll második fázisának első üzenete a *certificate* üzenet. Ez egy hitelesítésszolgáltató által aláírt tanúsítvány (vagy ilyen tanúsítványok lánca), amit az anonim Diffie–Hellman kivételével, minden kulccsere módszer esetén elküld a szerver. A tanúsítvány tartalma azonban változó. Tartalmazhat egy nyilvános RSA rejtelező kulcsot (RSA alapú kulccsere), vagy egy RSA, vagy DSS aláírás ellenőrző kulcsot (RSA alapú kulccsere), vagy egyszer használatos Diffie–Hellman), vagy a szerver Diffie–Hellman paramétereit (fix Diffie–Hellman).

A következő üzenet a **server-key-exchange** üzenet. Ezt csak abban az esetben küldi a szerver, ha az előző lépésben küldött tanúsítvány nem tartalmaz elegendő információt a kulccscsere befejezéséhez. Tipikusan, ha a tanúsítvány csak egy aláírásellenőrző kulcsot tartalmazott, akkor a szerver a **server-key-exchange** üzenetben küldi el a kliensnek a rejtjelezésre alkalmas RSA kulcsot (RSA alapú kulccscsere), vagy az egyszer használatos Diffie–Hellman paramétereit. A **server-key-exchange** üzenetet a szerver digitálisan aláírja. Ehhez először a **server-key-exchange** üzenetben elküldött kulccscsere paramétereket és az első fázisban kicserélt véletlenszámokat összefűzi, majd az eredmény hash értékén képezi az aláírást.

Ha a szerver szeretné hitelesíteni a klienst (ez opcionális), akkor küld egy **certificate-request** üzenetet, melyben specifikálja, hogy milyen tanúsítványokat vár a klienstől. Végül, a második fázist a **server-hello-done** üzenet zárja le, melyben a szerver jelzi a kliensnek, hogy befejezte a kulccscsere réseső részét.

Ha a szerver küldött **certificate-request** üzenetet, azaz szeretné hitelesíteni a klienst, akkor a protokoll harmadik fázisa egy **certificate** üzenettel kezdődik, melyben a kliens elküldi a szerver által kért tanúsítványokat.

A következő üzenet a **client-key-exchange**, melyet minden küld a kliens. A megegyezett kulccscsere módszertől függően, a **client-key-exchange** üzenet tartalmazhatja a kliens által generált és a szerver nyilvános RSA kulcsával rejtjelezett 48 bájtos véletlen blokkot (RSA alapú kulccscsere), vagy a kliens egyszer használatos Diffie–Hellman nyilvános értékét.

Végül, ha a kliens küldött **certificate** üzenetet, akkor a harmadik fázist a **certificate-verify** üzenettel fejezi be. Ez az üzenet tartalmazza az összes eddig (a kliens által vett és küldött) Handshake üzenet hash értékét digitálisan aláírva, ahol az aláírás a kliens által korábban küldött **certificate** üzenetben található aláírásellenőrző kulccsal ellenőrizhető.

A Handshake protokoll harmadik fázisa után a kliens és a szerver az következő módon előállít egy közös K mestertitkot: Jöljük K' -vel a kliens által generált 48 bájtos véletlen blokkot, melyet RSA alapú kulccscsere esetén a kliens a szerver nyilvános RSA kulcsával rejtjelezve juttat el a szervernek. Fix, egyszer használatos vagy anonim Diffie–Hellman-kulccscsere esetén pedig K' jelölje a Diffie–Hellman-algoritmus $g^{xy} \bmod p$ végeredményét. Ekkor

$$\begin{aligned} K = & \text{MD5}(K' | \text{SHA}("A" | K' | N_C | N_S)) | \\ & \text{MD5}(K' | \text{SHA}("BB" | K' | N_C | N_S)) | \\ & \text{MD5}(K' | \text{SHA}("CCC" | K' | N_C | N_S)), \end{aligned}$$

ahol N_C és N_S az első fázisban kicserélt véletlen számok, "A", "BB", és "CCC" pedig az "A", a "B" illetve a "C" ASCII karakterekből alkotott egy, kettő, illetve három hosszú karakterláncok. Mivel az MD5 hash függvény kimenetének mérete 16 bájt, ezért a három hash érték összefűzéséből nyert K mestertitok 48 bájt hosszú.

A Handshake protokoll negyedik fázisában a felek egy-egy **finished** üzenetet küldenek egymásnak. A **finished** üzenetek célja az egész Handshake hitelesítése, azaz az esetleges aktív támadások detektálása. Ezen cél elérése érdekében minden fél kiszámítja az összes vett és küldött Handshake üzenet, valamint az imént kiszámolt mestertitok összefűzésével nyert „szuper üzenet” hash ertékét. Ez a következő formula szerint történik:

$$\text{MD5}(K \mid pad_2 \mid \text{MD5}(msgs \mid sender \mid K \mid pad_1)) \mid \\ \text{SHA}(K \mid pad_2 \mid \text{SHA}(msgs \mid sender \mid K \mid pad_1)),$$

ahol *msgs* jelöli a Handshake üzeneteket, *sender* pedig egy, a küldő féltől függő konstans.

10.1. Példa. Tegyük fel, hogy a felek az RSA alapú kulccscsere módszerben egyeztek meg, a szerver tanúsítványa tartalmazza a szerver RSA rejtjelező kulcsát, és a szerver nem szeretné hitelesíteni a klienst. Ekkor a Handshake protokoll üzenetei a következőképpen alakulnak:

-
- | | | |
|------|--------------------|---------------------|
| (1) | $C \rightarrow S:$ | client-hello |
| (2) | $S \rightarrow C:$ | server-hello |
| (3) | $S \rightarrow C:$ | certificate |
| (6) | $S \rightarrow C:$ | server-hello-done |
| (8) | $C \rightarrow S:$ | client-key-exchange |
| | $C \rightarrow S:$ | change-cipher-spec |
| (10) | $C \rightarrow S:$ | client-finished |
| | $S \rightarrow C:$ | change-cipher-spec |
| (11) | $S \rightarrow C:$ | server-finished |
-

A **hello** üzenetek cseréje után a szerver a **certificate** üzenetben elküldi az RSA rejtjelező kulcsát tartalmazó tanúsítványát. A kliens generál egy 48 bájtos véletlen blokkot, majd a kapott rejtjelező kulcsot használva rejtjelezzi azt, és az eredményt a **client-key-exchange** üzenetben elküldi a szervernek. Ezután minden fél kiszámolja a mestertitkot, és a **change-cipher-spec**, valamint a **finished** üzenetek elküldésével befejezik a protokolلت. ♣

10.2. Példa. Tegyük fel, hogy a felek az egyszer használatos Diffie–Hellman-kulcscsere módszerben egyeztek meg, a szerver tanúsítványa egy DSS aláírásellenőrző kulcsot tartalmaz, és a szerver DSS aláírással szeretné hitelesíteni a klienst. Ekkor a Handshake protokoll minden üzenetét elküldik a felek.

A hello üzenetek cseréje után a szerver a certificate üzenetben elküldi a DSS aláírás ellenőrző kulcsát tartalmazó tanúsítványát. Utána generálja az egyszer használatos Diffie–Hellman paramétereit (p és g) és publikus értékét ($g^x \bmod p$), majd ezeket DSS aláírással ellátva a server-key-exchange üzenetben elküldi a kliensnek. Végül a certificate-request üzenetben DSS aláírás ellenőrző kulcsot tartalmazó tanúsítványt kér a klienstől.

A kliens ennek megfelelően a certificate üzenetben elküldi a DSS aláírás ellenőrző kulcsát tartalmazó tanúsítványát a szervernek. Ezután a szerver p és g Diffie–Hellman paramétereit használva, ő is generál egy egyszer használatos Diffie–Hellman publikus értéket ($g^y \bmod p$), majd elküldi azt a szervernek a client-key-exchange üzenetben. Végül az összes eddig vett és küldött Handshake üzenet hash értékének DSS aláírását küldi el a szervernek a certificate-verify üzenetben.

Ezután minden fél kiszámolja a közös mestertitkot, és a change-cipher-spec valamint a finished üzenetek elküldésével befejezik a protokollt. ♣

A finished üzeneteket a Record protokoll már az új algoritmusokat és kulcsokat használva kódolja. Az ehhez szükséges MAC kulcsokat, rejtjelező kulcsokat, IV -ket stb. a mestertitokból generálják a felek a következő módon: először a mester titokból létrehoznak egy megfelelő hosszúságú random bájtsorozatot, majd ezt a kulcsok és IV -k méretének megfelelő szeletekre vágják. A random bájtsorozat generálása a következő kifejezés szerint történik:

$$\begin{aligned} & \text{MD5}(K \mid \text{SHA("A" } | K | N_C | N_S)) \mid \\ & \text{MD5}(K \mid \text{SHA("BB" } | K | N_C | N_S)) \mid \\ & \text{MD5}(K \mid \text{SHA("CCC" } | K | N_C | N_S)) \mid \dots \end{aligned} \tag{10.1}$$

A fenti kifejezésben minden sor 16 bájtot állít elő, és ezeket fűzzük össze, hogy megfelelő mennyiségű véletlen bájtot kapunk. Ha például 80 bájtra van szükségünk, akkor azt öt 16 bájtos blokkból tudjuk összerakni, azaz ötször kell meghívni az MD5 és SHA függvényeket a fenti módon.

A teljes Handshake protokoll végrahajtására csak akkor van szükség, ha egy új viszonyt akarnak létrehozni a felek. Ha a kliens egy már meglévő

viszonyon belül szeretne egy új kapcsolatot létrehozni, és a szerver ezt engedélyezi („is resumable” bit 1 értékű), akkor a Handshake első fázisa után rögtön a harmadik fázis végére ugranak a felek a végrehajtásban, vagyis (10.1) alapján új kapcsolatkulcsokat generálnak a meglévő K mestertitok és az első fázisban generált friss N_C és N_S véletlenszámokat használva, majd elküldik change-cipher-spec és finished üzeneteiket. Ekkor tehát a protokoll futása sokkal rövidebb.

10.1.4. Az SSL protokoll analízise

Az SSL az évek során intenzív analízisnek volt kitéve. A 2.0 verzióban számos gyengeséget fedeztek fel, amit a 3.0 verzióban a Netscape kijavított. Az SSL 3.0 verziója tehát alapvetően stabil és biztonságos protokoll. Apróbb hiányosságok azonban még akadnak. Ezeket igyekszünk összefoglalni ebben a szakaszban. Úgy érezzük, hogy az itt leírt gondolatok megértése elmélyíti a fejezet első részének elolvasása során megszerzett ismereteket.

10.1.4.1. A Record protokoll analízise

A Record protokollt a következő négy szempont szerint vizsgáljuk:

- bizalmasság megsértésére irányuló támadások elleni védelem,
- forgalomanalízis elleni védelem,
- üzenet-visszajátszás elleni védelem,
- üzenetintegritás és hitelesség biztosítása.

Bizalmasság. Általában megállapítható, hogy a bizalmasság megsértésére irányuló támadások ellen jól védekezik az SSL Record protokollja. A védelem alapja a Record üzenetek rejtjelezése. A rejtjelezéshez az SSL rövid élettartamú kapcsolatkulcsokat használ, amiket a hosszabb élettartamú mestertitokból állít elő a kapcsolatban generált friss véletlen értékeket (a kliens és a szerver által generált friss véletlenszámokat) is felhasználva. A kapcsolatkulcsok előállítása egyirányú függvényel történik, ezért egy kapcsolatkulcs kompromittálódása nem veszélyezteti a mestertitkot. Külön pozitívumként mondható el, hogy az SSL különböző kapcsolatokban, és a kapcsolatokon belül különböző irányokban különböző kulcsokat használ. Ez nehezebb teszi a támadó dolgát, és általában jó tervezési hozzáállásnak minősül. Ezen kívül az SSL által támogatott rejtjelező algoritmusok mind erősek. Meg kell azonban jegyezni, hogy különböző amerikai export szabályok miatt, az al-

goritmusok csonkított (40 bites) kulcsokkal történő használatát is támogatja a protokoll, ami a biztonság szempontjából nem előnyös tuljadonság.

Forgalomanalízis. A Record protokoll egyáltalán nem védekezik a forgalomanalízis ellen. Az IP címeket és TCP port számokat nyilván nem is rejtheti, hiszen a TCP és az IP protokollok felett helyezkedik el. Az üzenetek méretét viszont rejthetné, de nem teszi. Ha a felek az RC4 kulcsfolyamatos rejtjelező használatában egyeznek meg, akkor nincs kitöltés, és így a rejtett üzenetek hossza megegyezik a nyílt üzenetek hosszával. Ha a felek blokk-rejtjelezőt használnak, akkor csak a minimálisan szükséges kitöltést alkalmazzák, így egy forgalmat figyelő támadó jó eséllyel tud tippelni a nyílt üzenetek hosszára.

Az átküldött nyílt üzenetek hosszának ismerete további információk meg szerzését teheti lehetővé. Tegyük fel például, hogy egy webböngésző és egy webszerver közötti HTTP forgalmat védjük az SSL protokollal. Ha a támadó egy legális felhasználó, aki szintén hozzáfér a szerveren tárolt weboldalakhoz, akkor könnyen megfigyelheti az oldalak és a hozzájuk tartozó URL-ek hosszát. Mivel az SSL nem rejt a nyílt üzenetek hosszát, ezért a támadó egy másik felhasználó rejtjelezett forgalmát lehallgatva, az üzenetek méretéből következtetni tud arra, hogy a felhasználó mely weboldalakat tölti le a szervertől.

Visszajátszás. A Record protokoll az üzenetek sorszámozásával védekezik a visszajátszás ellen. Mivel az üzeneteken számolt MAC tartalmazza a sor szám aktuális értékét is, ezért a támadó nem tudja az üzenetek sorrendjét észrevétlenül megváltoztatni, illetve nem tud üzeneteket törlni és beszúrni. A sorszám 64 bites, azaz praktikusan sosem fordul körbe.

Integritásvédelem. A Record protokoll minden üzenethez üzenethitelesítő kódot (MAC) csatol, mely védi az üzenet integritását és hitelesíti annak küldőjét. Az alkalmazott MAC algoritmus a HMAC egy korábbi verziója, a MAC kulcsok mérete pedig 128 bit, ami megfelelő biztonságot nyújt. A rejtjelezéshez hasonlóan, a MAC kulcsok is különbözök minden kapcsolatban, és azon belül minden két irányban.

A MAC számítással kapcsolatos egyetlen gyengeség, hogy a MAC függvény bemenete nem tartalmazza a Record üzenet verziómezőjét. Ez nem feltétlenül ad lehetőséget támadásra, de mindenenképpen felveti a következő kérdést: Ha a verziómező értékét a vevő felhasználja a Record üzenet feldol-

gozásánál, akkor azt a MAC számítás bemenetének tartalmaznia kellene; ha viszont a vevő nem használja a verziómezőt, akkor miért van egyáltalán erre a mezőre szükség?

10.1.4.2. A Handshake protokoll analízise

Cipher suite rollback. Az SSL 2.0 verziójában egy támadó el tudta érni azt, hogy a felek 40 bitesre csonkított kulcsokkal használják a rejtjelező algoritmust. Ehhez a támadó a **client-hello** üzenetet módosította, mégpedig úgy, hogy a kliens által javasolt biztonsági algoritmusok listájáról törölte a 40 bitnél nagyobb kulcsokat használó algoritmus-kombinációkat. Így a szerver már csak olyan algoritmus-kombinációt választhatott, ami 40 bites kulcsot használt rejtjelezésre. A kliens és a szerver a támadást nem detektálta, mert a Handshake protokoll 2.0 verziója nem használt **finished** üzeneteket, melyek az aktív támadások detektálását lehetővé tették volna.

A Handshake protokoll 3.0 verziója tehát a **finished** üzenetek segítségével detektálja a Cipher suite rollback támadást. Meg kell azonban jegyezni, hogy a **finished** üzenetek sikeres ellenőrzése csak azon feltétel mellett biztosítja a teljes Handshake hitelességét és sértetlenségét, hogy a mestertitok valóban titkos, sértetlen és hiteles.

A mestertitok hitelességét az alkalmazott kulccscsere módszertől függően többféleképpen biztosíthatja a protokoll. Ha a Handshake során a szerver küld **server-key-exchange** üzenetet, akkor az tartalmazza a szerver aláírást a szerver kulccscsere paraméterein, és ez explicit módon hitelesíti a kliens számára a később létrehozott mestertitket. Ha nincs **server-key-exchange** üzenet, akkor a mestertitok hitelesítése a kliens számára implicit. Például RSA alapú kulccscsere esetén a kliens tudja, hogy csak a szerver képes dekódolni a kliens által rejtjelezett 48 bájtos véletlen blokkot, a szerver nyilvános kulcsát pedig egy megbízható hitelesítés-szolgáltató hitelesíti.

A szerver számára a legtöbb esetben nem biztosított a mestertitok hitelesisége, hiszen a kliens hitelesítés opcionális, és ritkán használják. Más szavakkal, a szerver legtöbbször nem tudja biztosan, hogy ki a kliens, és így azt sem, hogy ki ismeri a mestertitket. Ugyanakkor, ha a szerver kéri a kliens hitelesítést, akkor a kliens digitális aláírása a **certificate-verify** üzenetben explicit módon hitelesíti a szerver számára a később létrehozott mestertitket.

Version rollback. Mivel az SSL 2.0 verziója számos biztonsági rést tartalmaz, ezért a támadó megpróbálhatja elérni, hogy a felek a 2.0 verziót futassák a 3.0 verziót helyett. Ez nyilván csak akkor lehetséges, ha a felek még

támogatják a 2.0 verziót. A támadónak ehhez csak annyit kell tennie, hogy a kliens **client-hello** üzenetében a verziómezőt 2.0-ra változtatja. A szerver így azt hiszi, hogy ez a kliens által támogatott legmagasabb verzió, és így nem tehet mászt, mint maga is 2.0-ra állítja a verzió mező értékét a **server-hello** üzenetben. A kliens ezt úgy értelmezi, hogy a szerver nem támogatja a 3.0 verziót, ezért helyette a 2.0 verziót javasolja. Mindketten azt hiszik tehát, hogy a másik fél által támogatott legmagasabb verzió a 2.0, ezért a továbbiakban azt használják. A dolog pikantériája, hogy a csalást észre sem veszik, mert a Handshake protokoll 2.0 verziójában nincsenek **finished** üzenetek, melyek ezt lehetővé tennék.

Szerencsére az SSL 3.0 verziója képes detektálni a version rollback támadást. Ezt a következő módon teszi. Ha a kliens támogatja a 3.0 verziót, akkor úgy generálja a mester titok alapját képző 48 bájtos véletlen blokkot, hogy abból igazából csak 46 bájt véletlen, az első két bájt pedig a kliens által támogatott legmagasabb verziót, azaz a 3.0 értéket tartalmazza. Ezt még akkor is így teszi, ha amúg a protokoll 2.0 verzióját futtatja. Az így előállított 48 bájtot küldi el aztán a szerver nyilvános RSA kulcsával rejtjelezve a szervernek¹. Ha a szerver támogatja a 3.0 verziót, akkor minden ellenőrzi a kapott blokk első két bájtját, még akkor is, ha amúg a 2.0 verziót futtatja. Vegyük észre, hogy a fenti version rollback támadás csak akkor minősül támadásnak, ha minden fél támogatja a 3.0 verziót. Ekkor azonban a kliens jelzi, a szerver pedig detektálja, hogy a kliens által támogatott legmagasabb verzió a 3.0. Ha ennek ellenére a 2.0 verziót használják, akkor a szerver megszakítja a kapcsolatot.

A **change-cipher-spec** üzenet elnyelése. Említettük, hogy a **change-cipher-spec** üzenet nem a Handshake része, és ezért a **finished** üzenetek számításánál és ellenőrzésénél a **change-cipher-spec** üzeneteket nem veszik figyelembe a felek. Elképzelhető továbbá, hogy valamely SSL implementáció megengedi a **finished** üzenet feldolgozását annak ellenére, hogy nem kapott **change-cipher-spec** üzenetet. Bár ez nem tűnik logikusnak, az SSL specifikációja nem zárja ki ezt a lehetőséget, és a Netscape egyik saját korábbi referencia implementációja (SSLRef 3.0b1) is pont ezt teszi. Ez súlyos hiba, amit egy támadó bizonyos esetekben ki tud használni.

Tegyük fel, hogy a felek olyan algoritmus-kombinációt szeretnének használni, amiben csak üzenethitelesítés van, de nincs rejtjelezés (például SSL-

¹ Megjegyezzük, hogy a Handshake 2.0 verziója csak az RSA alapú kulcscserét támogatja.

RSA-with-NUL-SHA). A támadó hagyja, hogy a Handshake első három fázisa lefusson, majd a kliens **change-cipher-spec** üzenetét elfogja és eldobja, így a szerver nem kapja azt meg. Ekkor a kliens és a szerver felemás állapotba kerülnek: a kliens már az új algoritmusokat használva, azaz MAC értékkel kiegészítve küldi üzeneteit, de a szerver még mindig MAC nélkül várja azokat. Az első üzenet, amit a kliens MAC értékkel ellátva küld a **finished** üzenet. A támadó ezt az üzenetet úgy módosítja, hogy a MAC értéket törli belőle. A szerver elfogadja az így módosított **finished** üzenetet, mert

- MAC nélkül várja a kliens üzeneteit, és
- a **finished** üzenet ellenőrzése sikeres lesz, hiszen annak számítása nem függ az elküldött vagy vett **change-cipher-spec** üzenetektől.

Hasonlóképpen, a támadó elnyeli a szerver **change-cipher-spec** üzenetét is, majd a szerver **finished** üzenetéből törli a MAC értéket. A kliens az előbbiekhez hasonlóan elfogadja a módosított **finished** üzenetet. A Handshake tehát sikeresen lefut, és innentől kezdve a támadó minden Record üzenetből törli a MAC értéket. Ez persze azt is jelenti, hogy a MAC törlése után minden Record üzenetet módosítani tud, és ezt a felek nem veszik észre, mert még mindig felemás állapotban vannak.

A támadás ellen úgy védekezhetünk, hogy a **finished** üzenet feldolgozását addig nem kezdjük el, amíg a **change-cipher-spec** üzenet meg nem érkezett. Az újabb implementációk erre már ügyelnek. Vegyük észre továbbá, hogy a támadás nem működik, ha rejtelezést is használnak a felek, mert a MAC értékkel ellentében a rejtelezést nem tudja eltüntetni a támadó.

Key exchange algorithm rollback. Egy másik lehetséges aktív támadás, mikor a támadó eléri, hogy a kliens RSA alapú kulcscserét, míg a szerver Diffie–Hellman-algoritmust futtasson egyazon Handshake során. Ekkor egy esetleges implementációs figyelmetlenség a következő végzetes támadáshoz vezethet.

Tegyük fel, hogy a kliens egyszer használatos RSA kulcscserét szeretne futtatni, a támadó azonban oly módon módosítja a kliens **client-hello** üzenetét, hogy a szerver számára úgy tűnjön, hogy a kliens egyszer használatos Diffie–Hellman-kulcscserét javasol (azaz a kliens által javasolt **SSL-RSA-with-...** azonosítókat **SSL-DHE-with-...** azonosítókra cseréli a **client-hello** üzenetben). A szerver így biztosan olyan algoritmus-kombinációt választ, melyben a kulcscseré módszer az egyszer használatos Diffie–Hellman. A támadó a **server-hello** üzenetet úgy módosítja, hogy a szerver által választott

SSL-DHE-with-... azonosítót SSL-RSA-with-... azonosítóra cseréli, így a kliens azt hiszi, a szerver RSA alapú kulcscserét választott.

A Handshake protokoll második fázisában a szerver elküldi egyszer használatos Diffie–Hellman paramétereit $(p, g, g^x \bmod p)$ a kliensnek. A kliens azonban a szerver egyszer használatos nyilvános RSA kulcsára vár, ezért a kapott üzenetet úgy értelmezheti, hogy p az RSA modulus, g pedig az exponent. A Handshake harmadik fázisában a kliens generál egy 48 bájtos véletlen blokkot, amit K'_1 -gyel jelölünk, és elküldi $(K'_1)^g \bmod p$ -t a szervernek. A támadó azonban elfogja ezt az üzenetet, és helyette a szerver által várt $g^y \bmod p$ -t küldi el a szervernek, ahol y -t a támadó választja.

Mivel p prím, ezért a támadó az elfogott $(K'_1)^g \bmod p$ -ből hatékonyan ki tudja számolni K'_1 -et. Továbbá, mivel a támadó generálta y -t, ezért könnyen ki tudja számolni $K'_2 = g^{xy} \bmod p$ -t is. K'_1 -ről a kliens azt hiszi, hogy az egy, a szerverrel megosztott közös titok, K'_2 pedig a szerver által kiszámolt érték, melyről a szerver azt hiszi, hogy a klienssel osztja meg. Valójában azonban mindenketten a támadóval osztják meg a titkukat. A támadó minden olyan információnak birtokában van, ami lehetővé teszi számára a K_1 és a K_2 mestertitkok kiszámítását K'_1 -ből, illetve K'_2 -ből. Ezek ismeretében azonban minden kulcsot ki tud számítani, és az ezen kulcsokkal védett Record üzeneteket fel tudja törni, illetve meg tudja változtatni. Mivel a finished üzenetek elküldésére csak a negyedik fázisban kerül sor, ezért azokat is tetszőlegesen manipulálni tudja a támadó, és ezzel képes eltüntetni az aktív támadás nyomait is. Más szavakkal, a finished üzenetek ellenére a kliens és a szerver nem szerez tudomást arról, hogy becsapták őket.

A key exchange algorithm rollback támadást az teszi lehetővé, hogy a server-key-exchange üzenetet a kliens nem hitelesített információkra támaszkodva dolgozza fel. Egészen pontosan, a kliens a szerver finished üzenetének sikeres ellenőrzéséig nem lehet biztos abban, hogy a szerver valóban RSA alapú kulcscserét választott. Ennek ellenére, a kliens az RSA alapú kulccscsere módszer szerint dolgozza fel a szerver üzenetét. Figyeljük meg, hogy a szerver aláírása a server-key-exchange üzenetben nem oldja meg a problémát, mert az csak az üzenet tartalmát hitelesíti. Itt azonban a feldolgozás kontextusa az, amit hitelesíteni kellene az üzenet értelmezése előtt. Sajnos a mestertitok kompromittálódása után a finished üzenetek már nem nyújtanak védelmet, hiszen korábban is kihangsúlyoztuk, hogy a finished üzenetek sikeres ellenőrzése csak azon feltétel mellett biztosítja a teljes Handshake hitelességét és sértelességét, hogy a mestertitok valóban titkos, sérteletlen és hiteles.

A problémát úgy lehetne megoldani, hogy a szerver nemcsak a kulcscsere paramétereit írná alá a server-key-exchange üzenetben, hanem például az összes addigi vett és küldött Handshake üzenet hash értékét is. Ez a megoldás összhangban volna az SSL filozófiájával abban az értelemben, hogy hasonló hash értéket ír alá a kliens a certificate-verify üzenetben, és a finished üzenetek számítása is az összes addigi Handshake üzenet hash értékéből történik.

10.1.5. A TLS protokoll

A TLS protokollt gyakorlatilag az SSL protokoll 3.1 verziójának is tekintjük, melyben a fenti elemzésben azonosított hiányosságokat kijavították. A TLS és az SSL közötti főbb különbségek a következők:

- a TLS Record protokollja véletlen hosszúságú kitöltést alkalmaz a forgalomanalízis megnehezítésére;
- a TLS Record protokollja a legújabb HMAC verziót használja és a MAC számítás magában foglalja a Record üzenet verziómezőjét is;
- a TLS Handshake protokollja a finished üzenetet addig nem dolgozza fel, amíg nem érkezett change-cipher-spec üzenet;
- az SSL Handshake során használt különböző MAC jellegű függvényeket (certificate-verify és finished üzenetek számítása, valamint a mestertitok és a kapcsolatkulcsok generálása) egységesítették a TLS-ben.

10.2. IPsec

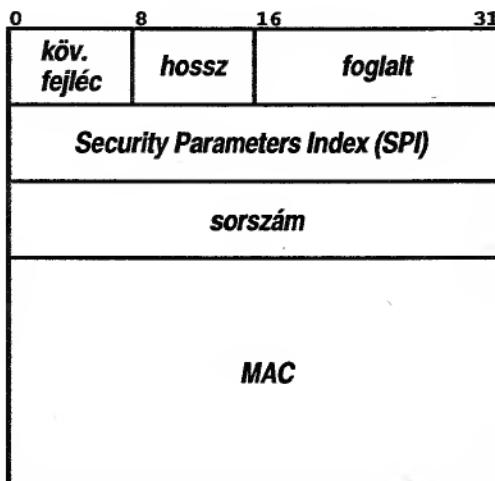
Az IPsec protokoll a TCP/IP architektúra hálózati rétegének szabványosított (RFC 2401, 2402, 2406, 2408, 2409) biztonsági protokollja. Ez azt jelenti, hogy az IP és minden fölötté található protokoll (TCP, UDP, ICMP, stb.) számára védelmet biztosít. Két alprotokollja van, az AH (Authentication Header) és az ESP (Encapsulated Security Payload). Az AH protokoll funkciói közé tartozik az IP csomagok integritásának védelme, eredetének hitelesítése és visszajátszásuk detektálása. Az ESP protokoll fő feladata az IP csomagok tartalmának rejtése, opcionálisan azonban az ESP is nyújthat integritásvédelem szolgáltatást. Természetesen az AH és az ESP protokollok kombinálhatók az IP csomagok teljeskörű védelme érdekében.

Az IPsec protokollhoz tartoznak még az ISAKMP (Internet Security Association and Key Management Protocol) és az IKE (Internet Key Exchange) protokollok. Mindketten kulcscserével kapcsolatos feladatakat látnak el. Az

ISAKMP egy általános célú keretprotokoll, mely bármely konkrét kulcscsere protokoll üzeneteit képes szállítani. Egy ilyen konkrét kulcscsere protokoll az IKE, mely jelenleg az IPSec hivatalos kulcscsere protokolljának szerepét tölti be. Az ISAKMP és az IKE protokollokat a teljesség kedvéért említettük meg, a továbbiakban azonban kizárolag az AH és az ESP protokollok ismertetésével foglalkozunk.

10.2.1. Az AH protokoll

Az AH protokoll tehát integritásvédelmet, eredethitelesítést és visszajátszás elleni védelmet biztosít az IP csomagok számára. Az integritásvédelmet és az eredethitelesítést úgy éri el, hogy az IP fejléc és az azt követő felsőbb szintű protokoll fejléce közé beszűr egy AH fejlécet, mely egy, a teljes IP csomagra számolt üzenethitelesítő kódot (MAC) tartalmaz. A visszajátszások detektálásának érdekében, az IP csomagokat sorszámozza. Az AH fejlécben található MAC érték a sorszámot is védi.

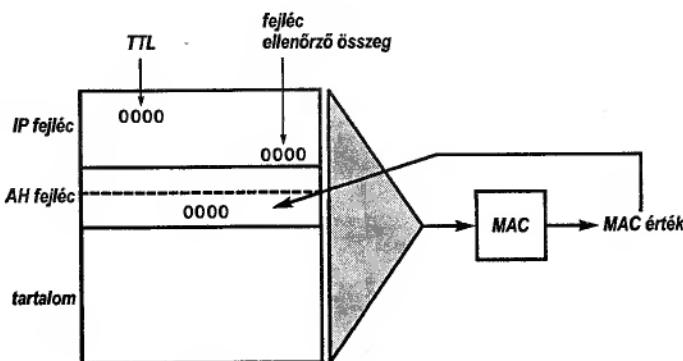


10.4. ábra. Az AH fejléc felépítése

Az AH fejléc felépítését a 10.4. ábra szemlélteti. A fejlécben található mezők és azok jelentése a következő:

- **következő fejléc (next header):** Ez a mező az AH fejlécet követő fejléc típusát adja meg, azaz az IP csomag tartalmára utal.

- *hossz (length)*: Ez a mező az AH fejléc 32 bites szavakban mért hosszára utal.
- *Security Parameter Index (SPI)*: Ez egy azonosító, mellyel a küldő azt jelzi a vevő számára, hogy milyen módon és mely kulcsokat használva kell az AH fejlécet feldolgozni. Az AH protokoll feltételezi, hogy a küldő és a vevő korábban már megegyezett az alkalmazható algoritmusokban és kulcsokban, tipikusan az ISAKMP/IKE protokollokat használva.
- *sorszám (sequence number)*: Ez a mező az aktuális IP csomag sorszámát tartalmazza.
- *üzenethitelesítő kód (MAC)*: Ez a teljes IP csomagra számolt üzenethitelesítő kód. A MAC számításának módját a 10.5. ábra szemlélteti. A számításhoz az IP fejléc változó mezőit (például TTL) és az AH fejléc MAC mezőjét nullával töltjük ki, majd az így kapott teljes csomagra ki-számoljuk a MAC értéket, és az eredményt visszairjuk az AH fejléc MAC mezőjébe.



10.5. ábra. Az AH fejléc MAC mezőjének számítása.

A régi IP csomagok visszajátszásának detektálását az AH protokoll az AH mezőben található sorszám alapján végzi. A vevőnek van egy konstans W méretű vételi ablaka. A vételi ablak e_r jobb széle megegyezik az eddig legnagyobb sorszámú sikeresen vett csomag sorszámaival. Az ablak bal széle pedig mindenkor $e_\ell = e_r - W + 1$. A vevő számon tartja, hogy az $[e_\ell, e_r]$ intervallumban mely csomagokat vette már egyszer. Tegyük fel, hogy egy új csomag érkezik, melynek sorszámát jelöljük s -sel. Ekkor a vevő a következőt teszi:

- Ha $s < e_\ell$, akkor a vevő eldobja a csomagot.

- Ha $e_\ell \leq s \leq e_r$ és s sorszámmal a vevő már vett egy csomagot, akkor eldobja a csomagot.
- Ha $e_\ell \leq s \leq e_r$ és s sorszámmal a vevő még nem vett csomagot, akkor a vevő ellenőrzi a csomag AH fejlécében található MAC értéket. Ha az ellenőrzés sikeres, akkor a vevő megjegyzi az s sorszámot, és elfogadja a csomagot.
- Ha $s > e_r$, akkor a vevő ellenőrzi a csomag AH fejlécében található MAC értéket. Ha az ellenőrzés sikeres, akkor a vevő megjegyzi az s sorszámot, a vételi ablak jobb szélét s -re állítja, és elfogadja a csomagot.

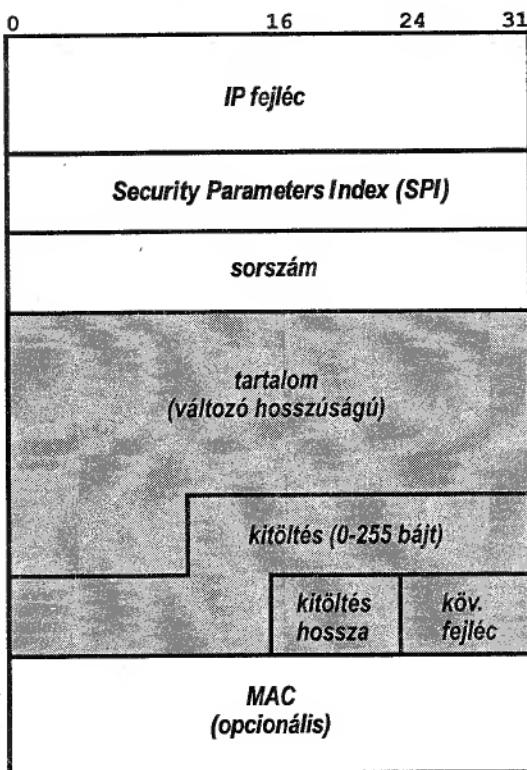
A vételi ablakra azért van szükség, mert az IP protokoll nem garantálja a csomagok sorrendhelyes kézbesítését. Ez azt jelenti, hogy normális körülmények között is érkezhet olyan csomag, melynek sorszáma kisebb, mint az eddigi legnagyobb vett sorszám. Ha ez a csomag még benne van az ablakban, akkor a vevő elfogadja. Az ablak véges mérete biztosítja, hogy „túl régi” csomagokat nem fogad el a vevő. A véges méret miatt a vevőnek nem kell az összes eddig vett sorszámot megjegyeznie, elegendő csak azokat számon tartani, melyek beleesnek az aktuális ablakba.

10.2.2. Az ESP protokoll

Az ESP protokoll feladata az IP csomag tartalmának rejtése, és opcionálisan a tartalom integritásának védelme. Az előbbi az IP csomag tartalmának rejtjelezésétől oldja meg a protokoll, az utóbbit pedig úgy, hogy az ESP fejlécre és a csomag tartalmára számít MAC kódot és azt a csomaghoz csatolja. Az AH-val ellentétben tehát az ESP MAC nem védi az IP fejléc mezőit.

A 10.6. ábra az ESP-vel védett IP csomag felépítését szemlélteti. Az ESP fejléc tartalmaz egy azonosítót (SPI), mely az AH-hoz hasonlóan itt is azt jelzi, hogy a vevőnek milyen módon és mely kulcsokat használva kell a csomagot feldolgozni. Az ESP protokoll is feltételezi, hogy a küldő és a vevő korábban már megegyezett az alkalmazható algoritmusokban és kulcsokban, tipikusan az ISAKMP/IKE protokollokat használva. Az ESP fejléchez tartozik még a csomag sorszáma is.

A 10.6. ábrán szürkével jelöltük a csomag rejtjelezett részét. Blokk-kódoló használata esetén a csomagot természetesen ki kell tölteni. A kitöltés hossza és az ESP fejlécet követő fejléc típusa (azaz a felsőbb protokoll fajtája) a kitöltés után kerül, és szintén rejtjelezve van. Végül az ESP MAC zárja le a csomagot, mely opcionális, és nincs rejtjelezve. A blokkrejtjelez-



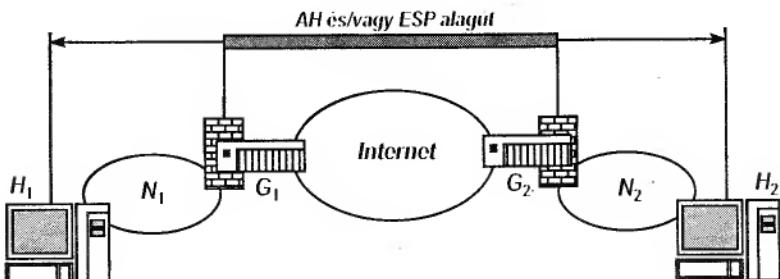
10.6. ábra. Az ESP-vel védett IP csomag felépítése

zőt CBC módban használja a protokoll. Az ehhez szükséges IV nyíltan kerül átvitelre a csomag tartalomrészének elején (az ESP sorszám után).

10.2.3. IPSec a gyakorlatban

Mind az AH, mind az ESP protokollt két üzemmódban lehet használni. Ezeket szállítási (transport) és alagút (tunnel) módoknak nevezzük. Szállítási módban az AH vagy az ESP fejléc a csomag eredeti IP fejléce és a felsőbb szintű protokoll (például TCP, UDP) fejléce közé kerül. Alagút módban azonban az eredeti IP csomagot teljes egészében beagyazzuk egy másik IP csomagba (IP tunneling), és az AH vagy az ESP fejléc az új, és az eredeti IP fejléc közé kerül. Ekkor tehát az AH fejléc vagy az ESP trailer következő fejléc mezője IP-re utal.

Az alagút módot tipikusan biztonsági átjárók (security gateway), például tűzfalak között használjuk. Segítségével könnyen létrehozhatunk virtuális magánhálózatokat (Virtual Private Network –VPN), ahol két tűzfallal védeott belső hálózatot az interneten keresztül, IPSec-et használva biztonságosan összekötünk. Ezt szemlélteti a 10.7. ábra. Tegyük fel például, hogy az N_1 belső hálózaton található H_1 gép egy IP csomagot küld az N_2 belső hálózaton található H_2 gépnek. Mikor az IP csomag eléri a G_1 tűzfalat, a tűzfal az egészet beágyazza egy G_2 -nek szóló IP csomagba, és azt IPSec védelemmel ellátva küldi tovább. Így a csomag biztonságban jut át az interneten a G_2 tűzfalhoz. G_2 elvégzi az IPSec feldolgozást (dekódolja a csomagot, ellenőrzi a MAC kódot stb.), majd a csomagban található eredeti IP csomagot (most már nyíltan) továbbküldi az eredeti címzettnek.



10.7. ábra. VPN létrehozása IPSec segítségével

A fenti példában a G_1 és G_2 tűzfalak alagút módban használják az AH és ESP protokollokat. Ekkor az IP csomagok a $H_1 - G_1$ és a $G_2 - H_2$ szakaszon védelem nélkül kerülnek továbbításra. Egy adott alkalmazásban ez esetleg nem jelent problémát, hiszen a belső hálózatok megbízhatónak tekinthetők. Ha ez nem így van, akkor biztonságos vég-vég kommunikációra van szükség H_1 és H_2 között. Erre a célra az AH és az ESP szállítási módja alkalmazható.

Az AH és ESP protokollok szállítási módú álapkombinációja (transport adjacency) az, amikor az IP csomagon először az ESP feldolgozást végezzük el hitelesítés szolgáltatás (azaz MAC) nélkül, majd az így kapott csomagra elvégezzük az AH feldolgozást is. Ekkor tehát a fejlécek sorrendje: IP, AH, ESP, TCP/UDP, ...

Alagút módban többféle kombinációt hozhatunk létre. Elékpzelhető például, hogy egy AH és/vagy ESP szállítási móddal védeott IP csomagot még egy AH és/vagy ESP alagút móddal védeott IP csomagba ágyazunk be. Sőt,

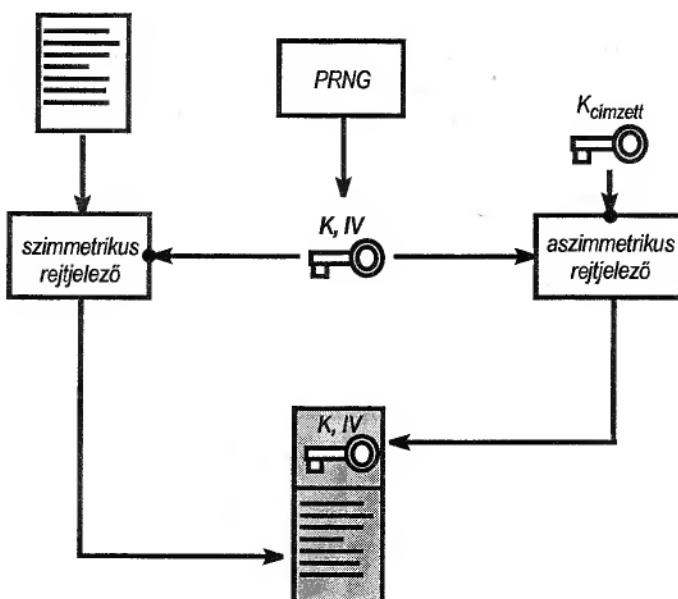
alagutak egymásba ágyazása is lehetséges, ennek alapkombinációi a következők: az alagutak két végpontja azonos (tipikusan hosztok között), az alagutak egyik végpontja azonos (tipikusan hoszt és tűzfal között) és az alagutak egyik végpontja sem közös (tipikusan tűzfalak között).

10.3. PGP (Pretty Good Privacy)

A PGP egy általános célú szoftver, melyet fájlok digitális aláírására és rejtjelezésére lehet használni. Azért tárgyaljuk mégis az internet biztonságáról szóló részben, mert a PGP megalkotásának fő motivációja az elektronikus levelezés védelme volt (alapjában véve az e-mail nem más, mint egy fájl). Ebben a vonatkozásában a PGP azért is érdekes, mert az SSL-től és az IPSec-től eltérően, egy aszinkron jellegű kommunikáció védelmét oldja meg. Az aszinkron jelzőt itt abban az értelemben használjuk, hogy a kommunikáló felek nincsenek egy időben jelen a kommunikáció során. Ez a tulajdonság alapvetően meghatározza a protokollt, hiszen az interakció lehetősége ki van zárva, és olyan üzenetekre van szükség, melyek önmagukban hordoznak minden információt, ami a feldolgozásukhoz szükséges.

A PGP alapvetően nyilvános kulcsú kriptográfiára épül. Fő szolgáltatásai közé tartozik az üzenetek digitális aláírása, az aláírt üzenetek tömörítése, és a tömörített üzenetek rejtjelezése. Ezen szolgáltatások közül a rejtjelezést tárgyaljuk részletesebben. A digitális aláírás a hash-and-sign módszerrel történik, amiről már korábban esett szó (lásd 7.2. fejezet).

Gondoljuk meg, hogy az aszinkron kommunikáció miatt a PGP-nek (és más hasonló rendszereknek) nincs lehetősége arra, hogy egy hagyományos kulccscsereprotokollt használva először létrehozzon egy kapcsolatkulcsot, majd azzal rejtjelezze az üzenetet (elektronikus levelet). Ezért kézenfekvőnek látszik a nyilvános kulcsú rejtjelezés használata, hiszen a címzett nyilvános kulcsa rendelkezésre állhat, vagy letölthető valamilyen kulcstárból. Tudjuk azonban, hogy hosszú üzenetek nyilvános kulcsú algoritmussal történő rejtjelezése nem praktikus. Ezért a PGP az ún. digitális boríték (digital envelop) technikát használja. Ennek az a lényege, hogy az üzenetet először egy frissen generált szimmetrikus kulccsal rejtjelezzük (például AES-sel), majd a szimmetrikus kulcsot kódoljuk a címzett nyilvános kulcsával, és a rejtjeles üzenetet a rejtjelezett szimmetrikus kulccsal együtt elküldjük a címzettnek. A címzett először a szimmetrikus kulcsot állítja vissza saját privát kulcsa segítségével, majd a visszaállított szimmetrikus kulccsal dekódolja az üzenetet. A digitális boríték technikát a 10.8. ábra szemlélteti.



10.8. ábra. A PGP által is használt digitális boríték technika szemléltetése

A PGP rendszer egy másik érdekessége a kulcsgondozás, illetve azon belül is a nyilvános kulcsok hitelesítésének módja. A PGP rendszerben ugyanis nem hitelesítés szolgáltatók (CA) bocsátják ki a felhasználók nyilvános kulcs tanúsítványait, hanem maguk a felhasználók. Ebben az értelemben tehát a PGP egy önszerveződő rendszer, nincs szüksége kiépített PKI-re.

A PGP rendszerben minden felhasználó kiadhat egy tanúsítványt más felhasználók nyilvános kulcsára. A tanúsítvány formája hasonlít a hagyományos PKI-ben használt tanúsítványhoz, azaz minimálisan tartalmazza a felhasználó nevét, nyilvános kulcsát és a tanúsítványt kibocsátó felhasználó aláírását. Ezeket a tanúsítványokat a felhasználók egymás között cserélgethetik, terjeszthetik, vagy valamelyen nyilvános adatbázisban elhelyezhetik.

Minden U felhasználónak van egy lokális publikus kulcsadatbázisa is. Ezen adatbázis minden bejegyzése tartalmaz egy V felhasználó-azonosítót, V vélt K_V nyilvános kulcsát, és más felhasználók által V számára kiadott tanúsítványokat. Ezek után U PGP szoftvere kiszámol egy hitelességmértéket, ami azt fejezi ki, hogy a rendelkezésre álló információk alapján U mennyire bízhat meg abban, hogy K_V valóban V kulcsa.

A hitelesség mértékének kiszámítása a következőképpen történik. U egy szubjektív bizalom értéket rendel minden tanúsítány kibocsátóhoz. A szubjektív bizalom értékek a következők lehetnek:

- ismeretlen felhasználó,
- általában nem megbízható felhasználó,
- általában megbízható felhasználó és
- mindig megbízható felhasználó,

ahol a megbízhatóság azt jelenti, hogy U mennyire bízik meg az adott felhasználó által kibocsátott tanúsítványokban. Ezután U szoftvere kiválasztja azokat a tanúsítókat, amelyek kulcsának hitelességéről már korábban megyőződött (az U által aláírt kulcsokat hitelesnek fogadja, minden további ellenőrzés nélkül), majd kiszámítja az ezen tanúsítókhöz rendelt bizalom értékek súlyozott összegét, ahol a súlyok U által konfigurálható paraméterek. Az eredmény adja a $V - K_V$ összerendelés hitelességének mértékét.

10.3. Példa. Tegyük fel, hogy a $V - K_V$ összerendelést A , B és C tanúsítja, ahol U a következő szubjektív bizalom értékeitet rendelte a tanúsítókhoz: A – általában megbízható, B – ismeretlen, C – általában megbízható. Tegyük fel továbbá, hogy A , B és C kulcsának hitelességéről már megyőződött U szoftvere. Legyenek a súlyok olyanok, hogy a kulcs hitelességének megállapításához legalább két, általában megbízható felhasználó, vagy legalább egy, minden megbízható felhasználó aláírása szükséges. Ekkor U szoftvere hitelesnek minősíti a $V - K_V$ összerendelést, mert két, általában megbízható felhasználó, A és C aláírta, és minden kettőjük kulcsát hitelesnek tartja U rendszere. Ellenben, ha például C kulcsa nem lenne hiteles, vagy C -t általában nem megbízható felhasználónak tartaná U , akkor K_V -t nem fogadná el hitelesnek.



11.

Mobil hálózatok biztonsága

Ebben a fejezetben két digitális mobiltelefon rendszer, a második generációs GSM és a harmadik generációs UMTS példáján keresztül mutatjuk be a mobil hálózatok biztonságával kapcsolatos problémákat és azok lehetséges megoldásait. A mobil kommunikációs rendszerek adatbiztonsági tervezés szempontjából speciálisak a következő okok miatt:

- fizikailag hozzáférhető rádiós csatornán is folyik kommunikáció,
- a mobilkészüléknek általában kicsi a számítási és a tárolási kapacitása,
- a mobilkészülék időben változtatja helyét.

Ennek megfelelően mind a GSM, mind az UMTS rendszerben lehetőség van a rádiós csatorna rejtelezésére, valamint minden rendszerben különös figyelmet fordítottak a hálózati funkciókhoz történő hozzáférés megfelelő védelmére. Figyelembe vették a mobilkészülékek kis számítási kapacitását is, ezért minden rendszer védelme szimmetrikus kulcsú kriptográfiára épül, ami általában nagyságrendekkel kisebb számítási kapacitást igényel, mint a nyilvános kulcsú algoritmusok. A mobilitással kapcsolatban minden rendszerben meg kellett oldani, hogy a mobilkészülékek idegen hálózatban (roaming közben) is azonosítani tudják magukat, és hozzáférjenek a hálózati funkciókhoz.

11.1. GSM biztonság

A GSM rendszer biztonsági architektúrájában fontos szerepet játszik a mobil készülékekben található SIM kártya. A SIM kártya lényegében az előfizető képviseli a rendszerben. Ennek megfelelően a SIM kártya tárolja az U előfizető és a HN hazai hálózat közötti, hosszú élettartamú $K_{U,HN}$ szimmetrikus kulcsot. Ezt a kulcsot használja az előfizető saját maga hitelesítésére, mikor egy beszélgetést kezdeményez, és ezen kulcs felhasználásával készülnek a beszélgetések rejtelzésére használt kapcsolatkulcsok is. A SIM kártyában vannak implementálva a biztonsági algoritmusok is, ezért a $K_{U,HN}$ kulcsnak soha nem kell elhagynia a SIM kártyát. A GSM biztonsági algoritmusainak (A3, A5, és A8) részletes leírása megtalálható a függelékben megadott hivatkozáson keresztül.

A GSM rendszer fő biztonsági funkciói az alábbiak:

- **hitelesítés:** A hálózat a következő kihívás-válasz alapú protokollt használva hitelesíti az előfizetőt:

GSM előfizető hitelesítés

- (1) $VN \rightarrow U$: Identity Request
 - (2) $U \rightarrow VN$: IMSI
 - (3) $VN \rightarrow HN$: IMSI
 - (4) $HN \rightarrow VN$: RAND | SRES | K_C
 - (5) $VN \rightarrow U$: RAND
 - (6) $U \rightarrow VN$: SRES'
 - (7) $VN \rightarrow U$: $\{TMSI\}_{K_C}$
-

ahol VN az idegen hálózat, melyben az U előfizető roaming közben szeretne hívást kezdeményezni, és HN az U hazai hálózata.

VN kérésére U elküldi nemzetközi $IMSI$ azonosítóját (International Mobile Subscriber Identity) VN -nek. VN nem tudja hitelesíteni U -t, mert nem ismeri a $K_{U,HN}$ kulcsot. Ezért továbbküldi az $IMSI$ azonosítót HN -nek, mellyel U hitelesítéséhez szükséges információkat kér HN -től. HN generál egy $RAND$ véletlenszámot, melyet kihívásnak szán U felé, kiszámítja a kihívásra adandó helyes $SRES = A3(K_{U,HN}, RAND)$ választ, és generál egy friss $K_C = A8(K_{U,HN}, RAND)$ kapcsolatkulcsot, ahol A3 és A8 két alkalmas függvény, melyet a szolgáltató akár maga választhat. Ezek után HN elküldi a $RAND | SRES | K_C$ hármat VN -nek.

VN elküldi a $RAND$ véletlenszámot U -nak, aki $K_{U,HN}$ ismeretében kiszámítja az $SRES' = A3(K_{U,HN}, RAND)$ választ és a K_C kapcsolatkulcsot. U

elküldi az *SRES'* választ *VN*-nek, aki összehasonlítja azt a *HN*-től kapott *SRES* értékkal. Egyezés esetén *U* hitelesítette magát. Végül *VN* generál egy ideiglenes *TMSI* azonosítót (Temporary Mobile Subscriber Identity) *U* számára, és azt *K_C*-vel rejtelezve elküldi *U*-nak. Mivel *U* is birtokában van *K_C*-nek, ezért dekódolja az üzenetet.

- *anonimitás*: A *VN* idegen hálózaton belül *U* a *TMSI* azonosítót használja az *IMSI* helyett, ezzel rejtvé valódi kilétét egy, a rádió interfészen hallgatózó támadó elől.
- *titkosítás*: *U* és *VN* a *K_C* kulcsot használják a mobilkészülék és a bázis állomás közötti forgalom rejtelezésére. A rejtelezéshez az *A5* kulcsfolyamatos rejtelezőt használják, mely a GSM szabványban van specifikálva.
- *hozzáférésvédelem*: A SIM kártya egy PIN kód segítségével hitelesíti az előfizetőt, és ezáltal biztosítja a rendszerhez történő illetéktelen hozzáférés megakadályozását.

A fenti négy biztonsági funkciót tekintve a következő észrevételeket tehetjük:

- A hozzáférésvédelmi megoldás a technológiailag biztonságos smart kártya alapú SIM felhasználásával jó megoldás, ami tovább él a harmadik generációs UMTS rendszerben is. Ezen a téren a biometrikus azonosítás technológiájában várható előretörés hozhat új megoldásokat a közeljövőben (újjlenyomatolvasó vagy hangazonosító a mobiltelefonban).
- A partner-hitelesítés csak egyirányú. Ugyanakkor az előfizetőnek is meg kellene tudnia győződni arról, hogy nem egy hamis bázisállomással áll-e szemben. Az UMTS ebben a vonatkozásban a kölcsönösséget vezeti be, s nemcsak az előfizető, de a hálózati infrastruktúra elemei is hitelesítik magukat.
- A forgalom rejtelezéses védelme csak a rádiós szakaszra vonatkozik, azaz a bázisállomásokat összekötő gerinchálózatra nem. Következetképpen némi megtévesztő a GSM szolgáltatással kapcsolatosan kiemelt rejtelezés funkció, mivel összességében nem biztonságosabb, mint a vezetékes telefonszolgáltatás. A rejtelezés a GSM-ben alapvetően nem egy szolgáltatás-minőség-emelő funkció, hanem inkább kényszerű intézkedés a rádiós szakaszon történő továbbítás miatt.
- Az előfizető a kapcsolat elején nyíltan küldi el az *IMSI* azonosítót, s így az anonimitást veszélyeztet. Ez a megoldás elkerülhetetlen, mivel a kapcsolat legelején nem lehet elkerülni bizonyos cím-információk nyílt továbbítását.

bítását. Meg kell továbbá jegyezni, hogy az anonimitást csak a rádiós szakasz lehallgató támadóval szemben biztosítja a *TMSI* használata, a rendszer többi résztvevőjével (pl. az idegen hálózattal) szemben nem. Az anonimitás-megoldások széleskörű kibővítése az egyik fontos új eleme a harmadik generációs rendszereknek.

11.1.1. Támadások a GSM ellen

Ezen a ponton célszerű röviden áttekinteni a fenti szolgáltatások hiányosságait kiaknázó ismert támadások elvét. Ezeket igyekeztek figyelembe venni az új, harmadik generációs rendszer tervezése során.

Kapcsolatfelvételi protokoll támadása (IMSI catcher). Az „IMSI catcher” támadás a MIM (Man-In-the-Middle) támadások egy példája, amelynek során a támadó megtudja a felhasználó valódi *IMSI* azonosítóját. A támadó, akit *X*-sel jelölünk, rendelkezik egy olyan eszközzel, mely a mobilkészülék felé bázisállomásként, a legális bázisállomás felé pedig mobilként lép fel. A támadás egy fizikai feltétele, hogy *X* a hiteles bázisállomásnál nagyobb teljesítménnyel sugározzon a mobilkészülék felé a kapcsolatfelvételi protokoll legelején, amikor az Identity Request parancsot kiadja. Ennek következtében a mobilkészülék nem a hiteles bázisállomásnak, hanem *X*-nek küldi el az *IMSI* azonosítót. *X* ezek után minden forgalmat reléz a mobilkészülék és a hiteles bázisállomás között. *X* a protokoll további lépései során arra is képes lehet, hogy a bázisállomástól érkező rejtjelezést bekapsoló parancsot (Cipher Mode Command (start cipher)) elnyomva, rejtjelezés nélküli állapotot állítson elő, azaz kikapsolja a rejtjelezés funkciót. Mindezen támadási lehetőségek egyértelműen protokollhibák eredményei.

SIM klónozás. A támadás célja más számlájára történő telefonhasználat. A támadás közvetlen célja a SIM kártyán tárolt titkos szimmetrikus kulcs megállapítása. A támadás viszonylag erős előfeltételei a következők:

1. a támadónak a támadás idejére fizikailag birtokolnia kell a támadott SIM kártyát és ismernie kell a kártyához tartozó PIN kódot,
2. a SIM kártyán az A3 és A8 algoritmusok egy COMP128 néven ismert implementációjának kell lennie.

A COMP128 algoritmus gyengeségei következtében a titkos kulcs nagyságrendileg 150 000 SIM kártyához intézett megfelelő kérés alapján megállapítható. Mivel ez az egyetlen titok a SIM-ben, ezzel már másolat is előáll-

lítható. A támadás megelőzhető lett volna, ha bevezetés előtt nyilvánosságra került volna az algoritmusok leírása, és kiderült volna azok gyengesége¹.

Hitelesítési adatok lehallgatása. Amikor az idegen hálózat lekéri a hitelesítéshez szükséges paramétereket ($RAND$, $SRES$, K_C) a hazai hálózat operátorától, akkor a paraméterek továbbítása tipikusan nem rejtjelezett csatornán történik a két hálózat között, s így azok lehallgathatók. A támadó az $SRES$ -t, mint a $RAND$ kihívásra adandó helyes választ megismerve hitelesen be tud jelentkezni a távoli bázisállomáshoz. Ezen kívül a K_C kulcs ismeretében a támadó le tudja hallgatni a bázisállomás és a mobilkészülék közötti rejtjelezett forgalmat.

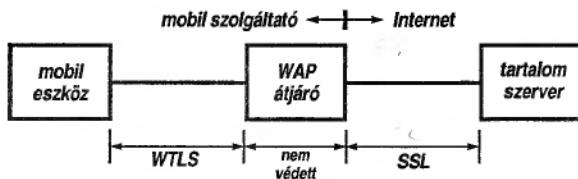
Számlázási adatok és adatok privát kezelése. Egy számlázási rekord (billing record) több olyan adatot tartalmaz, ami kényes, privát információnak tekinthető (pl. ki kivel, honnan, mikor, mennyi ideig beszélgetett). Továbbá minden rekordok a számlázóközpont adatbázisában az ügyfélhez csatolódva tárolódnak. Technikailag a szolgáltató ugyan indokolhatja ezek szükségességét a számlázáshoz, ugyanakkor ezen információk lehetővé teszik az ügyfelek folyamatos monitorozását, követését, bárhol is járnak a világban. Ennek folyamatosságát erősíti a periodikus helymeghatározás frissítés (location update). A számlázás kapcsán történő adatfelhalmozás elkerülhető a szolgáltatás előre fizetésével (pre-paid card).

11.1.2. A WAP szolgáltatás

A WAP (Wireless Application Protocol) alapvető célja az, hogy lehetővé tegye az internet elérést a mobiltelefonról. A WAP protokoll architektúra biztonságért felelős rétege a WTLS (Wireless Transport Layer Security) protokoll, mely lényegében az SSL vezetéknélküli környezetre módosított változata.

A 11.1. ábrán láthatjuk az adatbiztonsági modellt a WAP környezetben. A WAP átjáró (WAP gateway) alapvető feladata a HTML és a WML (Wireless Markup Language) közötti konverzió a szállított tartalom illetően. Ezen konverzióra azért van szükség, mert a mobil eszköz oldalán szűkösek az erőforrások (vezetéknélküli csatorna sávszélesége, illetve a mobil eszköz processzálási teljesítménye és kijelzőjének megjelenítési képessége). A WAP

¹ A részletek leírása megtalálható a <http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html> oldalon.



11.1. ábra. A WAP biztonsági modell

eszköz mikroböngészője a WML nyelven megjelenő tartalmakat képes kezelní (olvasni és interpretálni). Ebben a megközlítésben a WAP átjáró és a mobil eszköz együttese az internetes PC alapú böngészés egy vezetéknélküli megfelelője, az internet egyfajta vezetéknélküli meghosszabbítása, figyelembe véve az említett erőforrásszűkösségi szempontokat.

A mobil eszköz és a WAP átjáró között a WTLS, a WAP átjáró és a tartalomszerver között az SSL protokoll védi a kommunikációt. A WAP átjáró dekódol, majd újrakódol az SSL és a WTLS között. A WAP átjárón belül azonban nyíltan találhatók az adatok. Ez komoly adatbiztonsági kockázatot jelenthet. Tegyük fel például, hogy egy *X* ország bankja WAP-on kereszttüli szolgáltatást nyújt *B*-nek. *B* egy olyan *Y* országba utazik, ahol kevésbé kritikus kérdésként kezelik a biztonságot, s az *Y* ország mobil szolgáltatója által kezelt WAP átjáró kedvező támadási pontot nyújt.

Egy megoldás erre a problémára az, mikor a tartalomszerver közvetlenül WAP tartalmat (WML+WTLS) szolgáltat. Ezt egyszerűen úgy teheti meg, hogy a WAP átjáró képességet a saját web szerverére telepíti, s így ezen szerver és a mobil között egy vég-vég biztonságos kapcsolat épül fel. Ekkor a mobilszolgáltató által üzemeltetett WAP átjáró nem vesz részt a kapcsolatban. Ehhez azonban a mobil tulajdonosának a mobil konfigurációját úgy kell beállítania, hogy WAP átjáróként az adott tartalomszerver legyen megadva. A konfiguráció állítgatása azonban kényelmetlenséget okoz, főleg ha több különböző kedvenc szervert szeretnénk egymás után elérni. Amennyiben a WAP átjáró automatikusan felismerné, hogy már eleve WTLS protokollal véddett tartalom érkezik a tartalomszerver felől, akkor azt egyszerűen relézhetné a mobil felé, míg az SSL-el érkező HTML tartalmat szokásosan konvertálná WML+WTLS tartalomra.

11.2. UMTS biztonság

A harmadik generációs UMTS rendszer biztonsági architektúrája a GSM rendszerből nőtt ki, ezért alapjaiban sok hasonlóságot mutat a két rendszer. A GSM-hez hasonlóan, az UMTS rendszerben is, minden U felhasználónak van egy $K_{U,HN}$ titkos szimmetrikus kulcsa, amit a HN hazai hálózattal oszt meg. Ez a kulcs, és az azt használó f_1, f_2, f_3, f_4 , és f_5 algoritmusok a felhasználó mobilkészülékében a SIM kártyán vannak tárolva. A kulcson és az algoritmusokon kívül, U és HN nyilvántart egy számlálót, melynek aktuális értékét U -nál SQN_U -val, HN -nél pedig SQN_{HN} -nel jelöljük. A számlálót a felek a partner-hitelesítés és kulccscsere protokoll minden futása során növelik.

Protokoll. Az UMTS partner-hitelesítés és kulccscsere protokoll (a továbbiakban röviden UMTS protokoll) célja a felhasználó és a hálózat kölcsönös hitelesítése, kapcsolatkulcsok létrehozása rejtjelezésre és integritásvédelemre, és a kapcsolatkulcsok frissességének biztosítása a felek számára. Az ezen célokat megvalósító UMTS protokoll a következő üzeneteket használja:

UMTS partner-hitelesítés és kulesere protokoll

- (1) $VN \rightarrow U$: Identity Request
 - (2) $U \rightarrow VN$: *IMSI*
 - (3) $VN \rightarrow HN$: *IMSI*
 - (4) $HN \rightarrow VN$: $RAND|XRES|CK|IK|AUTH$
 - (5) $VN \rightarrow U$: $RAND|AUTH$
 - (6) $U \rightarrow VN$: *SRES*
-

A GSM protokollohoz hasonlóan, a felhasználó elküldi *IMSI* azonosítóját az őt kiszolgáló VN idegen hálóztnak. VN továbbküldi az *IMSI* azonosítót a felhasználó HN hazai hálózatának. HN válaszként egy

$RAND|XRES|CK|IK|AUTH$

öt elemből álló ún. *hitelesítő vektort* küld VN -nek, ahol

- $RAND$ egy frissen generált véletlenszám, melyet HN kihívásnak szán U számára,
- $XRES = f_2(K_{U,HN}, RAND)$ a kihívásra adandó helyes válasz,
- $CK = f_3(K_{U,HN}, RAND)$ a rejtjelezésre használandó kapcsolatkulcs,
- $IK = f_4(K_{U,HN}, RAND)$ integritásvédelemre használandó kapcsolatkulcs,
- $AUTH$ pedig egy hazai hálózatot U felé hitelesítő blokk.

AUTH számításához *HN* először előállít egy *SQN* sorozatszámot SQN_{HN} -ből, majd generál egy $AK = f_5(K_{U,HN}, RAND)$ anonimitás kulcsot és egy $MAC = f_1(K_{U,HN} | SQN | RAND | AMF)$ MAC értéket, ahol *AMF* egy operátor-specifikus kulcsmenedzsment paramétereket tartalmazó mező. Ezután *HN* eggyel megnöveli SQN_{HN} értékét, és a következő módon állítja össze a hitelesítő blokkot:

$$AUTH = (SQN \oplus AK) | AMF | MAC.$$

Miután a hitelesítő vektort megkapta, *VN* elküldi a *RAND* kihívást és az *AUTH* hitelesítő blokkot *U*-nak. Ezek alapján *U* kiszámolja az $AK = f_5(K_{U,HN}, RAND)$ kulcsot, majd ennek segítségével dekódolja az *AUTH* blokk első mezőjét, és visszaállítja a *HN* által használt *SQN* sorozatszámot. Ezután *U* ellenőrzi az *AUTH* blokk *MAC* mezőjét $f_1(K_{U,HN}, SQN | RAND | AMF)$ kiszámításával és a kapott értékhez történő hasonlítással. Ha a *MAC* helyes, akkor *U* ellenőrzi, hogy az *SQN* sorozatszám nagyobb-e, mint saját SQN_U értéke. Amennyiben nem, úgy *U* az üzenetet visszajátszásnak tekinti. Ha minden ellenőrzés sikeres, akkor *U* hitelesítette a hálózatot. Ezután *U* saját sorozatszámának értékét *SQN*-re állítja, majd kiszámolja az $SRES = f_2(K_{U,HN}, RAND)$ választ, és a $CK = f_3(K_{U,HN}, RAND)$ és $IK = f_4(K_{U,HN}, RAND)$ kulcsokat. Végül visszaküldi *SRES*-t *VN*-nek.

VN a kapott választ összehasonlítja az *XRES* értékkal. Egyezés esetén *U* hitelesítette magát. A továbbiakban *U* és *VN* a *CK* és *IK* kulcsokat használva védi a kettőjük közötti forgalmat. Külön protokoll gondoskodik a sorozatszámok beállításáról, ha *U* és *VN* kiesnek a szinkronból.

Analízis. A felhasználó hitelesítés a GSM-hez hasonló módon, egy kihívás-válasz protokollra épülve valósul meg. Az UMTS újdonsága a GSM-hez képest a hálózat hitelesítése a felhasználó felé. Ez az *AUTH* blokk *MAC* mezőjének segítségével történik, melyet csak a $K_{U,HN}$ ismeretében lehet előállítani. A *RAND* érték (és azon keresztül a *CK* és *IK* kulcsok) frissességét az *SQN* sorozatszám ellenőrzésével biztosítja a protokoll. A felhasználó anonimitását, a GSM-hez hasonlóan, egy ideiglenes azonosító használata biztosítja. Ezen kívül az *SQN* érték nem nyíltan kerül átvitelre, így a sorozatszámok megfigyelése és esetleges korrelációk keresése sem fedi fel a felhasználó identitását.

Az UMTS protokoll hibája, hogy az idegen hálózat nem hitelesíti magát *U* felé. Azaz *U* nem tudja biztosan, hogy melyik hálózatot használja a roaming alatt. Ezt egy támadó úgy használhatja ki, hogy *U* forgalmát titkon átírá-

nyítja egy másik, nem az U által választott hálózatba. Elképzelhető például, hogy különböző hálózatok különböző biztonsági követelményeket írnak elő, így a támadó megpróbálhatja a forgalmat egy olyan hálózatba terelni, ahol nincs előírva a rejtjelezés és rádiós szakasz sem. Ha az átirányítás sikeres, a támadó szabadon lehallgathatja a felhasználó forgalmát.

Az átirányításos támadás kivitelezéséhez a támadó egy „IMSI catcher” jellegű eszközt használ, mely a felhasználó felé bázisállomásként, a bázisállomások felé pedig mobil eszközöként viselkedik. A támadás menete a következő:

$$\begin{aligned} U \rightarrow X_{VN} : & \text{ IMSI} \\ X_U \rightarrow VN' : & \text{ IMSI} \\ VN' \rightarrow HN : & \text{ IMSI} \\ HN \rightarrow VN' : & \text{ RAND|XRES|CK|IK|AUTH} \\ VN' \rightarrow X_U : & \text{ RAND|AUTH} \\ X_{VN} \rightarrow U : & \text{ RAND|AUTH} \\ U \rightarrow X_{VN} : & \text{ SRES} \\ X_U \rightarrow VN' : & \text{ SRES.} \end{aligned}$$

Figyeljük meg, hogy a támadó csak annyit tesz, hogy U forgalmát átirányítja VN' -be, míg U azt hiszi, hogy VN -t használja. VN' sem sejt semmit, a fenti UMTS protokollnak megfelelő lépéseket hajta végre HN -nel. Még az is elképzelhető, hogy a gerinchálózaton valamilyen más protokollt használva hitelesíti magát HN -nek. Ez azonban nem segít U -nak észrevenni, hogy nem VN -t használja, mert a HN által generált $AUTH$ blokk semmilyen információt nem tartalmaz az idegen hálózat identitására vonatkozóan. Így végül U csak akkor értesül a csalásról, mikor hónap végén megkapja a számláját, és azon VN' használata lesz feltüntetve VN helyett.

Ezt a támadást el lehetne kerülni, ha az $AUTH$ blokk explicit utalást tartalmazna VN identitására vonatkozóan, azaz ha az UMTS protokoll betartaná a 8.6. fejezetben ismertetett 3. protokoll tervezési szabályt.

12.

Elektronikus fizetési protokollok

Az elektronikus – főképp az internetes – fizetési rendszer alapvető kérdése a biztonság. A biztonság lényegét tekintve azt jelenti, hogy a rendszer egyik résztvevőjének sem kell férnie attól, hogy pénzt fog veszíteni. A fizetési rendszer tervezésénél minden oldal (vásárlói, kereskedői és banki) biztonsági követelményeit figyelembe kell venni, s arra kell törekedni, hogy a felek biztonsága minél kisebb mértékben függön a többiek viselkedésétől. A fő biztonsági szempontok a következők:

- *autorizáció*: Az autorizáció követelménye azt jelenti, hogy egy banki ügyfél pénzét csak az ügyfél jóváhagyásával lehessen költeni. Ehhez szükséges az autorizáció (felhatalmazás, jóváhagyás), amely biztosítja, hogy a jóváhagyás tényleg a felhasználótól érkezett. Ezért megbízhatóan hitelesíteni kell az ügyfelet, aminek számos technikája ismert: személyes telefons jóváhagyás; jelszó vagy PIN kód alkalmazása; biometriai azonosítás; digitális aláírás.
- *adatok titkossága és hitelessége*: Biztosítani kell egy tranzakció adatainak titkosságát a tranzakcióban nem érintett felek számára. Erre a rejtjelezés technikáit használjuk. Hasonlóan, garantálni kell minden üzenet hitelességét, ami azt jelenti, hogy annak címzettje képes megállapítani, hogy az üzenet eredeti formában, az eredeti küldőtől érkezett-e meg. Ezen követelmények kielégítésére különböző kriptográfiai ellenőrzőösszeg-képző és digitális aláírás technikák állnak rendelkezésre.
- *elérhetőség és megbízhatóság*: Egy internetes fizetési rendszernek biztosítania kell a résztvevők számára a folyamatos elérhetőséget. Ez nem min-

dig teljes mértékben lehetséges, mivel az internet-infrastruktúra, amelyre épülnek, több nyilvános hálózat összekapcsolásával jött létre, emiatt viszonylag gyakori lehet egyes kommunikációs csatornák megszakadása, vagy telítettsége. Ez off-line rendszerek esetén kevesebb gondot okoz, pl. elég a deposit tranzakció elhalasztása. On-line rendszerek esetén a bank elérhetetlensége minden tranzakciót meghiúsít. Ezért akár tartalékkarendszerek kiépítése is szükséges, amely súlyos meghibásodás esetén működésbe lépne.

A tranzakcióknak atomiaknak kell lenniük, vagyis vagy a tranzakció minden műveletének kell végrehajtódnia, vagy egynek sem. A gyakorlatban azonban egy tranzakció megszakadhat végrehajtása közben egy külső ok miatt (pl. kommunikációs hálózat meghibásodása). Az így megszakadt tranzakciók nem szabad, hogy kárt okozzanak a résztvevőknek, valamint ebből nem profitálhat egy harmadik fél. A félbemaradt tranzakciók kezeléséhez ezért gondoskodni kell a megszakadás detektálásáról és jelzéséről, valamint a megszakadt tranzakciók folytatásának lehetőségről.

- ***személyes adatok védelme és anonimitás:*** Gondoskodni kell arról is, hogy egy tranzakció lebonyolítása során a rendszer többi résztvevőjéhez ne kerüljön a tranzakció szempontjából szükségtelen személyes adat. Egyes adatok lehetnek titkosak a tranzakcióban résztvevő egyik fél számára. Például az anonimitást biztosító rendszerekben a vásárló neve titkos marad a kereskedő előtt.

12.1. Elektronikus fizetési rendszerek (EPS) csoportosítása

Egy on-line tranzakciónak három szereplője van: a vásárló, a kereskedő és a Bank. (A Bank itt lehet valódi bank, hitelkártya társaság, illetve bármely olyan fél, amely hitelesíteni tudja a vásárlást fedezeti szempontból). A fizetési rendszerek osztályozása történhet az alábbi fő szempontok szerint:

- kredit vagy debit,
- on-line vagy off-line,
- azonosított vagy anonim.

A debit rendszer úgy működik, mint egy szokásos banki csekkezés. Először pénzt töltünk a számlára, majd abból fizetünk vásárlásokkor, mai szokások szerint, amikor például ATM-ről mágnescsíkos, PIN kódos bankkártyával pénzt veszünk le, vagy amikor a számlánkról havonta automatikusan átutal a bank közüzemi számláink kiegyenlítésére, a banki számlánkról

kerül leemelésre az összeg. Ha egy EPS a vásárlás során előbb ellenőrzi a bankszámlánkon levő összeget, s csak aztán jelez vissza a kereskedőnek, hogy lebonyolítható a fizetőképesség szempontjából a tranzakció, akkor debit rendszerről van szó. Azaz először befizetünk, aztán költünk. A kredit (hitel) rendszerű vásárlás során viszont először költünk – hitelünk mértékéig – és aztán fizetjük vissza a hitelben elköltött összeget.

Online vásárlás során a kereskedő kéri a harmadik fél (a Bank) jóváhagyását. A vásárló összeállít egy üzenetet, amely tartalmazza például a vásárolt termék/szolgáltatás megjelölését, a vásárló és kereskedő nevét, a pénzösszeget, a vásárló bankszámlaszámát, a vásárlás időpontját, és a vásárló által az üzenetre adott digitális aláírást. Ezt az üzenetet a vásárló elküldi a kereskedőnek, aki azonnal továbbítja azt a Bank felé jóváhagyás végett. A visszajelzés után a kereskedő elfogadja a vásárlás üzenetet, s rendbenhagyólag visszajelez a vásárlónak.

Offline vásárlás során a tranzakció a vásárló és a kereskedő között jön létre harmadik fél közbeiktatása nélkül, azaz a vásárló továbbítja a pénzt a kereskedőnek, aki maga meg tud győződni annak hitelességéről. A hétköznapi papírpénzes vásárlásaink ezen kategorizálásban off-line vásárlásnak felelnének meg.

Az azonosított fizetés során követhetők a vásárló korábbi tranzakciói. Például hitelkártyás vásárlás során egy papír dokumentum is készül a kereskedő boltjában, aki a „kártyalehúzós” termináljáról telefonon keresztül ellenőrzi a vásárló jelenlétében a hitelképességet, s egy papír dokumentumon megjelenik a vásárló azonosítója (név, kártyaazonosító), sőt a vásárló aláírását is adjja, hogy valóban megtörtént a vásárlás. Egy anonim fizetési rendszerben a vásárló adatai titokban maradnak, s nem követhetők a vásárlásai.

Az azonosított és anonim rendszereket tovább osztályozhatjuk a kommunikáció on-line illetve off-line jellege szerint:

- Azonosított on-line EPS esetén mind a kereskedő, mind a Bank azonosítja a vásárlót, s a kereskedő a Banktól visszaigazolást vár. Ezen rendszernek a napi megfelelője a kredit vagy a debit kártyás vásárlás egy szokásos áruházban.
- Anonim on-line EPS esetén a vásárló anonim marad a kereskedő számára. A kereskedő csak a vásárló gépének IP címét ismeri (Pontosabban azt az IP címet, amit vásárláskor használt). A kereskedő a Banktól visszaigazolást kap, ahol a Bank számára azonosított a vásárló, azaz a Bank követni tudja a vásárlásokat.

- Azonosított off-line EPS esetén mind a kereskedő, mind a Bank ismeri a vásárlót, de a banki számítógép on-line elérése valamelyen szempontból (pl. nincs meg a megfelelő infrastruktúra) nem lehetséges vagy kényelmetlen. Intelligens chipkártyák segítségével történő – vásárlás helyszíni – off-line fizetés a tipikus példa, ahol a chipkártya tárolja az elektronikus készpénzt, s a chipkártya garantálja annak fizikai és algoritmikus védelmét.
- Anonim off-line EPS esetén a vásárló anonim marad mind a kereskedő, mind a Bank számára. A kereskedő csak a vásárló gépének IP címét ismeri, azt az IP címet, amit vásárláskor használt.

12.2. Hitelkártyás fizetés az interneten: SET

A SET (Secure Electronic Transactions) a hagyományos bankkártyás fizetési modellből nőtt ki. Kifejlesztését a Visa és a MasterCard végezte a 90-es években. A cél egy olyan rendszer létrehozása volt, amely biztonságos kártyás fizetést tesz lehetővé az interneten. Ennek megfelelően a SET az alábbi biztonsági szolgáltatásokat nyújtja:

- a tranzakcióban részt vevő felek hitelesítése,
- a tranzakciós adatok titkosítása; bizonyos információk még a tranzakcióban résztvevő felek elől is rejtve vannak (pl. a kereskedő nem látja a vásárló hitelkártyaszámát), és
- a tranzakciós adatok integritásának védelme.

Építő elemek. A fenti követelemeyleket a SET nyilvános kulcsú kriptográfiai alapozva elégíti ki. A megoldásban központi szerepet kapnak a kulcsnálkülványok, a *digitális boríték* (*digital envelop*) technika és a *kettős aláírás* (*dual signature*) technika.

A digitális boríték technikáról a PGP tárgyalásánál, a 10.3. szakaszban már esett szó. Tegyük fel, hogy egy M üzenetet szeretnénk rejtjelezni B nyilvános kulcsával, K_B -vel. A hatékonyság növelése érdekében azonban nem tanácsos K_B -t közvetlenül M -en alkalmazni. Helyette M -et rejtjelezzük egy frissen generált k szimmetrikus kulccsal (és egy alkalmas szimmetrikus kulcsú rejtjelezővel), majd csak a szimmetrikus kulcsot rejtjelezzük K_B -vel. Az így nyert $\{k\}_{K_B} \mid \{M\}_k$ üzenetet nevezzük digitális borítéknak, és a továbbiakban $[M]_{K_B}$ -vel jelöljük.

A kettős aláírás lényege a következő. Legyen adva két dokumentum, M_1 és M_2 . Tegyük fel, hogy az A fél mindenkitől együttesen szeretné aláírni, de úgy, hogy két ellenőrző fél, B_1 és B_2 közül B_1 csak M_1 -et láthassa, B_2 pedig csak M_2 -t. Ha A egyszerűen képezné a $\text{sig}_A(M_1|M_2)$ aláírást, az nem lenne megfelelő, mert annak ellenőrzéséhez szükség van mind az M_1 , mind az M_2 dokumentumra. A helyzetben az sem segít, ha A az aláírást az összefűzött dokumentum $h(M_1|M_2)$ hash értékére képezi. A SET a problémát úgy oldja meg, hogy A a $h(h(M_1)|h(M_2))$ hash értéket írja alá, majd B_1 -nek elküldi az $M_1 | h(M_2) | \text{sig}_A(h(h(M_1)|h(M_2)))$ üzenetet, B_2 -nek pedig elküldi az $M_2 | h(M_1) | \text{sig}_A(h(h(M_1)|h(M_2)))$ üzenetet. Így B_1 csak M_1 -et látja, az aláírást mégis ellenőrizni tudja, mert megkapja az ehhez szükséges $h(M_2)$ hash értéket is, amiből viszont M_2 -t nem tudja kitalálni. Hasonlóképpen B_2 is ellenőrizni tudja az aláírást, de nem látja M_1 -et. Továbbá, az így konstruált aláírás összeláncolja az M_1 és M_2 dokumentumokat, mintha A azokat együtt írta volna alá. A további tárgyaláshoz a következő jelölést vezetjük be:

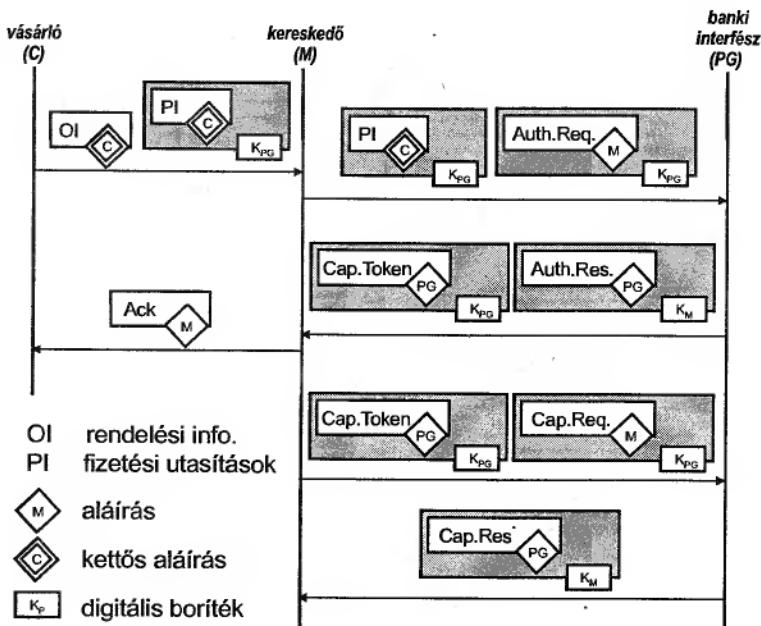
$$dsig_A(M_1, M_2) = M_1 | h(M_2) | \text{sig}_A(h(h(M_1)|h(M_2))). \quad (12.1)$$

Protokoll. A SET protokoll részvevői a következők:

- a C kártyatulajdonos vásárló (Cardholder),
- az M kereskedő (Merchant) és
- a PG banki interfész (Payment Gateway).

A protokoll minden résztvevője rendelkezik saját publikus-titkos kulcs-párral külön a rejtjelezés és külön a digitális aláírás céljára. Ez lehetővé teszi önálló szabályzás kialakítását a titkosítási illetve a hitelesítési (üzenetintegritás és forráshitelesítés) feladatokban. A SET saját, globális PKI-ra támaszkodik, mely hierarchikus felépítésű. A hierarchia csúcsán a gyökér hitelesítő szolgáltató (Root CA) található, majd alatta a bankkártya márkkák hitelesítő szervei állnak (Brand CA). Ezen utóbbiak hitelesítik a földrajzi hitelesítő szerveket (Geopolitical CA), majd azok külön az általuk lefedett kereskedőket, vásárlókat és pénzügyi intézményeket (bankokat).

A protokoll egyszerűsített sémája a 12.1. ábrán látható. A protokoll lépései a következők:



12.1. ábra. A SET protokoll fontosabb üzenetei

1. Böngészés

A tranzakció első lépése az, hogy a vásárló az interneten böngészve rátalál a kereskedő on-line áruházára, ott egy termékre, s jelzi vásárlási szándékát (pl. megnyomja a „Vegyél most!” gombot a termék mellett).

2. Tranzakció kezdés kérelem (Initiate Request)

A vásárló kitölți a rendelési űrlapot, megadja a használni kívánt bankártya műrkáját (pl. VISA), és elküldi ezen adatokat a kereskedőnek.

3. Tranzakció kezdés válasz (Initiate Response)

Miután a kereskedő megkapta a vásárló kezdési kérelmét, generál egy TID tranzakció-azonosítót és azt saját digitális aláírásával látja el. Ezután a következő választ küldi a vásárlónak:

$$\text{sig}_{K_M^{(\text{sig})}}(TID) \mid \text{Cert}(K_M^{(\text{sig})}) \mid \text{Cert}(K_M^{(\text{enc})}) \mid \text{Cert}(K_{PG}^{(\text{enc})}).$$

Az aláírt tranzakció-azonosítón kívül az üzenet tartalmaz még három tanúsítványt is: a kereskedő $K_M^{(\text{sig})}$ aláírás-ellenőrző kulcsának tanúsítvá-

nyát, a kereskedő $K_M^{(enc)}$ nyilvános rejtjelező (kulcscsere) kulcsának tanúsítványát és a banki interfész $K_{PG}^{(enc)}$ nyilvános rejtjelező (kulcscsere) kulcsának tanúsítványát¹. A vásárló szoftvere ellenőrzi a megkapott tanúsítványokat és a kereskedő aláírását a tranzakció azonosítón.

4. Vásárlási kérelem (Purchase Request)

A vásárló szoftvere előállítja a rendelési információkat (Order Information) tartalmazó OI dokumentumot és a fizetési utasításokat (Payment Instructions) tartalmazó PI dokumentumot. Mind OI , mind PI tartalmazza a TID tranzakció-azonosítót. A PI fizetési utasításokat a vásárló a saját bankjának szánja, ezért PI tartalmát el kívánja rejteni a kereskedő elől. Ugyanakkor megbízhatóan csatolni szeretné PI -t az OI rendelési információhoz, hogy később ellenőrizni lehessen, hogy az valóban az adott vásárláshoz tartozó fizetési utasításokat tartalmazta. A vásárló a kettős aláírás technikát használja, és a következő üzenetet küldi a kereskedőnek:

$$dsig_{K_C^{(sig)}}(OI, PI) \mid [dsig_{K_C^{(sig)}}(PI, OI)]_{K_{PG}^{(enc)}} \mid Cert(K_C^{(sig)}),$$

ahol $Cert(K_C^{(sig)})$ a vásárló aláírás-ellenőrző kulcsának tanúsítványa. Mint látható, a vásárló a digitális boríték technikát alkalmazza a (kettős aláírással ellátott) fizetési utasítások rejtésére. A banki interfész $K_{PG}^{(enc)}$ kulcsának hitelességét a 2. lépésben kapott $Cert(K_{PG}^{(enc)})$ tanúsítvány alapján tudja ellenőrizni a vásárló.

A kereskedő a megkapott tanúsítvány alapján ellenőrzi a vásárló $K_C^{(sig)}$ publikus kulcsát. Ennek a kulcsnak a segítségével le tudja ellenőrizni a $dsig_{K_C^{(sig)}}(OI, PI)$ üzenetrészben található aláírást. Mivel ez az üzenetrész nyíltan tartalmazza OI -t, ezért a kereskedő látja, hogy milyen rendelést kell teljesítenie. A PI fizetési utasítások azonban rejtve maradnak a kereskedő elől, hiszen a $dsig_{K_C^{(sig)}}(OI, PI)$ üzenetrész csak PI lenyomatát tartalmazza, a $dsig_{K_C^{(sig)}}(PI, OI)$ üzenetrész pedig rejtjelezve van.

5. Engedélyezés-kérés (Authorization Request)

Miután a kereskedő nem fér hozzá a vásárló által szolgáltatott fizetési utasításokhoz, ezért a bankot kell megkérnie, hogy ellenőrizze a vásárló fizetőképességét. Ennek megfelelően a kereskedő generál egy $AuthReq$

¹ Vegyük észre, hogy más kulcsok (és tanúsítványok) szolgálják az aláírás, és mások a rejtjelezés (kulcscsere) funkciókat.

engedélyezés-kérés üzenetet, amit digitálisan aláír és a banki interfész nyilvános kulcsával védett digitális borítékba helyez, majd a következő üzenetet küldi a banki interfésznek:

$$[dsig_{K_C^{(sig)}}(PI, OI)]_{K_{PG}^{(enc)}} \mid [sig_{K_M^{(sig)}}(AuthReq)]_{K_{PG}^{(enc)}} \mid \\ Cert(K_M^{(sig)}) \mid Cert(K_M^{(enc)}) \mid Cert(K_C^{(sig)}),$$

ahol tehát az üzenet tartalmazza a vásárlótól kapott rejtjelezett fizetési utasításokat és a kereskedő, illetve a vásárló tanúsítványait.

A banki interfész kibontja a digitális borítékokat és ellenőrzi a digitális aláírásokat. A banki interfész azt is ellenőrzi, hogy a kereskedő által az *AuthReq* üzenetben használt tranzakció-azonosító megegyezik-e a vásárló által a *PI* fizetési utasításokban használt azonosítóval. Ha az ellenőrzés sikeres, akkor a banki interfész elfogadja a kereskedő engedélyezés-kérését.

6. Engedélyezés

A banki interfész kapcsolatba lép a vásárló bankjával, és elküldi neki *PI*-t. A bank ellenőrzi, hogy a vásárló fizetési utasításai teljesíthetők-e. Amennyiben igen, a bank jelzi a banki interfésznek, hogy engedélyezi a tranzakciót. Ezek a lépések nem tartoznak szervesen a SET protokollba. A banki interfész és a bank nem az interneten keresztül kommunikál, hanem a pénzintézetek privát hálózatán.

7. Engedélyezés-kérésre adott válasz (*Authorization Response*)

Ha a bank kedvező választ ad, akkor a banki interfész generál egy *AuthRes* választ a kereskedő engedélyezés-kérésére, azt digitálisan aláírja, és a kereskedő nyilvános kulcsával védett digitális borítékba helyezi. A banki interfész a következő üzenetet küldi a kereskedőnek:

$$[sig_{K_{PG}^{(sig)}}(AuthRes)]_{K_M^{(enc)}} \mid [sig_{K_{PG}^{(sig)}}(CapToken)]_{K_{PG}^{(enc)}} \mid Cert(K_{PG}^{(sig)}).$$

Az üzenet opcionálisan tartalmazhat egy *CapToken* lehívási zsetont, amit a banki interfész generál, és amit a kereskedő a későbbiekben felhasználhat a fizetési tranzakció tényleges megindítására, azaz a vásárló pénzének lehívására és a kereskedő bankjába történő átutalására. A lehívási zsetont a banki interfész aláírja és saját nyilvános kulcsával védett digitális borítékba helyezi. Ez tehát azt jelenti, hogy a kereskedő nem férhet hozzá a nyílt lehívási zsetonhoz. A banki interfész lényegében magának küldi a lehívási zsetont egy, a kereskedő által megtestesített késleltő csatornán keresztül.

8. Vásárlási kérelemre adott válasz (*Purchase Response*)

Miután a kereskedő megkapta az engedélyezés-kérésre a kedvező választ, egy nyugtató üzenetet küld a vásárlónak, mely tartalmazza a kereskedő aláírását. A vásárló ellenőrzi a nyugta aláírását és eltárolja azt. Ezek után a kereskedő teljesíti a tranzakció során vállalt kötelezettségét, azaz elküldi a megrendelt árut, vagy biztosítja a megrendelt szolgáltatást a vásárlónak.

9. Lehívás kérelem (*Capture Request*)

A kereskedő később szeretné a tranzakciót befejezni, és az áruért vagy szolgáltatásért járó összeget a vásárló bankszámlájáról a saját bankszámlájára utaltatni. Ehhez generál egy *CapReq* lehívási kérelmet, ami tartalmazza a tranzakció adatait, többek között a tranzakció azonosítóját és a fizetendő összeget. A lehívási kérelmet a kereskedő aláírja, és a banki interfész nyilvános kulcsával védett digitális borítékba helyezi. Ezt aztán a korábban kapott lehívási zsetonnal együtt elküldi a banki interfésznek:

$$[\text{sig}_{K_M^{(\text{sig})}}(\text{CapReq})]_{K_{PG}^{(\text{enc})}} \mid [\text{sig}_{K_{PG}^{(\text{sig})}}(\text{CapToken})]_{K_{PG}^{(\text{enc})}}$$

A banki interfész kibontja a digitális borítékokat és ellenőrzi a digitális aláírásokat. Azt is ellenőrzi, hogy a lehívási kérelem és a lehívási zseton összetartoznak-e. Ha az ellenőrzések sikeresek, akkor a banki interfész kezdeményezi az összeg átutalását a vásárló bankjánál. Mindez azonban már a pénzintézetek privát hálózatán keresztül történik.

10. Lehívás kérelemre adott válasz (*Capture Response*)

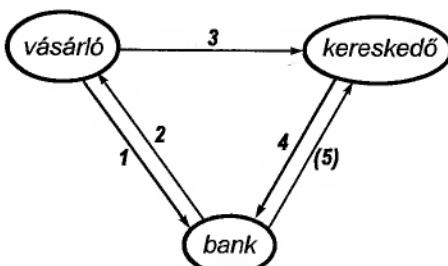
A banki interfész egy *CapRes* nyugtát generál a kereskedő lehívási kérelmére, amit aláír, a kereskedő nyilvános kulcsával védett digitális borítékba helyez, majd elküld a kereskedőnek:

$$[\text{sig}_{K_{PG}^{(\text{sig})}}(\text{CapRes})]_{K_M^{(\text{enc})}}.$$

A kereskedő kibontja a borítékot és ellenőrzi az aláírást. A kereskedő eltárolja a kapott *CapRes* választ az átutalás megtörténtének ellenőrzése céljából.

12.3. Digitális készpénz: DigiCash

A DigiCash egy on-line, anonim fizetési rendszer, mely a hagyományos (fizikai) készpénz modellje szerint működik. A digitális (elektronikus) készpénz (e-cash) egy bináris sorozat (egy szám), amely készpénznak felel meg, s ennek egyik számítógépről a másikra történő küldésével megtörténik a fizetés. Az elektronikus készpénz is anonim és újrafelhasználható, mint a szokásos bankjegy vagy pénzérme. Ezen tulajdonságai megkülönböztetik a hitelkártyás tranzakciótól, ahol a tranzakcióban részt vevő vásárló (aláírásával) hitelesíti magát.



12.2. ábra. A DigiCash rendszer résztvevői és a DigiCash tranzakció lépései

Protokoll. A DigiCash rendszer résztvevői és a DigiCash tranzakció lépései a 12.2. ábrán láthatók. A résztvevők a V vásárló, a K kereskedő, valamint a B on-line bank. Feltessük, hogy a vásárlónak számlája van a bankban, amelyen valamely összeg áll rendelkezésre. Ezen feltétel megvalósulása az alábbiakban részletezendő tranzakciós protokolltól független. A számlanyitáskor a vásárló választ egy (V_e, V_d) RSA kulcsparát, ahol V_e a nyilvános kulcs és V_d a titkos kulcs. A vásárló egy véletlen sorozatszámot is választ, amely véletlen érték az elektronikus készpénz (e-cash) sorozatszáma generátorának a véletlen magja lesz. Feltessük továbbá, hogy a bank rendelkezik $(B_e^{(i)}, B_d^{(i)})$ RSA kulcsparok egy sorozatával, ahol az egyes párok különböző pénzcímletekhez tartoznak (pl. $i \in \{1, 2, 5, 10, 20, 50, 100, \dots\}$).

A vásárló jelzi a bank felé pénzkivételi szándékát, amit aláírásával hitelesít. Az aláírást ellenőrzi a bank. Sikeres ellenőrzést követően a vásárló kezdeményezi megfelelő mennyiséggű elektronikus pénz „gyártását”. A pénz előállítása a fenti protokoll 1. és 2. lépése. A fenti protokoll 3–5. lépései a vásárlás során a kereskedő felé történő fizetéshez és a pénz ellenőrzéséhez tartoznak.

DigiCash protokoll

-
- (1) $V \rightarrow B: c, m = s \cdot r^e \bmod n$
 - (2) $B \rightarrow V: z = m^d \bmod n$
 - (3) $V \rightarrow K: P = (c, s, s^d \bmod n)$
 - (4) $K \rightarrow B: P = (c, s, s^d \bmod n)$
 - (5) $B \rightarrow K: P \text{ hamis} / P \text{ hiteles}$
-

1. A vásárló generálja a következő véletlen sorozatszámot (a véletlen mag felhasználásával), amely rendelkezik azon strukturáltsággal, hogy a baloldali fele egyértelmű determinisztikus kapcsolatban van a jobboldali feiével. Legyen ez a pillanatnyi sorozatszám s . Ha a vásárló ezen s sorozatszámmal szeretne generálni mondjuk 100 Ft címletű elektronikus pénzt, akkor kiválasztja az ezen címletnek megfelelő $e = B_e^{(c)}$ nyilvános kulcsot a bank kulcsai közül, és választ egy csak általa ismert, titkos r véletlen számot, majd elküldi az (1) üzenetet a banknak.
2. A bank az (1) üzenetben kapott m üzenetrészt aláírja az adott címletnek megfelelő d titkos kulcsával, s az eredményt a (2) üzenetben visszaküldi a vásárlónak. A vásárló a $z = (s \cdot r^e)^d = s^d \cdot r \bmod n$ és r mennyiségek ismeretében egy osztással előállítja a $w = z/r = s^d \bmod n$ mennyiséget². A $P = (c, s, s^d)$ hármas az elektronikus „bankjegy”. Ennek megfelelően leolvasható a címlete, az – egyedi – sorozatszáma, valamint hordozza a bank digitális aláírását.
3. A fizetés során a vásárló elküldi a megfelelő címletű elektronikus pénzek egy sorozatát a kereskedőnek³. Az egyszerűség kedvéért tegyük fel, hogy a vásárló a (3) üzenetet küldi, mely csak egyetlen bankjegyet tartalmaz.
4. A kereskedő a pénzt tárolhatja. Célszerűbb azonban, ha azt a (4) üzenetben elküldi a banknak, egyrészt ellenőrzésre, másrészt – hitelesség esetén – a bank a pénzt jóváírja a kereskedő számláján.

A bank két lépésben ellenőriz:

² Ezt az aláírás-technikát „vak” aláírásnak nevezik, ugyanis a vásárló úgy szerzi meg a bank aláírását az s sorozatszámon, hogy a bank nem látja s valódi értékét. Ezt az r vakító paraméter használata teszi lehetővé, melyet csak a vásárló tud eltávolítani a vakon aláírt sorozatszámról.

³ Természetesen ez a lépés tartalmazza előzetesen azt a részlépést is, hogy a vásárló ki-választ egy terméket (mondjuk egerével rákattintva egy „Buy one now!” feliratú gombra a kereskedő valamely weblapján), s kitölt egy űrlapot, amelyen megjelöldik a termék, annak mennyisége, ára, a vásárlás dátuma, majd V és K aláírásával hitelesíti ezen megrendelőt.

- (a) A bank ellenőrzi, hogy az s sorszám nem szerepelt-e már egy korábbi ellenőrzésnél, azaz nincs-e szó pénzduplikálásról⁴. Ha duplikálást észlel, azaz a pénzt egyszer már elköltötték, akkor azonnal a protokoll 5. lépéssére tér át, visszajelzve a kereskedőnek azt, hogy az általa ellenőrzésre bemutatott pénz másolat.
- (b) A bank ellenőrzi, hogy az s és s^d összetartozó pár-e, azaz az ō aláírását hordozza-e. Ezt a c címlethez tartozó nyilvános kulcsa felhasználásával végzi. Ha ez az ellenőrzés is rendben van, akkor a bank tárolja az s sorozatszámot, és azt ettől kezdve a forgalomból kivontnak tekinti.
5. A bank végül az (5) üzenetben tájékoztatja a kereskedőt, hogy az általa beváltani szándékozott pénz valódi volt-e vagy hamis (utóbbi alatt azt értjük, hogy a bank fenti ellenőrzései sikertelenek).

Analízis. Mint azt a protokoll lépéseinek magyarázatánál láttuk, egy másolt pénz (akár V , akár K készítené) a banki ellenőrzésen fennakad. Tekintsünk tehát ravaszabb hamisítás próbálkozást, azaz banki aláírás hamisítását. Ehhez a hamisítónak olyan u értéket kellene találnia, amelyet a címletnek megfelelő banki e nyilvános kitevőre emelve egy sorszámrakra jutna. Mivel azonban a sorszámoknak rögzített struktúrája van, megfelelő blokkméretek mellett gyakorlatilag kizárátható ezen támadás sikere.

Amikor a 2. lépésben B aláírását adja az m üzenetre, akkor az m felépítése miatt nem tudja megállapítani a V által éppen választott sorozatszámot, hiszen az ismeretlen r véletlennel való moduláris szorzás elfedi azt. Tehát úgy ad aláírást az m üzenet aláírásra megcélzott s részére, hogy nem láthatja annak valódi értékét. Az ilyen módon történő aláírást vak aláírásnak nevezük. Ezzel lesz a fizetőeszköz teljesen anonim, azaz a bank sem tudhatja, hogy egy később ellenőrzésre hozzákerülő elektronikus pénzt ki gyártotta le vele és mikor.

A protokoll lehetővé teszi, hogy a kereskedő ellenőrizze, hogy a 3. lépésben a vásárlótól kapott pénz adott címletű pénz-e, mivel ō is ismeri a bank megfelelő nyilvános kulcsát. Vegyük észre azonban, hogy ō (hacsak nem kizárálag nála vásárol az adott vásárló, és emelett a kereskedő is tárolja a korábbi sorozatszámokat) nem tudja eldönteni, hogy nem már felhasznált, másolt pénzt kapott-e. Pontosan ezért ellenőriztetи ezt a bankkal.

⁴ Ne felejtstük el, hogy most a pénz egy bitsorozat, amit másolni standard módon lehet, a klasszikus pénztől eltérően.

A fenti protokoll ebben a formájában még nem teszi lehetővé egy V és K közti letagadás-vita feloldását, azaz hogy a kereskedő ne hazudhassa azt, hogy a bank nem fogadta el a pénzt hitelesnek, s így a pénz nem került K bankszámlájára. Ezt olyan módosítással lehetne megoldani, hogy a V és K által is hitelesített (a fentiekben részletezett tartalmú) megrendelő is eljutna a bankhoz a pénzzel együtt, s ott tárolódna. Az anonimitás megőrzése érdekében az eredeti megrendelő helyett annak egy hash lenyomatát küldhetné el a vásárló, a bank azt tárolná, és csak vita esetén kellene az eredeti megrendelőt bemutatni.

Ha K jelezné V -nek, hogy a bankba történő beváltás előtt valahogy elvezett a 3. lépéshoz V -től megkapott pénz, akkor V -nek jelezni kellene tudnia a bank felé egy letiltást az adott pénzre vonatkozóan, például úgy, hogy megküldi a banknak az eltűnt pénz s sorozatszámát, amit a bank feljegyez.

Az s -re vonatkozó struktúramegkötést helyettesítheti egy strukturálthatlan véletlen sorszám nyilvános, egirányú függvényes leképezése is.

A fenti protokoll olyan változata is lehetséges, hogy a 3. lépés során csak a megrendelő kitöltése, s V és K általi kölcsönös hitelesítése történik meg, de V nem küldi át K -nak a pénzt ezen lépéshoz. Ehelyett a 4. lépéshoz V a B -nek elküld egy $\{K, H, P\}_k$ üzenetet, ahol k vásárló és a bank közötti titkos szimmetrikus kulcs, H a hitelesített rendelés lenyomata, és P a pénz. A bank ennek alapján, a P ellenőrzése után, a kereskedő számlájára ráteszi a P elektronikus pénz címletének megfelelő összeget, s értesíti erről a kereskedőt. Ezen protokoll-változatban az elektronikus pénz csak a vásárló és a bank között mozog.

A fentiekben leírt DigiCash protokoll a debit rendszerű, anonim és on-line fizetési rendszerek osztályába tartozik. Ugyanakkor V és B megállapodásától függően működhet kredit rendszerként is, azaz ha a bank engedi hitelben legyártatni nála az elektronikus pénzét. A részletezett protokoll szerinti rendszer on-line tulajdonsága azt jelenti, hogy amikor fizetésre került a sor, a kereskedő a 4. lépéshoz a bankot is bevonja a tranzakcióba azzal, hogy megnyugodhasson az elektronikus pénz valódi volta felől.

A DigiCash egyik hátránya, hogy az elköltött sorszámok tárolása (amelyre a duplaköltések elkerülése miatt van szükség) a rendszer hosszabb, folyamatos használata esetén kezelhetetlen méretűvé nőhet. E problémánál egyedüli megoldásként több „sorszámraktár” és több digitális pénzt kibocsátó bank bevonása képzelhető el. Egy másik hátránya, hogy a fizetések végösszege csak a címletértékek lineáris kombinációja lehet, nincs lehetőség visszafizetésre, ami miatt a felhasználóknak figyelniük kell a digitális pénztárcájukban ta-

látható érmék összetételere. Végül felhívjuk a figyelmet a teljes anonimitás kockázataira a pénzmosás kapcsán.

12.4. Mikrofizetési protokollok: PayWord

A SET és a DigiCash protokollok a valós, fizikai életben már létező fizetési módszereket próbálják meg adaptálni egy virtuális, elektronikus környezetre, melyet az új digitális hálózatok (elsősorban az internet) jelentenek. A SET a hitelkártyás fizetés, a DigiCash pedig a készpénzzel történő fizetés ekvivalens megvalósítása az elektronikus világban. Az új környezet azonban nemcsak a hagyományos módszerek újfajta megvalósítását, hanem merőben új fizetési módszerek kidolgozását is lehetővé teszi. Ebben a fejezetben egy ilyen új fizetési módszert tárgyalunk, amit mikrofizetésnek nevezünk.

A hálózaton keresztül nyújtott digitális szolgáltatások sokszor feldarabolhatóak. Gondoljunk például egy digitális folyóiratra, ahol egy szám minden cikke egy külön fájl; egy webes információs szolgáltatásra, mely HTML oldalakból épül fel; vagy akár egy zenei fájl letöltésére, ahol a teljes fájl bájtokból áll, stb. Miért kellene a vásárlónak a teljes szolgáltatást megvásárolnia, ha őt annak csak bizonyos részei érdeklők? Például, miért venné meg a vásárló a digitális folyóirat legfrissebb számát teljes egészében, ha abból csak egy cikket szeretne elolvasni? Ha a szolgáltatást fel lehet darabolni, akkor az egyes „szolgáltatás-darabkákat” lehet külön is árulni. A kérdés az, hogy hogyan fizessen a vásárló egy ilyen szolgáltatás-darabkáért. Használhatná a korábban tárgyalt fizetési módok valamelyikét, de azoknak nagy a tranzakciós költségük a megvásárolt szolgáltatás-darabka értékéhez viszonyítva. Egy hitelkártya tranzakció ára ma például kb. 100–300 Ft, melyet általában a kereskedő áll. Ekkor nyilván nem éri meg a tranzakciós költséggel összemérhető árú szolgáltatás-darabkákat külön árusítani, pedig elképzelhető olyan szolgáltatás, ahol amúgy lenne értelme ilyen méretű szolgáltatás-darabkákat árusítani (pl. egy telefonbeszélgetésnél van értelme a másodpercekért fizetni a percek helyett). A megoldást a mikrofizetési protokollok jelentik, melyek kifejezetten sok, kis értékű tranzakció lebonyolítására vannak optimalizálva.

Számos mikrofizetési protokollt javasoltak, néhányat közülük meg is valósítottak és alkalmaztak az interneten. Mi most a PayWord protokollt ismertetjük, mely egy egész mikrofizetési protokoll-család jellegzetes képviselője.

A PayWord protokollt Rivest és Shamir publikálta 1996-ban. A korábban bevezetett osztályozás szerint, a PayWord egy kredit alapú, off-line, azonosított rendszer. Nyilvános kulcsú kriptografiára épül, de azt igen hatékonyan

használja. Egészen pontosan a PayWord akkor hatékony, ha a vásárló egy kereskedőnek fizet egymás után sokszor, apró összegeket. Ekkor a tranzakció elején generált digitális aláírás feldolgozási költségét a protokoll „amortizálja”, azaz ezzel az egy digitális aláírással minden további mikrofizetési zsetont hitelesít.

A protokollnak három részvevője van: az U vásárló, az M kereskedő és a B bróker. A bróker a bankot képviseli a protokollban.

Regisztrációs fázis. Mielőtt U használná a rendszert, regisztrálnatnia kell magát egy brókerrel. Tegyük fel, hogy U a B brókert választotta. A regisztráció során B kibocsát egy $Cert_U$ tanúsítványt U számára, mely többek között a következő elemeket tartalmazza:

- B azonosítóját,
- U azonosítóját,
- U nyilvános aláírás ellenőrző kulcsát, K_U -t,
- a tanúsítvány érvényességi idejét és
- B aláírását az előző mezőkön.

A $Cert_U$ tanúsítvány lényegében B kijelentése, miszerint vállalja, hogy minden, a tanúsítvány érvényességi idején belül bemutatott, U által kibocsátott, azaz a K_U kulccsal hitelesíthető mikrofizetési zsetont a zseton megjelölt értékének megfelelően bevált.

Fizetési fázis. Tegyük fel, hogy U az M kereskedőnek szeretne fizetni apró összegeket. Ekkor U generál egy *Commit* üzenetet, mely többek között a következő mezőket tartalmazza:

- M azonosítóját,
- a $Cert_U$ tanúsítványt,
- a tranzakció során használt mikrofizetési zsetonok v értékét,
- a protokoll során átküldött mikrofizetési zsetonok beváltási határidejét,
- egy w_0 hash értéket, melynek szerepről alább szólunk és
- U aláírását az előző mezőkön.

A w_0 hash érték egy egyirányú hash lánc utolsó eleme. Ezt U a következőképpen állítja elő. Először generál egy w_n véletlenszámot, majd annak kiszámítja hash értékét, majd a kapott érték hash értékét, és így tovább. Formalisan $w_i = h(w_{i+1})$, $i = 0, 1, \dots, n-1$, ahol h egy egyirányú hash függvény.

n értékét U választja, mégpedig úgy, hogy megbecsüli, mennyi mikrofizetési zsetonra lesz szüksége a tranzakció során. U eltárolja az összes kiszámított w_i értéket, ezek alkotják majd a mikrofizetési zsetonokat. minden zseton v értékű.

U elküldi a *Commit* üzenetet M -nek. Ezzel lényegében engedélyezi, hogy a B bróker beváltsa a w_1, w_2, \dots, w_n zsetonokat M számára a megadott dátum előtt. Felhívjuk a figyelmet arra, hogy a *Commit* üzenet tartalmazza M azonosítóját. Ez azt jelenti, hogy a w_i zsetonok csak M számára értékesek, hiszen más úgysem válthatja be őket. Ezért a zsetonok átküldésekor nem kell különösebb gondot fordítani azok rejtésére.

M ellenőrzi a *Commit* üzenetet. Ez lényegében abból áll, hogy M ellenőrzi, hogy a zsetonok v értéke megfelel-e az M által nyújtott szolgáltatás-darabkák értékének, és hogy a beváltási határidő elegendően messze van-e. Továbbá, M ellenőrzi a $Cert_U$ tanúsítványt és az abban található K_U kulcs segítségével U aláírását a *Commit* üzeneten.

A protokoll során átküldött i -edik zseton nem más, mint az (i, w_i) pár. Mikor M megkapja w_i -t, akkor ellenőrzi, hogy annak hash értéke megegyezik-e az előzőleg kapott zseton értékével, ha $i > 1$, vagy a *Commit* üzenetben található w_0 értékkel, ha $i = 1$. Ha az ellenőrzés sikeres, akkor M elfogadja a zsetont. M minden csak az utoljára kapott zsetont tárolja el, ez elegendő lesz az összes többi beváltásához is.

A h hash függvény egyirányúsága miatt, w_i -ből nem lehet előállítani w_{i+1} -et, azaz bárki megfigyelheti az i -edik zsetont, az $(i+1)$ -edik zsetont nem tudja előállítani, csak U .

Beváltási fázis. A *Commit* üzenet beváltási határideje előtt M elküldi B -nek az utolsónak kapott (ℓ, w_ℓ) zsetont és a *Commit* üzenetet. B ellenőrzi a *Commit* üzenetet, majd ellenőrzi, hogy az abban található w_0 értékre fennáll-e a $w_0 = h^{(\ell)}(w_\ell)$ összefüggés. Ha az ellenőrzés sikeres, akkor U számláját megtereli $\ell \cdot v$ forinttal, M számláján pedig jóváír ugyanennyit.

Hatékonyság. A PayWord igen takarékosan bánik a digitális aláírással. U egyetlen aláírással az összes w_i zsetont hitelesíti. Az alkalmazott egyirányú hash lánctechnika hasonlít a partner-hitelesítő protokolloknál tárgyalt S/KEY egyszer használatos jelszó rendszerhez (9.1.3. szakasz).

U -nak a zsetonok előállításához n -szer meg kell hívnia a h hash függvényt. Ez n értékétől függően sok számítást vehet igénybe, ezt azonban U még az M -mel való kommunikáció megkezdése előtt, előre elvégezheti. U -

nak tárolnia kell továbbá az összes zsetont. Megjegyezzük, hogy különböző idő-memória kompromisszumokkal el lehet érni, hogy U -nak ne kelljen minden zsetont tárolni, ekkor azonban a hiányzó zsetonok előállítása időt vesz igénybe.

M -nek egyetlen digitális aláírást kell ellenőriznie tranzakciónként, és minden zseton ellenőrzésekor egyszer meg kell hívnia a h hash függvényt. Tárolnia csak a *Commit* üzenetet, valamint az utolsó zsetont és annak indexét kell.

B -nek egy digitális aláírást kell ellenőriznie, valamint ℓ -szer meg kell hívnia a h hash függvényt, ám ezeket off-line módon teheti meg.

IV.

Fejezetek a bizonyítható biztonság elméletéből

13.

Alapfogalmak

Shannon óta egy kriptográfiai algoritmusnak egy támadó korlátlan számítási erőforrásával szemben mutatott védeeltségét röviden információ-elméleti biztonságának, vagy feltétel nélküli biztonságának, míg a korlátos erőforrású esetet algoritmikus biztonságának, vagy feltételes biztonságának nevezzük. Rejtjelező algoritmusok közül egyetlen algoritmust ismerünk csak, amely feltétel nélkül biztonságos: a One-Time-Pad (OTP) rejtjelezőt. Tudjuk azonban, hogy az OTP is egy adott támadó modell mellett mutatja csak ezt a tulajdonsságot, nevezetesen amikor a támadó csak lehallgatást végezhet, s így csak rejtett szövegekhez juthat hozzá. A továbbiakban, bizonyítható biztonság kapcsán, feltételes biztonságú algoritmusokat keresünk.

Hogyan lehetünk biztosak abban, hogy kriptográfiai algoritmusunk biztonságos? A tradicionális (nem bizonyított biztonságú) megközelítésben úgy, hogy megkísérlünk egy támadást találni. Ha sikerrel járunk, akkor azt mondjuk, hogy nem biztonságos az algoritmusunk. Ha nem találunk támadást, akkor semmi megbízhatót nem tudunk mondani az algoritmus biztonságáról. Ha kiderül, hogy nem biztonságos az algoritmusunk, akkor a támadás erejétől, jellegétől függően vagy véglegesen elvetjük a konstrukciót, vagy megpróbáljuk megerősíteni a támadás jellegének megfelelően (azaz foluzzunk).

Ezzel szemben a bizonyított biztonság megközelítésben először definiálunk egy biztonságfogalmat, azaz megmondjuk, mit értünk pontosan a biztonságon (*definíció*). Ne feledjük, hogy alapvetően azért van szükség biztonsági algoritmusokra, mert támadható az a környezet, amelyben azok felhasználásra kerülnek. Más szóval, ha nem lenne feltételezett támadó, nem

lenne szükség kriptografiára sem. A biztonság fogalmai pontosan erre a körfülményre vonatkoznak, s a különböző fogalmak különböző támadási környezetek modellezését jelentik.

Kriptográfiai algoritmusunkat egy biztonságosnak feltételezett elemre, például egy nehéznek hitt feladatra építjük (*feltételezés*). Az algoritmusunk biztonságosságát ezek után indirekt módon bizonyítjuk: Feltesszük, hogy egy Z támadó sikerrel fel tudja törni (az alkalmazott biztonságfogalomnak megfelelő értelemben) a kriptográfiai algoritmusunkat. Ezután megmutatjuk, hogy Z támadási algoritmusát felhasználva egy Z' támadó sikерrel fel tudja törni a biztonságosnak hitt elemet (pl. a nehéznek hitt feladatra hatékony megoldó algoritmust mutat fel). Ezzel ellentmondásba kerülünk eredeti feltételezésünkkel. A szó igazi értelmében tehát nem bebizonyítjuk a biztonságot, hanem visszavezetést végezünk a feltételezésre (*redukció*). Azaz megmutatjuk, hogy ha egy Z támadó fel tudná törni a kriptográfiai algoritmusunkat, akkor létezne egy Z' támadó, amely fel tudná törni a feltételt.

A támadó erőforrását tipikusan t futási időkorláttal és orákulumokhoz küldhető legfeljebb q kéréssel modellezük. Például a választott nyílt szövegű támadás modellje a rejtelező orákulum, mely lehetővé teszi, hogy általunk választott nyílt szövegekhez azonnal megkapjuk a rejtejes szöveget. A támadás sikerét egy ϵ sikervalószínűsséggel jellemzzük. Azt mondjuk, hogy Z támadó (t, q, ϵ) -töri a kriptográfiai algoritmusunkat, ha sikervalószínűsége legalább ϵ . Redukciós bizonyításunk eredményeképp megmutatjuk, hogy Z' (t', q', ϵ') -töri a feltételt. Végső soron ezen vezetést – konkrétan a (t, q, ϵ) hármasból a (t', q', ϵ') hármasba való „átszámítást” – ellentétes irányban használhatjuk: Tegyük fel, hogy a feltételt tekintve megalapozott sejtésünk van (t', q', ϵ') -törésben a t' , a q' és az ϵ' értékére (nagyságrendjére), ugyanis a feltétel ismert, sokat vizsgált nehéz feladat. Így a (t', q', ϵ') hármasból kiindulva megállapítjuk a kriptográfiai algoritmusunk (t, q, ϵ) hármását, mint annak támadással szembeni minősítését.

A bizonyítható biztonság modern elméletének klasszikus korszaka az 1980-as évekre tehető. Ezen elmélet kiindulópontja az egyirányú függvény, s az annak létezésével kapcsolatos sejtés. Egyirányú függvényeken alapszik a bizonyíthatóan biztonságos álvéletlen-generátor. Ezen álvéletlen-generátor felhasználásával került definiálásra az álvéletlen függvény, majd arra építve az álvéletlen permutáció. Ugyanezen sorrendben épülnek egymásra a kapcsolódó konstrukciók is. Mindezen elemek a szimmetrikus kulcsú rejtelezés komplexitás-elméleti megalapozását adják. Az egyirányú függvények speciális osztálya a csapda egyirányú permutáció, illetve létezésének sejtése ala-

pozta meg az aszimmetrikus kulcsú biztonságos rejtjelezők első bizonyított biztonságú konstrukciót.

Az álvéletlen függvények fogalmát Goldreich, Goldwasser és Micali, az álvéletlen permutációk fogalmát Luby és Rackoff vezette be. Ezen elmélet komplexitáselméleti aszimptotikus megközelítést alkalmaz, amelynél az álvéletlen primitívek (álvéletlen-generátor, -függvény, -permutáció) végtelen sorozata kapcsán alapvetően azt vizsgáljuk, hogy egy polinom erőforrás-korlátú támadó képes-e megkülönböztetni azokat véletlen megfelelőiktől. A sorozatbeli index egy természetes módon adódó biztonsági paraméter értékének felel meg.

Ezen elmélet keretében született konstrukció – a már említett – bizonyított biztonságú szimmetrikus és aszimmetrikus kulcsú rejtjelezők mellett hash függvényekre, valamint egyes kriptográfiai protokollokra (pl. digitális aláírás) is. Ezek a konstrukciók azonban elsősorban elméleti jelentőségűek maradtak, ugyanis nagy számítási erőforrásigényük miatt kevésbé praktikusak. Ugyanakkor ezen klasszikus szakasz konstrukcióinak fontosságát elsősorban azok létezése adja, azaz a pozitív válasz arra a kérdésre, hogy lehet-e, s hogyan lehet bizonyítani algoritmikus biztonságot.

Az 1990-es évektől kezdődően egyre nőtt az igény arra, hogy gyakorlati alkalmazáshoz közelítő számításigényű, ugyanakkor bizonyított biztonságú kriptográfiai primitívek és protokollok szülessenek. A modern kriptográfia ezen új szakaszát Bellare és Rogaway korszaknyitó munkáiban „konkrét biztonság” (concrete security) megközelítésnek nevezte. A „konkrét” értelmét a nem aszimptotikus, hanem rögzített támadási erőforrás melletti vizsgálati módszer adja. Bár ez az irányzat is bonyolultság-elméleti technikákat alkalmaz, alapvető célja az, hogy minél élesebb redukcióval a gyakorlat számára hasznosabbá tegye a bizonyítható biztonság technikát.

13.1. Bonyolultságosztályok, orákulum, redukció

Az alábbiakban röviden összefoglaljuk az algoritmusok komplexitás-elméletének néhány fogalmát, amelyeket a továbbiakban használni fogunk. Megjegyezzük, hogy ezen áttekintés emlékeztető; az elméettel való ismerkedésre számos kiváló könyv áll rendelkezésre.

Egy algoritmus explicit lépésekre lebontott eljárás valamely számítás elvégzésére. Alapvetően az algoritmus egy bemeneti adatot (röviden bemenetet) dolgoz fel, s a számítások eredménye egy kimeneti adat (röviden kimenet). A bemenet mérete szokásosan a bemenet bináris alakjának hossza.

13.1. Definíció (polinom idejű algoritmus). *Egy algoritmust polinom idejű algoritmusnak hívunk, ha létezik egy c pozitív egész szám, hogy a legfeljebb n bites egészekkel végzett bitműveletek száma $O(n^c)$.*

(Emlékeztetünk a „nagy ordo” fogalmára: Legyen $f(n)$ és $g(n)$ két, a természetes számokon értelmezett, pozitív értékeket felvevő függvény. Ha létezik egy olyan c konstans, hogy minden elegendően nagy n értékre $f(n) \leq c \cdot g(n)$, azt mondjuk „ f nagy ordo g ”; jelölve: $f = O(g)$.)

13.2. Definíció (exponenciális idejű algoritmus). *Egy algoritmust exponenciális idejű algoritmusnak hívunk, ha létezik egy c konstans, hogy a legfeljebb n bites egészekkel végzett bitműveletek száma $O(e^{cn})$.*

Egy „probléma” egy feladat általános leírása, míg egy „eset” annak konkrét értékekre vonatkozó változata. Megkülönböztetjük a keresési problémát és a döntési problémát. A döntési problémák esetén a feladat egy eldöntendő kérdésként fogalmazódik meg, s a válasz kétértekű: „igen” vagy „nem”. A keresési problémák esetén a megoldást kettőnél nagyobb méretű halmazban keressük.

13.1. Példa. Egész szám faktorizációs keresési probléma: A feladat egy N egész szám egy M nemtriviális osztójának megkeresése, illetve annak megállapítása, hogy ilyen osztó nem létezik, azaz N prímszám. Ha egy konkrét N egész szám faktorizálása a feladat, akkor az egész szám faktorizációs probléma egy esetéről beszélünk. A problémát megoldó algoritmus bemenete és kimenete az alábbi:

Bemenet: N páratlan pozitív egész szám.

Kimenet: N -nek egy M nemtriviális osztója, vagy az „ N prímszám” üzenet.

Az egész szám faktorizációs döntési probléma egy esete kapcsán az eldönthető kérdés a következő: Van-e az N egész számnak egy M osztója a $[2, j]$ intervallumban, ahol N és j pozitív egész számok? A problémát eldöntő algoritmus bemenete és kimenete az alábbi:

Bemenet: N és j pozitív egész számok.

Kimenet: „igen” vagy „nem”.



Sok probléma keresési illetve döntési változata lényegében ekvivalens abban az értelemben, hogy ha ismerünk egy algoritmust az egyik változat megoldására, akkor abból kaphatunk algoritmust a másik változat megoldására is.

A számítástudomány három alapvető bonyolultságosztálya a P, az NP, illetve az NP-teljes döntési problémák osztálya.

13.3. Definíció (P bonyolultságosztály). *Egy döntési probléma a P bonyolultságosztályba esik, ha létezik egy algoritmus és egy $p(n)$ polinom, hogy a probléma tetszőleges esetét tekintve, ha annak bináris hossza (bemenet) nem nagyobb n -nél, akkor az algoritmus válaszideje nem nagyobb $p(n)$ -nél. Más megfogalmazásban: egy döntési probléma a P bonyolultságosztályba esik, ha létezik egy algoritmus és egy c konstans, hogy amennyiben a bemenet mérete nem nagyobb n -nél, akkor a válaszidő $O(n^c)$.*

Röviden úgy is fogalmazhatunk, hogy egy döntési probléma P -beli, ha van rá polinom idejű megoldó algoritmus.

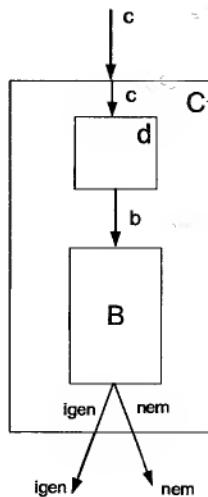
13.4. Definíció (NP bonyolultságosztály). *Egy döntési probléma az NP bonyolultságosztályba esik, ha korlátlan számítási kapacitás mellett nincs csak a válasz adható meg a probléma tetszőleges esetére, de „igen” válasz esetén még egy tanú (egy véges bináris sorozat) is felmutatható, amely tanú egy polinom idejű ellenőrző algoritmus bemeneteként igazolni tudja a válasz helyességét.*

A $P \subseteq NP$ tartalmazás nyilván fennáll: ugyanis egy P -beli döntési probléma eldöntő algoritmusát egyben polinom idejű ellenőrző algoritmusként is használhatjuk a tanút figyelmen kívül hagyva. Ha a 13.4. definícióbeli „igen”-t „nem”-re cseréljük, akkor a co-NP bonyolultságosztály definícióját kapjuk.

13.2. Példa. Tekintsük az egész szám faktorizációs problém a 13.1. példa-beli döntési probléma változatát. Ez valószínűleg nem P -beli, de biztosan NP -beli. Ugyanis egy M valódi osztót, ha létezik, minden meg tudunk találni, végigvizsgálva a $[2, N^{1/2}]$ intervallum egészeit. S ha létezik ilyen osztó, akkor az a tanú. ♣

A bizonyított biztonság bizonyítástechnikái közül a legfontosabb az algoritmikus visszavezetés (redukció) módszere. A módszer lényege a következő: Van egy algoritmusunk B probléma megoldására, s ezt az algoritmust szeretnénk felhasználni egy C probléma megoldására egy polinom idejű visszavezető algoritmus felhasználásával. Először tekintsük a döntési problémák visszavezetését (13.1. ábra).

13.5. Definíció (redukció). *Legyen B és C két döntési probléma. C problémát polinom időben redukáljuk B problémára, ha megadható egy d algoritmus, amely polinom idejű C bemenetében, s a C probléma egy c esetéhez előállítja a B probléma egy b esetét olyan módon, hogy a b kérdésre adott válasz megegyezik a c kérdésre adott válasszal.*



13.1. ábra. Döntési probléma visszavezetése

Így, ha B probléma eldöntésére találunk polinom idejű algoritmust, akkor az arra – polinom időben – visszavezetett C döntési problémára is van polinom idejű algoritmusunk. A visszavezetés egy másik, fordított felhasználási módja a következő: Ha tudjuk, hogy egy C döntési problémára nem létezik polinom idejű eldöntő algoritmus, akkor nem létezik a B döntési problémára sem, ugyanis ha létezne, akkor a polinom idejű visszavezetés miatt C problémára is léteznie kellene. A bizonyított biztonság levezetések alapvetően erre az indirekt következtetésre alapulnak.

Gyakran fogunk támaszkodni az orákulum fogalmára:

13.6. Definíció (orákulum). *Amikor azt mondjuk, hogy egy C probléma megoldó algoritmusa B orákulumot hívja, akkor ezen azt értjük, hogy van egy d algoritmusunk, amely előállítja B probléma egy esetét, amelyre mint bemenetre (orákulum-kérésre) a B orákulum azonnal előállítja a kimenetét (az orákulum-választ), amelyet d algoritmus továbbít C megoldó algoritmusának. Az „azonnal” azt jelenti, hogy az orákulum futási ideje nem számít bele C megoldó algoritmusának futási idejébe.*

A továbbiakban C^B jelölést használjuk akkor, ha egy C algoritmus B orákulumot használ (orákulumos algoritmus).

Az orákulum „természetes” módon fog előállni a bizonyításaink során. Így például a választott nyílt szöveg alapú támadást egy rejtjelező algoritmus ellen úgy modellezük, hogy megengedjük, hogy a támadó algoritmus hozzáérjen egy rejtjelező orákulumhoz. Hasonlóan a dekódoló orákulumhoz való hozzáérés a választott rejtett szövegű támadás modellje.

Orákulum modellben általánosítható a redukció definíciója általános (keresési vagy döntési) problémára, megengedve, hogy C probléma megoldó algoritmusa a probléma egy esetének megoldása során B probléma több esetének megoldásához hozzáérhessen:

13.7. Definíció (redukció). *Legyen B és C két probléma (döntési vagy keresési). C problémát polinom időben redukáljuk B problémára, ha létezik C problémára egy olyan C bemenetében polinom idejű megoldó algoritmus, amely C bemenetében legfeljebb polinom sokszor hívja B orákulumot.*

13.3. Példa. Az egész szám faktorizációs döntési problémát polinom időben redukálhatjuk az egész szám faktorizációs keresési problémára, ahol az utóbbit orákulumként használjuk. ♣

A bonyolultságosztályok között fontos szerepet játszik az NP-teljes problémák osztálya:

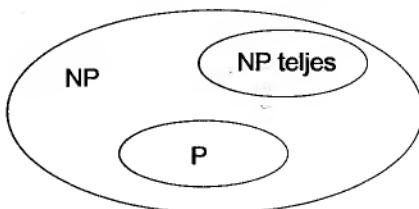
13.8. Definíció (NP-teljes). *Egy döntési probléma NP-teljes, ha minden NP-beli probléma visszavezethető rá polinom időben.*

Azaz, ha lenne egy polinom idejű megoldó algoritmusunk egy NP-teljes problémára, akkor lenne minden NP-beli problémára. Ebből következik az, hogy vagy minden NP-teljes probléma P-beli, vagy egyik sem az. A számítástudomány kapcsolatos alapvető sejtése, hogy P valódi részhalmaza NP-nek, amely szerint létezik olyan NP-beli probléma, amelynek nincsen polinom idejű megoldó algoritmusa. A 13.2. ábra szemlélteti a P, az NP és az NP-teljes bonyolultságosztályok sejtett viszonyát.

A $P \neq NP$ sejtés – mint azt látni fogjuk – alapvető a bizonyított biztonság elmélete szempontjából is:

13.9. Definíció ($P \neq NP$ sejtés). *A $P \neq NP$ sejtés szerint létezik olyan NP-beli probléma, amelyhez nem létezik P-beli megoldó algoritmus.*

A kriptografiában és speciálisan a bizonyított biztonság elméletében is igen fontosak a véletlen biteket is használó (továbbiakban röviden randomizált) algoritmusok.



13.2. ábra. A P, NP és NP teljes osztályok sejtett viszonya

13.10. Definíció (randomizált polinom idejű algoritmus).

Randomizált polinom idejű algoritmusnak hívunk egy véletlen biteket használó polinom idejű algoritmust.

Egy randomizált polinom idejű algoritmus nyilván legfeljebb polinomiálisan sok véletlen bitet használhat. A polinom idejű algoritmusokat (randomizált vagy determinisztikus) a továbbiakban, röviden, hatékony algoritmusnak hívjuk. Hasonlóan, nehéz problémán azt értjük, hogy megoldására nem ismert hatékony algoritmus.

13.11. Definíció (RP-beli (Random Polynomial-time) döntési probléma).

Egy döntési probléma RP-beli, ha randomizált polinom idejű algoritmussal eldönthető, s ahol az „igen” válasz esetében a válasz biztosan helyes, míg a „nem” válasz helyességének valószínűsége nagyobb, mint $\frac{1}{2}$.

Példaként gondoljunk egy randomizált prímtesztelelő algoritmusra, amelynek válasza biztos, ha azt mondja, hogy a tesztelt szám összetett, míg prímgyanú esetén annak valószínűsége, hogy a tesztelt szám mégis összetett, legfeljebb $\frac{1}{2}$.

Az RP osztálynál szélesebb az úgynevezett BPP (Bounded-Probability Polinomial-time) osztály, amelynél az „igen” válasz esetén is megengedett a tévedés:

13.12. Definíció (BPP-beli döntési probléma).

Egy döntési probléma BPP-beli, ha véletlen biteket használó polinom idejű algoritmussal eldönthető, és létezik $\delta > 0$ konstans, hogy az „igen” illetve „nem” válasz helyességének valószínűsége nagyobb, mint $\frac{1}{2} + \delta$.

Sajnos a továbbiak számára, a fentiekben röviden áttekintett osztályozás nem teljesen kielégítő. Fontos ugyanis észrevenni, hogy a fentebb említett bonyolultságosztályok a legrosszabb esetet tekintik, abban az értelemben,

hogy a megoldó algoritmusnak képesnek kell lennie a bonyolultságosztályba eső probléma legnehezebb esetének megoldására is. Így egy olyan kijelentés, hogy egy probléma megoldására nincs hatékony algoritmus, jelentheti „csak” azt, hogy például a legnehezebb esetekre nincs, ugyanakkor javarészt van. Márpedig kriptográfiai alkalmazásban egy ilyen „garancia” elfogadhatatlan. Másrészt alkalmazásainkban általában is megfelelhetne egy problémára olyan megoldó algoritmus is, amely a legtöbb esetre ad megoldást. Más szavakkal, ha egy esetet az esetek halmazából véletlenszerűen választunk, akkor megelégednénk, ha legfeljebb „elhanyagolhatóan kicsi” valószínűségű esemény kivételével adna megoldást. A két gondolatot összetéve megállapíthatjuk, hogy kriptográfiai szempontból nem elegendő azt tudnunk, hogy egy kriptográfiai törési probléma legnehezebb esetei tényleg nehezek; azt szeretnénk, hogy bármely véletlenszerűen választott eset majdnem minden nehéz legyen.

Ezen igényeknek megfelelően került Levin által bevezetésre az egyirányú függvény. Az egyirányú függvény fogalma, s annak sejtett létezése alapvető a kriptográfiai bizonyított biztonság elméletében és konstrukcióiban.

13.2. Egyirányú függvény (One Way Function – OWF)

A szokásos megfogalmazás szerint az „egyirányú függvényt könnyű kiértékelni az ōsképtér bármely elemére, ugyanakkor nehéz az inverzet kiszámítani a képtér elemeire”. Az erősen egyirányú függvény definíciója az alábbi:

13.13. Definíció (erősen egyirányú függvény).

Az $f : \{0,1\}^ \rightarrow \{0,1\}^*$ függvény (erősen) egyirányú, ha a következő két feltétel teljesül:*

1. *könnyű kiszámítani: létezik olyan hatékony T algoritmus, amely x bemenetre az $f(x)$ kimenetet adja,*
2. *nehéz invertálni: tetszőleges hatékony T' algoritmus, és tetszőleges $p(n)$ polinom, valamint elegendően nagy n esetén*

$$\Pr\{f(T'(f(U_n), 1^n)) = f(U_n)\} < \frac{1}{p(n)}. \quad (13.1)$$

A (13.1) formula szerint egyenletes eloszlással választunk ōsképet, majd annak képét adjuk bemenetként a T' invertáló algoritmusnak azzal, hogy adjon meg egy ōsképtérbeli elemet, amelynek azonos a képe. Erősen egyirányú függvény esetén ezt a feladatot csak elhanyagolhatóan kis valószínűséggel képes megoldani bármely, a feltételeknek megfelelő algoritmus.

Megjegyzés. A (13.1) kifejezésben a T' algoritmus bemenetében szereplő 1^n , azaz n darab 1 bitből álló sorozat értelmét az adja, hogy ezáltal biztosítható, hogy T' algoritmus a bemenete és kimenete együttes hosszában polinom időben futhasson. Ha ugyanis ezt elhagynánk, akkor például az $f(x) = |x|$ függvény, azaz amely bemenete hosszát adja meg bináris ábrázolásban (pl. $f(1001110) = 111$), annak okán lehetne egyirányú függvény, hogy a T' invertáló algoritmust futtató gép nem lenne képes még arra sem, hogy kiírja a kimenetre az inverzet, hiszen annak a kimeneti hossza a bemenet hosszának exponenciális függvénye.

13.14. Definíció (elhanyagolható függvény). *Egy $\mu : N \rightarrow N$ függvényt elhanyagolhatónak nevezünk, ha tetszőleges $p(n)$ polinom esetén létezik M pozitív egész szám, hogy tetszőleges $n > M$ esetén $\mu(n) < 1/p(n)$.*

Például $2^{-\sqrt{n}}$ és $n^{-\log_2 n}$ elhanyagolható függvények n -ben.

13.2.1. Példák egyirányú függvényekre

Számos egyirányú függvény jelöltet ismerünk, amelyekre mind ez ideig nem ismeretes hatékony invertáló algoritmus a fenti definíciók szerint, s amelykről sejtjük az egyirányú tulajdonságot. Példaként három jól ismert jelöltet említtünk meg: az egész szám faktorizációt, a hibajavító véletlen lineáris blokk kódok dekódolását, valamint a részletösszeg problémát.

Egész szám faktorizáció. Tekintsük a következő függvényt:

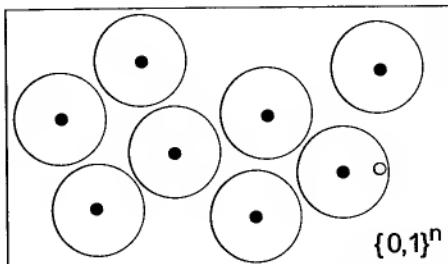
$$f : \{0, 1\}^* \rightarrow \{0, 1\}^*, \quad f(x, y) = x \cdot y, \quad |x| = |y|, \quad (13.2)$$

ahol $x \cdot y$ a bináris ábrázolású x és y egészek szorzata, bináris alakban meghatározva. Azaz két, véletlenszerűen választott $n/2$ bites egész szorzatát ismerve elhanyagolható (tetszőleges $p(n)$ polinomra aszimptotikusan $1/p(n)$ -nél kisebb valószínűségű) azon esetek száma, amikor sikeres a faktorizáció.

Hibajavító véletlen lineáris blokk kódok dekódolása. Tekintsünk egy $C(n, r, n, \delta \cdot n)$, $0 < r, \delta < 1$ bináris lineáris kódot. Az n bites bináris kódszavakat $(r \cdot n) \times n$ méretű G bináris generátor mátrix állítja elő. Jelöljön m egy $r \cdot n$ bites üzenetvektort. Egy e hibavektor egy n bites vektor, melynek Hamming-súlya kisebb, mint $\delta \cdot n/2$. Továbbá legyen $r < 1 - h(\delta)$, ahol $h(\cdot)$ a bináris entrópia függvény. A hibajavító kódok elméletéből ismert, hogy ha $r < 1 - h(\delta)$, akkor egy G véletlen $(r \cdot n) \times n$ méretű bináris generátor mátrix minden esetben egy $d \geq \delta \cdot n$ minimális Hamming-távolságú

kódot generál. Egy ilyen kód képes javítani tetszőleges $\delta \cdot n/2$ -nél kisebb Hamming-súlyú hibát. Ezen jelölésekkel a kapcsolatos egyirányú függvény:

$$f : \{0,1\}^* \rightarrow \{0,1\}^*, \quad f(G, m, e) = (G, mG + e). \quad (13.3)$$



13.3. ábra. A kódszavak terének szemléltetése (tele fekete körök a kódszavakat, tele szürke kör egy hibás kódszót jelöl, nagy körök a kódszavak körii $\delta \cdot n/2$ sugarú Hamming-gömböket jelölik)

A fenti dimenzionális megkötéseknek megfelelően véletlenszerűen választott G generátor mátrix ismeretében, valamint véletlenszerűen választott üzenet és hibavektor mellett adódó $mG + e$ hibás kódszó ismeretében – az egyirányúság fenti definíciója szerinti értelemben – nehéz feladatnak sejtett az m üzenet dekódolása.

A feladat a következőképp is megfogalmazható: ha egy altér adott az n bites vektorok lineáris terében a G mátrix sorai mint véletlen bázis által, akkor ismerve ezen bázist, valamint az altérben valamely, véletlenszerűen választott c vektor $\delta \cdot n/2$ sugarú környezetén belül véletlenszerűen választott vektort, a c vektor meghatározása nehéz feladat (13.3. ábra).

Részletösszeg probléma. Legyen

$$f : \{0,1\}^* \rightarrow \{0,1\}^*, \quad f(x_1, x_2, \dots, x_n, I) = (x_1, x_2, \dots, x_n, \sum_{i \in I} x_i), \quad (13.4)$$

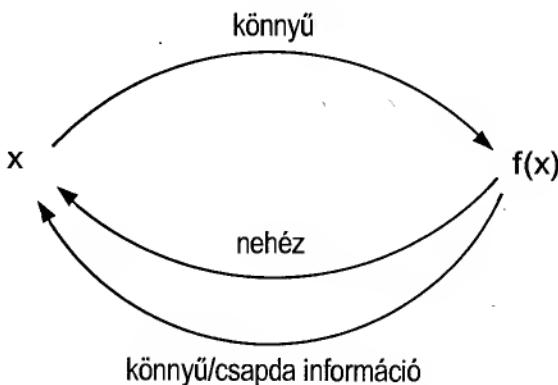
ahol $|x_1| = \dots = |x_n| = n$ és $I \subseteq \{1, 2, \dots, n\}$.

Azaz, ha véletlenszerűen választunk egy I részhalmazt véletlenszerűen választott n darab n bites vektor alkotta halmazból, majd a részhalmaz elemeit (koordinátánként egész számkként) összegezzük, akkor az összeg alapján – az egyirányúság fenti definíciója szerinti értelemben – nehéz feladatnak sejtett az összetevők azonosítása.

Az ún. hártsáktípusú (knapsack) algoritmusok a részletösszeg probléma nehézségére épülnek. Mindezidáig azonban ezen problémára épülő biztonságos rejtjelező algoritmust nem sikerült találni. Ezzel kapcsolatban részletes magyar nyelvű leírás található korábbi könyünkben (lásd irodalom jegyzék).

13.3. Csapda egyirányú permutáció

A csapda egyirányú permutáció egy speciális egyirányú függvény, amely egy bitsorozat (a csapda) ismeretében hatékonyan invertálható, ennek hiányában azonban egyirányú. Először a függvényegyüttetés, valamint a hatékonyan kiszámítható függvényegyüttetés fogalmát definiáljuk.



13.4. ábra. Csapda egyirányú permutáció illusztráció

13.15. Definíció (függvények együttese). Az F_n -függvények $F = \{F_n\}_{n \in N}$ indexelt halmazát függvények együttesének nevezzük.

13.16. Definíció (hatékonyan kiszámítható függvényegyüttetés). Egy függvényegyüttetés hatékony kiszámíthatósága az alábbi két feltétel teljesülését jelenti:

1. Hatékonyan indexelhető: létezzen hatékony indexelő $I : \{0, 1\}^* \rightarrow \{0, 1\}^*$, továbbá az indexet mint bináris sorozatot függvénybe képező $\rho : \{0, 1\}^* \rightarrow \{f : \{0, 1\}^n \rightarrow \{0, 1\}^n\}$ algoritmus.

Az I algoritmus az 1^n input (valamely biztonsági paraméter értéke) alapján megcímmez egy függvényt. Jelölje $I(1^n) = i$ ezt a címet, amely alapján $\rho(i) = f_i$, $f_i : \{0, 1\}^n \rightarrow \{0, 1\}^n$ a megcímzett függvény.

2. Hatékonyan kiértékelhető: létezzen egy randomizált polinom idejű F^* algoritmus, amelyre $F^*(i, x) = f_i(x)$.

13.4. Példa. Hatékonyan kiszámítható függvényegyüttésre tekintsünk példaként egy speciális együttest, a diszkrét hatványozást, amely permutáció és egyirányú is egyben, s az alkalmazások szempontjából igen fontos. Ezen példa után bevezetjük az egyirányú permutáció-együttes speciális esetét, a nyilvános kulcsú rejtjelezés és digitális aláírás szempontjából alapvető, csapda egyirányú permutáció-együttest. A diszkrét hatványozás – eddigi ismertek szerint – nem rendelkezik csapda tulajdonsággal.

Legyen p egy nagy prímszám (például 500 bit méretű). Tekintsük az $\text{EXP}_{p,g} : \mathbb{Z}_{p-1} \rightarrow \mathbb{Z}_p^*$ függvényt:

$$\text{EXP}_{p,g}(x) = g^x \bmod p,$$

ahol g generátor eleme a \mathbb{Z}_p^* ciklikus csoporthnak. Ez a leképezés bijektív.

Az I indexelő algoritmus az 1^n biztonsági paraméter inputra véletlenszerűen generál egy $2^{n-1} \leq p < 2^n$ prímet, majd egy $g \in \mathbb{Z}_p^*$ generátorelemet. Ezen feladatok hatékonyan megoldhatók. A p, g pár kiválasztásával az $\text{EXP}_{p,g}$ transzformációt is kiválasztottuk. A diszkrét hatványozás hatékonyan kiszámítható az ismételt négyzetreemelés és szorzás módszerével. Az inverzképzés nehézsége azt jelenti, hogy az

$$f(p, g, x) = (p, g, \text{EXP}_{p,g}(x))$$

leképezést egyirányúnak feltételezzük. ♣

A csapda egyirányú permutáció alábbi definíciójában a könnyebb követhetőség kedvéért a tipikus használat, a nyilvános kulcsú rejtjelezés jelöléseit alkalmazzuk.

13.17. Definíció (csapda permutáció-együttes). Függvények

$$F = \{E_{pk} : X_{pk} \rightarrow X_{pk}\}_{pk \in PK}$$

együttese csapda permutáció-együttes, ha

1. Kulcsgenerálás: létezik egy hatékony G algoritmus, amelynek $G(n)$ kimenete (pk, sk) , ahol $|pk| = n$, $|sk| = q(n)$ valamely $q(n)$ polinomra.
2. Egyirányú tulajdonság: létezik egy hatékony E^* algoritmus, amelyre teljesül, hogy $E^*(pk, x) = E_{pk}(x)$ tetszőleges $x \in X_{pk}$, $pk \in PK$ esetén, továbbá

$$\Pr_{\substack{x \leftarrow r_{X_{pk}} \\ (pk, sk) \leftarrow G(n)}} \{E_{pk}(T(pk, E_{pk}(x))) = E_{pk}(x)\} < \frac{1}{p(n)} \quad (13.5)$$

tetszőleges hatékony T algoritmus és $p(n)$ polinom, valamint elegendően nagy n esetén. A valószínűséget az alábbi valósínűségi változók eloszlása felett képezzük: (pk, sk) nyilvános-titkos kulcspár, amit $G(1^n)$ kulcsgeneráló algoritmus állít elő, x üzenet, amit az ōsképtérből egyenletes eloszlás szerint véletlenül választunk, valamint T hatékony algoritmus véletlen elemei, amiket számítása során felhasznál. Továbbá $v \leftarrow_r V$ egy V véges halmazból egyenletes eloszlással választott v valószínűségi változót jelöl.

3. *Csapda permutáció tulajdonság: létezik egy hatékony D algoritmus, hogy $D(sk, E_{pk}(x)) = x$ tetszőleges $x \in X_{pk}$, $pk \in PK$ esetén.*

A továbbiakban az egyszerűség (és a szokásos megnevezés) okán, ahol az nem félreérthető, a rövidebb *csapda egyirányú permutáció*, illetve a *csapda permutáció* megnevezést használjuk. Ismét vegyük észre, hogy az inverz számításának nem egy adott kulcsához tartozó permutáció esetén kell „nehéz feladatnak” lennie, hanem adott n biztonsági paraméter mellett kapható összes kulcs és összes ōskép átlagában.

A csapda permutáció modell mint a nyilvános kulcsú rejtjelező kódolók alapja azonban önmagában több szempontból sem kielégítő. A 13.17. definíció nem állítja, hogy az invertálás nehéz, ha az x bemenet speciálisan választott térből származik (az egyirányú tulajdonságot a teljes ōsképtérből egyenletesen választott bemenetre mondjuk ki). Szélsőséges, de a mondani-valót jól illusztráló példa az, amikor az üzenettér két elemű, azaz 0 vagy 1. Nyilván könnyű megkülönböztetni a lehetséges két rejtjeles szöveget, például azáltal, hogy kérjük az egyik üzenetre a rejtjelezettjét egy választott szövegű támadásnál.

Továbbá abból, hogy nehéz invertálni egy csapda egyirányú függvényt (vagy általában egy egyirányú függvényt) nem következik, hogy nehéz feladat részleges információt szerezni az ōsképről (pl. megtudni egy bitjét).

Az is előfordulhat, hogy különböző rejtjelezett üzenetek között relációk fedezhetők fel. A legegyszerűbb ilyen eset az, amikor azonos üzenetet különböző időpontokban rejtjelezve küldünk. Determinisztikus lévén a rejtjelezés, a rejtjeles üzenetek nyilván azonosak, ami információt jelent a támadó számára. Ezen probléma illusztrációjául tekintsük az RSA rejtjelezést $e = 3$ nyilvános kitevővel, s tegyük fel, hogy először x majd $x + 1$ üzenetet rejtjelezzük. Ha a támadó megfigyeli a rejtjeles üzeneteket, ki tudja számítani x értékét. Ugyanis $y_1 = x^3 \pmod{N}$ és $y_2 = (x + 1)^3 = x^3 + 3x + 3x^2 + 1 = y_1 + 3x + 3x^2 + 1 \pmod{N}$, így

$$\frac{y_2 + 2y_1 - 1}{y_2 - y_1 + 2} = \frac{x(3x^2 + 3x + 3)}{3x^2 + 3x + 3} = x$$

adódik (ahol a kijelölt osztás művelete az RSA modulusra vett inverzzel való szorzást jelenti, s feltesszük az inverz létezését).

13.3.1. Példák csapda egyirányú függvényre

RSA csapda permutáció. Az RSA csapda permutáció létezésének alapja az alábbi RSA feltételezés.

13.18. Definíció (RSA feltételezés). *Tetszőleges hatékony T algoritmus, tetszőleges $p(n)$ polinom és elegendően nagy n esetén*

$$\Pr_{N,e}\{T(N, e, E_{N,e}(x)) = x\} \leq \frac{1}{p(n)},$$

ahol a valószínűséget az alábbi valószínűségi változók eloszlása felett képezzük: (N, e) az RSA publikus kulcsa, amit $G(1^n)$ kulcsgeneráló algoritmus állít elő, x üzenet, amit az űsképtérből egyenletes eloszlás szerint véletlenül választunk, T hatékony algoritmus véletlen elemei, amiket számítása során felhasznál.

1. A $G(1^n)$ kulcsgeneráló algoritmus generál két, véletlenül választott p, q prímszámot, amelyek mindegyike $2^{n/2}$ nagyságrendű, majd kiszámítja az $((N, e), d)$ nyilvános-titkos kulcs párt, ahol $N = pq$ az RSA modulus, továbbá $ed = 1 \pmod{\varphi(N)}$. Az $X_{(n,e)}$ űsképtér a $0, 1, \dots, n-1$ halmaz.

Van hatékony $G(1^n)$ algoritmus kulcsgenerálás céljára: először is van hatékony algoritmus a prím választásra, majd ezek szorzatával előáll az N modulus, illetve a prímek eggyel csökkengett értékének szorzatából $\varphi(N)$ értéke. A e kitevő megtalálásához, $e = 3$ értékről indulva, növeljük egyenként e értékét mindaddig, amíg $l.n.k.o.(e, \varphi(N)) = 1$ feltétel nem teljesül. Mivel – φ definíciójából levezethetően – fennáll a

$$\hat{\varphi}(\varphi(N)) \geq \frac{\varphi(N)}{c \ln \varphi(N)}$$

egyenlőtlenség valamely c számra, ezért $\approx c \ln \varphi(N)$ szám tesztelése elegendő az e kitevő megtalálásához.

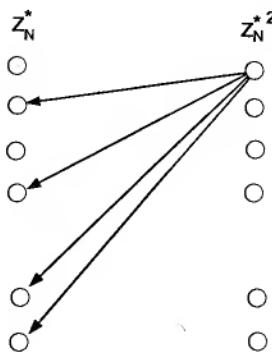
Szigorú értelemben (az Euler–Fermat-tétel kapcsán) az űsképtér nem Z_N , hanem Z_N^* , nem tartalmazza azon x üzeneteket, amelyekre $l.n.k.o.(x, N) \neq$

1. Mivel azonban $\frac{|Z_N^*|}{|Z_N|} = \left(1 - \frac{1}{p}\right) \left(1 - \frac{1}{q}\right) \approx 1$, ezért gyakorlatilag nincs szükség erre a megszorításra. Másrészt, ahogy láttuk az RSA dekódolás kapcsán, a dekódoló (szerencsésen) visszaállítja az üzenetet még az $l.n.k.o.(x, N) \neq 1$ esetben is.
2. A kódolás $E_{N,e}(x) = x^e \bmod N$, amely a d titkos kulcs ismerete nélkül, az RSA feltételezés következtében, nem invertálható. A kódolás művelete egy diszkrét hatványozás, ami hatékonyan végrehajtható.
3. A d titkos kulcs ismeretében a dekódoló: $(E_{(N,e)}(x))^d = x \bmod N$, amely determinisztikus polinom idejű algoritmussal kiszámítható.

Rabin-függvény. A $G(1^n)$ kulcsgeneráló algoritmus generál két, véletlen p, q prímszámot, amelyek mindegyike $2^{n/2}$ nagyságrendű. A nyilvános kulcs $N = p \cdot q$, a csapda információ p, q . Tekintsük az

$$F_{(N,e)}(x) = x^2 \bmod N$$

függvényt. Ezen függvény invertálása nehéz feladat, ha csak az N értéke ismert (lásd az alábbi tételel). Ugyanakkor az invertálás könnyű feladat, ha az N prímtényezői ismertek.



13.5. ábra. A Rabin-függvény nem bijektív: egy képtérbeli elemmek (kvadratikus maradéknak) 4 ősképe van

Az invertálás nehézsége pontosabban a következőt jelenti:

13.19. Tétel. Akkor és csak akkor létezik hatékony algoritmus a Rabin-függvény invertálására, ha létezik az egész faktorizációs feladatra is.

Bizonyítás: Mivel kvadratikus maradékból történő négyzetgyökvonás feladata modulo prímszám aritmetikában polinom idejű megoldó algoritmus ismert, ezért, ha N két faktorját ki tudjuk számítani, akkor a Rabin-függvényt hatékonyan tudjuk invertálni. Ha viszont ezen faktorokat nem ismerjük, akkor az invertálás nehéz feladat:

Indirekt bizonyításhoz tegyük fel, hogy van egy gyökvonó algoritmusunk, amely hatékonyan kiszámítja egy

$$x^2 = c \pmod{N} \quad (13.6)$$

egyenlet egy megoldását tetszőleges c kvadratikus maradék esetén. Válaszszunk véletlenszerűen egy d , $0 < d < N$ számot, és számítsuk ki a négyzetet \pmod{N} . Legyen $c = d^2 \pmod{N}$. Gyökvonó algoritmusunknak adjuk bemenetként ezen c kvadratikus maradékot, s legyen az algoritmusunk kimenete d' . Nyilván $d^2 - d'^2 = 0 \pmod{N}$, azaz

$$N|(d - d')(d + d'). \quad (13.7)$$

Az algoritmusunk kimenete lényegében vagy d (azaz $d' = \pm d \pmod{N}$), vagy nem az (azaz $d' \neq \pm d \pmod{N}$), amely két lehetőség mindegyikének valószínűsége $1/2$, d szám véletlenszerű választása miatt.

Ha a $d' = \pm d$ esemény következik be, azaz, ha $\{d - d' = 0 \text{ vagy } d + d' = N\}$, akkor a (13.7) oszthatóság triviális. A másik esetben viszont, sem $d - d'$ sem $d + d'$ nem osztható N -nel, ezért ekkor akár az $l.n.k.o.(N, |d - d'|)$, akár az $l.n.k.o.(N, d + d')$ legnagyobb közös osztó számítással N prímosztóját kapnánk meg, amivel ellentmondásra jutunk, hiszen az egész szám faktORIZÁLÁS feladatot nehéznek sejtjük. □

A Rabin-függvény formulája nagyon hasonló az RSA kódoló formulához („mintha $e = 2$ publikus kitevőjű RSA lenne”). Felmerül a kérdés, hogy lehetne-e a Rabin-függvényt nyilvános kulcsú rejtelező kódolóként gyakorlatban is használni. Sajnos több gond merül fel ezzel kapcsolatban:

Először is probléma van az egyértelmű inverz biztosításával, azaz amikor a vételi oldalra megérkezik a rejteles üzenet, s négyzetgyököt vonunk, 4 különböző megoldást, lehetséges nyílt üzenetet kapunk. Ezen az üzenettér leszűkítésével próbálhatunk meg segíteni. Egy lehetséges megoldás a következő: a nyílt szöveg alsó j bitjét (pl. alsó 100 bitjét) véletlen bitekkel töltjük fel, azaz az n bites blokkból csak $n - j$ bitet használunk egy m üzenet átvitelére: m üzenet esetén egy $x = [m, r]$ üzenetet képezünk, ahol r egy j bit hosszú véletlen sorozat, továbbá rejteles üzenetként $[x^2 \pmod{N}, r]$ kerül

továbbításra. A dekódoló azt a gyököt választja, amelynek alsó j bitjén r áll. Ezzel elhanyagolható lesz annak a valószínűsége, hogy a választás nem egyértelmű.

Az is lehetséges, hogy az üzenetek halmaza eredetileg is ritkított, például valamelyen természetes formátum követelménynek eleve eleget tesz. Vegyük azonban észre, hogy amint az egyértelmű dekódolhatóság szempontot követve ritkítjuk az üzenetteret, egyúttal megszüntetjük a faktorizációra redukálás (bizonyított biztonság) 13.19. tételebeli ötletét.

Ha viszont nem ritkítjuk az üzenetteret, akkor a támadó választott rejttet szövegű támadással dekódolást kérhet egy tetszőleges rejttet szöveget, és a 13.19. téTEL ötlete alapján faktorizálni tudná a modulust, s így a csapda információhoz jutna.

13.4. Keménybit

Abból a feltételezésből, hogy egy f függvény egyirányú, nem következik, hogy nehéz az ōsképekre vonatkozóan részleges információt kiszámítani, tudniillik az invertálás nehézsége a teljes ōskép, azaz minden az n bitjének kiszámítási nehézségére vonatkozik.

Például, ha f függvény egyirányú, akkor – indirekt igazolással – könnyen láthatóan

$$g(x, r) = (f(x), r), \quad |x| = |r|$$

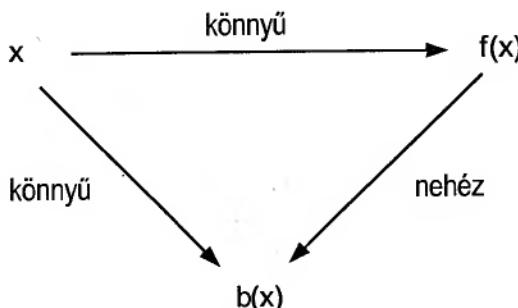
is egyirányú, ugyanakkor ezen g egyirányú függvény ōsképe bitjeinek fele – nyilván – könnyen „kiszámítható”.

Sejthetően kell, hogy legyen az ōsképnek olyan bitje, amelyet nem tudunk kiszámítani, amelynek kiszámítása nem sikeresebb bármely hatékony algoritmus számára annál, mintha egyszerűen pénzfeldobással döntene annak értékéről. Ezt a heurisztikát formalizálja egy egyirányú függvény keménybit-jének alábbi definíciója:

13.20. Definíció (keménybit). *Egy $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ egyirányú függvény keménybit leképezése a $b : \{0, 1\}^* \rightarrow \{0, 1\}$ függvény, ha b hatékonyan kiszámítható, továbbá tetszőleges hatékony T' algoritmus és tetszőleges $p(n)$ polinom, valamint elegendően nagy n esetén*

$$\Pr\{T'(f(U_n)) = b(U_n)\} < \frac{1}{2} + \frac{1}{p(n)}. \quad (13.8)$$

A b függvény értékét az f függvény keménybitjének hívjuk (13.6. ábra).



13.6. ábra. Keménybit leképezés

A következő keménybit tétele a továbbiakban kifejtésre kerülő több biztonságos konstrukció elméleti alapja. A tétele állításának ismerete is elégsges felhasználásának követéséhez, ezért első olvasásnál átugorható.

13.21. Tétel (Goldreich–Levin). *Legyen f egy hossztartó egyirányú függvény. Az*

$$f'(x, w) = (f(x), w), \quad |x| = |w|, \quad x, w \in \{0, 1\}^*$$

egyirányú függvénynek $b(x, w) = \langle x, w \rangle$ keménybit leképezése, ahol $\langle x, w \rangle$ a skaláris szorzást jelöli, azaz $\langle x, w \rangle = \sum_j x_j \cdot w_j \bmod 2$.

Bizonyítás: Indirekt feltételezésként tegyük fel, hogy létezik egy hatékony B algoritmus, továbbá $p(n)$ polinom, hogy elegendően nagy n esetén

$$\Pr_{x, w} \{B(f(x), w) = \langle x, w \rangle\} \geq \frac{1}{2} + \frac{1}{p(n)}. \quad (13.9)$$

Ezen B algoritmusra redukálunk egy INV algoritmust, amely az f egyirányú függvényt invertálja.

13.1. Algoritmus. Az INV algoritmus működése a következő:

1. A $\{0, 1\}^n$ halmazból véletlenszerűen kiválasztunk Y_0, Y_1, \dots, Y_t elemeket, ahol $t = c \cdot \log_2(n)$. A c konstansot a továbbiakban meg fogjuk határozni.
2. Kiválasztunk $t+1$ véletlen bitet: $G_j \in \{0, 1\}$, $j = 0, 1, \dots, t$.
3. Legyen $Z_I = Y_0 + \sum_{i \in I} Y_i \pmod{2}$, tetszőleges $I \subset \{1, 2, \dots, t\}$ részhalmazra.
4. minden $k \in \{1, 2, \dots, n\}$ esetén INV futtatja B algoritmust $(f(x), Z_I \oplus e_k)$ bemenettel, ahol $e_k \in \{0, 1\}^n$ egységvektor a k -adik bitje kivételével zérus.

Definiáljuk a $V_k^I = B(f(x), Z_I \oplus e_k) + G_0 + \sum_{i \in I} G_i \pmod{2}$ biteket, $k \in \{1, 2, \dots, n\}$.

5. minden egyes $k \in \{1, 2, \dots, n\}$ esetén az alábbi többségi döntést hozzuk:

$$\bar{x}_k = \text{majority}_I(V_k^I).$$

6. Ellenőrizzük az $f(x) = f(\bar{x})$ egyenlőség teljesülését, ahol $\bar{x} = \bar{x}_1 \dots \bar{x}_n$ és $x = x_1 \dots x_n$, s ha fennáll az egyenlőség, akkor INV kimenete \bar{x} .

Azt fogjuk mondani, hogy egy x „jó”, ha

$$\Pr_w \{B(f(x), w) = \langle x, w \rangle\} \geq \frac{1}{2} + \frac{1}{2p(n)}.$$

Indirekt bizonyításként azt mutatjuk meg, hogy INV algoritmus sikervalósínűsége nem elhanyagolható, ha a (13.9) fennáll. A bizonyítás során ezen sikervalósínűségre adunk egy nem elhanyagolható alsó korlátot a következő egyenlőtlenségből kiindulva:

$$\begin{aligned} \Pr_x \{INV(f(x)) = x\} &\geq & (13.10) \\ &\geq \Pr_x \{x \text{ jó}\} \cdot \Pr_x \{G_j = \langle x, Y_j \rangle, j \in \{0, 1, \dots, t\} \mid x \text{ jó}\} \cdot \\ &\quad \Pr_x \{INV(f(x)) = x \mid x \text{ jó} \text{ és } G_j = \langle x, Y_j \rangle, j \in \{0, 1, \dots, t\}\}. \end{aligned}$$

A továbbiakban (13.10) jobb oldali három valószínűségére adunk külön-külön alsó korlátot, majd ezeket egyesítjük.

1. állítás: $\Pr_x \{x \text{ jó}\} \geq \frac{1}{2p(n)}$.

A (13.9) indirekt feltételezésből kiindulva

$$\begin{aligned} \frac{1}{2} + \frac{1}{p(n)} &\leq \Pr_{x,w} \{B(f(x), w) = \langle x, w \rangle\} \\ &= \Pr_{x,w} \{B(f(x), w) = \langle x, w \rangle \mid x \text{ jó}\} \Pr_{x,w} \{x \text{ jó}\} + \\ &\quad \Pr_{x,w} \{B(f(x), w) = \langle x, w \rangle \mid x \text{ nem jó}\} \Pr_{x,w} \{x \text{ nem jó}\} \\ &\leq \Pr_x \{x \text{ jó}\} + \left(\frac{1}{2} + \frac{1}{2p(n)} \right) \Pr_x \{x \text{ nem jó}\} \\ &\leq \Pr_x \{x \text{ jó}\} + \left(\frac{1}{2} + \frac{1}{2p(n)} \right), \end{aligned}$$

ahonnan következik az 1. állítás.

2. állítás: $\Pr\{G_j = \langle x, Y_j \rangle, j \in \{0, 1, \dots, t\} \mid x \text{ jó}\} = \frac{1}{2n^c}$

Mivel G_j bináris valószínűségi változók függetlenek egymástól és a többi valószínűségi változótól:

$$\begin{aligned} \Pr\{G_j = \langle x, Y_j \rangle, j \in \{0, \dots, t\} \mid x \text{ jó}\} &= \quad (13.11) \\ &= \prod_{j=0}^t \Pr\{G_j = \langle x, Y_j \rangle\} \\ &= \prod_{j=0}^t \frac{1}{2} = \frac{1}{2^{t+1}} = \frac{1}{2n^c}, \end{aligned}$$

amivel beláttuk a 2. állítást.

Rövidség kedvéért jelölje W az $\{x \text{ jó} \text{ és } G_j = \langle x, Y_j \rangle, j = 0, 1, \dots, t\}$ feltételt.

3. állítás: $\Pr_x\{INV(f(x)) = x \mid W\} \geq 1 - \frac{1}{n}$

Ezen állítás alábbi bizonyításában a valószínűségek W feltétel mellettiek; a formalizmus könnyebb követhetőségét remélve elhagyjuk ezen feltétel kiírását. Ha minden $i = 0, 1, \dots, t$ esetén $G_i = \langle x, Y_i \rangle$, akkor

$$G_0 + \sum_{i \in I} G_i \bmod 2 = \langle x, Z_I \rangle.$$

Definiáljuk a

$$C_k^I = \begin{cases} 1, & \text{ha } V_k^I = x_k \\ 0, & \text{ha } V_k^I \neq x_k \end{cases}$$

függvényt. Vegyük észre, hogy C_k^I a V_k^I definícióján keresztül csak Z_I függvénye. Így, mivel a különböző I halmazok esetén Z_I valószínűségi változók páronként függetlenek, ugyanez igaz C_k^I valószínűségi változók halmzárára is. C_k^I definíciója alapján

$$\Pr\{x_k = \bar{x}_k\} = \Pr\{\sum_I C_k^I > \frac{2^t}{2}\}, \quad (13.12)$$

mivel 2^t részhalmaza van $\{1, 2, \dots, t\}$ halmaznak. Ugyancsak C_k^I definíciója alapján $E[C_k^I] = P[V_k^I = x_k] > 1/2 + 1/(2p(n))$. Így

$$E\left[\sum_I C_k^I\right] > 2^t \left(\frac{1}{2} + \frac{1}{2p(n)}\right) \quad (13.13)$$

adódik. (13.12) és (13.13) felhasználásával kapjuk, hogy

$$\Pr\{x_k \neq \bar{x}_k\} \leq \Pr\left\{\left|\sum_l C_k^l - E\left(\sum_l C_k^l\right)\right| \geq \frac{2^t}{2p(n)}\right\}.$$

Innen a Csebisev-egyenlőtlenség felhasználásával a következő felső becslést kaphatjuk:

$$\frac{\text{Var}\left[\sum_l C_k^l\right] 4[p(n)]^2}{2^{2t}} \leq \frac{E\left[\sum_l C_k^l\right] 4[p(n)]^2}{2^{2t}} \leq \frac{4[p(n)]^2}{2^t}, \quad (13.14)$$

ahol az első egyenlőtlenségnél kihasználtuk, hogy 0-1 értékű, páronként független valószínűségi változók összegét tekintjük. Ha a c konstansra $n^c \geq 4(p(n))^2 n^2$ egyenlőtlenség teljesül, figyelembe véve a $t = c \cdot \log_2(n)$ összefüggést, először a $4(p(n))^2 / 2^t \leq 1/n^2$, majd (13.14) alapján $P[x_k \neq \bar{x}_k] \leq 1/n^2$ formulára jutunk. Innen adódik, hogy

$$\Pr\{x = \bar{x}\} \geq 1 - \sum_k \Pr\{x_k = \bar{x}_k\} \geq 1 - n \cdot (1/n^2) = 1 - 1/n,$$

s ezzel beláttuk a 3. állítást.

Visszatérve a (13.10) egyenlőtlenséghez, figyelembe véve a fentiekben kapott három részteredményt az alábbi alsó korlátot kapjuk az INV algoritmus sikervalószínűségére:

$$\Pr_x\{INV(f(x)) = x\} \geq \frac{1}{2p(n)} \frac{1}{2n^c} \left(1 - \frac{1}{n}\right).$$

Válasszuk c konstansot $n^c = 4(p(n))^2 n^2$ szerint, amivel

$$\frac{1}{16p(n)^3 n^2} \left(1 - \frac{1}{n}\right)$$

alsó becslésre jutunk, ami ellentmondana az f függvény egyirányú tulajdon-ságának. \square

A Goldreich–Levin-(GL)-tételet alkalmazásaként tekintsünk egy F csapda egyirányú permutációt. Képezzünk egy F' csapda egyirányú permutációt az F permutáció – 13.17. definíció szerinti – kódoló és dekódoló algoritmusainak alábbi módosításával (a kulcsgeneráló algoritmus ázonos marad):

$$\begin{aligned} E'(pk, (x, r)) &= E(pk, x), r \\ D'(sk, (z, r)) &= D(sk, z), r \end{aligned}$$

ahol r tetszőleges bitsorozat, amelyre $|r| = |x| = |z|$. Az így kapott csapda permutáció keménybit leképezése $b_{pk}(x, r) = \langle r, x \rangle$. Így a GL-tétel esetfüggetlen konstrukciót mond a keménybit leképezésre, ráadásul a leképezés nagyon egyszerű. Tehát a keménybit definíciója értelmében, ha véletlenszerűen választjuk az x és az r elemeket, továbbá a (pk, sk) kulcs-párt, akkor $1/2$ felett elhanyagolhatóan kicsi annak valószínűsége, hogy az E' kódoló kimenete ismeretében helyes döntést lehessen hozni a keménybitre.

13.4.1. Az RSA algoritmus egyszerű keménybit-leképezése

RSA esetén az üzenet lsb (legkisebb helyiértékű) bitje keménybit. Ezt mondja ki az alábbi téTEL:

13.22. TétEL (RSA lsb bit). *Tetszőleges hatékony Z algoritmus, tetszőleges $p(n)$ polinom és elegendően nagy n esetén*

$$\Pr_{N,e}\{Z(N, e, E_{N,e}(x)) = \text{lsb}(x)\} \leq \frac{1}{2} + \frac{1}{p(n)},$$

ahol a valószínűség számításával kapcsolatos megjegyzésünk az RSA feltételezés definiciójánál mondottakkal azonos.

Egy kapcsolódó, egyszerűbben igazolható, enyhébb állítást bebizonyítunk az alábbiakban, amelynek bizonyításlogikája azonban jól mutatja a 13.22. téTEL bizonyítása lényegi ötleteit a hosszabb technikai részletek nélkül.

13.23. TétEL. *Tetszőleges hatékony Z algoritmus, tetszőleges $p(n)$ polinom és elegendően nagy n esetén*

$$\Pr_{N,e}\{Z(N, e, E_{N,e}(x)) = \text{lsb}(x)\} < 1.$$

Bizonyítás: Tegyük fel, hogy a téTEL állítása nem igaz, azaz létezik olyan hatékony algoritmus, amely 1 valószínűsséggel megmondja az RSA rejtelezés által rejtelezett üzenet ismeretében az üzenet lsb bitjét. Legyen $\bar{x}_i = 0$, ha $x \in (t \cdot N/2^i, (t+1) \cdot N/2^i)$ egy t páros számra (a zérust is beleértve), illetve $\bar{x}_i = 1$, ha $x \in (t \cdot N/2^i, (t+1) \cdot N/2^i)$, egy t páratlan számra, $i = 1, 2, \dots, \lfloor \log_2 N \rfloor$. (Vegyük észre, hogy x nem eshet a tartományok határára N RSA modulus képzése miatt.)

Például, ha x 0 és $N/2$ közé esik, akkor $\bar{x}_1 = 0$, ha pedig x $N/2$ és N közé esik, akkor $\bar{x}_1 = 1$. \bar{x}_2 értéke tovább felezi az értéktartományt: így például, ha x $N/2$ és $3N/4$ közé esik, akkor $\bar{x}_1 = 1$ és $\bar{x}_2 = 0$. Nem nehéz belátni, hogy

$$\bar{x}_i = \text{lsb}(2^i x \bmod N).$$

Például, ha $\bar{x}_i = 0$, akkor $x \in (t \cdot N/2^i, (t+1)N/2^i)$, ahol t páros, következésképp $2^i x \bmod N = 2^i x - t \cdot N$, ahol a jobb oldal párossága miatt a bal oldal is páros, azaz lsb bitje valóban 0. Hasonlóan gondolható végig az $\bar{x}_i = 1$ eset is.

Számítassuk ki a tételelbeli Z algoritmussal $\bar{x}_i = Z(n, e, V_i)$ biteket, ahol

$$V_i = E_{N,e}(2^i x) = E_{N,e}(x) \cdot E_{N,e}(2^i) \pmod{N}$$

$i = 1, 2, \dots, \lfloor \log_2 N \rfloor$. Az x üzenet az \bar{x}_i bitek alapján már könnyen rekonstruálható a támadó számára. \square

13.5. Feladatok

13.1. Feladat*. Mutassuk meg, hogy ha van algoritmusunk az egész szám faktorizáció keresési feladatra, akkor van a kapcsolatos döntési feladatra is!

13.2. Feladat. Tekintsük a prímség döntési problémát, mint RP-beli döntési problémát. A Rabin–Miller-prímteszt esetén adja meg a döntési pontosságot!

13.3. Feladat. Tekintsük a BPP bonyolultságosztályt. Legyen M véletlen polinom idejű Turing gép $\frac{1}{2} + \frac{1}{p(n)}$ döntési pontossággal, ahol $p(n)$ egy polinom, továbbá x az M gép n bites bemenete. Mutassuk meg, hogy M alapján konstruálható egy M' gép, amelynek pontossága legalább $2/3$!

13.4. Feladat. Legyen $f_{len}(x)$ az x bemenetének bitben mért hosszát bináris számként megadó függvény. Egyirányú-e ez a függvény, ha az invertáló algoritmus időkorlátja

1. $p(|f_{len}(x)|)$
2. $p(|x|)$ valamely $p(n)$ polinomra!

13.5. Feladat. Egy Z „invertáló” algoritmus véletlenszerűen választ egy beállítást. Mekkora a sikeres valószínűsége, ha

1. f permutáció
2. f egy konstansba képez le!

13.6. Feladat. Igaz vagy sem a következő állítás: ha $f(x)$ egy erősen egyirányú függvény, akkor g függvény is erősen egyirányú függvény, ahol $g(x, r) =$

$(f(x), r)$, ha $r \log_2 |x|$ számú zérussal kezdődik, egyébként legyen $g(x, r) = (x, r)$, $|x| = |r| = n$.

13.7. Feladat. Adja meg az $f(c, x) = (0, x)$, $c \in \{0, 1\}$, $x \in \{0, 1\}^*$ függvény keménybit leképezését!

13.8. Feladat. Igazoljuk, hogy egy $f(x)$ függvény $b(x)$ keménybit leképzése kiegyenlített, azaz a $|\Pr\{b(U_n) = 0\} - \Pr\{b(U_n) = 1\}|$ differencia elhalányagolhatóan kicsi!

13.9. Feladat. Mutasson példát olyan függvényre, amely triviálisan nem egyirányú, ugyanakkor van keménybitje!

13.10. Feladat. Legyen $g(x_1, x_2) = [f(x_1), f(x_2)]$, $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ alakú gyengén egyirányú függvény. Tegyük fel, hogy egy Z invertáló támadó a g függvény kimenete ismeretében képes ősképet meghatározni tetszőleges (x_1, x_2) bemenet esetén, kivéve az x_1 koordináta lehetséges értékei halma-zának $1/4$ -nél kisebb részéhez tartozó kimeneti értékeket. Adj meg egy olyan algoritmust, amely tetszőlegesen nagy valószínűséggel képes invertálni az f függvényt.

13.11. Feladat. Legyen $f : \Sigma^* \rightarrow \Sigma^*$ egy egyirányú függvény, és legyen $g : \Sigma^* \rightarrow \Sigma^*$ egy polinom időben kiszámítható függvény.

1. Tekintsük az $f'(w) = [f(w), g(w)]$ függvényt. Mutassuk meg, hogy g választható úgy, hogy bármely f esetén f' nem egyirányú.
2. Egyirányú-e az $f \circ g$, illetve a $g \circ f$ összetett függvény?
3. Tegyük fel, hogy $\Sigma = \{0, 1\}$. Legyen f és g hossztartó. Egyirányú-e az $f \oplus g(w) = f(w) \oplus g(w)$ függvény?
4. Ha létezik f egyirányú függvény, mutassuk meg, hogy létezik olyan f' egyirányú függvény, amelyhez létezik polinom idejű Z algoritmus, amely ki tudja számítani w paritását $f'(w)$ ismeretében!

13.12. Feladat. Legyen $f : \Sigma^* \rightarrow \Sigma^*$ egy hossztartó egyirányú függvény, és legyen $g : \Sigma^* \rightarrow \Sigma^*$ egy hossztartó, polinom időben kiszámítható függvény. Mutassuk meg, hogy az $f'(x, y) = [f(x), g(y)]$ függvény egyirányú.

13.13. Feladat*. Mutassuk meg, hogy ha van hatékony algoritmus az egész szám faktorizáció feladatra, akkor hatékonyan tudjuk invertálni az RSA kódolást.

13.14. Feladat*. Mutassuk meg, hogy ha hatékonyan tudjuk kiszámítani $\phi(N)$ értékét, akkor hatékonyan tudjuk faktorizálni N modulust is.

13.15. Feladat*. Mutassuk meg, hogy ha az e nyilvános kitevő 3, és ha hozzájutunk a d titkos kitevőhöz, akkor a modulust hatékonyan faktorizálni tudjuk.

13.16. Feladat*. Az RSA multiplikatív tulajdonságú: tetszőleges m_1, m_2 üzenetpárra, $E_{pk}(m_1m_2) = E_{pk}(m_1)E_{pk}(m_2) \pmod{N}$. Ezen tulajdonság ismeretében mutassuk meg, hogy ha a támadó az üzenetek 1 százalékának hatékony dekódolására képes, akkor képes minden, pk publikus kulccsal kódolt üzenet hatékony dekódolására is.

13.17. Feladat. Tekintsük az $x^u \pmod{N}$ leképezést, ahol $N = p \cdot q$, különböző prímek szorzata. Lehet-e ez a leképezés egyértelműen dekódolható, ha $(u, \phi(N)) > 1$?

13.18. Feladat. Mi történik az RSA dekódolás kapcsán, ha prímteszten „átcsúszott” összetett számot (álprímet) használunk? Mi a helyzet akkor, ha az álprím Carmichael-szám?

13.19. Feladat. Gondot jelenthet-e, ha e nyilvános kitevő rendje mod $\phi(N)$ nem elég nagy?

14.

Véletlen és algoritmikus megkülönböztethetőség

A véletlen elem (bit, bitsorozat) nagy szerepet játszik a kriptográfiai algoritmusokban. Gondoljunk egy rejtjelező vagy digitális aláíró algoritmus titkos kulcsára, a kriptográfiai protokollok friss elemére, az üzenet véletlenítésére rejtjelezést megelőzően vagy például prímtesztelelő algoritmusokra az RSA esetén. A véletlen elem minősége kritikus ezen alkalmazások legtöbbjében, mivel a statisztikai függés, a redundancia a támadás erejét növeli. Ezért ideálisan valódi véletlen bitek (pl. pénzfeldobás-sorozat elemei) alkalmazása kívánatos. Ez azonban tipikusan kölcsönös megoldás, különösképp nagyobb rendszerek kriptográfiai alapú védelme, s legtöbb PC alapú, kommerciális implementáció esetén.

A véletlenség első algoritmikus definícióját Kolmogorov adta, amely szerint egy bináris z sorozat véletlenségét, Kolmogorov-komplexitását, a sorozatot reprodukáló algoritmusok közül a legkisebb bináris leírású algoritmus mérete adja. Következésképpen z sorozat valódi véletlen, ha a legrövidebb reprodukáló algoritmus mérete pontosan $|z|$ (azaz nem kisebb ennél). Sajnos nincsen olyan hatékony algoritmus, amely ki tudná számítani egy bemenetére adott z sorozat Kolmogorov-komplexitását.

Az algoritmikusan véletlen fogalmának egy sokkal használhatóbb megközelítése az az álvéletlenség fogalom, amely a valódi véletlentől hatékony algoritmussal való megkülönböztethetetlenségen alapul. Ezen fogalom szerint az álvéletlen elem értékét az adja, hogy valódi véletlen elemekkel ekvivalens módon alkalmazhatjuk polinom erőforrás-korlátú algoritmusainkban.

Miután a bizonyítható biztonság számos fogalma is valószínűség-eloszlások algoritmikus megkülönböztethetőségére vezethető vissza, vagy azzal

kerül definiálásra, ezért a jelen fejezetben igyekszünk ezen fontos koncepciót a továbbiakhoz szükséges mértékben körüljárni.

14.1. Valószínűség-eloszlások algoritmikus megkülönböztethetősége

Tekintsük a véges hosszúságú, bináris sorozatok (megszámlálható) halmazát és ezen halmaz felett D és D' valószínűség-eloszlásokat. Tekintsünk továbbá tetszőleges $Z : \{0, 1\}^* \rightarrow \{0, 1\}$ megkülönböztető algoritmust, amely véges t erőforrás mellett kívánja megkülönböztetni a D és D' diszkrét valószínűség-eloszlásokat az abból vett minta alapján. Z kimenete 1, ha D eloszlásra dönt, illetve kimenete 0, ha D' eloszlásra dönt. Mérje

$$|\Pr_{x \leftarrow D}\{Z(x) = 1\} - \Pr_{x' \leftarrow D'}\{Z(x') = 1\}|$$

a Z algoritmus megkülönböztető erejét, ahol $x \leftarrow D$ a D eloszlásból vett x mintát jelzi. Tehát Z algoritmus megkülönböztető ereje D és D' eloszlások vonatkozásában két feltételes valószínűség különbségének abszolút értéke, ahol az egyik feltételes valószínűség azon esemény valószínűsége, hogy Z algoritmus helyesen a D eloszlásra dönt feltéve, hogy a minta a D eloszlásból származik, a másik feltételes valószínűség azon esemény valószínűsége, hogy hibásan a D eloszlásra dönt miközben a minta a D' eloszlásból származik.

14.1. Definíció ((t, ε)-megkülönböztethetetlen). D és D' diszkrét valószínűség-eloszlás párt (t, ε)-megkülönböztethetetlennek nevezzük, ha

$$\max_Z |\Pr_{x \leftarrow D}\{Z(x) = 1\} - \Pr_{x' \leftarrow D'}\{Z(x') = 1\}| \leq \varepsilon, \quad (14.1)$$

ahol a maximumot a megkülönböztető algoritmusok felett vesszük.

A (14.1) maximumra az alábbiakban a $d_t(D, D')$ rövidebb jelölést használjuk. A (14.1) kifejezés alapján $0 \leq d_t(D, D') \leq 1$, továbbá annál jobban megkülönböztethetőnek tartjuk D és D' eloszlás-párt, minél nagyobb a $d_t(D, D')$ érték. Vegyük észre, hogy (14.1) kifejezésben el is hagyhattuk volna az abszolút értéket: származtassunk ugyanis egy Z' algoritmust Z algoritmus (bináris) kimenete negálásával, ekkor

$$\begin{aligned} & \Pr_{x \leftarrow D}\{Z(x) = 1\} - \Pr_{x' \leftarrow D'}\{Z(x') = 1\} \\ &= -(\Pr_{x \leftarrow D}\{Z'(x) = 1\} - \Pr_{x' \leftarrow D'}\{Z'(x') = 1\}), \end{aligned}$$

tehát, ha egy algoritmusra a maximális megkülönböztető erő adódik, akkor ez az algoritmus vagy a fenti egyszerű származtatása nemnegatív különbséget ad.

Valószínűségi változókra is kimondjuk a megkülönböztetés definícióját. Legyen X illetve Y valószínűségi változó D illetve D' eloszlású, amelyek Z algoritmus általi megkülönböztetésén az alábbi kifejezést értjük:

$$|\Pr\{Z(X) = 1\} - \Pr\{Z(Y) = 1\}|.$$

Alkalmazásainkban Z algoritmus tipikusan egy támadót (támadást) modellez. Ekkor a megkülönböztethetlenség 14.1. definíciója a támadással szembeni ellenálló képességek, biztonságnak a mértékét adja. A t erőforrás a támadó erőforrásainak egy listáját adhatja meg: például számítási lépések száma, tárkapacitás, orákulum-kérések száma (orákulum-hozzáférésként modellezve például egy választott nyílt szövegű támadást).

Hasznosak lesznek a továbbiakban a $d_t(D, D')$ leképezés alábbi tulajdonságai:

1. Tulajdonság: $d_t(D, D')$ leképezés távolság.

Bizonyítás: Lássuk be, hogy fennáll az alábbi háromszög-egyenlőtlenség:

$$d_t(D, D'') \leq d_t(D, D') + d_t(D', D'') \quad (14.2)$$

teszőleges D, D', D'' eloszlásokra. A (14.1) definíciót felhasználva

$$\begin{aligned} d_t(D, D'') &= \max_Z |\Pr_{x \leftarrow D} \{Z(x) = 1\} - \Pr_{x'' \leftarrow D''} \{Z(x'') = 1\}| \\ &= \max_Z |\Pr_{x \leftarrow D} \{Z(x) = 1\} - \Pr_{x' \leftarrow D'} \{Z(x') = 1\} \\ &\quad + \Pr_{x' \leftarrow D'} \{Z(x') = 1\} - \Pr_{x'' \leftarrow D''} \{Z(x'') = 1\}| \\ &\leq \max_Z |\Pr_{x \leftarrow D} \{Z(x) = 1\} - \Pr_{x' \leftarrow D'} \{Z(x') = 1\}| \\ &\quad + \max_Z |\Pr_{x' \leftarrow D'} \{Z(x') = 1\} - \Pr_{x'' \leftarrow D''} \{Z(x'') = 1\}| \end{aligned}$$

□

Jelölje D és D' eloszlás eloszlásfüggvényét $P_D(x)$ illetve $P_{D'}(x)$. A matematikai statisztikából ismert statisztikai távolság (variációs távolság) definíciója:

$$V(D, D') = \frac{1}{2} \sum_x |P_D(x) - P_{D'}(x)|, \quad (14.3)$$

azaz a két eloszlás L_1 távolságának a fele. $t = \infty$ jelölje a korlátlan erőforrást. A (14.1) kifejezés szerinti távolság és a variációs távolság kapcsolatát mondja ki az alábbi tulajdonság.

2. Tulajdonság:

$$V(D, D') = d_\infty(D, D') \quad (14.4)$$

Bizonyítás: Először is vegyük észre, hogy

$$V(D, D') = P_D(S^*) - P_{D'}(S^*), \quad (14.5)$$

ahol $S^* = \{x : P_D(x) > P_{D'}(x)\}$ és $P(S) = \sum_{x \in S} P(x)$. Ezt felhasználva

$$\begin{aligned} d_\infty(D, D') &= \max_Z \left| \Pr_{x \leftarrow D} \{Z(x) = 1\} - \Pr_{x' \leftarrow D'} \{Z(x') = 1\} \right| \\ &= \max_Z \left| \sum_{x: Z(x)=1} [P_D(x) - P_{D'}(x)] \right| \\ &= \max_Z \sum_{x: Z(x)=1} [P_D(x) - P_{D'}(x)] \\ &= \sum_{x \in S^*} [P_D(x) - P_{D'}(x)] = V(D, D'), \end{aligned}$$

ahol az utolsó előtti sor az általánosság megszorítása nélkül adódik, mivel Z kimenete negálásával egy, csak ebben különböző algoritmus adható meg helyette, továbbá az utolsó egyenlőség S^* definíciója alapján látható. A korlátlan erőforrás azért kell, hogy az x minta S^* halmaiba tartozása eldönthető legyen tetszőleges D, D' eloszlás páros esetén (egy általános eloszlás tipikusan nem írható le polinomiális erőforrással, ti. megadásához fel kell sorolni $P_D(x)$ valószínűséget az összes x értékhez). \square

2. Tulajdonság következménye:

Tetszőleges véges t értékre

$$d_t(D, D') \leq V(D, D') \quad (14.6)$$

Bizonyítás: Az állítás a 2.Tulajdonság, valamint a nyilvánvaló $d_t(D, D') \leq d_\infty(D, D')$ egyenlőtlenség következménye. \square

A (14.6) egyenlőtlenség azt fejezi ki, hogy ha statisztikai távolságban két eloszlás közel van, akkor algoritmikus megkülönböztethetőség szempontjából is közel van. Megfordítva az állítás nem igaz: konstruálható eloszlások párja, amelyek algoritmikusan közeliak, ugyanakkor a statisztikai távolságuk nagy.

A 14.1. definíció egy minta alapján történő döntésre vonatkozik. A definíció kiterjeszhető $m \geq 2$ minta alapján történő megkülönböztetés esetére. Tekintsük először az $m = 2$ esetet. Jelöljön $D_1 \times D_2$ kétdimenziós eloszlást, amelynek D_1, D_2 peremeloszlásai függetlenek, azaz

$$P_{D_1 \times D_2}(x_1, x_2) = P_{D_1}(x_1) \cdot P_{D_2}(x_2), \forall x_1, x_2.$$

3. Tulajdonság:

$$d_t(D \times D, D' \times D') \leq 2d_t(D, D') \quad (14.7)$$

Bizonyítás: Először lássuk be a következő segéd-egyenlőtlenséget:

$$d_t(D \times D, D \times D') \leq d_t(D, D').$$

Ugyanis, ha létezik Z algoritmus t erőforrással, amely d előnnyel megkülönbözteti $D \times D$ és $D \times D'$ eloszlásokat, akkor létezik Z' algoritmus, amely ugyancsak t erőforrással és ugyancsak d előnnyel megkülönbözteti D és D' eloszlásokat: ha Z' bemenete y , akkor Z' futtatja Z algoritmust (r, y) , $r \leftarrow D$ bemenettel. A háromszög-egyenlőtlenség és ezen segédegyenlőtlenség alkalmazásával adódik a (14.7) állítás:

$$d_t(D \times D, D' \times D') \leq d_t(D \times D, D \times D') + d_t(D \times D', D' \times D') \leq 2d_t(D, D').$$

□

14.2. Polinomiális időben megkülönböztethetőség

A 14.1. definíciót terjesszük ki eloszlások $D = \{D_n\}_{n \in N}$, $D' = \{D'_n\}_{n \in N}$ végtelen halmaza pájjára. A továbbiakban a tömör szóhasználat kedvéért indexelt elemekből álló végtelen halmazt együttesnek (pl. eloszlások együttese, valószínűségi változók együttese) nevezünk. Legyen X_n illetve Y_n valószínűsségi változó D_n illetve D'_n eloszlású. Speciális szerepet játszó valószínűsségi változó együttes $U = \{U_n\}_{n \in N}$, ahol U_n egyenletes eloszlású valószínűsségi változó.

Tipikus alkalmazásban ezen valószínűsségi változók értékkészlete n bites sorozatok halmaza, ahol n például egy biztonsági paraméter (pl. kulcs, kimenet/bemenet blokk) mérete. Az aszimptotikus vizsgálatoknál arra vagyunk kíváncsiak, hogy az n paraméter értékének növekedésével a számszerűsített biztonság hogyan növekszik. A 14.1. definícióban mind a t erőforráskorlát,

mind pedig a megkülönböztetés ε mértéke rögzítettek voltak. Most a Z támadó erőforrására és a megkülönböztetés mértékére n paraméterrel változó megkötést teszünk:

14.2. Definíció (polinomiálisan megkülönböztethetetlen).

Valószínűségi változók két együttese, $X = \{X_n\}_{n \in N}$ és $Y = \{Y_n\}_{n \in N}$ polinomiális időben megkülönböztethetetlen, ha tetszőleges hatékony Z algoritmus, valamint tetszőleges $p(n)$ polinom és elegendően nagy n esetén

$$|\Pr\{Z(X_n) = 1\} - \Pr\{Z(Y_n) = 1\}| < \frac{1}{p(n)}. \quad (14.8)$$

A definíció megfordításaként azt mondhatjuk, hogy valószínűségi változók két együttese, $X = \{X_n\}_{n \in N}$ és $Y = \{Y_n\}_{n \in N}$ polinomiális időben megkülönböztethető, ha létezik hatékony Z algoritmus és létezik egy $p(n)$ polinom, hogy végtelen sok n értékére

$$|\Pr\{Z(X_n) = 1\} - \Pr\{Z(Y_n) = 1\}| \geq \frac{1}{p(n)}. \quad (14.9)$$

A 14.2. definíció egy minta alapján történő döntsére vonatkozik. Általánosítható polinomiális sok, $m = q(n)$ mintára, ahol $q(n)$ egy polinom. Az alábbi téTEL bizonyítását a hibrid bizonyítástechnika széles körben alkalmazott módszerével végezzük, s egyben ezen technikát is bemutatjuk.

14.3. Tétel. *Polinomiálisan sok minta alapján sem lehet megkülönböztetni két valószínűségi változó együttest, ha egy minta alapján nem lehet.*

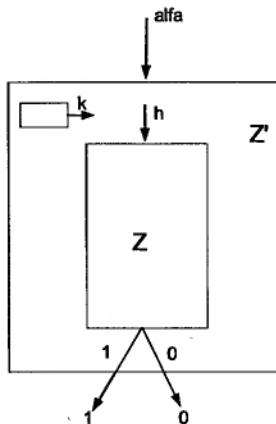
Bizonyítás: Z támadó bemenetként m független, azonos eloszlású mintát kap, amelyek D vagy D' eloszlásból származnak, így az m minta eloszlása vagy $D_1 \times D_2 \times \dots \times D_m$ vagy $D'_1 \times D'_2 \times \dots \times D'_m$. Indirekt bizonyításhoz tegyük fel, hogy Z sikeres ezen utóbbi eloszláspár megkülönböztetésében. Konstruálunk ennek alapján egy Z' támadót, amely Z felhasználásával megkülönbözteti D, D' eloszlás-párt, s ezzel ellentmondásra jutunk. Vezessük be a

$$H^k = D_1 \times \dots \times D_k \times D'_{k+1} \times \dots \times D'_m$$

ún. k -adik hibridet, $k = 0, 1, \dots, m$. A $H^m = D_1 \times D_2 \times \dots \times D_m$ és $H^0 = D'_1 \times D'_2 \times \dots \times D'_m$ hibridek az ún. extrém hibridek. Tegyük fel, hogy Z sikeres – az új fogalmaink szerint – az extrém hibridek megkülönböztetése feladatban. Legyen Z' támadó az alábbi:

1. Z' bemenete α , amely - Z' szemében - 1/2 valószínűsséggel D vagy D' .

2. Z' kisorsol egy k értéket a $\{0, 1, \dots, m-1\}$ halmazból.
 3. Z' a Z számára bemenetként a következő eloszlást küldi:
- $$h = D_1 \times \cdots \times D_k \times \alpha \times D'_{k+2} \cdots \times D'_m.$$
- Z' bemenetétől függően, $h = H^{k+1}$ (ha $\alpha = D$), illetve $h = H^k$ (ha $\alpha = D'$).
4. Z' kimenete egyezzen meg Z kimenetével.



14.1. ábra. Redukció

Következésképp

$$\Pr\{Z'(D) = 1\} = \frac{1}{m} \sum_{k=0}^{m-1} \Pr\{Z(H^{k+1}) = 1\},$$

$$\Pr\{Z'(D') = 1\} = \frac{1}{m} \sum_{k=0}^{m-1} \Pr\{Z(H^k) = 1\},$$

így

$$\begin{aligned}
 & |\Pr\{Z'(D) = 1\} - \Pr\{Z'(D') = 1\}| = \\
 & = \frac{1}{m} \left| \sum_{k=0}^{m-1} \Pr\{Z(H^{k+1}) = 1\} - \sum_{k=0}^{m-1} \Pr\{Z(H^k) = 1\} \right| \\
 & = \frac{1}{m} |\Pr\{Z(H^m) = 1\} - \Pr\{Z(H^0) = 1\}| \\
 & \geq \frac{1}{m} \frac{1}{p(n)} = \frac{1}{p'(n)},
 \end{aligned}$$

ahol $p'(n) = q(n)p(n)$, azaz D és D' eloszlásokat egy minta alapján nem elhanyagolható mértékben meg tudnánk különböztetni, amivel ellentmondásra jutottunk. \square

Az elhanyagolhatóság definíciójának egyik értelmét az adja, hogy hatékony algoritmussal csak elhanyagolható mértékben megkülönböztethető valószínűségi változó együtteseket nem tudunk megkülönböztetni egymástól. Ugyanakkor, amennyiben két eloszlás megkülönböztethető, elegendő számú minta vételével tetszőlegesen pontos megkülönböztetés hajtható végre.

14.3. Feladatok

14.1. Feladat. Tekintse az alábbi eloszlás-párt:

$$D(x) = \begin{cases} \frac{2}{3}2^{-n} & \text{ha } 0 \leq x < 2^{n-1} \\ \frac{4}{3}2^{-n} & \text{ha } 2^{n-1} \leq x < 2^n \end{cases} \quad \text{és}$$

$$D'(x) = \begin{cases} \frac{4}{3}2^{-n} & \text{ha } 0 \leq x < 2^{n-1} \\ \frac{2}{3}2^{-n} & \text{ha } 2^{n-1} \leq x < 2^n. \end{cases}$$

1. Adjon megkülönböztető algoritmust!
2. Lehet-e $1/6$ -odnál jobb a megkülönböztető képesség?

14.2. Feladat. Szeretnénk megkülönböztetni egy igazi pénzérmét egy külssőre teljesen azonos hamis pénzérmétől. Azt tudjuk, hogy a hamis érme feldobáskor $1/2 + 1/p(n)$ valószínűséggel esik fej oldalára, ahol $p(n)$ egy polinom. Kezünkbe adják a két érme közül az egyiket.

Vegyük észre a kapcsolatot a 14.1. definícióval: $D = \{1/2 + 1/p(n), 1/2 - 1/p(n)\}$, $D' = \{1/2, 1/2\}$. Tekintsük a következő megkülönböztető algoritmust: a fejdobások relatív gyakoriságát számoljuk, s az algoritmus kiemelte 1 (hamis pénzre dönt), ha a relatív gyakoriság nagyobb, mint $1/2 + 1/(2p(n))$. Az analízishez a Chernoff-féle felső becslést alkalmazzuk. Hogyan alakul n -ben aszimptotikusan a megkülönböztetés valószínűsége polinomiális számú dobás esetén?

14.3. Feladat. Egy F eloszlású valószínűségi változót egy t' erőforrású $L : \{0,1\}^* \rightarrow \{0,1\}^*$ algoritmussal leképezünk, s az eredmény eloszlását jelölje $L(F)$. Mutassuk meg, hogy $d_t(L(D), L(D')) \leq d_{t+t'}(D, D')$, ahol $t + t'$ az egyesített erőforrást jelenti!

14.4. Feladat. Jelölje $D|E$ a D eloszlású valószínűsségi változó E eseményre, mint feltételre vett feltételes eloszlását. Igazoljuk a következő egyenlőtlenséget:

$$d_t(D, D') \leq d_t(D|E, D'|E) + P(\bar{E}),$$

ahol \bar{E} az E komplemente!

14.5. Feladat*. Nagy variációs távolságú eloszlások is lehetnek hatékony algoritmussal megkülönböztethetetlenek. Legyen D egyenletes eloszlás a $\{0, 1\}^n$ halmazon, továbbá legyen D' eloszlás koncentrált: egyenletes eloszlás a $\{0, 1\}^n$ egy $2^{n/2}$ méretű részhalmazán. A variációs távolság $1 - 2^{n/2}$, azaz közel maximális. Igazoljuk – ennek ellenére – a polinomiális-idejű megkülönböztethetetlenséget!

14.6. Feladat. Valószínűsségi változók két együttese $X = \{X_n\}_{n \in N}$ és $Y = \{Y_n\}_{n \in N}$ polinomiális időben megkülönböztethetetlen. Legyen L egy hatékony algoritmus. Mutassuk meg, hogy $X' = \{L(X_n)\}_{n \in N}$ és $Y' = \{L(Y_n)\}_{n \in N}$ valószínűsségi változó együttesek is polinomiális időben megkülönböztethetetlenek!

14.7. Feladat. Valószínűsségi változók két sorozata, $X = \{X_n\}_{n \in N}$ és $Y = \{Y_n\}_{n \in N}$ legyen variációs távolságban közel, s legyen $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ tetszőleges függvény. Mutassuk meg, hogy ekkor az $X' = \{f(X_n)\}_{n \in N}$ és $Y' = \{f(Y_n)\}_{n \in N}$ valószínűsségi változó sorozatok is közeliek variációs távolságban!

14.8. Feladat. Mutassuk meg, hogy a variációs távolság egy ekvivalens definíciója az alábbi: $\{X_n\}_{n \in N}$ és $\{Y_n\}_{n \in N}$ valószínűsségi változó sorozat akkor és csak akkor közel variációs távolságban, ha tetszőleges $S \subseteq \{0, 1\}^*$ esetén

$$\Delta_S(n) = |P(X_n \in S) - P(Y_n \in S)|$$

elhanyagolhatóan kicsi n -ben!

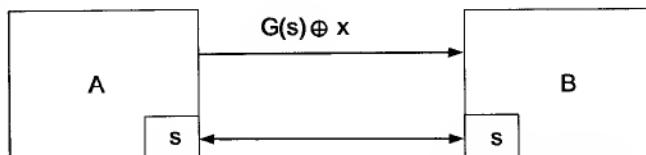
14.9. Feladat*. Terjessük ki a 3. Tulajdonságot $m > 2$ minta esetére!

15.

Álvéletlen-generátor

Kriptográfiai alkalmazások (sok résztvevő protokollok) sok véletlen bitet igényelhetnek, ugyanakkor a valódi véletlen bitek előállítása költséges, s legtöbbször speciális hardvert igényel. Felmerül a kérdés, hogy lehetséges-e viszonylag kevés valódi véletlen elemet „szaporítani” hatékony algoritmussal olyan módon, hogy az eredményül kapott álvéletlen biteket a valódi véletlentől ne lehessen megkülönböztetni.

Egy G biztonságos álvéletlen-generátor helyettesítheti a one-time pad véletlen kulcsfolyamát (15.1. ábra). Ennek az az előnye, hogy egy „szokásos” szimmetrikus kulcs méretű, valódi véletlen bitsorozatot (s) kell csak titokban megosztani a két félnek, a rejtjelezendő (tömörített) üzenetek bithosszának megfelelő számú valódi bit helyett. Hátránya azonban az, hogy a létező biztonságos álvéletlen-generátorok sok számítást kell elvégezzenek egy-egy álvéletlen bit generálásakor, míg egy valódi véletlen kulcsú one-time pad esetén ez a művelet egyszerűen a következő véletlen bit tárba való olvasásának idejét jelenti.



15.1. ábra. One-time pad rejtjelezés álvéletlen kulcsfolyammal

15.1. Álvéletlen-generátor és az egyirányú függvény

15.1.1. Definíció (valószínűségi változók álvéletlen együttese).

Valószínűségi változók $X = \{X_n\}_{n \in N}$ együttese álvéletlen, ha polinomiális időben megkülönböztethetően az egyenletes eloszlású valószínűségi változók $U = \{U_n\}_{n \in N}$ együttesétől.

Ha egy együttes – a fenti definíció értelmében – álvéletlen, akkor az ekvivalens használható valódi véletlen sorozat helyettesítőjeként tetszőleges hatékony T algoritmusban. Ha ugyanis megkülönböztethető lenne a T algoritmus futása a két esetben, akkor a T algoritmus maga, egyben egy hatékony megkülönböztető algoritmus lenne, ami ellentmondás.

A továbbiakban az X_n valószínűségi változó az $n \in N$ indextől függő hosszságú bináris vektor: $X_n = (x_{n,1}, x_{n,2}, \dots, x_{n,m(n)})$.

15.2. Definíció (predikálhatóság). *Valószínűségi változók együttese akkor predikálható, ha létezik hatékony algoritmus, amely X_n valószínűségi vektorváltozó tetszőleges j bites prefíxe alapján a $(j+1)$ -edik bitet nem elhangolható valószínűséggel predikálni képes, $1 \leq j < m(n)$.*

15.3. Tétel. *Valószínűségi változók együttese akkor és csak akkor álvéletlen, ha nem predikálható.*

Bizonyítás: Lásd a 15.1. feladatot. □

15.4. Definíció (álvéletlen-generátor). *A $G : \{0,1\}^* \rightarrow \{0,1\}^*$ álvéletlen-generátor (PRG, PseudoRandom Generator) egy determinisztikus, polinomiális idejű algoritmus, amely teljesíti a következő két feltételt:*

1. *hossznövelés: létezik egy $l : N \rightarrow N$ függvény, amelyre $l(n) > n$ tetszőleges $n \in N$ értékre, továbbá $|G(s)| = l(|s|)$, ahol $s \in \{0,1\}^*$;*
2. *álvéletlenség: valószínűségi változók $\{G(U_n)\}_{n \in N}$ együttese álvéletlen.*

A generátor s bemenetét szokás a generálás magjának nevezni. A konstrukció első lépése, az egy bittel növelés, azaz az $l(n) = n + 1$ eset igényli a lényegi ötletet. Innen, ugyan nem triviális, de alapvetően egy redukciós technikai lépéssel lehet a tetszőleges polinomiális hosszra növelést elvégezni.

A konstrukció egyirányú függvényre épül. Pontosabban, a bemutatandó konstrukció egyirányú permutációt alapul. Ez a „legegyszerűbb” eset, ami ugyanakkor a tetszőleges egyirányú függvényre épülő konstrukció lényeges ötleteit is tartalmazza, az egyébként bonyolult technikai részletek minimumon tartása mellett. Ugyancsak nem mutatjuk be annak a fontos egzisztencia téTELnek a teljes bizonyítását, amely kimondja, hogy:

15.5. Tétel. Álvéletlen-generátor akkor és csak akkor létezik, ha létezik egy-irányú függvény.

Az egyik irányban azonban bebizonyítjuk a következő állítást:

15.6. Tétel. Legyen G egy álvéletlen-generátor, $l(n) = 2n$ hossznövelő függvénnyel. Ekkor az $f(x, y) = G(x)$, $|x| = |y|$ függvény (erősen) egyirányú.

Bizonyítás: Először is megállapíthatjuk, hogy az álvéletlen-generátor, valamint az f függvény definíciója alapján az f függvény hatékonyan kiszámítható.

Indirekt bizonyításul tegyük fel, hogy f nem egyirányú, azaz létezik egy Z algoritmus, valamint egy $p(n)$ polinom, hogy Z algoritmus $f(U_{2n})$ bemenettel végtelen sok n esetén, legalább $1/p(n)$ sikervalószínűsgéggel képes ősképet kiszámítani, azaz $\Pr\{f(Z(f(U_{2n}))) = f(U_{2n})\} \geq 1/p(n)$ ezen n értékek esetén. Ennek alapján redukciós technikával konstruálunk egy Z' mekgülönböztető algoritmust, amely ezen n értékeknél mekgülönbözteti U_{2n} és $G(U_n)$ valószínűsségi változókat. Tekintsük a következő algoritmust:

15.1. Algoritmus.

1. Z' bemenete $\alpha \in \{U_{2n}, G(U_n)\}$,
2. Z' futtatja Z algoritmust α bemenettel:
 - ha $f(Z(\alpha)) = \alpha$, akkor Z' kimenete 1,
 - ha $f(Z(\alpha)) \neq \alpha$, akkor Z' kimenete 0.

Mivel $f(U_{2n}) = f(U_n, U_n) = G(U_n)$, ezért – indirekt feltételezésünk alapján – valamely $p(n)$ polinomra és végtelen sok n értékre $\Pr\{Z'(G(U_n)) = 1\} \geq 1/p(n)$. Az f függvény a $2n$ bites sorozatok terében egy legfeljebb 2^n méretű részhalmazra képez. Innen következik, hogy tetszőleges invertáló algoritmus, amelynek a bemenetére U_{2n} érkezik ($\alpha = U_{2n}$ eset), legfeljebb 2^{-n} valószínűsséggel lehet sikeres, azaz akkor, ha U_{2n} realizációja a $\{G(x), x \in \{0, 1\}^n\}$ halmazba esik. Összegezve az eddigieket, valamely $p(n)$ polinomra és végtelen sok n értékre kapjuk, hogy

$$|\Pr\{Z'(G(U_n)) = 1\} - \Pr\{Z'(U_{2n}) = 1\}| \geq 1/p(n) - 2^{-n} \geq 1/(2p(n))$$

ami ellentmond annak, hogy G álvéletlen-generátor. □

15.2. Álvéletlen-generátor konstrukció

Az 1 bittel hossznövelő álvéletlen-generátor konstrukciója a keménybittel növeli meg egy egyirányú permutáció kimenetét. Az álvéletlen-generátort az alábbi téTEL mondja ki:

15.7. TÉTEL (Blum–Micali–Yao-konstrukció). *Legyen az f erősen egyirányú permutáció. Legyen b az f keménybit leképezése. A $G(s) = [f(s), b(s)]$ leképezéssel álvéletlen-generátort kapunk.*

Bizonyítás: Ismét a redukciós technikát alkalmazzuk. Indirekt feltételezéssel tegyük fel, hogy létezik Z megkülönböztető algoritmus, azaz létezik $p(n)$ polinom, hogy végtelen sok n értékre

$$|\Pr\{Z(G(U_n)) = 1\} - \Pr\{Z(U_{n+1}) = 1\}| \geq 1/p(n).$$

Konstruálunk egy olyan hatékony Z' algoritmust, amely sikeresen képes predikálni a $b(U_n)$ keménybitet $f(U_n)$ alapján. Innen a 15.3. téTEL figyelembe vételével már igazolt a téTEL.

15.2. Algoritmus.

1. Z' bemenete $y = f(U_n)$,
2. Z' sorsol egy véletlen σ bitet,
3. Z' futtatja Z algoritmust $[y, \sigma]$ bemenettel:
 - ha $Z([y, \sigma]) = 1$, akkor Z' kimenete σ
 - ha $Z([y, \sigma]) = 0$, akkor Z' kimenete $\bar{\sigma}$.

Az alábbiakban valószínűségszámítási lépésekkel el fogunk jutni a

$$\Pr\{Z'(f(U_n)) = b(U_n)\} \geq 1/2 + 1/p(n)$$

alsó becslésig, ami a keménybit nem elhanyagolhatóan sikeres becslését, s a konstrukció alapján egyúttal a generátor kimenetének predikálhatóságát is jelentené. A főbb lépések a következők:

$$\begin{aligned} \Pr\{Z'(f(U_n)) = b(U_n)\} &= \\ &= \Pr\{\sigma = b(U_n)\} \cdot \Pr\{Z'(f(U_n)) = b(U_n) \mid \sigma = b(U_n)\} \\ &\quad + \Pr\{\sigma \neq b(U_n)\} \cdot \Pr\{Z'(f(U_n)) = b(U_n) \mid \sigma \neq b(U_n)\} \\ &\stackrel{(2)}{=} 1/2 \cdot [\Pr\{Z[f(U_n), b(U_n)] = 1\} + 1 - \Pr\{Z[f(U_n), \bar{b}(U_n)] = 1\}] \\ &\stackrel{(3)}{=} 1/2 + [\Pr\{Z(G(U_n)) = 1\} - \Pr\{Z(U_{n+1}) = 1\}] \\ &\geq 1/2 + 1/p(n), \end{aligned}$$

ahol $\stackrel{(2)}{=}$ illetve $\stackrel{(3)}{=}$ egyenlőségek magyarázata a következő:

A $\stackrel{(2)}{=}$ egyenlőségnél a

$$\begin{aligned} \Pr\{Z'(f(U_n)) = b(U_n) | \sigma = b(U_n)\} &= \\ &= \Pr\{Z'(f(U_n)) = \sigma | \sigma = b(U_n)\} \\ &\stackrel{(4)}{=} \Pr\{Z[f(U_n), \sigma] = 1 | \sigma = b(U_n)\} \\ &= \Pr\{Z[f(U_n), b(U_n)] = 1 | \sigma = b(U_n)\} \\ &= \Pr\{Z[f(U_n), b(U_n)] = 1\}, \end{aligned}$$

ahol $\stackrel{(4)}{=}$ egyenlőség a Z' algoritmus konstrukciója alapján látható: $\{Z(y, \sigma) = 1\} = \{Z'(y) = \sigma\}$, $\{Z(y, \sigma) = 0\} = \{Z'(y) = \bar{\sigma}\}$, illetve

$$\begin{aligned} \Pr\{Z'(f(U_n)) = b(U_n) | \sigma \neq b(U_n)\} &= \\ &= \Pr\{Z'(f(U_n)) = \bar{\sigma} | \sigma \neq b(U_n)\} \\ &= \Pr\{Z[f(U_n), \sigma] = 0 | \sigma = \bar{b}(U_n)\} \\ &= \Pr\{Z[f(U_n), \bar{b}(U_n)] = 0 | \sigma = \bar{b}(U_n)\} \\ &= \Pr\{Z[f(U_n), \bar{b}(U_n)] = 0\} \\ &= 1 - \Pr\{Z[f(U_n), \bar{b}(U_n)] = 1\}. \end{aligned}$$

A $\stackrel{(3)}{=}$ egyenlőségnél a

$$\begin{aligned} \Pr\{Z(U_{n+1}) = 1\} &= \\ &= 1/2 [\Pr\{Z[f(U_n), b(U_n)] = 1\} + \Pr\{Z[f(U_n), \bar{b}(U_n)] = 1\}] \end{aligned}$$

összefüggést használtuk, valamint figyelembe vettük, hogy f permutáció, s így U_{n+1} és $[f(U_n), U_1]$ valószínűségi változók azonos eloszlásúak. \square

15.3. Feladatok

15.1. Feladat. Igazoljuk, hogy

1. az álvéletlenség implikálja a predikálhatatlanságot, és
2. a predikálhatatlanság implikálja az álvéletlenséget!

15.2. Feladat. Igazoljuk, hogy az álvéletlen-generátor kimenete nem lehet valódi véletlen!

15.3. Feladat. Mutassunk konstrukciót arra az esetre, amikor

1. egy PRG (álvétlen-generátor) egy-egy értelmű leképezés,
2. egy PRG különböző bemeneti értéket (különböző magot) azonos kimenetbe (álvétlen sorozatba) képez le!

15.4. Feladat. Tegyük fel, hogy $\{G(U_n)\}_{n \in N}$ és $\{U_{h(n)}\}_{n \in N}$ algoritmikusan megkülönböztethetetlenek. Bizonyítsuk be, hogy $G(U_n)$ bithossza elhanyagolhatóan kicsi valószínűséggel nem $h(n)$, azaz $\Pr\{|G(U_n)| \neq h(n)\}$ valószínűség elhanyagolható (n -ben)!

15.5. Feladat. A keménybittel hossznövelő PRG konstrukciójánál az f egyirányú függvény $f(s)$ kimenete végéhez toldjuk a $b(s)$ keménybitet, azaz $[f(s), b(s)]$ a PRG kimenete. Igazoljuk, hogy ha $[b(s), f(s)]$ a generátor kimenete, szintén PRG-t kapunk!

15.6. Feladat. Legyen G egy PRG, továbbá h egy polinomiális időben számítható permutáció. Igazoljuk, hogy ekkor $G'(s) = G(h(s))$, illetve $G''(s) = h(G(s))$ is PRG!

15.7. Feladat. Legyen G egy PRG $l(n) = n + 1$ hossznöveléssel. Legyen $G'(s)$ az a leképezés, amelyet G $p(|s|)$ számú iteratív alkalmazásával nyertünk:

$$\begin{aligned} G'(s) &= G^{p(|s|)}(s) \\ G^{(0)}(s) &= s, \quad G^{(i+1)}(s) = G(C(G^{(i)}(s))), \end{aligned}$$

ahol C csonkoló leképezés az $n + 1$ bites argumentuma n bitjét veszi rögzített n pozícióból (azaz 1 bitet elhagyunk). Igazoljuk, hogy $G'(s)$ is PRG!

15.8. Feladat. Legyen G egy PRG, amelyre $|G(w)| = |w|^2, \forall w \in \{0, 1\}^*$ (azaz négyzetesen hossznövelő). Használjuk ezen PRG-t kulcsfolyam-generátorként szimmetrikus kulcsú rejtjelezésben a szokásos mod 2 összegző módon. Lehetséges-e, hogy egy Z támadó felmutat m_0, m_1 üzenetpárt, ahol az üzenetek n^2 bit hosszúak, s amelyre

$$|\Pr\{Z(G(U_n) \oplus m_0) = 1\} - \Pr\{Z(G(U_n) \oplus m_1) = 1\}| \geq 1/q(n),$$

azaz a kapcsolatos rejtjeles szövegek megkülönböztethetők?

15.9. Feladat. Igazoljuk, hogy egyirányú függvény akkor és csak akkor létezik, ha létezik randomizált polinom idejű algoritmusok $B(1^n)$ és $C(1^n)$ párja, amelyek n -bites kimenetre képeznek, továbbá kimeneti eloszlásuk polinomiálisan megkülönböztethetetlen, valamint kimeneti halmazaik metszete legfeljebb $2^{n/2}$ elemet tartalmaz! ($B(1^n)$, $C(1^n)$ jelölés értelme a következő: B és C algoritmusok 1^n bemenetet kapnak, belül véletlen elemeket sorsolnak, számításokat végeznek és kiadnak egy-egy kimenetet, azaz kimenetük valószínűségi változó. Amikor $B(1^n)$, $C(1^n)$ eloszlásról beszélünk, akkor ezen valószínűségi változók eloszlását értjük alatta.)

Az állítás informálisabb megfogalmazása: Egyirányú függvény akkor és csak akkor létezik, ha létezik két hatékonyan mintavételezhető eloszlás, lényegében diszjunkt tartóval, amelyek polinomiálisan nem megkülönböztethetők egymástól.

16.

Álvéletlen függvény, álvéletlen permutáció

A véletlen függvény és a véletlen permutáció végtelen erőforrás melletti ideális primitívek (lenyomatképzés, szimmetrikus kulcsú blokkrejtjelezés) matematikai modelljei. Ezen ideális objektumok hatékony változatai az álvéletlen függvény és az álvéletlen permutáció, amelyeket polinom erőforrás-korlát mellett tudunk generálni, ugyanakkor egy ugyancsak polinom erőforrás-korlátú támadó nem tud az ideálistól megkülönböztetni. Ezen álvéletlen függvények a valós (implementálható) primitívek erőforrás-megszorítás melletti idealizált modelljei.

Szimmetrikus kulcsú rejtjelezés esetén a két fél megegyezik egy titkos kulcsban, amellyel transzformációk egy nagy halmazából egyet választanak. Ezen megoldás előnye, hogy egy véletlenül választott kisméretű közös titkot kell csak megosztaniuk. Ideálisan esetben a két fél egy közös (egyenletes eloszlású) véletlen függvényt birtokolna. A véletlen függvény előnye, hogy ideálisan biztonságos algoritmusokat lehetne rá építeni, anélkül, hogy egy támadó erőforrására korlátozást kellene tennünk, továbbá – technikailag – viszonylag könnyen tudnánk felső becslést adni a támadó sikervalószínűségére. Sajnos ez nem valósítható meg a gyakorlatban, mivel bemenet-kimenet dimenzió megkötésnek eleget tevő összes függvények halmaza olyan nagy, hogy még csak meg sem tudnánk – hatékonyan – címezni az elemeit.

Ugyanakkor bizonyítástechnikai elemként használhatjuk a véletlen függvényt a következőképp: orákulumként működtetjük, azaz ha egy kérést küldünk ezen orákulumnak, az azonnal visszaadja válaszként a kérés véletlen függvény szerinti leképezését. Ha ezen orákulum-modellben tudunk konstruálni egy biztonságos, és ezen orákulum-modellben polinom idejű algorit-

must, akkor helyettesítve az orákulumot egy álvéletlen függvényel, az algoritmusunk biztonsága nem változik, feltéve, hogy a támadó erőforrását ezen álvéletlen függvény definíciója szerint korlátozzuk.

Az álvéletlen függvény egy másik szempontú megközelítése a közvetlen címzésű véletlen blokk generálás, szemben a soros elérésű álvéletlen-generátorral. Az álvéletlen-generátorral egy n bites valódi véletlen magot biztonságosan (tőle hatékony algoritmussal megkülönböztethetetlenül) ki tudtunk nyújtani n -ben polinom hosszúságú álvéletlen sorozatra. Ez a fajta álvéletlen-generálás azonban nem illeszkedik olyan felhasználásokhoz, amelynél egy algoritmus futása során több ponton is ugyanazon véletlen bitsorozatot szeretnénk megcímezni. Ehhez egy álvéletlen függvény lenne jó, amely azonos bemenetre azonos kimenetet ad, míg különböző bemenetekre különböző álvéletlen kimeneteket. Elvileg ez a feladat megoldható lenne PRG felhasználásával is, csak hogy ehhez (a soros generálás miatt) tárolni kellene a teljes álvéletlen sorozatot, hogy annak a címzéssel meghatározott szeletét vehessük. Ugyanezen feladatot véletlen függvényel az n bites véletlen mag tárolási költségén meg tudjuk oldani.

16.1. Véletlen függvény, álvéletlen függvény

16.1. Definíció (véletlen függvény). *F_n véletlen függvény egy olyan valószínűségi változó, amelynek minden lehetséges értéke egy n bitból n bitbe képező függvény. Ha ezen valószínűségi változó eloszlása egyenletes, akkor egyenletes eloszlású véletlen függvényről beszélünk, s ekkor a H_n jelölést használjuk.*

Szokásosan, véletlen függvény alatt az egyenletes eloszlású véletlen függvényt (H_n) értjük.

16.2. Definíció (véletlen függvények együttese). *Az F_n véletlen függvények $F = \{F_n\}_{n \in N}$ halmazát véletlen függvények együttesének nevezzük.*

A továbbiakban $H = \{H_n\}_{n \in N}$ jelöli az egyenletes eloszlású véletlen függvények együttesét.

Álvéletlen függvények együttesének tagjai dimenzióban megegyeznek a megfelelő egyenletes eloszlású véletlen függvények együttesének tagjaival, de hatékonyan generálhatók, s az együttes hatékony algoritmussal attól nem különböztethető meg. Formálisan:

16.3. Definíció (álváleletlen függvények együttese). $F = \{F_n\}_{n \in N}$ álváleletlen függvények együttese, ha tetszőleges randomizált, polinom idejű Z orákulumos algoritmus, tetszőleges $p(n)$ polinom és elegendően nagy n esetén

$$|\Pr\{Z^{F_n}(1^n) = 1\} - \Pr\{Z^{H_n}(1^n) = 1\}| < \frac{1}{p(n)}. \quad (16.1)$$

A jelölés egyszerűsítése kedvéért az álváleletlen függvény jelölésében meg-tartottuk az F_n jelölést, ugyanis az álváleletlen függvény egy speciális véletlen függvény (ti. a véletlen függvény definíciójában nem tettünk erőforrás-megkötést).

A véletlen permutáció, a véletlen permutációk együttese, valamint az álváleletlen permutációk együttese a megfelelő, függvényre szóló definíciók – értelemszerű – speciális esetei. Két vonatkozásban azonban finomodnak a kapcsolatos definíciók: a hatékony kiszámítás függvény-együttesekre vonatkozó definíciója mellett a permutáció-együttesek esetén a hatékony kiszámítás és hatékony invertálás definíciójára bővíthető, amely egy erősebb – de a gyakorlati szempontból természetes – megkötést jelent. További modell-finomítással jutunk az erősen álváleletlen permutáció-együttes fogalmához, amelynek definíciója annyiban bővül, hogy az orákulumos algoritmus az invertáló orákulumhoz is hozzáfér, mégsem képes megkülönböztetésre, s ennek megfelelően (16.1) egyenlőtlenség az alábbi alakot ölti:

$$|\Pr\{Z^{F_n, F_n^{-1}}(1^n) = 1\} - \Pr\{Z^{H_n, H_n^{-1}}(1^n) = 1\}| < \frac{1}{p(n)}. \quad (16.2)$$

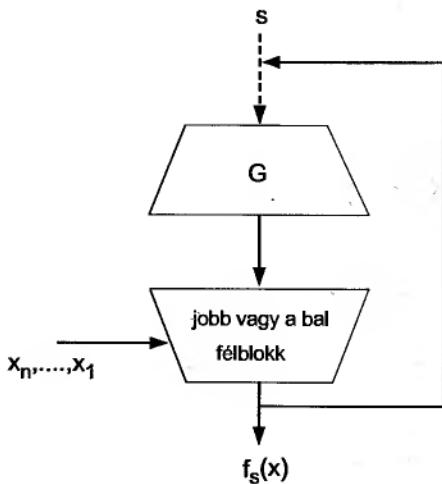
Az álváleletlen függvényekre röviden a PRF (PseudoRandom Function), az álváleletlen permutációkra röviden a PRP (PseudoRandom Permutation) megnevezést is használni fogjuk.

16.2. Álváleletlen függvény konstrukció

Legyen G egy kétszeresen hossznövelő ($l(n) = 2n$) álváleletlen-generátor. Az $x = (x_1, \dots, x_n) \in \{0, 1\}^n$ argumentum esetén definiáljuk az $f_s : \{0, 1\}^n \rightarrow \{0, 1\}^n$ függvény értékét az alábbi módon (16.1. ábra):

$$f_s(x_1, \dots, x_n) = G_{x_n}(\dots(G_{x_2}(G_{x_1}(s))\dots)). \quad (16.3)$$

ahol, $s \in \{0, 1\}^n$, továbbá $G_0(.)$ és $G_1(.)$ a G kimenetének bal élletve jobb fele. Azaz az x bemenet bitjeinek megfelelően, G kimenetének a jobb vagy a bal felét választjuk, s a választott fél lesz G következő bemenete.



16.1. ábra. PRF konstrukció

16.4. Tétel. A (16.3) konstrukció szerinti f függvény egyenletes eloszlás szerint véletlenszerűen választott s bemenettel $F = \{F_n\}_{n \in N}$ álvéletlen függvényegyüttést eredményez, ahol $F_n = f_s$.

Bizonyítás: Először is megállapíthatjuk, hogy F hatékony kiszámítható. F álvéletlenségének bizonyításához a hibrid technikát alkalmazzuk:

A H^k , $0 \leq k \leq n$ hibrideket a következőképp definiáljuk:

$$\begin{aligned} k = 0 : H^0 &= F_n \\ k = n : H^n &= H_n \\ 1 \leq k < n : H^k &= g_{U_n^{(1)}, \dots, U_n^{(2^k)}}, \end{aligned}$$

ahol

$$g_{z_1, \dots, z_{2^k}}(x_1, \dots, x_n) = G_{x_n}(\dots(G_{x_{k+2}}(G_{x_{k+1}}(z_{h(x_k, \dots, x_1)}))\dots),$$

továbbá $h : \{0,1\}^k \rightarrow \{1, \dots, 2^k\}$ leképezés a k bites bemenetet egy egész szám bináris ábrázolásának tekinti, s azon egész szám eggyel növelt értékébe képezi le. A 0-dik hibrid F_n , azaz a (16.3) szerinti álvéletlen függvény, az n -edik hibrid az egyenletes eloszlású véletlen függvény, H_n . A $1 \leq$

$k < n$ esetben az x argumentum x_1, \dots, x_k bitjei alapján h leképezéssel kiválasztjuk az $U_n^{(i)}$, n biten egyenletes eloszlású valószínűségi változót, ahol $h(x_1, \dots, x_k) = i$. Ez a magja annak a (16.3) formulához hasonló struktúrájú g leképezésnek, ahol a G transzformációt $(n - k)$ -szor alkalmazzuk.

A levezetés logikája a következő lesz: a G álvéletlen-generátor kimenetének egyenletes eloszlástól való megkülönböztetését redukáljuk a szomszédos hibridek megkülönböztetésére. Továbbá belátjuk, hogy az extrém hibridek ($k = 0, k = n$) megkülönböztetésének képességeből a (k -ban) szomszédos hibridek megkülönböztetésének képessége következik. Ha tehát – indirekt feltételezés szerint – a (16.3) konstrukció nem vezet álvéletlen együttesre, s így az extrém hibridek megkülönböztethetők, akkor a szomszédos hibridek is megkülönböztethetők, s ezáltal a szomszédos hibridekre redukált G álvéletlen-generátor kimenet megkülönböztethetőségének feladata is sikkerrel végrehajtható. Ezen utóbbi végkövetkeztetés viszont ellentmond tételeink feltételének, következésképp (16.3) konstrukció valóban egy álvéletlen együttesre vezet.

Tegyük fel tehát, hogy létezik egy randomizált, polinom-idejű Z orákulumos algoritmus és tetszőleges $p(n)$ polinom, hogy végtelen sok n -re

$$|\Pr\{Z^{F_n}(1^n) = 1\} - \Pr\{Z^{H_n}(1^n) = 1\}| > \frac{1}{p(n)}. \quad (16.4)$$

Legyen $t(n)$ polinom Z erőforrás korlátja. Következésképp $t(n)$ korlátot jelent a Z orákulum-kéréseinek darabszámára is. A Z álvéletlen függvény megkülönböztető algoritmusra redukáljuk az álvéletlen-generátor megkülönböztetési feladatot a következő módon:

16.1. Algoritmus.

1. Z' álvéletlen-generátor megkülönböztető algoritmus bemenetére $\alpha = (\alpha_1, \dots, \alpha_{t(n)})$, $\alpha_i \in \{0, 1\}^{2n}$ vektor érkezik, amely $\alpha^{(1)} = (G(U_n^{(1)}), \dots, G(U_n^{(t(n))}))$ vagy $\alpha^{(0)} = (U_{2n}^{(1)}, \dots, U_{2n}^{(t(n))})$ $1/2 - 1/2$ valószínűséggel.
2. Z' kisorsol egy $\kappa \in \{0, 1, \dots, n-1\}$ értéket véletlenszerűen.
3. Z' futtatja Z orákulumos algoritmust, s amikor az a j -edik orákulum-kérést, $q_j \in \{0, 1\}^n$ küldi, Z' a következő választ adja rá (szimulálja)

$$A_j = G_{q_{j,n-1}}(\dots(G_{q_{j,\kappa+2}}(G_{q_{j,\kappa+1}}(\alpha_{j,(q_{j,\kappa})})))\dots), \quad (16.5)$$

ahol $j \leq t(n)$, $q_j = (q_{j,0}, \dots, q_{j,n-1})$, továbbá $\alpha_j = (\alpha_{j,(0)}, \alpha_{j,(1)})$ az α_j két n -bites felére bontása.

4. Z' kimenete = Z kimenete.

(16.5) alapján látható, hogy ha $\alpha = \alpha^{(1)}$, akkor A_j eloszlása $H_n^K(q_{j,0}, \dots, q_{j,n-1})$ eloszlásával, míg ha $\alpha = \alpha^{(0)}$, akkor A_j eloszlása $H_n^{K+1}(q_{j,0}, \dots, q_{j,n-1})$ eloszlásával egyezik meg. Következésképpen a Z' bemenetétől független, Z nézőpontjából a (16.4) formula szerinti F_n helyén H_n^K hibrid vagy H_n^{K+1} hibrid áll, vagyis:

$$\Pr\{Z'[G(U_n^{(1)}), \dots, G(U_n^{(t(n))})] = 1 | \kappa = k\} = \Pr\{Z^{H_n^k}(1^n) = 1\}$$

$$\Pr\{Z'[U_{2n}^{(1)}, \dots, U_{2n}^{(t(n))}] = 1 | \kappa = k\} = \Pr\{Z^{H_n^{k+1}}(1^n) = 1\}.$$

A fentiekkel felhasználva

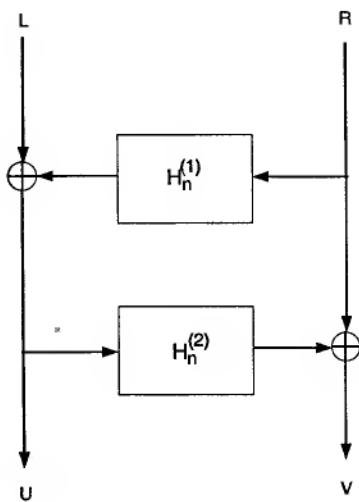
$$\begin{aligned} & |\Pr\{Z'[G(U_n^{(1)}), \dots, G(U_n^{(t(n))})] = 1\} - \Pr\{Z'[U_{2n}^{(1)}, \dots, U_{2n}^{(t(n))}] = 1\}| = \\ &= \left| \frac{1}{n} \sum_{k=0}^{n-1} \Pr\{Z'[G(U_n^{(1)}), \dots, G(U_n^{(t(n))})] = 1 | \kappa = k\} \right. \\ &\quad \left. - \frac{1}{n} \sum_{k=0}^{n-1} \Pr\{Z'[U_{2n}^{(1)}, \dots, U_{2n}^{(t(n))}] = 1 | \kappa = k\} \right| \\ &= \left| \frac{1}{n} \sum_{k=0}^{n-1} \Pr\{Z^{H_n^k}(1^n) = 1\} - \frac{1}{n} \sum_{k=0}^{n-1} \Pr\{Z^{H_n^{k+1}}(1^n) = 1\} \right| \\ &= \frac{1}{n} |\Pr\{Z^{H_n^0}(1^n) = 1\} - \Pr\{Z^{H_n^n}(1^n) = 1\}| \\ &\geq \frac{1}{np(n)}, \end{aligned}$$

amivel ellentmondásra jutottunk. □

16.3. Álvéletlen permutáció konstrukció

A Feistel-féle rétegekből felépülő struktúra az alapja az alábbiakban bemutatásra kerülő álvéletlen permutáció konstrukcióknak. A Feistel-rétegezés garantálja, hogy permutációt kapunk. Tekintsük először a 16.2. ábra két rétegből álló $DES_{H_n}^2$ mini-DES struktúráját. A DES f leképezését, rétegenként független, egyenletes eloszlású véletlen függvénnyel helyettesítettük. Először belátjuk, hogy két Feistel-réteggel nem képezhető PRP.

16.5. Tétel. *A $DES_{H_n}^2$ nem PRP.*

16.2. ábra. $DES_{H_n}^2$ leképezés

Bizonyítás: A rövidség kedvéért legyen $F = DES_{H_n}^2$. Ezen $F : \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$ leképezés az alábbi:

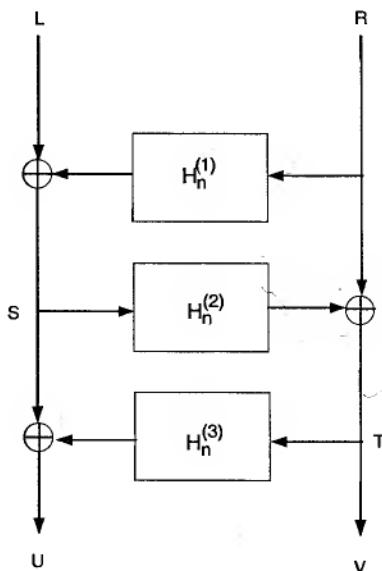
$$F_{H_n^{(1)}, H_n^{(2)}}(L, R) = (L \oplus H_n^{(1)}(R), R \oplus H_n^{(2)}(L \oplus H_n^{(1)}(R))),$$

ahol $H_n^{(1)}$ és $H_n^{(2)} : \{0,1\}^n \rightarrow \{0,1\}^n$ egyenletes eloszlású véletlen függvények, amelyeket tekinthetünk az eredeti DES transzformáció f leképezése idealizált modelljének.

Z^{F^*} megkülönböztető támadó hozzáfér egy F^* orákulumhoz, amely orákulum $1/2$ valószínűsséggel F vagy H_{2n} leképezés szerint működik. A támadó az orákulumnak $x_1 = (L_1, R)$, illetve $x_2 = (L_2, R)$ kérést küldi, ahol $L_2 = L_1 \oplus w$. Könnyen ellenőrizhető, hogy ha $F^* = F$, továbbá $F^*(x_1) = (U_1, V_1)$, $F^*(x_2) = (U_2, V_2)$, akkor $U_2 = U_1 \oplus w$.

Ennek alapján $Z^{F^*} = 1$, ha $F^*(x_1) \oplus F^*(x_2) = (w, .)$, és $Z^{F^*} = 0$, egyébként. Mivel $\Pr\{H_{2n}(x_1) \oplus H_{2n}(x_2) = (w, .)\} = 2^{-n}$, ezért F és H_{2n} megkülönböztetési valószínűsége $1 - 2^{-n}$. \square

Ha azonban a 16.2. ábra szerinti konstrukciót 3 rétegűre bővítjük, akkor a kapott $DES_{H_n}^3 : \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$ transzformáció már álvéletlen permutáció lesz (16.3. ábra).

16.3. ábra. $DES_{H_n}^3$ leképezés

16.6. Tétel. A $DES_{H_n}^3$ PRP.

Bizonyítás: Azt kell belátni, hogy egy hatékony támadó a $DES_{H_n}^3 : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ leképezést a $H_{2n} : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ leképezéstől csak elhanyagolható valószínűsséggel képes megkülönböztetni (a (16.1) formula értelmében). Jelölje q az orákulum-kérések számát, amit Z^{F^*} megkülönböztető támadó az F^* orákulumhoz küld, amely orákulum $1/2$ valószínűsséggel $DES_{H_n}^3$ vagy H_{2n} leképezés szerint működik.

Tekintsük a 16.3. ábrát, s legyen B a következő esemény: az $S_i = S_j$ vagy $T_i = T_j$ valamely $1 \leq i < j \leq q$ kérés-párra. A \bar{B} komplementes esemény: $S_i \neq S_j$ és $T_i \neq T_j$ minden $1 \leq i < j \leq q$ kérés-párra. Vegyük észre, hogy \bar{B} feltétel mellett U_i és T_i (V_i), $1 \leq i \leq q$ független, egyenletes eloszlású valószínűségi változók. Következésképpen \bar{B} feltétel mellett nem megkülönböztethető Z^{F^*} számára az, ha $DES_{H_n}^3$ leképezés helyett H_{2n} leképezés szerint kapta volna a válaszokat kéréseire: ugyanis $H_n^{(2)}$ és $H_n^{(3)}$ valódi véletlen leképezések kimenete one-time-pad kulcs jelleggel véletlenítí a kimeneti blokkokat.

Innen már „csak” technikai lépések következnek:

$$\Pr\{B\} \leq \sum_{i \neq j} (\Pr\{S_i = S_j\} + \Pr\{T_i = T_j\}) \leq 2 \cdot \binom{q}{2} 2^{-n} \leq q^2/2^n.$$

Így a $d_t(D, D') \leq d_t(D|E, D'|E) + P(\overline{E})$ egyenlőtlenség felhasználásával, ahol $E = \overline{B}$, $t = q$, D a $Z^{DES_{H_n}^3}$ kimenet-eloszlása, továbbá D' a $Z^{H_{2n}}$ kimenet-eloszlása:

$$\begin{aligned} |\Pr\{Z^{DES_{H_n}^3} = 1\} - \Pr\{Z^{H_{2n}} = 1\}| &\leq \\ &\leq |\Pr\{Z^{DES_{H_n}^3} = 1|\overline{B}\} - \Pr\{Z^{H_{2n}} = 1|\overline{B}\}| + \Pr\{B\} \\ &= |0| + \Pr\{B\} \leq q^2/2^n. \end{aligned}$$

Tehát $DES_{H_n}^3$ álvéletlen permutáció: a megkülönböztethetőség $q(n)$ polinom kérésszámkorlát mellett n paraméterrel exponenciálisan gyorsan (így bárminely polinom reciprokánál gyorsabban) csökken, s így elhanyagolható. \square

A $DES_{H_n}^3$ konstrukció véletlen függvényt használ, azaz nem hatékony generálású. Kérdés, mi történik, ha $F_n^{(i)} : \{0, 1\}^n \rightarrow \{0, 1\}^n$, $i = 1, 2, 3$ álvéletlen függvényekkel helyettesítjük a $H_n^{(i)} : \{0, 1\}^n \rightarrow \{0, 1\}^n$, $i = 1, 2, 3$ egyenletes eloszlású véletlen függvényeket. Szerencsére továbbra is álvéletlen permutációt kapunk (lásd a 16.4. feladatot). Ugyanakkor a $DES_{H_n}^3$ ($DES_{F_n}^3$) leképezés még nem erős álvéletlen permutáció, azaz ha az inverz orákulumhoz is hozzáfér a támadó, akkor nem elhanyagolható sikerrel a 3-rétegű mini-DES konstrukciót képes megkülönböztetni a H_{2n} leképezéstől.

16.7. Tétel. A $DES_{H_n}^3$ nem erős PRP.

Bizonyítás: A támadás logikája rokon a 2-rétegű DES támadással, itt is be-menetblokkok differenciáival ügyeskedünk:

A rövidség kedvéért legyen $F = DES_{H_n}^3$, továbbá a Z megkülönböztető támadó F^* orákulumnak, illetve F^* inverzének küldhet kéréseket, ahol F^* 1/2 valószínűséggel F vagy H_{2n} leképezés szerint működik. Legyen $F(L, R) = (U, V)$. Továbbá legyen $w_1, w_2, w_3 \in \{0, 1\}^n$ olyan, hogy $F(L \oplus w_1 \oplus w_3, R \oplus w_2) = (U', V')$, amelyre $U' = U \oplus w_1$, $V' = V$ teljesüljön. Ezen megkötés a w_1, w_2, w_3 hármasra is megkötést eredményez: a 16.3. ábra jelútjait követve

$$\begin{aligned} U' &= (L \oplus w_1 \oplus w_3) \oplus H_n^{(1)}(R \oplus w_2) \oplus H_n^{(3)}(V') \\ &= U \oplus w_1 \\ &= [L \oplus H_n^{(1)}(R) \oplus H_n^{(3)}(V)] \oplus w_1, \end{aligned}$$

ahonnan egyszerűsítés után a w_2 és w_3 között az alábbi összefüggés adódik:

$$w_3 = H_n^{(1)}(R) \oplus H_n^{(1)}(R \oplus w_2). \quad (16.6)$$

Tekintsük most az $(L \oplus w_3, R \oplus w_2)$ bemenetet F számára, s jelölje a kimenetet (U_1, V_1) . (16.6) felhasználásával

$$\begin{aligned} V_1 &= R \oplus w_2 \oplus H_n^{(2)}(L \oplus w_3 \oplus H_n^{(1)}(R \oplus w_2)) \\ &= R \oplus H_n^{(2)}(L \oplus H_n^{(1)}(R)) \oplus w_2 \\ &= V \oplus w_2. \end{aligned}$$

A könnyebb áttekinthetőség kedvéért a 16.1. táblázatban összefoglaltuk az F^* orákulumhoz küldött három kérést (bemenet blokkot), valamint F^* orákulum válaszát (kimenet blokkot), azon esetben, amikor $F^* = F$.

F^* kérés	F^* válasz	összefüggés
(L, R)	(U, V)	
$(L \oplus w_1 \oplus w_3, R \oplus w_2)$	(U', V')	$U' = U \oplus w_1, V' = V$
$(L \oplus w_3, R \oplus w_2)$	(U_1, V_1)	$V_1 = V \oplus w_2$

16.1. táblázat. $DES_{H_n}^3$ támadása

Z támadó ismerve a a 16.1. táblázatbeli összefüggéseket, először a kimenet felől támad:

A támadó $F^* = F$ feltételezéssel él, s próbálja ezt megerősíteni vagy megcáfolni. A támadó az F^* invertáló orákulumának ad bemenetként egyrészt egy (U, V) blokkot, másrészt egy tetszőlegesen választott $w_1 \in \{0, 1\}^n$ mellett az $(U \oplus w_1, V)$ blokkot, amelyekre – az $F^* = F$ feltételezés mellett – az invertáló orákulum válaszként az (L, R) illetve az $(L \oplus w_1 \oplus w_3, R \oplus w_2)$ blokkot küldi. Ezen két válaszblokk alapján a támadó már tudja w_2 és w_3 értékét.

A támadó most az F^* orákulumnak elküldi az $(L \oplus w_3, R \oplus w_2)$ bemenetet. Az arra kapott válasz alapján a támadó az alábbi megkülönböztető döntést hozza: ha $F^*(L \oplus w_3, R \oplus w_2) = (\cdot, V \oplus w_2)$, akkor Z kimenete 1 ($F^* = F$ mellett dönt), egyébként Z kimenete 0 ($F^* = H_{2n}$ mellett dönt).

Ha $F^* = H_{2n}$, akkor a különböző bemenetekhez tartozó kimenetek független egyenletes eloszlású valószínűségi változók, következésképp a fenti döntés megkülönböztető ereje $1 - 2^{-n}$. \square

A fenti technikákkal az is megmutatható, hogy $DES_{F_n}^4$ viszont már erős álvéletlen permutáció.

16.4. PRF alkalmazás példák

Kihívás-válasz alapú partnerazonosítás. A és B fél megegyeznek egy közös k kulcsban. A fél szeretné a távoli B felet azonosítani, s ehhez generál egy véletlen r elemet egy PRF függvény együttes $f_k(\cdot)$ tagjának ősképteréből, s elküldi kihívásként B félnek, aki alkalmazza rá a közös titkos függvényt, majd az eredményt válaszként visszaküldi A félnek:

$$\begin{array}{ll} (1) & A \rightarrow B: \quad r \\ (2) & B \rightarrow A: \quad f_k(r) \end{array}$$

Szimmetrikus kulcsú rejtjelezés. A és B fél megegyeznek egy közös k kulcsban. A fél szeretne a távoli B félnek egy m üzenetet rejtjelezetten továbbítani. Ezt a következőképp teszi:

$$E_k(m) = (r, f_k(r) \oplus m).$$

A PRF továbbá kriptográfiai hash függvény modellezésére is alkalmas (16.6. feladat).

16.5. Feladatok

16.1. Feladat. Igazoljuk ellenpéldával, hogy miért nem helyes az alábbi definíció egy PRF-re: F_n egy PRF, ha létezik M^{F_n} polinom idejű orákulumos algoritmus, ami az alábbi pontossággal képes becslőlni az F_n kimenetét

$$\Pr\{M^{F_n}(U_n) = F_n(U_n)\} < \frac{1}{2^{|F_n|}} + \frac{1}{p(n)}$$

tetszőleges $p(n)$ polinomra, s elegendően nagy n -re (M természetesen nem küldheti saját bemenetét kérésként az F_n orákulumhoz).

16.2. Feladat. Igazoljuk ellenpéldával, hogy miért nem helyes az alábbi alternatív konstrukció PRF-re:

$$f_s(x) = G_{\sigma_n}(\dots (G_{\sigma_2}(G_{\sigma_1}(x))\dots),$$

ahol $s = \sigma_1 \dots \sigma_n$. Azaz x és s szerepe felcserélésre került a standard konstrukcióhoz képest.

16.3. Feladat. Zárat gyártunk, amelyeken olvasható az N azonosító szám, míg a zár W kombinációját titkosan kell kezeljük. A kombinációt egy n bites szó azonosítja. Hogyan járunk el a W kombináció megválasztásánál, ha a feltételek a következők:

- az N nyilvános azonosító alapján ne legyen kitalálható,
- ha az ügyfél a kombinációt később elfelejti, azt közölnünk kell tudni vele,
- minimalizálni szeretnénk a (sorozatszám, kombináció) párokkal kapcsolatos biztonsági tárolási igényt,

ha

1. élhetünk az egyirányú függvények létezésének feltételezésével
2. az ügyfelek hipochonderek, félnek, hogy esetleg kiderül, hogy $P = NP$, ezért csak feltétel nélküli biztonság garantálását fogadják el.

16.4. Feladat. Bizonyítsuk be, hogy ha a $DES_{H_n}^3$ konstrukcióban álvéletlen függvényeket használunk, akkor is álvéletlen permutációra jutunk!

16.5. Feladat. A csapda permutáció lényegében egy egyirányú permutáció. Mint egy nyilvános kulcsú rejtjelező modellje, a csapda permutáció tulajdonság garantálja a választott nyílt szövegű támadás elleni védeottséget (nyilvános kulcs ismeretében orákulum nélkül is tudunk választott nyílt szövegű támadást indítani). Igazoljuk, hogy amennyiben létezik (G, F, I) csapda permutáció, akkor létezik erős álvéletlen permutáció is.

Informálisan: ha létezik választott nyílt szövegű támadásra biztonságos nyilvános kulcsú rejtjelező, akkor létezik választott rejtett szövegű támadásra biztonságos szimmetrikus kulcsú rejtjelező is.

Segítség: A feladat összetett, ezért megoldási segítséget adunk hozzá. A biztonságos csapda permutáció léte implikálja az egyirányú függvény létezését. Egyirányú függvényből kiindulva viszont konstruálható álvéletlen permutáció, s így létezik olyan $E(k, x)$ algoritmus, amely k véletlen választása mellett álvéletlen permutáció az x bemenet vonatkozásában. Jelölje $D(k, x)$ a kapcsolatos inverzet. Ezen (E, D) álvéletlen permutáció és a (G, F, I) csapda permutáció kaszkádba kötésével generáljuk a kívánt (G', F', I') permutációt. (G, F, I) csapda permutáció esetén $F(pk, x)$ a kódoló és $I(sk, y)$ a dekódoló transzformáció. (G', F', I') permutáció esetén $F'((k, pk), x) = E(k, F(pk, x))$

a kódoló és $I'((k, sk), z) = I(sk, D(k, z))$ a dekódoló transzformáció, azaz

$$\begin{aligned} F' : x &\rightarrow F(pk, .) \rightarrow E(k, .) \rightarrow z \\ I' : z &\rightarrow D(k, .) \rightarrow I(sk, .) \rightarrow x. \end{aligned}$$

Továbbá G' generálja (k, pk, sk) kulcshármast, ahol a (pk, sk) párt G felhasználásával, míg k kulcsot véletlenszerűen választva állítja elő.

Igazoljuk tehát, hogy a fenti módon generált (G', F', I') álvéletlen permutáció és csapda permutáció is egyben!

16.6. Feladat*. Vizsgálja meg a véletlen függvény, illetve a PRF által modellezett kriptográfiai hash függvény tulajdonságait egyirányúság, illetve ütközésmentesség vonatkozásban.

17.

Szimmetrikus kulcsú rejtjelező leképezés modelljei

A hatékonyan kiszámítható függvényegyüttés definíciója magában foglalja a hatékony indexelést, valamint a kiválasztott függvény hatékony kiértékelését argumentuma bármely elemére. A szimmetrikus kulcsú rejtjelezés esetén az indexelés természetes módon adódik: a biztonsági paramétert indexbe képező I algoritmus az $I(1^n) = K$ kulcs kisorsolását, míg az indexet tagfüggvénybe képező ρ algoritmus a $\rho(K) = E_K$ rejtjelező függvény hozzárendelést végzi.

Az álvéletlen függvény, valamint álvéletlen permutáció szimmetrikus kulcsú blokkrejtjelező transzformációk és kriptográfiai hash függvények modelljei. Az álvéletlen leképezések együtteseinek eddigi definícióiban elsősorban az n biztonsági paraméter növekedése melletti aszimptotikus megkülönbözthetetlenségre koncentráltunk. Konkrét rejtjelező transzformációk vizsgálatánál azonban nagyon fontos lenne rögzített erőforráskorlátok mellett analizálni azok biztonságát.

Ehhez először definiálni kell azt, hogy mit értünk egy rejtjelezés biztonságosságán. Első pillantásra talán egyszerűnek tűnhet a biztonságosság meghatározása, de kicsit utánagondolva rájöhetünk, hogy ez nem is egyszerű feladat. Mondhatnánk például azt, hogy az a rejtjelező transzformáció biztonságos, amelynek kulcsát a támadó nem képes megismerni (kiszámlálni). Sajnos, egy blokk-kódoló kulcsfejtés elleni biztonsága nem elégsges feltétele a kódoló biztonságának. Triviálisnak tűnő példa erre a következő: legyen a kódoló transzformáció a helybenhagyás: $E_K(x) = x$, azaz amikor végül is nem kódolunk. Ez a kódoló nyilván használhatatlan rejtjelezésre, ugyanakkor a támadó semmilyen módon nem képes a kulcsot meghatározni. Gondolhat-

nánk például arra is, hogy a támadó ne tudja rekonstruálni a nyílt szöveget. De mi a helyzet akkor, ha a teljes nyílt szöveg vonatkozásában erre ugyan nem képes a támadó, de a nyílt szöveg néhány bitjét mégis ki tudja deríteni, s ez utóbbi is hordoz érzékeny információt.

Ebben a fejezetben „konkrét biztonság” megközelítésben a szimmetrikus kulcsú rejtjelezés kapcsán tekintjük át a biztonság definícióit. Ezen definíciók alapvetően két eloszlás megkülönböztetésére vonatkoznak. Számos példával igyekszünk elmélyíteni a definíciók megértését.

A szimmetrikus kulcsú rejtjelezéssel kapcsolatos biztonság definíciók közül a következőket tekintjük:

1. véletlen függvénytől (véletlen permutációtól) megkülönböztető támadás (*prf*- és *prp*-megkülönböztetés),
2. kulcsfejtő támadás (*kr*, key recovery),
3. nyílt szöveg visszafejtő támadás (*pr*, plaintext recovery),
4. üzenet megkülönböztető támadás (*lor*-megkülönböztetés¹).

17.1. Véletlen függvénytől megkülönböztetés

A véletlen függvénytől megkülönböztetés kapcsán azt vizsgáljuk, hogy rejtjelező függvények kulccsal indexelt együttese úgy viselkedik-e, mint egy által véletlen együttes, azaz (egyenletes eloszlású) véletlen függvények együttesétől, a megkülönböztetés mértéke szerint, milyen mértékben különböznek. A konkrét biztonság megközelítés kapcsán az erőforrás paraméterek és a biztonsági paraméter minél pontosabb kifejezésével szeretnénk becsülni ezen megkülönböztetés mértékét.

Ahol nem okoz félreérést, az egyenletes eloszlású véletlen függvény röviden véletlen függvénynek nevezzük.

Annak érdekében, hogy könnyebben követhetők legyenek a biztonság fogalmak az alkalmazásaik során, kissé részletesebbé tesszük a kapcsolatos megkülönböztetés formulák megjelenését. A kapcsolatos jelöléseket véletlen függvénytől megkülönböztetés kapcsán mutatjuk be.

Jelölje $\{prf - cpa - 1_E(Z^8) = 1\}$ azt az eseményt, ahol

1. Z a támadó (megkülönböztető) algoritmus,

¹ left-or-right indistinguishability

2. Z támadó az E leképezést szeretné megkülönböztetni a megfelelő dimenziójú véletlen függvénytől, s erre utal a prf jelölés. (Véletlen permutációtól való megkülönböztetés esetén prp jelölést használunk.)
3. Z támadó egy g orákulumnak küld kéréseket, ahol az orákulum a támadás kezdetén véletlen bináris döntést hoz, s vagy az E leképezés, vagy a véletlen függvény szerint adja meg a választ. Ha a kérések az orákulum által számolt leképezés bemenetét adják, akkor ez megfelel a szokásos választott nyílt szövegű támadásnak, ezért ezt az esetet cpa (chosen plaintext attack) rövidítéssel jelöljük.
Hasonlóan, amikor a Z támadó az inverz leképezésekre is küldhet kéréseket, az megfelel a választott rejtett szövegű támadásnak, amit a cca (chosen ciphertext attack) rövidítéssel jelölünk.
4. $prf - cpa - 1$ jelölésben az 1 szám azt jelzi, hogy az orákulum az E leképezés szerint számol. Hasonlóan, a $prf - cpa - 0$ esetben az orákulum a véletlen leképezést futtatja.
5. Az egyenlőség utáni 1 jelenti Z támadó döntését, azaz szerinte az orákulum az E leképezés szerint számol.

Összefoglalóan tehát, a támadás-azonosító szerkezete a következő: támadás kódjele – orákulum hozzáférés típusa – kisorsolt leképezés 1/0 jele. Elvileg meg kellene jelölni még az erőforráskorlátot is, ezt azonban – az amúgy is összetett támadás-azonosítóból – fentebb elhagytuk, ugyanakkor az összes különböző támadó feletti megkülönböztető képesség kifejezésénél már jelöljük, ahogy az látható az alábbi definícióban is:

17.1. Definíció (prf -megkülönböztetés). Z támadó prf megkülönböztető képessége a

$$\Delta_E^{prf-cpa}(Z) = \Pr\{prf - cpa - 1_E(Z) = 1\} - \Pr\{prf - cpa - 0_E(Z) = 1\} \quad (17.1)$$

differenciával, az E leképezés megkülönböztethetősége a

$$\Delta_E^{prf-cpa}(t, q_e, \mu_e) = \max_Z \Delta_E^{prf-cpa}(Z) \quad (17.2)$$

maximummal kerül definiálásra, ahol (t, q, μ_e) jelöli a támadó erőforrás-korlátját, ahol

1. t : futási idő,

2. q_e : kódoló orákulumhoz küldött kérések száma,
3. μ_e : kódoló orákulum válaszainak összhossza.

Megjegyzés: A (17.1) kifejezéssel kapcsolatban emlékeztetünk arra, hogy ezen differencia előjele mindenkorán pozitívra, definálva egy olyan támadót, amelynek bináris kimenete az eredeti komplemente. A megkülönböztetés ereje ugyanaz marad.

Ha a támadó egy dekódoló orákulumhoz is hozzáfér (*cca*), akkor $(t, q_e, \mu_e, q_d, \mu_d)$ jelöli a támadó erőforráskorlátját, ahol

1. q_d : a dekódoló orákulumhoz küldött kérések száma,
2. μ_d : a dekódoló orákulum válaszai összhossza.

17.1. Példa. A születésnap paradoxon egy alkalmazása:

Tekintsük most az $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ szimmetrikus kulcsú rejtjelezést mint kulccsal indexelt permutáció-együtttest. Legyen a feladat E megkülönböztetése a véletlen függvénytől.

Tekintsük a következő Z^g megkülönböztető algoritmust: Z kérdezze meg g orákulumot q alkalommal, s ha minden a q válasz különböző, döntsön álvéletlen permutációra (kimenet= 1), egyébként döntsön véletlen függvényre (kimenet= 0). A születésnapi paradoxon alapján Z megkülönböztető erejére a következő alsó korlát kapható:

$$\Delta_E^{prp-cpa}(t, q, qn) \geq 0.3 \frac{q(q-1)}{2^n}. \quad (17.3)$$



17.2. Példa. Lineáris rejtjelező esete:

Legyen E lineáris leképezések halmaza: $y = Ax$, ahol A egy $n \times n$ méretű invertálható bináris mátrix, a kulcs. Ezen E rejtjelező *prf*-megkülönböztethetősége közel 1 értékű, következésképpen ezen rejtjelező leképezésként nem biztonságos:

Tekintsük a következő Z^g megkülönböztető algoritmust: ha $g(0^n) = 0^n$, akkor Z kimenete 1, egyébként Z kimenete 0. Azaz zérus bemenetre kérünk választ az orákulumtól, amely $g = E$ esetén minden zérus választ ad, míg ha g véletlen függvény szerint számol, akkor csak 2^{-n} valószínűséggel kapjuk a zérus kimenet értéket. Következésképp

$$\Delta_E^{prf-cpa}(t, 1, n) = 1 - 2^{-n},$$

(ahol t processzálási idő feltételezettel elegendő az orákulum válasz 0^n sorozattal való összehasonlításához, s ennek eredményétől függően a $0/1$ kimenet írásához).



17.3. Példa. A DES, illetve az AES rejtjelező kódolók biztonságára a kimerítő kulcskeresés, illetve a lineáris kriptanalízis számításigényének felhasználásával, az alábbi alsó korlátot adhatjuk:

1. $\Delta_{DES}^{prf-cpa}(t, q_e, 64 \cdot q_e) \geq \max\{c_1 \frac{t/T_{DES}}{2^{55}}, c_2 \frac{q_e}{2^{40}}\},$
2. $\Delta_{AES}^{prf-cpa}(t, q_e, 128 \cdot q_e) \geq \max\{c_3 \frac{t/T_{AES}}{2^{128}}, c_4 \frac{q_e}{2^{128}}\},$

ahol T_x az x kódoló esetén egy nyílt szöveg blokk-kódolásának ideje, továbbá c_i együtthatók megfelelő konstansok. Ugyanis, ha egy p valószínűsséggel ismerjük a blokkrejtjelező kulcsát, akkor ugyanekkora valószínűsséggel ismerjük a dekódoló transzformációt is, s ezen utóbbi transzformációt alkalmazva a cpa -orákulum válaszára, legalább p valószínűsséggel sikeresen meg tudjuk különböztetni az orákulum által futatott algoritmus kétféle választását.



A prf-cpa és prp-cpa biztonság kapcsolata. A PRP egy blokkrejtjelező természetes modellje. Ugyanakkor a PRF modell analízise egyszerűbb. Ezért érdekes az a kérdés, hogy a prf -megkülönböztethetőség illetve a prp -megkülönböztethetőség között milyen kapcsolat van, azaz milyen kapcsolat van a véletlen függvénytől, illetve a véletlen permutációtól való megkülönböztethetőség között bemenet kéréseket küldve a kapcsolatos megkülönböztető orákulumnak (cpa).

17.2. Tétel. *Tekintsünk egy E álvéletlen permutációt:*

$$\Delta_E^{prf-cpa}(t, q, qn) \leq \Delta_E^{prp-cpa}(t, q, qn) + \frac{q(q-1)}{2^n}. \quad (17.4)$$

Bizonyítás: Az észrevétel a következő: egy E álvéletlen permutáció véletlen függvénytől való megkülönböztethetőségét segíti az, hogy a véletlen függvény különböző bemenetekhez rendelhet azonos kimenetet, míg a permutáció nem.

A bizonyítás leírását technikailag egyszerűsíti egy V támadó bevezetése, amely hozzon a Z támadóval komplexens kimenetelű döntést, azaz ha Z kimenete 1, akkor V kimenete 0, és megfordítva.

$$\begin{aligned}
\Delta_E^{prf-cpa}(Z) &= \\
&= \Pr\{prf - cpa - 1_E(Z) = 1\} - \Pr\{prf - cpa - 0_E(Z) = 1\} \\
&= (1 - \Pr\{prf - cpa - 1_E(V) = 1\}) - (1 - \Pr\{prf - cpa - 0_E(V) = 1\}) \\
&= \Pr\{prf - cpa - 0_E(V) = 1\} - \Pr\{prf - cpa - 1_E(V) = 1\} \\
&= \Pr\{prf - cpa - 0_E(V) = 1\} - \Pr\{prp - cpa - 1_E(V) = 1\} (*) \\
&= \Pr\{prf - cpa - 0_E(V) = 1\} - \Pr\{prp - cpa - 0_E(V) = 1\} \\
&\quad + \Pr\{prp - cpa - 0_E(V) = 1\} - \Pr\{prp - cpa - 1_E(V) = 1\} \\
&= \Pr\{prf - cpa - 0_E(V) = 1\} - \Pr\{prp - cpa - 0_E(V) = 1\} \\
&\quad + \Delta_E^{prp-cpa}(V),
\end{aligned}$$

ahol a negyedik sor második tagja megegyezik a harmadik sor második tagjával, mivel a $prf - cpa - 1$, illetve $prp - cpa - 1$ azt jelenti, hogy a g orákulum az E szerint válaszol (különbség $prf - cpa - 0$ és $prp - cpa - 0$ között van, ahol az egyikben véletlen függvény, a másikban véletlen permutáció szerint érkezik a válasz).

Ezek után az állítás bizonyításához tehát azt kell belátni, hogy

$$\Pr\{prf - cpa - 0_E(V) = 1\} - \Pr\{prp - cpa - 0_E(V) = 1\} \leq \frac{q(q-1)}{2^{n+1}}.$$

Jelölje D azt az eseményt, hogy a g orákulumhoz intézett q különböző kérisre q különböző válasz érkezett:

$$\begin{aligned}
&\Pr\{prf - cpa - 0_E(V) = 1\} = \\
&= \Pr\{prf - cpa - 0_E(V) = 1|D\} \Pr\{D\} \\
&\quad + \Pr\{prf - cpa - 0_E(V) = 1|\bar{D}\} \Pr\{\bar{D}\} \\
&\leq \Pr\{prf - cpa - 0_E(V) = 1|D\} + \Pr\{\bar{D}\} \\
&= \Pr\{prp - cpa - 0_E(V) = 1\} + \Pr\{\bar{D}\} \\
&\leq \Pr\{prp - cpa - 0_E(V) = 1\} + \frac{q(q-1)}{2^{n+1}},
\end{aligned}$$

ahol a negyedik sor első tagjánál azt az észrevételt használtuk, hogy D feltétel mellett a $prf - cpa - 0$ környezet nem különböztehető meg a támadó számára a $prp - cpa - 0$ környezettől. \square

17.2. Kulcsfejtés elleni biztonság

Klasszikus támadási célkitűzés az, hogy egy *cpa*-támadó q számú nyílt-rejtett pár alapján meghatározza a kulcsot. Ezen támadási célt tűzik ki a ma ismert legerősebb támadási módszerek, a differenciális és a lineáris kriptanalízis is. Mint a fejezet bevezetőjében láttuk, a kulcsfejtés elleni biztonság nem elégsgéges feltétele a kódoló biztonságának.

Egy Z támadó által egy rejtjelező ellen végrehajtott kr kulcsfejtő támadáson a következő lépések végrehajtását értjük:

17.1. Algoritmus.

1. Egy K kulcs kisorsolásra kerül a kulcstérből, amely kulcs Z számára nem ismert.
2. Z támadó választott nyílt szövegére mint kérésre, az E_K orákulum annak rejtjelezettjét küldi válaszként. A küldhető kérések száma q_e .
3. A nyílt-rejtett szövegpárok alapján Z a K' döntést hozza a kulcsra.
4. A kérések K' kulccsal történő kódolásával Z ellenőrzi döntése helyességet. Ha döntése helyes, akkor kimenetén 1, ellenkező esetben 0 értéket ad.

A Z támadó kr kulcsfejtő támadásának erejét – értelemszerűen – a

$$\Delta_E^{kr}(Z) = \Pr\{kr_E(Z) = 1\}$$

valószínűség definiálja, s ennek alapján az E_K rejtjelező kr -biztonsága

$$\Delta_E^{kr}(t, q_e, \mu_e) = \max_Z \Pr\{kr_E(Z) = 1\}. \quad (17.5)$$

17.4. Példa. (DES): Ha feltesszük, hogy két nyílt-rejtett páron ellenőrizve a kimerítő kereséses támadást, az minden sikeres végeredményű (ami gyakorlatilag jó közelítés), akkor

$$\Delta_{DES}^{kr}(2^{55}T_{DES}, 2, 2 \cdot 64) \simeq 1,$$

hasonlóan lineáris kriptanalízis esetére:

$$\Delta_{DES}^{kr}(2^{43}T_{DES}, 2^{43}, 2^{43} \cdot 64) \simeq 1,$$

ahol T_{DES} egy DES rejtjelezés számítás ideje. Lineáris kriptanalízis esetén jóval kisebb a számításigény, de jóval nagyobb a nyílt-rejtett párok szükséges száma. A példa jól illusztrálja a számításidő – adatigény mérleget is. ♣

A gyakorlatban használt $E_K(x)$ szimmetrikus kulcsú rejtjelező leképezésekkel kapcsolatosan elvárjuk, hogy K kulcs bemenetében, x rögzített (támadó által is ismert) értéke mellett egyirányú leképezést nyújtson. Ezen elvárásunk helyességét be is tudjuk bizonyítani PRF tulajdonságú rejtjelező esetén.

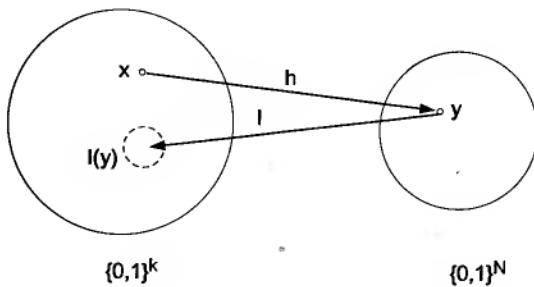
Emlékeztetünk az egyirányú függvény definíciójára. Tekintsünk egy $h : \{0,1\}^k \rightarrow \{0,1\}^N$ függvényt, és legyen I ezen függvény egy invertáló algoritmusai. Jelölje $P_h^{owf}(I)$ az I invertáló algoritmus sikervalószínűségét (17.1. ábra):

$$P_h^{owf}(I) = \Pr\{h(I(h(U_k))) = h(U_k)\}. \quad (17.6)$$

Ezen valószínűség t futási idő erőforráskorlátnak megfelelő invertáló algoritmusok feletti maximuma

$$P_h^{owf}(t) = \max_I P_h^{owf}(I) \quad (17.7)$$

definiálja a h leképezés *owf*-biztonságát (azaz jelen esetben nem két valószínűség különbsége).



17.1. ábra. Illusztráció a 17.3. tételehez

17.3. Tétel. Ha $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^N$ rejtjelező prf – cpa-biztonságos, akkor $h(K) = E(K, 0^n) : \{0,1\}^k \rightarrow \{0,1\}^N$ függvény *owf*-biztonságos:

$$P_h^{owf}(t) \leq \frac{1}{1 - 2^{k-N}} \Delta_E^{prf-cpa}(t + t_E, 1, N), \quad (17.8)$$

ahol $k \leq N - 1$, továbbá t_E (lényegében) egy rejtjelezési lépés ideje.

Bizonyítás: Lényegében a következőt kell bebizonyítani: ha a támadó a kulcsot nem elhanyagolható sikерrel képes kiszámítani, akkor képes a rejtjelező transzformációt is sikeresen megkülönböztetni a véletlen leképezéstől. Legyen $R : \{0,1\}^n \rightarrow \{0,1\}^N$ egy véletlen függvény. Legyen g egy rejtjelező orákulum, amelyről nem tudja a támadó, hogy az E vagy az R leképezést futtatja-e. Legyen Z egy I invertáló algoritmust felhasználó megkülönböztető támadó, amely a g orákulumhoz tesz fel kérdéseket, s a válaszok alapján eldönti, hogy az orákulum milyen algoritmust futtat:

17.2. Algoritmus.

1. Z támadó a g orákulumnak a 0^n üzenetet (kérést) küldi, amelyre az orákulum az y választ adja.
2. Z támadó futtatja az I invertáló algoritmust, s bemenetként y -t adja, majd beolvassa az $x = I(y)$ kimenetet.
3. Z támadó ellenőriz és dönt: ha $E(x, 0^n) = y$, akkor Z kimenete 1, egyébként kimenete 0.

Legyen továbbá

$$P_h^{owf}(I) = \epsilon. \quad (17.9)$$

Alább belátjuk, hogy

$$\Pr\{prf - cpa - 1_E(Z) = 1\} = \epsilon, \quad (17.10)$$

továbbá

$$\Pr\{prf - cpa - 0_E(Z) = 1\} \leq \frac{2^k}{2^N} \cdot \epsilon. \quad (17.11)$$

Ha ezt már beláttuk, akkor

$$\begin{aligned} \Delta_E^{prf - cpa}(Z) &= \\ &= \Pr\{prf - cpa - 1_E(Z) = 1\} - \Pr\{prf - cpa - 0_E(Z) = 1\} \\ &\geq \epsilon - \frac{2^k}{2^N} \cdot \epsilon = (1 - 2^{k-N})\epsilon \end{aligned} \quad (17.12)$$

alapján (17.9) figyelembevételével a (17.8) állítás adódik.

Ugyanis (17.9) és (17.12) alapján tetszőleges I esetén

$$P_h^{owf}(I) \leq \frac{1}{1 - 2^{k-N}} \cdot \Delta_E^{prf - cpa}(Z),$$

majd vegyük ezen egyenlőtlenség kétoldali mennyiségeinek maximumát I -ben (erőforrás-korlát mellett), valamint vegyük figyelembe a jobb oldalon, hogy

$$\max_I \Delta_E^{prf-cpa}(Z) \leq \max_V \Delta_E^{prf-cpa}(V).$$

Lássuk be tehát (17.10) és (17.11) állításokat.

A $g = E$ esetre vonatkozó (17.10) egyenlőség a (17.6) formula mögötti kísérlet (azaz h invertálása és a sikeres ellenőrzése) és Z algoritmus lépéseinak egybeesése miatt fennáll.

(17.11) felső becslés a következőképp látható be. Legyen

$$\begin{aligned} X &= \{x \in \{0,1\}^k : h(I(h(x))) = h(x)\}, \\ Y &= \{y \in \{0,1\}^N : h(I(y)) = y\}. \end{aligned}$$

A feladat $prf - cpa - 0_E(Z)$ esetén az alábbi valószínűség becslése:

$$\delta = \Pr\{E(I(U_N), 0^n) = U_N\} = \Pr\{h(I(U_N)) = U_N\}. \quad (17.13)$$

Nem nehéz látni, hogy

$$\delta = \frac{|Y|}{2^N} \leq \frac{|X|}{2^N} = \frac{2^k \varepsilon}{2^N},$$

ahol az első egyenlőség a δ illetve Y definíciója alapján látható. Az egyenlőtlenség az $|Y| \leq |X|$ alapján következik (ti. h több különböző bemenetet képezhet azonos kimenetbe). A második egyenlőség (17.6) és (17.9) formulák, valamint X definíciója alapján fennálló $\varepsilon = |X|/2^k$ egyenlőség következménye, ugyanis $\Pr\{h(I(h(U_k))) = h(U_k)\} = \varepsilon$. \square

17.3. Nyílt szöveg visszafejtő támadás

Egy Z támadó által egy E_K rejtjelező ellen végrehajtott nyílt szöveg fejtő támadáson (pr, plaintext recovery) a következő lépések végrehajtását értjük:

17.3. Algoritmus.

1. Egy K kulcs és egy m üzenet kisorsolásra kerül a kulcsstérből, illetve üzenettérből, amely sorsolási eredmények Z számára nem ismertek.
2. Z támadó megkapja a $c = E_K(m)$ rejtjeles szöveget.
3. Z támadó nyílt szöveg kérésekkel fordulhat az orákulumhoz. A küldhető kérések száma q_e .

4. A nyílt-rejtett szövegpárok alapján Z döntést hoz az m üzenetre: döntése m' .
5. Z támadó a meghozott döntését ellenőrzi: m' üzenetet elküldi az orákulumnak, s a választ egybeveti c rejtett szöveggel. Ha ezen egybevetés eredményes, akkor kimenetén 1, ellenkező esetben 0 értéket adja.

Megjegyzés: Az 5. ellenőrző lépést nem számítottuk be az orákulumkérések engedélyezett számába. Végső soron Z kimenetén az m' üzenetet is küldhetné (ti. ez a gyakorlati célja), azonban szerettünk volna ragaszkodni az eddigiekben is használt bináris kimenetű támadó algoritmushoz.

A Z támadó nyílt szöveg fejtő erejét a

$$\Delta_E^{pr-cpa}(Z) = \Pr\{pr - cpa_E(Z) = 1\}$$

valószínűség definiálja, s ennek alapján a rejtjelező nyílt szöveg visszafejtés elleni biztonsága:

$$\Delta_E^{pr-cpa}(t, q_e, \mu_e) = \max_Z \Pr\{pr - cpa_E(Z) = 1\}. \quad (17.14)$$

17.4. Üzenet-megkülönböztető támadás

Az üzenet-megkülönböztető támadás (*lor*) erős biztonság-definíció, amely nemcsak a szimmetrikus kulcsú rejtjelezőkre, de kis módosítással az aszimmetrikus kulcsú rejtjelezőkre is alkalmazható. Ezen biztonság garantálja a rejtjelező biztonságát kulcsvisszafejtő támadással, valamint a nyílt szöveget fejtő támadással szemben is. A rejtjelező ebben az értelemben akkor biztonságos, ha a támadó nem tud felmutatni két nyílt szöveget, amelynek rejtjezetjeit nem elhanyagolható mértékben meg tudja különböztetni.

Hozzáférés kódoló orákulumhoz (lor – cpa). Z támadó kiválaszthat tetszőleges (m_0, m_1) üzenetpárt, s ezt a párta rejtjelező orákulumhoz küldi kéresként. Az orákulum először véletlenszerűen kisorsol egy $b \in \{0, 1\}$ bináris értéket, s ennek alapján kiválasztja az m_b üzenetet, majd azt rejtjelezzi, s a rejtjelezett üzenetet válaszként visszaküldi Z támadónak. Formálisan ezt úgy jelöljük, hogy Z támadó egy $E_K \circ \partial_b$ orákulumhoz fér hozzá, ahol $\partial_b(m_0, m_1) = m_b$, továbbá E_K a szimmetrikus kulcsú rejtjelező transzformáció K kulccsal. A Z támadó futási idő korlátja t , legfeljebb q_e kérést küldhet az orákulumnak, amelyek összhossza nem nagyobb, mint μ_e . Az orákulum,

a legelső kérésnél kisorsoltnak megfelelően, a további kérésekre minden az üzenetpár ugyanazon elemét választja ki. A Z támadó feladata annak megál-lapítása, hogy az üzenetpár melyik tagjára kapja a rejtejes választ, azaz egy b' döntést hoz b értékére. Ennek alapján:

17.4. Definíció („bal-vagy-jobb”-megkülönböztetés, $lor - cpa$).

Z támadó $\Delta_E^{lor-cpa}(Z)$ lor – cpa-megkülönböztető képességét

$$\Delta_E^{lor-cpa}(Z) = \Pr\{lor - cpa - 1_E(Z) = 1\} - \Pr\{lor - cpa - 0_E(Z) = 1\} \quad (17.15)$$

defi niálja, illetve az E_K lor – cpa-megkülönböztethetősége a

$$\Delta_E^{lor-cpa}(t, q_e, \mu_e) = \max_Z \Delta_E^{lor-cpa}(Z) \quad (17.16)$$

maximummal kerül defi niálásra. A lor – cpa – 1 támadás-azonosító a $b = 1$, a lor – cpa – 0 támadás-azonosító a $b = 0$ esetre vonatkozik.

Ha $\Delta_E^{lor-cpa}(t, q_e, \mu_e) < \epsilon$, azt mondjuk, hogy E_K rejtelező $(t, q_e, \mu_e, \epsilon)$ -biztonságú. Hasonlóan, ha $\Delta_E^{lor-cpa}(Z) \geq \epsilon$, azt mondjuk, hogy Z támadó a rejtelezőt $(t, q_e, \mu_e, \epsilon)$ -tőri.

A (17.15) differencia a támadó b' döntését kiemelve

$$\Pr\{b' = 1|b = 1\} - \Pr\{b' = 1|b = 0\}$$

alakba is írható, ahonnan elemi lépésekkel

$$\begin{aligned} \Pr\{b' = 1|b = 1\} - \Pr\{b' = 1|b = 0\} &= \\ &= \Pr\{b' = 1|b = 1\} - (1 - \Pr\{b' = 0|b = 0\}) \\ &= 2 \cdot \left[\frac{1}{2} \Pr\{b' = 1|b = 1\} - \frac{1}{2} \Pr\{b' = 0|b = 0\} \right] - 1 \\ &= 2 \cdot \Pr\{b' = b\} - 1 \end{aligned}$$

alakot kapjuk. Ennek felhasználásával a (17.15) definícióval ekvivalens

$$\Delta_E^{lor-cpa}(Z) = 2 \Pr\{lor - cpa - b_E(Z) = b\} - 1 \quad (17.17)$$

definícióra jutunk.

Az alábbi három példa azt jelzi, hogy nem könnyű megfelelni ezen biztonságfogalomnak: az első példa az ECB, a második a CBC blokkrejtjelezési módra mutatja meg, hogy nem lor – cpa-biztonságosak. A harmadik példában viszont megmutatjuk, hogy a CTR (számláló) mód lor – cpa-biztonságos.

17.5. Példa. Az ECB blokkrejtjelezési mód nem *lor – cpa*-biztonságos. Sőt egyetlen orákulum-kérés elegendő a megkülönböztető támadás sikeréhez. Ennek belátásához tekintsük a következő, két blokk hosszú üzenetekből álló párt:

$$(m_0 = [u, u], m_1 = [u, v]),$$

ahol u és v különböző blokkok, $|u| = |v| = n$. Mivel a rejtjelezés blokkonként történik, ezért Z megkülönböztető algoritmus azt vizsgálja, hogy azonos vagy különböző-e a rejtjelezett válasz két blokkja. Ekkor

$$\Delta_E^{lor-cpa}(t, 1, 2n) = 1,$$

ahol t az (m_0, m_1) üzenetpár orákulumhoz küldésének, illetve a válasz két fele egybevetésének ideje. ♣

17.6. Példa. A CBC blokkrejtjelezési mód nem *lor – cpa*-biztonságos. Két kéréssel fordul a támadó a *lor – cpa*-orákulumhoz: az

$$(m_{01} = (r_1, r_2), m_{11} = (u, u))$$

üzenetpárral, és az

$$(m_{02} = (r_3, r_4), m_{12} = (u, v))$$

üzenetpárral, ahol r_i, u, v üzenetblokk méretűek, továbbá r_i blokkok véletlenek. A Z megkülönböztető algoritmus a rejtett szövegek első blokkjának azonosságát vizsgálja a két kérés alapján. Így

$$\Delta_E^{lor-cpa}(t, 2, 4n) = 1 - 2^{-n}.$$

A CTR (Counter) mód használatos randomizált (állapotmentes), illetve számlálóalapú (állapotos) változatban. Mindkettő kulcsfolyamatos rejtjelezés. A randomizált kulcs-inicializálós mód (CTR\$) esetén a kulcsfolyam

$$E_K(r+1), E_K(r+2), \dots, E_K(r+i),$$

míg a számlálós kulcs-inicializálós mód (CTRC) esetén a kulcsfolyam:

$$E_K(ctr+1), E_K(ctr+2), \dots, E_K(ctr+i),$$

ahol r , illetve ctr nyílt szöveg tébeli véletlen, illetve konstans elemek. A kulcsfolyam i blokk hosszú, amely szám megegyezik a rejtjelezendő nyílt

szöveg blokkjainak számával. A kulcsfolyamot bitenkénti mod 2 összeadás-sal adjuk a nyílt szöveghez. Az r véletlen elemet minden egyes újabb nyílt szöveg rejtjelezése előtt újrasorsoljuk. A ctr számláló kezdeti értéke zérus, értéke minden rejtjelezéssel növekszik és tárolódik (állapotos rejtjelezési mód). A számláló értéke nem haladhatja meg a nyílt szövegek terének méretét, ha ezt eléri, a rejtjelezés nem folytatható, ekkor új K kulcsot sorsolunk, s a ctr számláló értékét nullázzuk. Az E blokk-kódolóval történő CTR módú rejtjelezésre a $ctr(E)$ jelölést használjuk.

Az alábbiakban mutatjuk, hogy a CTR mód biztonságos: a biztonságos E kódoló elemet biztonságosan terjeszti ki hosszabb üzenetekre. Az alábbi állítást látjuk be a CTRC mód esetére:

17.5. Tétel. Legyen $E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^N$. CTRC mód esetén

$$\Delta_{ctr(E)}^{lor-cpa}(t, q, \mu) \leq 2 \cdot \Delta_E^{prf-cpa}(t, q', nq'), \quad (17.18)$$

ahol $q' = \mu/N$, $\mu < N2^n$.

(A téTEL megengedi, hogy a E blokkrejtjelező hossznövelő legyen, azaz az $N > n$ esetet is.)

Bizonyítás: A bizonyítás egyik fontos ötlete az, hogy szétválasztjuk a protokoll (CTR mód) biztonságosságának vizsgálatát a kódoló transzformáció biztonságosságától egy $R : \{0, 1\}^n \rightarrow \{0, 1\}^N$ véletlen függvény felhasználásával.

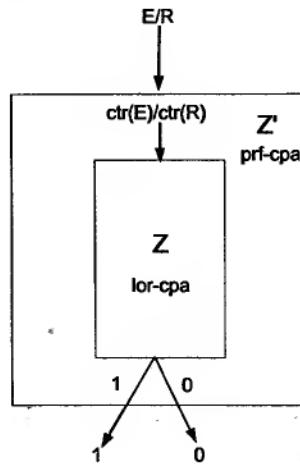
Legyen Z egy $lor - cpa$ -támadó a $ctr(E)$ rejtjelezéssel szemben. Erre redukálja Z' $prf - cpa$ -támadó a saját feladatát, amely az E rejtjelező megkülönböztetése az R véletlen függvénytől: Z' kódolási kéréseket küldhet egy g orákulumnak, amely vagy E vagy R leképezés szerint számol (17.2. ábra).

Z' támadó szimulál egy $lor - cpa$ -orákulumot Z számára, azaz Z' válaszol Z orákulum-kéréseire: Z' támadó kisorsol egy b bináris értéket, s a Z kéréssében érkező (m_0, m_1) üzenetpárból Z' b értékének megfelelően kiválasztja az egyik üzenetet, majd saját g orákulumának felhasználásával szimulálja Z orákulumát. A szimulált orákulum válaszát továbbítja Z felé.

Z döntése alapján Z' a következő döntést hozza g orákulum tartalmáról: ha Z eltalálta b értékét, akkor Z' úgy dönt, hogy az E leképezést futtatja a g orákulum (Z' kimenete 1), ha pedig Z döntése sikertelen, Z' arra dönt, hogy az R véletlen leképezést futtatja a g orákulum (Z' kimenete 0).

Belájtuk, hogy

$$\Delta_{ctr(E)}^{lor-cpa}(Z) = 2 \cdot \Delta_E^{prf-cpa}(Z'), \quad (17.19)$$



17.2. ábra. Redukció (17.5. téTEL)

ahonnan (17.18) formula már következik, mivel (17.19) két oldalán Z szeménti maximumot képezve, a jobb oldal legfeljebb akkora lesz, mint $\max_V [2 \cdot \Delta_E^{prf-cpa}(V)]$.

$$\begin{aligned}
 & \Pr\{prf - cpa - 1_E(Z') = 1\} = \\
 &= \frac{1}{2} \Pr\{lor - cpa - 1_{ctr(E)}(Z) = 1\} \\
 &\quad + \frac{1}{2} \Pr\{lor - cpa - 0_{ctr(E)}(Z) = 0\} \\
 &= \frac{1}{2} \Pr\{lor - cpa - 1_{ctr(E)}(Z) = 1\} \\
 &\quad + \frac{1}{2} (1 - \Pr\{lor - cpa - 0_{ctr(E)}(Z) = 1\}) \\
 &= \frac{1}{2} + \frac{1}{2} (\Pr\{lor - cpa - 1_{ctr(E)}(Z) = 1\} \\
 &\quad - \Pr\{lor - cpa - 0_{ctr(E)}(Z) = 1\}) \\
 &= \frac{1}{2} + \frac{1}{2} \Delta_{ctr(E)}^{lor-cpa}(Z).
 \end{aligned}$$

Hasonlóan kapjuk, hogy

$$\Pr\{prf - cpa - 0_E(Z') = 1\} = \frac{1}{2} + \frac{1}{2} \Delta_{ctr(R)}^{lor-cpa}(Z),$$

ahonnan

$$\begin{aligned}\Delta_E^{prf-cpa}(Z') &= \Pr\{prf - cpa - 1_E(Z') = 1\} \\ &\quad - \Pr\{prf - cpa - 0_E(Z') = 1\} \\ &= \frac{1}{2}\Delta_{ctr(E)}^{lor-cpa}(Z) - \frac{1}{2}\Delta_{ctr(R)}^{lor-cpa}(Z) \\ &= \frac{1}{2}\Delta_{ctr(E)}^{lor-cpa}(Z),\end{aligned}$$

ahol felhasználtuk, hogy

$$\Delta_{ctr(R)}^{lor-cpa}(Z) = 0. \quad (17.20)$$

(17.20) állítás intuitíve is látható: ha véletlen függvényel rejtjelezünk CTR módban, akkor tulajdonképpen one-time-pad kulcsfolyammal rejtjelezünk mindenkor, amíg nem merítettük ki a teljes számlálóteret. Innen (17.19) felhasználásával (17.18) állítás adódik. \square

A $lor - cpa$ -biztonság fogalom erejét a fenti blokkkódolási mód analízis példák mellett az alábbi két téTEL is jól mutatja. Az első eredmény szerint $lor - cpa$ -biztonság garantálja a rejtjelező biztonságát kulcsvisszafejtő támadással szemben, míg a második eredmény ugyanezt garantálja ismert rejtett szöveg alapján, nyílt szöveget fejtő támadással szemben.

17.6. TétEL. *Ha egy szimmetrikus kulcsú rejtjelező $lor - cpa$ -biztonságos, akkor biztonságos kulcsvisszafejtő támadással szemben, azaz kr-biztonságos is.*

Bizonyítás: Indirekt bizonyítással tegyük fel, hogy Z támadó sikeres az E_K rejtjelező K kulcsának visszafejtésében. Z alapján egy Z' $lor - cpa$ -támadó konstruálható az alábbi módon:

17.4. Algoritmus.

1. Z' támadó a Z kulcsfejtő algoritmust futtatja.
2. A Z kulcsfejtő algoritmus m választott üzenetre kéri a rejtjelezett üzenetet.
3. Z' az (m, m) üzenetpárt küldi az $E_K \circ \partial_b$ orákulumnak, s az $E_K \circ \partial_b(m, m)$ orákulum-választ adja válaszként a Z algoritmusnak.
4. Z kimenetként K' becsült kulcsot adjja.
5. Z' az (m_0, m_1) üzenetpárt küldi az $E_K \circ \partial_b$ $lor - cpa$ -orákulumnak, s az orákulum válasza alapján a következő döntést hozza: ha $D_{K'}(E_K \circ \partial_b(m_0, m_1)) = m_1$, akkor Z' kimenete 1, egyébként 0.

Bevezetve a $B = \{K' = K\}$ eseményt, Z támadó sikeressége azt jelenti, hogy $\Pr\{B\} \geq \varepsilon$, ahol ε nem elhanyagolható. Innen

$$\begin{aligned} & |\Pr\{\text{lor} - \text{cpa} - 1_E(Z') = 1\} - \Pr\{\text{lor} - \text{cpa} - 0_E(Z') = 1\}| = \\ &= |(\Pr\{\text{lor} - \text{cpa} - 1_E(Z') = 1|B\} \\ &\quad - \Pr\{\text{lor} - \text{cpa} - 0_E(Z') = 1|B\}) \cdot P\{B\} \\ &\quad + (\Pr\{\text{lor} - \text{cpa} - 1_E(Z') = 1|\bar{B}\} \\ &\quad - \Pr\{\text{lor} - \text{cpa} - 0_E(Z') = 1|\bar{B}\}) \cdot \Pr\{\bar{B}\}| \\ &= |(1 - 0) \cdot \Pr\{B\} + (2^{-n} - 2^{-n}) \cdot \Pr\{\bar{B}\}| = \Pr\{B\} \geq \varepsilon, \end{aligned}$$

azaz E_k nem lenne $\text{lor} - \text{cpa}$ -biztonságos, ami ellentmond a feltételünknek. \square

17.7. Tétel. *Ha egy E_K szimmetrikus kulcsú rejtjelező $\text{lor} - \text{cpa}$ -biztonságos, akkor biztonságos ismert rejtett szöveg alapján nyílt szöveget fejtő támadással szemben, azaz pr-biztonságos is.*

Bizonyítás: (Vázlat) Tegyük fel, hogy állításunkkal ellentében létezik egy Z nyílt szöveg fejtő támadó, amelynek sikervalószínűsége nem elhanyagolható. Z alapján egy Z' $\text{lor} - \text{cpa}$ -támadó konstruálható, ahol Z' kimenete 1, ha $Z[E_K \circ \partial_b(m_0, m_1)] = m_1$, egyébként Z' kimenete 0. \square

Hozzáférés kódoló és dekódoló orákulumhoz is ($\text{lor} - \text{cca}$). Választott rejtett szövegű támadás (cca) esetén megengedjük, hogy a támadó ne csak bal-vagy-jobb rejtjelező orákulumhoz férjen hozzá, de általa választott rejtett szöveghez is kérhesse a nyílt szöveget. Ezen utóbbit azon – természetes – megszorítás mellett teheti, hogy olyan rejtett szöveget adhat csak meg kérés-ként, amely korábban nem szerepelt egy nyílt szövegű kérésre adott rejtjelező orákulum-válaszként. Z támadó tehát az $E_K \circ \partial_b$ és D_K orákulumokhoz fér hozzá, ahol D_K a dekódoló transzformáció.

Ezen $\text{lor} - \text{cca}$ -támadó várhatóan erősebb, mint a $\text{lor} - \text{cpa}$ -támadó. Ezt az alábbi példával is illusztráljuk, megmutatva, hogy meg tudja törni a CTR (Számláló Mód) rejtjelezési módot is.

17.7. Példa. Legyen c az m nyílt szöveg rejtjelezettje. Negáljuk c j -edik bit-jét, amellyel c' rejtett szöveget kapjuk. Ezt a c' rejtett szöveget adjuk fejtésre az orákulumnak, amely m' nyílt szöveget adja válaszul, ahol m és m' a j -edik bitben különböznek. Következésképp az m üzenetet is megismertük anélkül,

hogy olyan rejtett szöveget adtunk volna kérésként az orákulumnak bemenetként, amelyet az kimenetként már korábban előállított. A fejezet elején bevezetett erőforrás-jelöléseinkkel azt kaptuk tehát, hogy

$$\Delta_E^{lor-cpa}(t, 1, n, 1, N) = 1,$$

azaz egy n bit hosszú kérést intéztünk a *lor* (kódoló) orákulumhoz, egy N bites kérést pedig a dekódoló orákulumhoz.



18.

Biztonságos nyilvános kulcsú rejtjelezés

Ebben a fejezetben a nyilvános kulcsú rejtjelezéshez kapcsolódó biztonsági fogalmak kerülnek kifejtésre. A nyilvános kulcsú rejtjelezést egy (G, E, D) hatékony algoritmus-hármasossal definiáljuk, ahol

1. $G(1^n)$ kulcsgeneráló algoritmus outputja a (pk, sk) (nyilvános kulcs, titkos kulcs) kulcspár,
2. E rejtjelező algoritmus $(m \in \{0,1\}^n, pk)$ inputra a $c = E_{pk}(m)$ rejtjeles szöveget számolja ki,
3. D dekódoló algoritmus $(c \in \{0,1\}^n, sk)$ inputra az $m = D(E_{pk}(m), sk)$ nyílt szöveget számolja ki.

A legismertebb nyilvános kulcsú rejtjelező az RSA. Az alábbi példában egy kevésbé ismertet, az ElGamal rejtjelezőt mutatjuk be.

18.1. Példa. (ElGamal rejtjelező): Tekintsünk egy q elemű G csoportot g generátor elemmel. A kulcsgeneráló algoritmus outputja $(pk = X, sk = x)$, ahol $X = g^x$, továbbá x véletlenszerűen kerül kiválasztásra az $S = \{1, 2, \dots, q\}$ halmazból ($x \leftarrow_r S$). Egy $m \in G$ üzenet rejtjelezettje

$$E_{pk}(m) = (Y, \alpha),$$

ahol $Y = g^y$, $y \leftarrow_r S$, $\alpha = Km$, $K = X^y$. A rejtett üzenet dekódolása

$$D_{sk}(Y, \alpha) = \alpha/K = m,$$

ahol $K = Y^x$.

Az ElGamal rejtjelező biztonsági analízisére a fejezetben visszatérünk. ♣

A nyilvános kulcsú rejtjelezők alapvető biztonsági fogalmai a következők:

1. szemantikai biztonság (*ss*, semantic security),
2. üzenet-megkülönböztetés biztonság (*ind*, indistinguishability),
3. rejtjeles szöveg módosíthatatlanság biztonság (*nm*, non-malleability).

Az üzenet-megkülönböztetés biztonság lényegében megfelel a szimmetrikus kulcsú rejtjelezők üzenet-megkülönböztetés biztonság (*lor*) fogalmának. Mivel inkább az *ind* rövidítés használatos az üzenet-megkülönböztetés biztonság fogalomra nyilvános kulcsú esetben, ezért mi is ezt a megjelölést alkalmazzuk.

A biztonsági fogalmakon belüli támadási modellek két fő típusa a – szimmetrikus kulcsú rejtjelezés kapcsán már taglalt – *cpa* (nyílt szöveg választás alapú támadás) és *cca* (rejtett szöveg választás alapú támadás). A *cca* modellen belül megkülönböztetjük a *cca1* és a *cca2* modellt. A *cca2* modellben a támadó adaptívan küldheti orákulum-kéréseit, azaz következő kérése az előző kérésekhez kapott orákulum-válaszoktól is függhet, ezért szokás adaptív választott rejtett szövegű támadásnak nevezni. A *cca1* modell esetén a támadó egyben küldi el összes orákulum-kérését, amely kérések nem függhetnek az adott támadás során általa kapott rejtett vagy fejtett mennyiségektől.

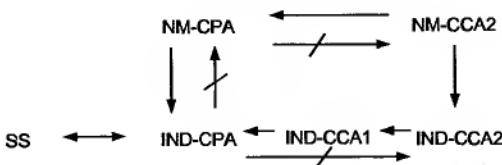
18.2. Példa. Az ElGamal rejtjelező nem biztonságos adaptív választott rejtett szövegű támadás (*cca2*) ellen. Ha a támadó kezébe kerül egy $E_{pk}(m) = (Y, \alpha)$ rejtjelezett üzenet, akkor egy $(Y, g\alpha)$ kérést küldhet a dekódoló orákulumnak, amely azt a $gm \in G$ üzenetbe dekódolja. Ennek alapján az m üzenet már könnyen kiszámítható. ♣

A 18.1. ábra szemlélteti a biztonság-definíciók implikációs gráfját. Láthatjuk például, hogy az *nm* – *cca2* biztonságfogalom a legerősebb (a vizsgáltak közül), mivel ennek teljesülése maga után vonja a többi biztonságfogalom szerinti biztonságot is.

A továbbiakban sorra vesszük a fenti biztonság-definíciókat.

18.1. Szemantikai biztonság

Shannon ideális rejtjelezője esetén, lehallgatott rejtjeles szöveg ismeretében a nyílt szövegre vonatkozóan tetszőlegesen nagy számítási kapacitás mellett sem tudunk meg a priori ismereteinknél több információt. Ennek a fogalomnak erőforrás-megszorítás mellett megfelelője a szemantikai biztonság fogalma.



18.1. ábra. Nyilvános kulcsú rejtjelezés biztonság fogalmai

Legyen $g : M \rightarrow \{0,1\}^*$ a részinformáció leképezés az üzenetek M teréből a véges bináris sorozatok terébe. Például $g(m) = 1$, $m \in \{0,1\}^n$, ha az üzenetben "111" részsorozat előfordul.

Kisorsolunk egy m üzenetet az üzenetek teréből, a priori a támadó által ismert eloszlás szerint. Szemantikai biztonság fennállása esetén, a támadó semmilyen g részinformációs függvény esetén sem tudja pontosabban becsülni $g(m)$ értékét azon esetben, ha megtudja az m üzenet $c = E_{pk}(m)$ rejtjelezettjét, ahhoz képest, mintha ezen rejtjeles üzenetet nem ismerné. Formálisan:

18.1. Definíció (szemantikai biztonság). A (G, E, D) nyilvános kulcsú rejtjelezés $(t(n), \varepsilon(n))$ szemantikailag biztonságos, ha nyílt üzenetek tetszőleges X eloszlása, tetszőleges g részinformációs függvény és tetszőleges $t(n)$ erőforrás-korlátú hatékony Z támadó algoritmus esetén létezik azonos erőforrás-korlátú hatékony Z' támadó algoritmus, hogy a

$$\Pr_{\substack{m \leftarrow X \\ (pk, sk) \leftarrow G(n)}} \{Z(1^n, E(m, pk), pk) = g(m)\} = \Pr_{\substack{m \leftarrow X \\ (pk, sk) \leftarrow G(n)}} \{Z'(1^n, pk) = g(m)\} \quad (18.1)$$

differencia nem nagyobb, mint $\varepsilon(n)$, ahol Z , illetve Z' algoritmusok mögött zárójelben azok inputja áll. Továbbá, a valószínűség \Pr jele alatt azon valószínűségi változókat jelöltük, amelyek felett a valószínűséget számítjuk. Így a (18.1) kifejezésben m üzenet valószínűségi változó az X eloszlás szerint kerül kiválasztásra, továbbá a kulcs-pár véletlenszerűen kerül kiválasztásra a G hatékony kulcsgeneráló algoritmus outputjaként az 1^n (biztonsági paraméter) inputra.

Vegyük észre, hogy (18.1) baloldala nemnegatív, miután Z algoritmusok halmaza magába foglalja Z' algoritmusok halmazát. A Z' algoritmust szokás szimulációs algoritmusnak is hívni.

Az alábbi példa egy fontos tényre hívja fel a figyelmet.

18.3. Példa. Tekintsünk egy nyilvános kulcsú rejtjelezést, amelynél E determinisztikus leképezés, továbbá a nyílt üzenetek X eloszlása konstans méretű (azaz n -től független méretű) üzenethalmazra koncentrált, például $M = \{0, 1\}$. Ez a rejtjelezés nem lehet szemantikailag biztonságos. Ugyanis Z támadó 1 valószínűséggel sikeres tud lenni, ha – a nyilvános kódoló kulcs, pk ismeretében – megkeresi azon üzenetet, amelynek a rejtjelezettjét megismerte. Ez az észrevétel is a véletlenítés rejtjelezésben betöltött szerepének fontosságát jelzi. ♣

18.2. Üzenet-megkülönböztethetetlenség biztonság

18.2.1. Megkülönböztethetetlenség dekódoló orákulumhoz való hozzá-férés nélkül ($ind - cpa$)

Szóban kifejtve: egy (G, E, D) nyilvános kulcsú rejtjelezés üzenet-megkülönböztető támadással szemben biztonságos, ha a támadó nem tud megadni egy olyan üzenetpárt, amelyből visszakapja az egyik, véletlenszerűen ki-választott üzenet rejtjelezettjét, s nem elhanyagolható valószínűséggel meg tudja állapítani, hogy az melyik üzenethez tartozik.

18.2. Definíció (üzenet-megkülönböztethetetlenség). *Egy (G, E, D) nyilvános kulcsú rejtjelezés üzenet-megkülönböztető támadással szemben $(t(n), \epsilon(n))$ -biztonságú, ha tetszőleges $m_0, m_1 \in \{0, 1\}^n$ üzenetpár, tetszőleges, $t(n)$ erőforrás-korlátú hatékony Z támadó algoritmus esetén*

$$\Pr_{(pk, sk) \leftarrow G(n)} \{Z(1^n, m_0, m_1, E(m_1, pk), pk) = 1\} - \Pr_{(pk, sk) \leftarrow G(n)} \{Z(1^n, m_0, m_1, E(m_0, pk), pk) = 1\} \leq \epsilon(n) \quad (18.2)$$

Nem nehéz belátni, hogy

$$\Pr_{\substack{b \in \{0, 1\} \\ (pk, sk) \leftarrow G(n)}} \{Z(1^n, m_0, m_1, E(m_b, pk), pk) = b\} \leq 1/2 + \epsilon(n)/2. \quad (18.3)$$

Ugyanis – egyszerűsített jelölésekkel – kiindulva (18.2) egyenlőtlenség bal oldalából:

$$\begin{aligned}
& \Pr\{Z(.E(m_0.)) = 0 | b = 0\} \cdot \Pr\{b = 0\} \\
& + \Pr\{Z(.E(m_1.)) = 1 | b = 1\} \cdot \Pr\{b = 1\} \\
& = [1 - \Pr\{Z(.E(m_0.)) = 1\}] \cdot \Pr\{b = 0\} \\
& + \Pr\{Z(.E(m_1.)) = 1\} \cdot \Pr\{b = 1\} \\
& = 1/2 + 1/2 \cdot [\Pr\{Z(.E(m_1.)) = 1\} - \Pr\{Z(.E(m_0.)) = 1\}] \\
& \leq 1/2 + \epsilon(n)/2,
\end{aligned}$$

ahonnan átrendezéssel (18.3) alakra juthatunk.

Az ElGamal rejtjelező és az ind – cpa-biztonság. Legyen p egy prímszám, s tekintsük a mod p szorzócsoportot. A támadó az m_0 üzenetet kvadratikus maradéknak ($q.r.$), az m_1 üzenetet nem kvadratikus maradéknak ($n.q.r.$) választja, azaz a $z^2 = m_0 \pmod p$ egyenletnek van, míg a $z^2 = m_1 \pmod p$ egyenletnek nincs megoldása a csoportban. A támadó rendelkezésére áll g^x nyilvános kulcs, továbbá az ind – cpa-biztonság szerint egy $(g^y, g^{x,y}m_i)$ rejtett üzenet.

A kvadratikus maradékokkal kapcsolatosan hivatkozunk az alábbi tulajdonságokra:

Két csoportbeli elem szorzata akkor és csak akkor kvadratikus maradék, ha minden két elem kvadratikus maradék. $g^{x,y}$ akkor és csak akkor kvadratikus maradék, ha $X = g^x$ vagy $Y = g^y$ kvadratikus maradék. Könnyű feladat az adott csoportban a kvadratikus maradék tulajdonság eldöntése.

Ezek alapján a támadó a 18.1. táblázat szerint sikeresen támad.

g^{xy}	$g^{x,y}m_i$	m_i
$q.r.$	$q.r.$	$q.r. \rightarrow m_0$
$q.r.$	$n.q.r.$	$n.q.r. \rightarrow m_1$
$n.q.r.$	$q.r.$	$n.q.r. \rightarrow m_1$
$n.q.r.$	$n.q.r.$	$q.r. \rightarrow m_0$

18.1. táblázat. Támadás az ElGamal rejtjelezés ellen

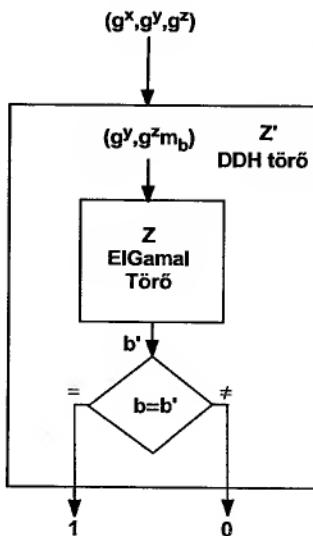
Ha azonban egy $G = \langle g \rangle$ csoportban a Diffie–Hellman döntési probléma (DDH-probléma) nehéz, akkor ezen csoport felett az ElGamal rejtjelező már ind – cpa-biztonságú, ahol a DDH-probléma a következő:

18.3. Definíció (DDH-probléma). *Tekintsünk egy $G = \langle g \rangle$ csoportot, s adott (g^x, g^y, g^z) hármas esetén el kell döntení, hogy $z = x \cdot y \pmod{|G|}$ összefüggés fennáll-e a kitevők között.*

Az $ind - cpa$ -biztonság bizonyítása redukcióval végezhető (18.2. ábra): Legyen Z egy támadó, amely sikeresen töri az ElGamal rejtjelezőt. Konstrálunk egy Z' támadót, amely „törí” a DDH-problémát a következőképp:

18.1. Algoritmus.

1. Z' támadó Z támadónak g^x csoportelemet adja, mint nyilvános kulcsot.
2. Z támadó előállít egy m_0, m_1 üzenet-párt.
3. Z' kisorsol egy b bitet, s rejtjeles üzenetként $(g^y, g^z m_b)$ elemet küldi Z -nek.
4. Ha Z outputja megegyezik b bittel, akkor Z' outputja 1, azaz Z' döntése az, hogy (g^x, g^y, g^z) egy DDH-hármas, egyébként Z' outputja 0.



18.2. ábra. Redukció (ElGamal rejtjelezés)

A szemantikai biztonság és az $ind - cpa$ -biztonság ekvivalensek. A (18.1) és (18.2) definíciók egybevetéséből nem nehéz látni az $ss \rightarrow ind - cpa$ implikációt:

18.4. Tétel. $ss\text{-biztonság} \rightarrow ind\text{-}cpa\text{-biztonság}$

Bizonyítás: A (18.1) formulában legyen $g(m_0) = 0$, $g(m_1) = 1$, továbbá legyen X üzeneteloszlás egyenletes az $\{m_0, m_1\}$ halmazon. Ekkor egy Z ss -támadó egyben ind -támadó is. A (18.1) formula alapján ugyanis

$$\begin{aligned} & \Pr_{\substack{b \in \{0,1\} \\ (pk, sk) \leftarrow G(n)}} \{Z(1^n, m_0, m_1, E(m_b, pk), pk) = b\} \\ & \leq \Pr_{\substack{b \in \{0,1\} \\ (pk, sk) \leftarrow G(n)}} \{Z'(1^n, pk) = b\} + \varepsilon(n), \end{aligned} \quad (18.4)$$

ahol észrevehetjük, hogy az egyenlőtlenség jobb oldalának első tagja $1/2$, ugyanis b valószínűségi változót az orákulum egyenletes eloszlással sorsolja.

□

Ellenkező irányban is fennáll az implikáció:

18.5. Tétel. $ind\text{-}cpa\text{-biztonság} \rightarrow ss\text{-biztonság}$

Bizonyítás: Indirekt bizonyítás. Legyen Z egy ss -támadó, amely (t, ε) -töri a rejtjelezőt, aza (18.1) különbség $(> \varepsilon)$, tetszőleges, az erőforráskorlátnak eleget tevő Z' szimulációs algoritmus esetén. Megmutatjuk, hogy Z használható $ind\text{-}cpa$ -támadásra. A kapocs a kétféle biztonság-definíció között az az észrevétel, hogy az ss -biztonság definíciójábeli Z' szimulációs algoritmusra fennáll a

$$Z'(1^n, pk) = Z'(1^n, E(0, pk), pk) \quad (18.5)$$

egyenlőség, hiszen $E(0, pk)$ előállítható kizárolag pk ismeretében. Mivel (18.1) különbség $(> \varepsilon)$ tetszőleges, erőforráskorlátnak eleget tevő Z' szimulációs algoritmus esetén fennáll, így fennáll akkor is, ha szimulációs algoritmusként Z' helyett Z algoritmust használjuk. Formalizáljuk ezen észrevételeket. Tömörítés céljából legyen

$$U(v) = \{Z(1^n, E(v, pk), pk) = g(v)\},$$

$$W(v) = \{Z(1^n, E(0, pk), pk) = g(v)\}.$$

A sikeresnek feltételezett ss -támadás kapcsán (18.5) figyelembevételével):

$$\Pr_{\substack{m \leftarrow X \\ (pk, sk) \leftarrow G(n)}} \{U(m)\} - \Pr_{\substack{m \leftarrow X \\ (pk, sk) \leftarrow G(n)}} \{W(m)\} > \varepsilon(n),$$

azaz

$$\sum_{m'} \Pr\{m'\} \left(\Pr_{(pk, sk) \leftarrow G(n)} \{U(m')\} - \Pr_{(pk, sk) \leftarrow G(n)} \{W(m')\} \right) > \varepsilon(n),$$

ahonnan a skatulya-elv alapján létezik pozitív valószínűségű m^* üzenet, hogy

$$\Pr_{(pk, sk) \leftarrow G(n)} \{U(m^*)\} - \Pr_{(pk, sk) \leftarrow G(n)} \{W(m^*)\} > \varepsilon(n). \quad (18.6)$$

18.2. Algoritmus. Definiáljuk a $Z^* \text{ ind} - \text{cpa}$ támadó algoritmust következőképp:

1. Z^* inputja $(1^n, m_0 = 0, m_1 = m, E(m_b, pk), pk)$.
2. Z^* meghívja Z algoritmust $(1^n, E(m_b, pk), pk)$ inputtal.
3. Z^* algoritmus outputja 1, ha Z outputja $g(m)$, egyébként Z^* outputja 0.

Ekkor $Z^* \text{ ind} - \text{cpa}$ -támadó $\varepsilon/2$ -nél nagyobb sikervalószínűségű lenne. \square

Felmerül a kérdés: hogyan konstruálunk $\text{ind} - \text{cpa}$ - vagy ss -biztonságú rejtjelező algoritmust? Fentebb láttunk egy ElGamal alapú megoldást. Az alábbiakban először egy – elsősorban elméleti szempontból érdekes – konstrukciót mutatunk be, amely hasonlóan az álvéletlen-generátor alapkonstrukcióhoz, egyirányú függvény keménybitjére támaszkodik.

Az álvéletlen függvény kapcsán bemutatott konstrukció egyirányú permutációra támaszkodik. Az alábbi rejtjelező algoritmus csapda egyirányú permutációt tételez fel, s annak keménybitje felhasználásával 1 bit üzenetet rejtjelez.

A Goldwasser–Micali rejtjelező algoritmus. Adott $\{E_{pk} : X_{pk} \rightarrow X_{pk}\}_{pk \in PK}$ csapda permutáció együttes. Véletlenszerűen választott $m \in \{0, 1\}$ bitet rejtjelezünk az alábbi módon:

18.3. Algoritmus.

1. Kulcsgenerálás: a G algoritmus futtatásával a (pk, sk) nyilvános-titkos kulcspár képzése.
2. Rejtjelezés: $E'(pk, m) = [E_{pk}(x), m \oplus b_{pk}(x)]$, ahol az $x \in X_{pk}$ elemet véletlenszerűen választjuk.
3. Dekódolás: $D'(pk, sk, E'(pk, m)) = m$, részletezve:

$$b_{pk}(D(sk, E_{pk}(x)) \oplus (m \oplus b_{pk}(x))) = m.$$

Azaz a rejtjeles üzenetet úgy képezzük, hogy a csapda permutáció ősképtérből egy véletlenül választott x elemet leképezzünk a csapda permutációval, majd ezt meghosszabbítjuk egy bittel, amely az üzenetbit és a véletlen ősképhez tartozó keménybit mod 2 összege.

Az m üzenetbitről feltettük, hogy egyenletes eloszlással választottuk, elenkező esetben (lévén, hogy az üzeneteloszlás a támadó a priori ismeretei között is megtalálható), kizártlag az üzeneteloszlás ismeretében is hozhatna sikeres döntést a támadó.

Ezen rejtjelezés $ind - cpa$ -biztonságosságát alább igazoljuk. Heurisztikus gondolatmenetünk az, hogy a $b_{pk}(x)$ keménybitet kellene ahhoz ismernünk, hogy az üzenetbitet dekódoljuk. Ezen keménybitre vonatkozó pontos információt tartalmazza $E_{pk}(x)$, de ennek alapján csak az tudja ezen bitet kinyerni hatékony algoritmussal, aki a csapda-információ, azaz dekódoló kulcs birtokában van. Az egy bitnyi üzenet kódolása s bites $m = (m_1, m_2, \dots, m_s)$ bináris üzenetekre kiterjeszthető minden egyes bitre, sorban elvégezve az egy bites kódolást.

A Goldwasser–Micali-rejtjelezésnek elsősorban elméleti jelentősége van. Gyakorlati haszna kicsi az 1 üzenetbitre jutó nagy műveletigény, a drasztikus hossznövekedés, valamint amiatt, hogy üzenetbitenként annak sokszorosa mennyisége véletlen bitet igényel (az x véletlen választása kapcsán). Mielőtt elsierte a következetést, a one-time-pad véletlenbit „pazarláshoz” hasonlítanánk ezt a rejtjelezést, vegyük észre, hogy ugyan valóban elő kell tudni állítani a nagy mennyisége véletlen bitet a kódoló oldalon, de az nem játszik kulcsnak megfelelő szerepet, s így annak másolatát nem kell előzetesen eljuttatnunk a dekódoló oldalra.

18.6. Tétel. A Goldwasser–Micali-algoritmus $ind - cpa$ -biztonságú.

Bizonyítás:

$s = 1$ eset (egy bites üzenet kódolása):

Az üzenet-megkülönböztethetetlenség biztonságot az üzenet-párok rejtjeles alakjainak megkülönböztethetetlenségével definiáltuk. Esetünkben összesen két üzenet van, $m_0 = 0$ és $m_1 = 1$, így ezek rejtjelezettjei megkülönböztethetetlenségét kell bizonyítanunk. Indirekt bizonyításként tegyük fel, hogy a téTEL állítása nem igaz, azaz létezik $t(n)$ erőforrás-korlátú Z támadó, és létezik $p(n)$ polinom, hogy végtelen sok n esetén

$$\Pr_{\substack{m \in \{0,1\} \\ (pk, sk) \leftarrow G(1^n) \\ x \leftarrow {}_r X_{pk}(1^n)}} \{Z(1^n, E_{pk}(x), m \oplus b_{pk}(x), pk) = m\} > 1/2 + 1/p(n). \quad (18.7)$$

Bevezetve a $c = m \oplus b_{pk}(x)$ új változót, amely – a one-time-pad elven – ugyancsak egyenletes eloszlású, x változótól független bináris valószínűségi változó:

$$\Pr_{\substack{c \in \{0,1\} \\ (pk, sk) \leftarrow G(1^n) \\ x \leftarrow_r X_{pk}(1^n)}} \{Z(1^n, E_{pk}(x), c, pk) = c \oplus b_{pk}(x)\} > 1/2 + 1/p(n),$$

ahonnan elemi átrendezéssel

$$\Pr_{\substack{c \in \{0,1\} \\ (pk, sk) \leftarrow G(1^n) \\ x \leftarrow_r X_{pk}(1^n)}} \{Z(1^n, E_{pk}(x), c, pk) \oplus c = b_{pk}(x)\} > 1/2 + 1/p(n)$$

alakot kapjuk. Ennek alapján létezik egy $Z'(1^n, E_{pk}(x), pk)$ algoritmus, amelyik véletlenszerűen kisorsol egy c bitet, majd Z algoritmus futtatásával ki-számítja $Z(1^n, E_{pk}(x), c, pk) \oplus c$ outputot, hogy

$$\Pr_{\substack{(pk, sk) \leftarrow G(1^n) \\ x \leftarrow_r X_{pk}(1^n)}} \{Z'(1^n, E_{pk}(x), pk) = b_{pk}(x)\} > 1/2 + 1/p(n),$$

ami viszont ellentmond annak, hogy b keménybit leképezés.

$s > 1$ eset: (vázlat)

A hibrid bizonyítástechnika segítségével igazolhatjuk az általános esetet, ahol az $m = (m_1, m_2, \dots, m_s)$, $m' = (m'_1, m'_2, \dots, m'_s)$ üzenetpárhoz tartoznak az extrém hibridek, s az általános, k -adik hibrid alakja

$$H^k = E(pk, m_1), \dots, E(pk, m_k), E(pk, m'_{k+1}), \dots, E(pk, m'_s),$$

$k = 0, 1, \dots, s$, ahol $E(pk, m)$ az egy bites rejtelezés. A 14.3. téTEL bizonyítási módszerét adaptálva, ellentmondásra jutunk az egy bites rejtelezés $ind - cpa$ -biztonsága vonatkozásában. \square

A Goldreich–Levin egyirányú permutációkra vonatkozó keménybit-tétel csapda permutációkra vonatkozó élesítése felhasználásával az RSA algoritmusra alapozva megkonstruálhatjuk az első, $ind - cpa$ értelemben bizonyíthatóan biztonságos – de még továbbra is csak elvi jelentőségű – rejtelezőket.

1 bit ind – cpa-biztonságú rejtjelezése RSA algoritmussal.

Tekintsük az RSA algoritmust. Egy m üzenetbit rejtjelezéséhez először válasszunk véletlenszerűen az RSA ősképteréből egy x bitsorozatot, valamint ugyanilyen méretű, ugyancsak véletlenszerűen választott r bitsorozatot, s hajtsuk végre az alábbi műveletet:

$$E'((n, e), m) = [r, E_{(n, e)}(x), m \oplus \langle r, x \rangle]. \quad (18.8)$$

Speciálisan az RSA algoritmus esetén tudjuk, hogy az üzenet legkisebb helyiértékű (lsb) bitje keménybit, azaz $b_{(n, e)}(x) = lsb(x)$ keménybit leképezés. Tehát RSA esetén nemcsak a véletlenszerűn választott r bitsorozatok átlagában, de konstans $r^* = (1, 0, \dots, 0)$ mellett $b_{pk}(x) = \langle r^*, x \rangle$ keménybit leképezés. Ezenfelül kis ügyeskedéssel – általános esetben is – le tudunk faragni 1 bitet a rejtjeles blokk méretéből a következő észrevétellel: véletlenszerűen választunk $x \in X_{pk}$ elemet úgy, hogy arra teljesüljön a $b_{pk}(x) = m$ megkötés is, ahol m a rejtjelezendő 1 bites üzenet. A dekódoló ismeri a (nyilvános) keménybit leképezést, s így $E_{pk}(x)$ invertálásával kapott x alapján, $b_{pk}(x)$ kiszámításával dekódolhatja az üzenetet. Ekkor tehát az RSA algoritmus esetére a (18.8) formula szerinti kódolás a következőképp egyszerűsödik:

$$E'((n, e), m) = E_{(n, e)}(x), \quad (18.9)$$

ahol x bitsorozatot véletlenszerűen választjuk, azzal a megkötéssel, hogy $m = lsb(x)$.

18.2.2. Megkülönböztethetetlenség dekódoló orákulumhoz való hozzáféréssel ($ind - cca$)

Intuitíve sejthető, hogy az $ind - cpa$ -biztonság gyengébb, mint az $ind - cca$ -biztonság.

18.7. Tétel. *Létezik olyan nyilvános kulcsú rejtjelező algoritmus, amelyik $ind - cpa$ -biztonságos, de nem $ind - cca2$ -biztonságos.*

Bizonyítás: Tekintsük egy m 1 bites üzenet alábbi Goldwasser–Micali-rejtjelezését RSA felhasználásával ((18.8) formula alapján):

$$E((n, e), m) = [r, x^3 \bmod n, m \oplus \langle r, x \rangle].$$

Az $ind - cca2$ -biztonság támadás modellje adaptív választott szövegű támadás. A dekódoló orákulumhoz (a vak aláírás ötletéhez hasonlóan) küldjük el

$u^3x^3 \pmod n$ elemet a rejtett szövegek teréből, ahol u véletlen elem invertálható $\pmod n$. Ezzel hozzájutunk x elemhez, s innen r ismeretében dekódolni tudjuk az m üzenetet. \square

A Cramer–Shoup-rejtjelező *ind–cca2*-biztonságú. Az ElGamal rejtjelezés nem *ind–cca2*-biztonságú. A Cramer–Shoup-kódolás az ElGamal-kódolás egy ebből a szempontból tökéletesített változata, amely a DDH-probléma nehézségét feltételezi.

Az egyik ötlet az, hogy egy hitelesítő elemet is kiszámol a kódoló. A dekódoló először ellenőrzi ezen hitelesítő elem helyességét, s csak sikeres esetén dekódolja az üzenetet. Ezzel azt kívánja elérni a rejtjelezés, hogy a támadó csak olyan kódolt szöveget küldhessen a dekódolónak, amit a kódoló oráku-lum állított elő. Ezzel lényegében visszaszorítjuk a támadót egy *cpa*-támadás körülményei közé a lehetőségeit illetően, ugyanis így a támadó nem képes a dekódolónak általa fabrikált rejtett szöveget küldeni (a valódi kódoló oráku-lumtól származó rejtett szöveget pedig nincs értelme a dekódolónak küldeni, hiszen ezek a támadó nyílt szöveg kérésére keletkeztek).

A Cramer–Shoup-rejtjelező kódolás a következő: (az ElGamal rejtjelezés-nél bevezetett jelöléseket használjuk):

18.4. Algoritmus.

1. Kulcsgenerálás: Legyenek $g_1, g_2 \in G$, $x_1, x_2, y_1, y_2, z \in Z_q$ véletlen elemek, ahol $q = |G|$. A nyilvános kulcs öt elemű:

$$(g_1, g_2, c = g_1^{x_1} g_2^{x_2}, d = g_1^{y_1} g_2^{y_2}, h = g_1^z, H),$$

ahol H egy univerzális egyirányú hash függvény (UOWHF). A titkos kulcs szintén öt elemű:

$$(x_1, x_2, y_1, y_2, z).$$

2. Rejtjelező kódolás: Legyen $m \in G$ az üzenet. A kódoló algoritmus sorsol egy $r \in Z_q$ véletlen elemet. A rejtett szöveg (u_1, u_2, e, v) , ahol

$$u_1 = g_1^r, u_2 = g_2^r, e = h^r m, v = c^r d^{r\alpha},$$

ahol $\alpha = H(u_1, u_2, e)$, továbbá v a hitelesítő elem.

3. Dekódolás: Egy (u_1, u_2, e, v) rejtett szöveg inputra a dekódoló először el- lenőrzi a v hitelesítő elem helyességét az (x_1, x_2, y_1, y_2) titkos kulcsele-mek és a nyilvános H hash transzformáció felhasználásával, azaz képezi

az $\alpha = H(u_1, u_2, e)$ lenyomatot, majd ellenőrzi az alábbi egyenlőség fennállását:

$$u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha} = v.$$

Ha nem áll fenn az egyenlőség, akkor outputja a „visszutasítva” szöveg, ellenkező esetben elvégzi az alábbi műveletet, amivel dekódolja az üzenetet:

$$m = e/u_1^z.$$

18.8. Tétel. A Cramer–Shoup-rejtjelezés ind – cca2-biztonságú

Bizonyítás: A bizonyítás indirekt. Tegyük fel, hogy Z támadó sikeres $ind - cca2$ -támadó, akkor konstruálható egy Z' támadó, amely nem elhanyagolható sikerrel oldja a DDH-problémát, s ezzel ellentmondásra jutunk. A Z' támadó Z számára szimulálja a kulcsgenerálást, a kódolást és a dekódolást. Az egyszerűbb megkülönböztetés kedvéért alább a Z algoritmust nevezük röviden támadónak, Z' algoritmust szimulátornak (18.3. ábra).

A szimulátor inputja (g_1, g_2, u_1, u_2) , s azt kell megállapítania, hogy ez a negyes véletlen választás (továbbiakban R eloszlás szerinti), vagy $u_1 = g_1^r$, $u_2 = g_2^r$ (továbbiakban D eloszlás szerinti). A szimulátor működése a következő:

18.5. Algoritmus.

1. A szimulátor kulcsgeneráló algoritmusa $x_1, x_2, y_1, y_2, z_1, z_2 \in Z_q$ véletlen elemeket választ, majd kiszámítja a

$$c = g_1^{x_1} g_2^{x_2}, d = g_1^{y_1} g_2^{y_2}, h = g_1^{z_1} g_2^{z_2}$$

kulcselemeket, továbbá véletlenszerűen választ egy H hash függvényt (UOWHF). Így a szimulátor (g_1, g_2, c, d, h, H) nyilvános kulcsot generálja a (Z) támadó számára.

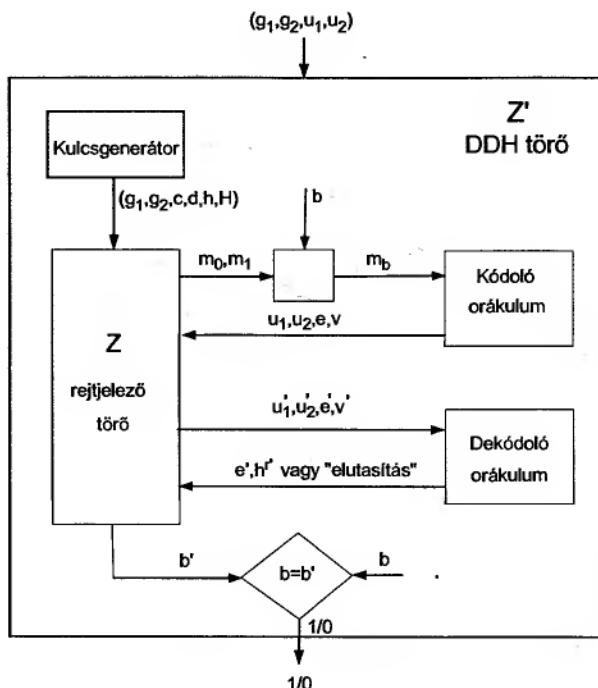
2. A kódoló orákulum a következő: (a támadó által) adott m_0, m_1 üzenetpárhoz a szimulátor véletlenszerűen választ egy b bitet és az

$$e = u_1^z u_2^{z_2} m_b, \alpha = H(u_1, u_2, e), v = u_1^{x_1+y_1\alpha} u_2^{x_2+y_2\alpha}$$

kiszámítása után (u_1, u_2, e, v) rejtett szöveg outputot adja.

3. A dekódoló orákulum a valós dekódolás fentebb bemutatott lépéseiit végzi, kivéve az $m = e/u_1^z$ lépést, amely helyett $m = e/(u_1^{z_1} u_2^{z_2})$ lépést végzi.

Egy $(u'_1, u'_2, e', v') \in G^4$ négyest $\log_{g_1}(u'_1) = \log_{g_2}(u'_2)$ esetén érvényes rejtett szövegnek, ellenkező esetben érvénytelen rejtett szövegnek hívunk.



18.3. ábra. Redukció (Cramer–Shoup-rejtelező)

D eloszlás szerinti szimulátor input. Tegyük fel először, hogy a *D* eloszlás szerint választódott a szimulátor inputja, azaz az input (g_1, g_2, u_1, u_2) , ahol $u_1 = g_1^r$, $u_2 = g_2^r$. Ekkor a kódoló és dekódoló orákulum szimulációja tökéletes: a támadó számára nincs különbség a valós és a szimulált környezete között, következésképp ez esetben az *ind – cca2* szerinti sikervalószínűsége is azonos kell legyen.

A szimulált kódoló pontosan úgy működik, mint a valós esetben: az $r \in Z_q$ véletlen elemet nem kell választania, mivel az inputjaként kapott (g_1, g_2, u_1, u_2) négyesben $u_1 = g_1^r$, $u_2 = g_2^r$ már tartalmazza ezt a véletlen kitevőt. A dekódoló szimulátor – hasonlóan a valóshoz – elhanyagolhatóan kis valószínűségű eseménytől eltekintve visszautasít minden érvénytelen rejtett szöveget.

A (*Z*) támadó nem ismeri az (x_1, x_2, y_1, y_2) titkos kulcslemekeket, de azt tudja, hogy a következő 4 egyenletből álló rendszert elégítik ki:

1. $\log_{g_1} c = x_1 + w x_2$ a c nyilvános kulcslelem alapján.
2. $\log_{g_1} d = y_1 + w y_2$ a d nyilvános kulcslelem alapján.

Egy, a támadó által generált érvénytelen (u'_1, u'_2, e', v') rejtett szöveg csak akkor kerül dekódolásra, ha a dekódoló (orákulum) hiteles elem ellenőrzőjét becsapja, azaz ha

3. $\log_{g_1} v' = r'_1 x_1 + wr'_2 x_2 + \alpha r'_1 y_1 + \alpha r'_2 w y_2$, ahol $\log_{g_1} g_2 = w$, $\log_{g_1} u'_1 = r'_1$, $\log_{g_1} u'_2 = wr'_2$, $r_1 \neq r_2$.

Ha a kódoló orákulumot használja a támadó, annak outputja alapján:

4. $\log_{g_1} v = rx_1 + wrx_2 + \alpha ry_1 + \alpha rwy_2$, amely azonban az első két egyenlettel lineárisan összefüggő, így a kódoló kérdezése nem juttatja többlet információhoz a titkos kulcselemekkel kapcsolatban a támadót.

Természetesen ez az egyenletrendszer ezen lineáris alakban nem áll a támadó rendelkezésére, mivel a diszkrét logaritmusképzés nehéz feladat, de ezen rendszer konkrét megoldása – szerencsére – nem is szükséges a bizonyításhoz.

Az első három egyenlet (x_1, x_2, y_1, y_2) négy ismeretlenet tartalmazza. Gonodatban átrendezve, elimináljuk a harmadik egyenletből például az x_2 , y_2 ismeretleneket az első két egyenlet felhasználásával, s eredményül

$$\log_{g_1} v' = ax_1 + by_1$$

egyenletre jutunk. A támadó nem ismeri a titkos kulcselemeket, ezért annak valószínűsége, hogy ezen utóbbi egyenlőség fennáll (és így v' elfogadásra kerül, mint „helyes” hitelesítő elem), $1/q$, ami elhanyagolhatóan kicsi.

R eloszlás szerinti szimulátor input. Tegyük fel most, hogy *R eloszlás* szerint választódott a szimulátor inputja, azaz az input (g_1, g_2, u_1, u_2) , ahol $u_1 = g_1^{r_1}$, $u_2 = g_2^{r_2}$, s nagy valószínűséggel $r_1 \neq r_2$. Ekkor a támadó elhanyagolhatóan kicsi valószínűségű eseménytől eltekintve nem képes sikeres üzenetmegkülböztetésre. Ezt két lépésben látjuk be. Először megmutatjuk, hogy azon feltétel mellett, hogy a dekódoló orákulum minden érvénytelen rejtett szöveget elutasít, pénzfeldobásnál jobb lehetősége nincs a támadónak a megkülböztetés feladatban. Második lépésként pedig azt, hogy elhanyagolhatóan kicsi valószínűséggel fogad csak el a dekódoló érvénytelen rejtett szöveget.

Tegyük fel tehát, hogy tökéletes a dekódoló orákulum érvénytelen rejtett szöveg elutasító képessége. A (Z) támadó nem ismeri a (z_1, z_2) titkos kulcselemeket, de azt tudja, hogy a következő egyenletekből álló rendszert elégítik ki:

1. $\log_{g_1} h = z_1 + wz_2$ a h nyilvános kulcs alapján.

Feltételezésünk szerint egy, a támadó által generált (u'_1, u'_2, e', v') rejtett szöveg csak akkor kerül dekódolásra, ha az érvényes, azaz

$$(u'_1)^{z_1} (u'_2)^{z_2} = (g_1)^{r' z_1} (g_2)^{r' z_2} = h'$$

alapján

2. $r' \log_{g_1} h = r' z_1 + w r' z_2$ egyenlet fennáll, amely azonban az első egyenlettel lineárisan összefüggő, így további információt nem nyújt a támadónak.

A kódoló orákulum segítségével kapható

$$(u_1, u_2, e, v), e = \gamma \cdot m_b, \gamma = u_1^{z_1} u_2^{z_2}$$

rejtett szöveg alapján

3. $\log_{g_1} \gamma = r_1 z_1 + w r_2 z_2$ egyenlet áll fenn.

Ezen első és harmadik egyenlet lineárisan független, s belőle a

$$\log_{g_1} \gamma = a + b z_2$$

alakú egyenletet kaphatjuk. Mivel z_2 a támadó számára ismeretlen, s véletlen, ezért ugyanez igaz γ vonatkozásában is. Ezt összevetve az m_b üzenet

$$e = \gamma \cdot m_b$$

alakú kódolásával láthatóan one-time-pad típusú kódolást kaptunk, következésképp a támadónak pénzfeldobásnál nincs jobb lehetősége a b bitre hozandó döntésnél.

Most már csak azt kell belátnunk, hogy elhanyagolhatóan kicsi valószínűséggel fogad csal el a dekódoló érvénytelen rejtett szöveget. A módszer az eddigiekhez hasonló. Azt vizsgáljuk, hogy a támadó számára ismeretlen, véletlen választású (x_1, x_2, y_1, y_2) kulcselem négyes milyen egyenletrendszeret elégít ki. Ez az egyenletrendszer a következő:

1. $\log_{g_1} c = x_1 + w x_2$ a c nyilvános kulcs alapján.
2. $\log_{g_1} d = y_1 + w y_2$ a d nyilvános kulcs alapján.
3. $\log_{g_1} v = r_1 x_1 + w r_2 x_2 + \alpha r_1 y_1 + \alpha r_2 w y_2$ a (u_1, u_2, e, v) kódoló output alapján.

A támadó (u'_1, u'_2, e', v') érvénytelen rejtett szövegéről, ahol $\log_{g_1} g_2 = w$, $\log_{g_1} u'_1 = r'_1$, $\log_{g_1} u'_2 = w r'_2$, $r_1 \neq r_2$, $\alpha' = H(u'_1, u'_2, e')$ kapjuk

$$4. \log_{g_1} v' = r'_1 x_1 + w r'_2 x_2 + \alpha r'_1 y_1 + \alpha r'_2 w y_2 - t.$$

Vizsgáljuk a három lehetséges esetet (u'_1, u'_2, e', v') négyessel kapcsolatban:

i.) $(u'_1, u'_2, e') = (u_1, u_2, e), v \neq v'$

Ez nem lehetséges, mivel a hitelesítő leképezés egy függvény.

ii.) $(u'_1, u'_2, e') \neq (u_1, u_2, e), \alpha \neq \alpha'$

Ez esetben a fenti négy egyenletből álló egyenletrendszer együtthatómátrixának determinánsa nemzérus:

$$w^2(r_2 - r_1)(r'_2 - r'_1)(\alpha - \alpha') \neq 0,$$

ezért csak egyetlen (x_1, x_2, y_1, y_2) kulcselem négyes mellett van megoldása. Mivel a kulcselemelek véletlenek a támadó számára, így a sikeres valószínűsége – a fenti egyenletrendszer kielégülése – elhanyagolható valószínűségű.

iii.) $(u'_1, u'_2, e') \neq (u_1, u_2, e), \alpha = \alpha'$

Ez nem lehetséges nem elhanyagolható gyakorisággal, mivel ez ütközés előállítását jelentené, s ellentmondana annak, hogy H hash függvényt UOWHF családból választottuk. Mivel az – enyhébb feltételt jelentő – univerzális hash függvényt (UOWHF) tételeztünk fel az ütközésmentes hash függvény (CRHF) helyett, ezért ezen állítás még magyarázatot igényel. Ha iii.) fennállhatna, akkor támadót kaphatnánk az UOWHF ütközésmentességének megtörésére.

Ennek belátásához módosítsuk a kódoló orákulumunkat úgy, hogy végezze el az összes korábbi lépést, de (u_1, u_2, e, v) rejtjeles szövegbe véglül is véletlenszerűen választott e kerül. Ezen módosított szimuláció a támadó szemében nem különözböztethető meg az eredeti szimuláltortól mindenkorábban, míg a iii.) esemény elő nem áll, azaz amikor a támadó ütközésre jut. Mivel (u_1, u_2, e) , a H függvény argumentuma független H választásától, ezért azt választhatnánk azelőtt is, hogy a H függvényt kiválasztjuk, következetesképp az UOWHF ütközés feltételét is teljesíteni tudnánk nem elhanyagolható valószínűséggel. □

18.3. Rejtjeles szöveg módosíthatatlanság biztonság

Ezen biztonság szerint egy támadó nem tudja a rejtett szöveget úgy módosítani, hogy a módosított rejtett szöveghez tartozó nyílt szöveg szándékainak megfelelően megváltozzon az eredeti rejtett szöveghez tartozó nyílt szöveghez képest. Például egy kulcsfolyamatos rejtjelezés nem nm -biztonságú, mivel ekkor mod 2 összegzéssel adjuk össze a kulcsbitet a nyílt szöveg bittel, s így egy támadó invertálva egy rejtett bitet közvetlenül invertálja a megfelelő nyílt szöveg bitet. Mielőtt formálisan definiálnánk ezen biztonságfogalmat, néhány további motiváló példát mutatunk be.

18.4. Példa. Aukciós ajánlattétel lopása:

Tekintsük a következő aukciós protokollvázat, ahol B az egyik ajánlattevő, S a kikiáltó (eladó). Az ajánlattételi szakasz kezdetének kulcslépése:

$$B \rightarrow S : E_{pk_B}(B, 10000Ft)$$

Az ajánlattételi szakasz lezárása után az ajánlatok felfedésének kulcslépése a következő:

$$B \rightarrow S : B, 10000\text{ Ft}, sk_B,$$

azaz ajánlattételkor az ajánlattevő elküldi a nevét, valamint az ajánlati összeget, saját publikus kulcsával rejtjelezve. Az ajánlattételi szakasz lezárása után felfedi a titkos kulcsát. Tegyük fel, hogy Goldwasser–Micali-rejtjelezőt használunk a bizonyított biztonság kedvéért. Ez a rejtjelező bitenkénti rejtjelezést végez. Ezért az $E_{pk_B}(B, 10000Ft)$ rejtett szöveg $E_{pk_B}(B)$, $E_{pk_B}(10000Ft)$ felekre bontható. Egy C támadó el tudja lopni B ajánlatát, kicsérélve az első felet az $E_{pk_B}(C)$ rejtett szövegre. C támadó tehát anélkül módosította szándékai szerint a rejtett szöveget, hogy megfejtette volna. ♣

18.5. Példa. Távoli pénzfeldobásos döntés:

B és C személyek távol vannak egymástól, egymásban nem bíznak meg, s szeretnék döntésre jutni valamiben. Tegyük fel, hogy a döntést pénzfeldobás jelleggel szeretnék meghozni. Ehhez a következő protokollt használják

(1)	$B \rightarrow C$:	$E_{pk_B}(r)$
(2)	$C \rightarrow B$:	$E_{pk_C}(s)$
(3)	$B \rightarrow C$:	sk_B
(4)	$C \rightarrow B$:	sk_C
(5)	B, C :	$r \oplus s$

ahol s és r bitek 0, 1 értékű pénzfeldobás kimenetelek, továbbá E egy nyilvános kulcsú rejtjelező. A felek megállapodása szerint, ha $r \oplus s = 0$, akkor B nyer, egyébként C nyer. Egy támadó az első vagy a második lépésben a rejt-

jeles szöveget manipulálva, az üzenetbitet invertálva megfordíthatja a végeredményt. A támadónak ekkor sem kell ismernie a tényleges nyílt üzenetet (s vagy r értékét).



18.6. Példa. WEP támadása:

A WEP (Wired Equivalent Privacy) protokollban a kliens a TCP csomagját kulcsfolyamatos rejtjelezést nyújtó RC4 rejtjelező algoritmussal rejtjelezve küldi a WEP bázis állomásnak, amely dekódolja azt, s a nyílt üzenetet küldi tovább a szervernek. Kulcsfolyamatos rejtjelezésnél mod 2 összegzéssel adjuk össze a kulcsbitet a nyílt szöveg bittel, ezért ha a támadó invertál egy rejtett bitet, közvetlenül invertálja a megfelelő nyílt szöveg bitet. A protokoll szerint a dekódolt nyílt üzenet integritás-ellenőrzése nem érzékeny arra az esetre, ha két olyan pozícióban invertáljuk a biteket, ahol azok azonos értékűek (azaz $(0,0)$ pár változtatása $(1,1)$ párra vagy viszont). Ha viszont ellentétes értékű biteken történik az inverzió, azt detektálja (azaz $(1,0)$ pár változtatása $(0,1)$ párra vagy viszont). Integritáshiba detektálása esetén nem küld a szerver nyugtázó üzenetet, egyébként küld. Ennek ismeretében a támadó el tudja dönteni, hogy invertált bitpozíció párokon azonos vagy különböző nyílt szöveg bitek állnak. A támadónak tehát – ismét – nem kell ismernie a tényleges nyílt üzenetet.



Ezen intuitív előkészületek után formálisan definiáljuk az nm -biztonság fogalmát:

18.9. Definíció (nm -biztonság). Egy (G, E, D) nyilvános kulcsú rejtjelezés szöveg módosításával szemben $(t(n), \epsilon(n))$ -biztonságú, ha az üzenetek M tere feletti tetszőleges üzeneteloszlás és üzenetpárok feletti tetszőleges $R : M \times M \rightarrow \{0, 1\}$ reláció esetén, továbbá $t(n)$ erőforrás-korlátú, tetszőleges Z támadóhoz létezik olyan $t(n)$ erőforrás-korlátú Z' támadó, hogy a

$$\Pr_{\substack{m \in M \\ (pk, sk) \leftarrow G(n)}} \{R(m, D_{sk}(Z^O(pk, E_{pk}(m))))\} - \Pr_{\substack{m \in M \\ (pk, sk) \leftarrow G(n)}} \{R(m, D_{sk}(Z'(pk)))\} \quad (18.10)$$

differencia nem nagyobb, mint $\epsilon(n)$, ahol Z^O egy O orákulumhoz hozzáférő Z nm -támadó, amely orákulum nm -cpa-biztonság esetén hiányzik, míg nm -cca-biztonság esetén a D_{sk} dekódolás.

A (18.10) kifejezés értelmében tehát Z támadó $E_{pk}(m)$ rejtjeles szöveget kíván módosítani úgy, hogy a módosított rejtjeles szöveg olyan nyílt szöveg-

nek felel meg, amely R relációban áll az eredeti (m) nyílt szöveggel. A rejtjelező nm -biztonságos, ha ezt csak elhanyagolhatóan nagyobb sikерrel képes tenni, mint az azonos erőforrás-korlátú legerősebb támadó, amelynek nem áll az $E_{pk}(m)$ rejtjeles szöveg a rendelkezésére. Z' algoritmust szokás szimulációs algoritmusnak is nevezni.

Az alábbi két téTEL az nm -biztonság és az ind -biztonság kapcsolatát fejezi ki.

18.10. TétEL. *Ha egy nyilvános kulcsú rejtjelező $nm - O$ -biztonságos, akkor $ind - O$ -biztonságos is, ahol $O \in \{cpa, cca1, cca2\}$.*

Bizonyítás: Tegyük fel, hogy a rejtjelezőt egy Y^O támadó $(t, \varepsilon) - ind$ -töri, ahol $O cpa, cca1$ vagy $cca2$ orákulum, azaz létezik egy (m_0, m_1) üzenetpár, amelyre a

$$\begin{aligned} & \Pr_{(pk, sk) \leftarrow G(n)} \{Y^O(m_0, m_1, E(m_0, pk), pk) = 1\} \\ & - \Pr_{(pk, sk) \leftarrow G(n)} \{Y^O(m_0, m_1, E(m_1, pk), pk) = 1\} \end{aligned} \quad (18.11)$$

differencia nagyobb, mint $\varepsilon(n)$. Tekintsünk az üzenetek M halmaza felett egy eloszlást, amelynél az m_0, m_1 üzenetek valószínűsége $1/2 - 1/2$. Az R reláció legyen a következő: $R(u, v) = 1$, ha $u = v$, illetve $R(u, v) = 0$, ha $u \neq v$. Definiáljuk a

$$Z^O(pk, c) = E_{pk}(m_\eta)$$

nm -támadót, ahol $\eta = Y^O(pk, m_0, m_1, c)$, továbbá c helyére rejtjelezett üzenet kerül. Megmutatjuk, hogy (18.11) fennállása esetén, a rejtjelező nem nm -biztonságos. Meg kell mutatnunk, hogy a Z' legjobb szimulációs algoritmus mellett is a

$$\Pr_{\substack{m \in M \\ (pk, sk) \leftarrow G(n)}} \{R(m, D_{sk}(Z^O(pk, E_{pk}(m))))\} - \Pr_{\substack{m \in M \\ (pk, sk) \leftarrow G(n)}} \{R(m, D_{sk}(Z'(pk)))\} \quad (18.12)$$

differencia nagyobb, mint $\varepsilon(n)/2$. Vegyük észre, hogy a (18.12) különbség második tagja:

$$\Pr_{b \in \{0,1\}} \{m_b = D_{sk}(Z'(pk))\} = 1/2.$$

Az első tag:

$$\begin{aligned}
 & \Pr_{\substack{b \in \{0,1\} \\ (pk, sk) \leftarrow G(n)}} \{R(m, D_{sk}(Z^O(pk, E_{pk}(m_b))))\} = \\
 &= \Pr_{\substack{b \in \{0,1\} \\ (pk, sk) \leftarrow G(n)}} \{Y^O(pk, m_0, m_1, E_{pk}(m_b)) = b\} \\
 &= \frac{1}{2} \Pr_{(pk, sk) \leftarrow G(n)} \{Y^O(pk, m_0, m_1, E_{pk}(m_0)) = 0\} \\
 &\quad + \frac{1}{2} \Pr_{(pk, sk) \leftarrow G(n)} \{Y^O(pk, m_0, m_1, E_{pk}(m_1)) = 1\} \\
 &= \frac{1}{2} \Pr_{(pk, sk) \leftarrow G(n)} \{Y^O(pk, m_0, m_1, E_{pk}(m_1)) = 1\} \\
 &\quad + \frac{1}{2} (1 - \Pr_{(pk, sk) \leftarrow G(n)} \{Y^O(pk, m_0, m_1, E_{pk}(m_0)) = 1\}) \\
 &> 1/2 + \epsilon(n)/2.
 \end{aligned}$$

□

18.11. Tétel. *Létezik olyan rejtjelező, amely ind – cpa-biztonságos, de nem nm – cpa-biztonságos.*

Bizonyítás: Legyen (E, D, G) egy ind – cpa-biztonságos rejtjelező. Konstruálunk ennek alapján egy (E', D', G') rejtjelezőt, amely ind – cpa-biztonságos, de nem nm – cpa-biztonságos. Legyen $E'_{pk}(x) = [0, E_{pk}(x)]$, azaz egy zéró bittel bővítjük a rejtjeles üzenetet. Legyen $D'_{sk}[b, y] = D_{sk}(y)$, azaz a dekódolás figyelmen kívül hagyja a zérus bitet. (E', D', G') rejtjelező nyilván ind – cpa-biztonságos, hiszen egy konstans bittel növeltük a rejtjeles blokkot: ugyanis, ha két üzenet sikeresen megkülönböztethető lenne a megnövelt blokkhosszú rejtjelezés esetén, akkor a megkülönböztető algoritmus ugyanazon két üzenetet az eredeti rejtjelezés esetén is sikeresen megkülönböztetné. Ugyanakkor (E', D', G') rejtjelező nem nm – cpa-biztonságos, ugyanis, ha $E'_{pk}(x)$ rejtjeles szöveget $[1, E_{pk}(x)]$ blokkra módosítjuk, a módosított rejtjeles szöveg is az eredeti nyílt szövegbe dekódolódik, azaz az azonosságreláció szerint teljes sikerű nm – cpa-támadást hajtottunk végre. □

18.4. Feladatok

18.1. Feladat. Mutassuk meg, hogy az *ind – cpa*-biztonság implikálja a következőket:

1. Nehéz feladat a publikus kulcsból annak titkos párját kiszámítani.
2. Nehéz feladat a nyílt szövég első bitjét meghatározni a rejtett szöveg alapján.

18.2. Feladat*. A 18.4., 18.5. és 18.6. példák esetén adja meg a 18.9. definíció szerinti *R* relációt!

19.

A véletlen orákulum bizonyítástechnika

Véletlen orákulum alatt az alábbiakban egy olyan véletlen függvényt értünk, mely nyilvános elérésű, azaz mindenki által – így a támadó által is – hozzáférhető, s amely ugyanazon inputra ugyanazon outputot adja (függvény), különböző inputokra viszont független, az output térben egyenletesen választott outputot ad.

A véletlen orákulum modell alkalmazását biztonság bizonyítása kapcsán több konstrukción is bemutatjuk, jelen pontban egy rejtelező kódolás esetén. Véletlen orákulummal modellezük a konstrukció egy vagy több építőelemét (például egy hash leképezést), ami lehetővé teszi a biztonsági analízis egyszerűbb végrehajtását. A fő kritika ezen módszerrel szemben a modellezés helyessége vagy egyáltalán annak jogossága. Ezen utóbbi izgalmas ellenvetést nem elemezzük a továbbiakban, ugyanakkor megjegyezzük a következőket. Az is igen értékes, ha egy konstrukcióról (legalább!) egy modellen belül korrekt biztonsági analízist tudunk elvégezni. Ha ezen modellen belül a konstrukció „megbukik”, akkor nagy valószínűséggel el kell azt vetni vagy át kell tervezni, míg ha nem, legalább annyit korrektül kijelenthetünk, hogy a konstrukció „átment” a véletlen orákulum modelles teszten.

Az alábbi két – elsősorban elméleti – példa azt mutatja, hogy a véletlen orákulum modell feltételezése egy nagy erőforrás feltételezését jelenti, mivel ebben a „világban” természetes módon, biztonságosan állnak elő egyes kriptográfiai primitívek.

19.1. Példa. Legyen $R : \{0,1\}^n \rightarrow \{0,1\}^{3n}$ egy véletlen orákulum. Ebben a modellben létezik $(t, t/2^n)$ -biztonságú álvéletlen-generátor (PRG).

G^R generátorként közvetlenül az R orákulumot használjuk, azaz $G^R(x) = R(x)$ x mag esetén.

Emlékeztetőül: $G^R : X \rightarrow Y$ egy (t, ε) -biztonságú PRG, ha

$$|\Pr\{Z^R(G^R(x)) = 1\} - \Pr\{Z^R(y) = 1\}| \leq \varepsilon$$

tetszőleges, legfeljebb t erőforrás-igényű Z^R megkülönböztető algoritmus esetén, ahol x és y az X , illetve Y halmazból véletlenszerűen választott vektrok (eddig jelöléseinkekkel $x \leftarrow_r X, y \leftarrow_r Y$). A valószínűséget x, y, R véletlen választása, valamint Z esetleges véletlen elemei felett számoljuk.

A Z támadó feladata, hogy erőforráskorlátja mellett minél jobban megkülönböztesse $R(x)$ valószínűségi változó eloszlását az $y \leftarrow_r \{0,1\}^{3n}$ valószínűségi változó eloszlásától. Ha t számú kérést küld Z az R orákulumnak, amelyek x_1, x_2, \dots, x_t , és amelyre az $R(x_1), R(x_2), \dots, R(x_t)$ választ kapja, a következő két vektor valószínűségi változót kell megkülönböztetnie:

$$U = [R(x), R(x_1), R(x_2), \dots, R(x_t)], V = [y, R(x_1), R(x_2), \dots, R(x_t)].$$

U és V valószínűségi változók eloszlása csak akkor különbözik, ha létezik i , hogy $x_i = x$, mivel R véletlen függvényként azonos inputra azonos outputot ad, különböző inputra pedig független outputot. Jelölje B a $\{\exists i, x_i = x\}$ eseményt. Nyilván

$$\Pr\{Z(R(x)) = 1 | \bar{B}\} = \Pr\{Z(y) = 1 | \bar{B}\}.$$

Továbbá, annak valószínűsége, hogy a Z számára ismeretlen, és a PRG üzemeltetője által véletlen magként választott x megegyezik valamely általa választott kéréssel, felülről becsülhető az unió becslési technikával

$$\Pr\{B\} \leq t/2^n$$

módon. Innen

$$\begin{aligned} & |\Pr\{Z^R(G^R(x)) = 1\} - \Pr\{Z^R(y) = 1\}| = \\ &= |\Pr\{Z^R(G^R(x)) = 1 | B\} \cdot \Pr\{B\} + \Pr\{Z^R(G^R(x)) = 1 | \bar{B}\} \cdot \Pr\{\bar{B}\} \\ &\quad - \Pr\{Z^R(y) = 1 | B\} \cdot \Pr\{B\} - \Pr\{Z^R(y) = 1 | \bar{B}\} \cdot \Pr\{\bar{B}\}| \\ &= |\Pr\{Z^R(G^R(x)) = 1 | B\} - \Pr\{Z^R(y) = 1 | B\}| \cdot \Pr\{B\} \\ &\leq \Pr\{B\} \leq t/2^n, \end{aligned}$$

azaz G^R egy PRG.



19.2. Példa. Legyen $R : \{0, 1\}^n \rightarrow \{0, 1\}^{3n}$ egy véletlen orákulum. Ebben a modellben létezik $(t, 2t/2^n)$ biztonságú egyirányú függvény.

Egyirányú függvényként (ismét) magát az R orákulumot használjuk. Emlékezetől: $f^R : X \rightarrow Y$ egy (t, ε) -biztonságú egyirányú függvény, ha

$$\Pr\{f^R(Z^R(f^R(x))) = f^R(x)\} \leq \varepsilon$$

tetszőleges, legfeljebb t erőforrás-igényű Z^R invertáló algoritmus esetén, ahol $x \leftarrow_r X$. A valószínűséget x és R véletlen választása, valamint Z esetleges véletlen elemei felett számoljuk. Z támadó inputja $y = f^R(x)$. Mivel f véletlenszerűen választott, ezért Z nem lehet jobbat, mint véletlenszerűen választ egy elemet az inputok teréből. A feladat annak a valószínűségnek a kiszámítása, hogy olyan x' elemet választ, amelyre $f(x') = y$. Ez az esemény bekövetkezik, ha $A = \{x' = x\}$, vagy ha $B = \{x' \neq x : f(x') = y\}$. Mivel $\Pr\{A\} = 1/2^n$, illetve $\Pr\{B\} = 1/2^{3n}$, ezért t számú invertálási kísérlet (orákulum kérés) esetén – unió felső becsléssel – $t(1/2^n + 1/2^{3n}) < 2t/2^n$ adódik. ♣

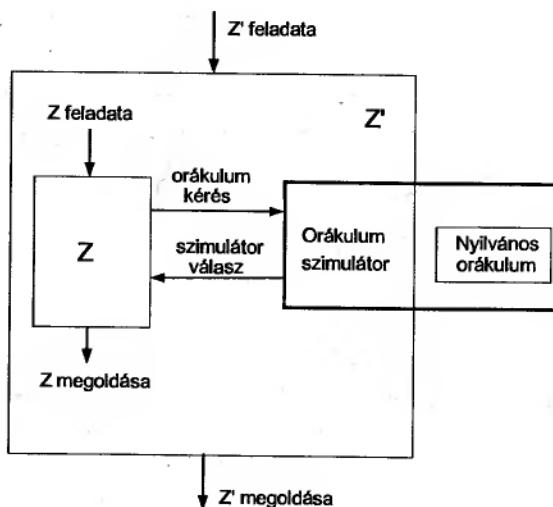
19.1. *ind – cpa*-biztonság véletlen orákulum modellben

A véletlen orákulumos modellen belüli bizonyítások továbbra is redukciók, ahol valamely standard nehéz feladat megoldhatóságát redukáljuk a konstruált biztonsági algoritmusunk támadási feladatának megoldhatóságára: megadunk egy hatékony Z' algoritmust, amely futtatja a biztonsági algoritmusunkat sikeresen támadó hatékony Z algoritmust, s ennek eredményeképpen sikeresen megoldja a nehéz feladatot (19.1. ábra).

Ahhoz, hogy Z lefusson, annak valódi környezetét (inputjait, orákulumait) Z által észrevehetetlen módon Z' -nek szimulálnia kell tudni. A szimulációt azonban nem minden tudja Z' tökéletesen végezni, például akkor sem, ha nem nyilvános transzformációval kapcsolatos kérésre kell választ adnia. Ezzel együtt is, a szimulációs sikertelenség valószínűségét elhanyagolható kis értéken (tipikusan biztonsági paraméterben exponenciálisan csökkenő felső korláttal) kellene tudnia tartani. Z valós környezetének szimulációja sikerében nagyban segíti Z' szimuláltot a véletlen orákulum modell.

A biztonság bizonyítása során a fő lépések így a következők:

1. Z' inputjának beágyazása Z inputjába.
2. Az orákulum(ok) minél tökéletesebb szimulációja.
3. Z' kívánt outputjának kinyerése Z kommunikációjából.



19.1. ábra. Redukciós technika és a véletlen orákulum

4. Z' sikervalószínűségének minél pontosabb alsó becslése.

Tekintsük az alábbi rejtjelezést:

$$E^G(x) = [f(r), G(r) \oplus x], \quad (19.1)$$

ahol

f egy egyirányú csapda permutáció,

r véletlen elem a permutáció ōsképteréből,

x a rejtjelezendő üzenet,

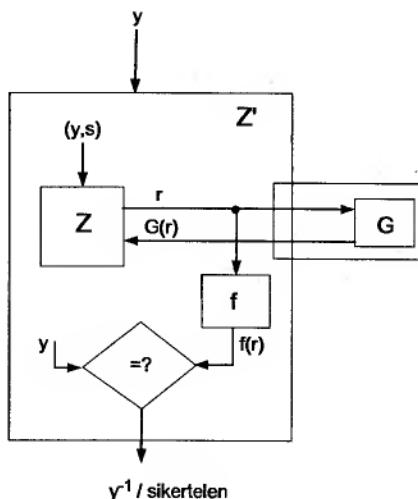
G véletlen-generátor, amelynek outputja üzenet méretű, és amelyhez orákulum hozzáférése van a támadónak is (véletlen orákulum modell).

$E^G(x_b)$ egy véletlen bitvektor: r (egyenletes eloszlású) véletlen elem f permutáció input teréből, ezért $f(r)$ is (egyenletes eloszlású) véletlen, továbbá $G(r)$ is (egyenletes eloszlású) véletlen kimenet, ami r -től és x -től független, így az üzenet egy OTP (one-time-pad) kódolását kapjuk, s a kódszó eloszlása is egyenletes.

A (19.1) konstrukció mögötti intuíció a következő: akkor tudjuk csak dekódolni az üzenetet, ha képesek vagyunk először az r ōsképet előállítani,

mivel r ismerete nélkül $G(r)$ a támadó számára egy véletlen bitvektor, s így x üzenet one-time-pad rejtjelezett.

Tegyük fel, hogy létezik egy sikeres Z *ind-cpa*-támadó, s mutassuk meg, hogy ez esetben konstruálható Z' támadó, amely adott y esetén, ahol y az f egy outputja, $y = f(r^*)$, sikerkel elő tudja állítani annak r^* ōsképét, azaz sikerkel invertálja f egyirányú csapda permutációt.



19.2. ábra. Redukció

Z inputja, az *ind-cpa*-biztonságnak megfelelően, a valós (nem szimulált) környezetében m_0, m_1 üzenetpár valamelyik tagjának $E^G(m_b)$ rejtjezetét várja a kódoló orákulumtól. Z' inputja az invertálandó y , és Z' kisorsol egy megfelelő méretű s véletlen elemet, majd egy $y' = (y, s)$ „rejtjelezett” szöveget ad vissza Z számára.

A véletlen orákulum modell alkalmazása segít Z' számára áthidalni azt a problémát, hogy r^* ismerete nélkül egy nem véletlen orákulummal helyettesített, valós G leképezés esetén az r^* ōsképet kellene tudni kitalálnia, hogy konzisztens lehessen a rejtjelezett szöveg az m_0, m_1 üzenetpárral. Ha ugyanis G véletlen orákulum, akkor $G(r) \oplus m_b$ OTP kódolás lévén, a kódolás eredménye véletlen és m_b -től független. Vagy másként megfogalmazva, (y, s) térszöleges m_b -vel konzisztens, abban az értelemben, hogy $G(r) = s \oplus m_b$, azaz lényegében konzisztensre definiáljuk a véletlen orákulum kimenetét.

Z feladata az, hogy $((y, s), m_0, m_1)$ input alapján a b -re sikeres döntést hozzon az $ind - cpa$ -biztonság definíciójának megfelelően. Z' figyeli, hogy Z milyen orákulum kérésekkel fordul G orákulumhoz. Legyen B az az esemény, hogy ezen kérések között szerepel r^* , amit Z' ellenőrizni tud, hiszen f nyilvános. Azaz Z' az $f(r) = y$ egyenlőség fennállását ellenőrzi minden r kérésre. Ha az ellenőrzés sikeres (B teljesül), akkor Z' outputja r^* . Z sikervalósínűsége az indirekt feltételezés szerint nem elhanyagolható, így

$$\begin{aligned} 1/2 + \varepsilon &< \Pr\{Z \text{ sikeres}\} \\ &= \Pr\{Z \text{ sikeres} | B\} \Pr\{B\} + \Pr\{Z \text{ sikeres} | \bar{B}\} \Pr\{\bar{B}\} \\ &\leq 1 \cdot \Pr\{B\} + \Pr\{Z \text{ sikeres} | \bar{B}\} \\ &\leq \Pr\{B\} + 1/2, \end{aligned}$$

ahol felhasználtuk, hogy \bar{B} esetén, azaz ha Z nem kérdezi r^* inputtal G orákulumot, akkor Z' szimulátornak pénzfeldobásnál nincs nagyobb esélye $G(r)$ kitalálására. Következésképp $\Pr\{B\} > \varepsilon$, ami ellentmond f egyirányú tulajdonságának.

20.

Biztonságos digitális aláírás

Tekintsünk egy (G, Sig, V) nyilvános kulcsú aláíró sémát, ahol G, Sig, V randomizált polinomiális idejű algoritmusok a következők:

1. Kulcsgeneráló algoritmus: $G : 1^n \rightarrow (\text{PK}, \text{SK})$, ahol PK és SK a nyilvános illetve titkos kulcsok tere.
2. Aláíró algoritmus: $\text{Sig} : \text{SK} \times \{0,1\}^* \rightarrow Y$, ahol Y az aláírások tere.
3. Aláírást ellenőrző algoritmus: $V : \text{PK} \times \{0,1\}^* \times Y \rightarrow \{0,1\}$, ahol 1 output a sikeres ellenőrzésnek felel meg.

Egy (G, Sig, V) nyilvános kulcsú aláíró séma (t, q, ϵ) -biztonságú, ha bár-mely Z^{Sig} hatékony algoritmus esetén, amely a Sig aláíró orákulumnak q kérdezést tehet fel, t ideig futhat, annak a valószínűsége, hogy outputján előállít egy új, az orákulumtól korábban nem kérdezett $[m', y']$ [üzenet, aláírás]-párt, legfeljebb ϵ , azaz

$$\Pr_{(pk, sk) \leftarrow G} \{V_{pk}(m', y') = 1, \text{ ahol } (m', y') \leftarrow Z^{\text{Sig}}(pk)\} \leq \epsilon, \quad (20.1)$$

illetve tömörebben

$$\Pr_{(pk, sk) \leftarrow G} \{V_{pk}(Z^{\text{Sig}}(pk)) = 1\} \leq \epsilon. \quad (20.2)$$

20.1. Biztonságos one-time aláírás

Legyen $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ egy egyirányú permutáció. Egy $m \in \{0, 1\}^l$ üzenet aláírásához véletlenszerűen generálunk $2l$ darab, egyenként n bit méretű blokkot, amelyet titokban tartunk, s ezek együttese képezi az sk titkos kulcsot. A titkos kulcs n bites blokkjainak f szerinti leképezésével kapható $2l$ darab n bit méretű blokk együttese alkotja a pk nyilvános kulcsot. Tehát

$$\begin{aligned} sk &= ((r_{1,0}, r_{1,1}), \dots, (r_{l,0}, r_{l,1})), \\ pk &= ((f(r_{1,0}), f(r_{1,1})), \dots, (f(r_{l,0}), f(r_{l,1}))). \end{aligned} \quad (20.3)$$

Az m üzenet aláírását bitenként képezzük: ha $m_i = 0$, akkor az aláírás r_{i0} , míg ha $m_i = 1$, akkor az aláírása r_{i1} , így

$$Sig(m) = (r_{1,m_1}, \dots, r_{l,m_l}). \quad (20.4)$$

A V ellenőrző algoritmus az

$$[m, Sig(m)] \quad (20.5)$$

input, a pk nyilvános kulcs és f leképezés felhasználásával üzenetbitenként ellenőrzi az aláírás helyességét.

20.1. Tétel. Tekintsük a (20.3)-(20.5) aláíró algoritmust.

a.) Ha a támadó legalább két kéréssel fordulhat az aláíró orákulumhoz, továbbá $l \geq 2$, akkor sikervalószínűsége 1, azaz az aláírás séma $(t, 2, 1)$ -biztonságú.

b.) Ha a támadó csak egy kéréssel fordulhat az aláíró orákulumhoz és f egy (t, ϵ) -biztonságú egyirányú permutáció, továbbá $l \geq 1$, akkor az aláírás séma $(t, 1, 2\epsilon)$ -biztonságú.

Bizonyítás:

a.) Z támadó aláírást kér a 0^l és 1^l üzenetekre, amivel a titkos kulcs birtokába jut. Az $l \geq 2$ feltétel biztosítja, hogy ezek után a támadó választhat új üzenetet, amelyre aláírást tud adni (ti. $l = 1$ esetben 0 és 1 a két lehetséges üzenet).

b.) A bizonyítás indirekt. Tegyük fel, hogy az aláírás séma $(t, 1, 2\epsilon)$ -törhető egy Z támadó által, azaz Z sikervalószínűsége nagyobb, mint 2ϵ . Z' támadó egy $y = f(x)$ inputra az x inverzet szeretré kiszámítáni. Z' támadó, a Z számára megkülönböztethetetlenül, G és Sig helyett G' és Sig' szimulált algoritmusokat nyújtja a következőképp:

Z kulcsgenerálás kérése esetén G' kisorsol egy n bites r véletlen blokkot, majd egy véletlen bit alapján („feldob egy pénzt”) a következő választ adja: ha 0 a kimenetel, akkor G' válasza $K_0 = [pk = (f(r), y), sk = (r, ?)]$, ha pedig 1 a kimenetel, akkor G' válasza $K_1 = [pk = (y, f(r)), sk = (?, r)]$, ahol „?” azt jelzi, hogy Z' nem tudja szimulálni ezeket a mennyiségeket, mivel nem ismeri y inverzét.

Erzután Z' futtatja Z algoritmust, amely legfeljebb 1 kéréssel fordul a Sig' szimulált algoritmushoz. Ha K_b volt a kisorsolt kulcs, és Z az $m = b$, egy bites üzenetre kér aláírást, akkor Sig' szimulátor helyesen képes megválaszolni, mivel ekkor válasza r . Ha azonban K_b volt a kisorsolt kulcs, és Z az $m = \bar{b}$, egy bites üzenetre kér aláírást, szimulációs hiba keletkezik. Ha ilyen módon Sig' sikeres a hozzá intézett kérés megválaszolásában, s ezek után Z sikeresen képes új aláírást generálni, ez azt jelenti, hogy kiszámítja y ősképét. Következésképpen ezen ősképszámítás sikervalószínűsége (Z' sikervalószínűsége) nagyobb, mint $1/2 \cdot (2\epsilon) = \epsilon$ □

Megjegyezzük, hogy b.) állítás általánosítható l bitnél hosszabb üzenet esetére.

Az alábbiakban az RSA algoritmus felhasználásával képzett aláírást, s a kapcsolatos algoritmusok biztonságának főbb elemeit tekintjük át.

20.2. Aláírás az RSA algoritmus felhasználásával

20.2.1. Egyszerű RSA aláírás

Legyen az m üzenet legfeljebb egy blokk méretű, s az aláírás az alábbi:

$$Sig(m) = m^d \pmod{N}. \quad (20.6)$$

A (20.6) aláírás támadásához kihasználhatjuk az aláíró leképezés multiplikatív tulajdonságát, azaz a tetszőleges m_1, m_2 üzenetpár esetén fennálló

$$Sig(m_1 \cdot m_2) = Sig(m_1) \cdot Sig(m_2) \pmod{N}$$

összefüggést: Az aláíró orákulumtól kérjünk aláírást például az $m_1 = 2$, illetve az $m_2 = 2 \cdot m$ üzenetekre, amely alapján előállíthatjuk az m üzenet-hez tartozó aláírást. Ugyanis $Sig(2 \cdot m) = Sig(2) \cdot Sig(m) \pmod{N}$ alapján, $Sig(m) = Sig(2)^{-1} \cdot Sig(2 \cdot m) \pmod{N}$, ahol $Sig(2)$ inverze mod N létezik, mivel $Sig(2)$ és N relatív prímek.

Sőt, a (20.1) biztonság-definíció esetén sikkerrel lehet támadni az egyszerű RSA aláírást anélkül is, hogy egyáltalán kéréssel fordulnánk az aláírás orákulumhoz, azaz $q = 0$ esetben is! Tekintsünk ugyanis egy $m = y^e \pmod{N}$

alakú m üzenetet. Ennek az üzenetnek az y nyilván aláírása, s ehhez a támadáshoz csak a nyilvános kulcsot használtuk.

20.2.2. Lenyomat aláírása

Ne korlátozzuk az üzenet hosszát egy blokk méretre, s alkalmazzuk a lenyomatképzés technikát. Egy m üzenet és H lenyomatképző (hash) függvény esetén tekintsük a

$$\text{Sig}(m) = [00\dots 00|H(m)]^d \pmod{N} \quad (20.7)$$

aláírást, ahol zérusokkal teljes blokkra egészítjük ki a lenyomatot a felső helyiértékeken (hash-and-sign).

Ez az algoritmus is támadható, például, ha a H függvény nem egyirányú, vagy ha multiplikatív tulajdonságú (azaz $H(m_1 \cdot m_2) = H(m_1) \cdot H(m_2)$ tésszéges m_1, m_2 üzenetpár esetén): ha H függvény nem egyirányú, akkor egy ismert aláíráshoz a támadó hatékonyan kiszámolhat egy csaló ősképet (üzenetet). Ha H függvény multiplikatív, akkor az aláírás is öröklí a multiplikatív tulajdonságot, s ez esetben a fentiekben említett támadás alkalmazható.

20.2.3. PKCS-1 szabványú aláírás

Egy m üzenet és h hash függvény esetén a

$$\text{Sig}(m) = [1FF\dots FF00|H(m)]^d \pmod{N} \quad (20.8)$$

aláírást alkalmazzuk, azaz az előző, lenyomat aláírása módszerbeli, zérusokkal való feltöltés helyett $1FF\dots FF00$ feltöltést alkalmazzuk (ahol $1, F, 0$ hexadecimális karakterek). Ekkor már nem működik egy multiplikatív lenyomatképző függvény esetére bemutatott fenti támadás. Támadás ugyan még nem került publikálásra ezen szabványos aláírás ellen, ennek ellenére a (20.8) formulabeli módosítás nem tekinthető lényegi előrelépésnak, s támadhatósága valószínűsíthető.

20.2.4. Teljes blokknyi lenyomat (Full Domain Hash, FDH)

FDH hash függvény outputjának mérete megegyezik az RSA blokkméretével (nem annak töredéke), azaz egy m üzenet és FDH hash függvény esetén a

$$\text{Sig}(m) = \text{FDH}(m)^d \pmod{N} \quad (20.9)$$

aláírást alkalmazzuk.

Az alábbiakban ezen aláírás biztonságát analizáljuk véletlen orákulum modellben: a támadó hozzáfér az FDH hash függvényt modellező, véletlen függvényként működő H publikus véletlen orákulumhoz, amelyhez legfeljebb u darab kérést intézhet. Ezenkívül továbbra is hozzáfér a Sig aláíró orákulumhoz, amelyhez legfeljebb q kérést intézhet.

20.2. Tétel. *Tegyük fel, hogy az RSA (u, ϵ) -biztonságú csapda permutáció. Ekkor a véletlen orákulum modellben a (20.9) aláírás séma a.) $(u, q = 0, \epsilon)$ -biztonságú (a támadó nem használja az aláíró orákulumot), b.) $(u, q = 1, \delta)$ -biztonságú, ahol $\delta = (u \cdot \epsilon + 2^{-n}) / (1 - 1/u)$ (a támadó egy kérést intéz az aláíró orákulumhoz).*

Bizonyítás: a.) A bizonyítás indirekt. Tegyük fel, hogy létezik egy Z algoritmus, amely ϵ -nál nagyobb valószínűséggel töri a (20.9) algoritmust. Mutatjuk, hogy ekkor létezik egy Z' algoritmus, amely egy RSA kódolású z rejtegt szöveget ϵ -nál nagyobb valószínűséggel dekódol.

Z' algoritmus Z algoritmusnak a H orákulumhoz szánt kérését H' hash orákulum-szimulátorhoz küldi, amely szimulátor működése a következő: Legyen Z i -edik kérése m_i , amelyre

$$H'(m_i) = r_i^\epsilon z \pmod{N}$$

a válasz, $i = 1, 2, \dots, u$, ahol $r_i \leftarrow_r \{0, 1\}^n$ véletlen elem. Z számára H és H' output eloszlása megkülönböztethetetlen, így a szimulált esetre is megmarad Z aláírástörő sikergyakorisága. Z feltételezésünknek megfelelő gyakorisággal ki tudja számolni valamelyik válasz dekódoltját, azaz Z outputja, siker esetén

$$[m_i, (r_i^\epsilon z)^d = r_i z^d \pmod{N}].$$

Z' először m_i alapján kikeresi r_i véletlen elemet (képes rá, hiszen ő futtatja a H' szimulátort, s így annak minden részletéhez hozzáfér), majd egy moduláris osztással előállítja z rejtegt szöveg z^d dekódoltját, amivel ellentmondásra jutottunk. Mivel Z' akkor és csak akkor sikeres, amikor Z sikeres, így a két sikervalószínűség is azonos. Z' sikervalószínűsége (azaz RSA-törés valószínűsége) azonban nem lehet nagyobb, mint ϵ , így az aláírást támadó Z sikervalószínűsége sem lehet ennél jobb.

b.) Z' kisorsol egy j , $1 \leq j \leq u$ sorszámot véletlenszerűen. A H' és Sig' szimulátorok működése legyen a következő: H' -höz intézett j -edik kérés esetén annak válasza z , továbbá, ha $i \neq j$, akkor $H'(m_i) = r_i^\epsilon \pmod{N}$ a

válasz. Így, ha Sig' szimulátorhoz küldött kérés nem z , akkor $Sig'(m_i) = r_i \pmod{N}$. Z' azonban nem tud helyes alírást szimulálni akkor, ha ezen kérés z .

Tekintsük az alábbi eseményeket:

1. $Sim = \{Z$ szimulálni képes a valós aláíró orákulumot},
2. $\overline{Sim} = \{Z$ egy z „lenyomatra” kér alírást az aláíró orákulum-szimuláltortól (szimulációs hiba esemény)},
3. $C = \{Z$ egy z „lenyomatra” számol ki alírást},
4. $F = \{Z$ az alírást olyan üzenetre adja, amelyhez megkérdezte H orákulumtól (azaz H' szimuláltortól) a lenyomatot},
5. $U = \{Z$ sikeres, azaz egy helyes alírás outputot ad}.

Ezen jelölések, valamint $\Pr\{U|\overline{Sim}\} = 0$ felhasználásával kapjuk

$$\begin{aligned}\Pr\{U\} &= \Pr\{U|\overline{Sim}\} \Pr\{\overline{Sim}\} + \Pr\{U|Sim\} \Pr\{Sim\} \\ &= 0 \cdot \Pr\{\overline{Sim}\} + \Pr\{U|Sim\} \Pr\{Sim\},\end{aligned}$$

azaz $\Pr\{U\} = \Pr\{U \cap Sim\}$. Indirekt feltételezésünk szerint az állítással ellentétben $\Pr\{U\} > \delta$. Ekkor

$$\begin{aligned}\delta \cdot \Pr\{Sim\} &< \Pr\{U|Sim \cap \overline{F}\} \Pr\{Sim \cap \overline{F}\} + \Pr\{U|Sim \cap F\} \Pr\{Sim \cap F\} \\ &= 2^{-n} \cdot \Pr\{Sim \cap \overline{F}\} + \Pr\{U \cap Sim \cap F\} \\ &\leq 2^{-n} + \Pr\{U \cap Sim \cap F\} \\ &\leq 2^{-n} + \sum_{i=1}^u \Pr\{U \cap Sim \cap \{H'(m_i) \text{ alírása, } (Z \text{ által})\}\},\end{aligned}$$

ahol az első egyenlőségnél felhasználtuk, hogy ha Z nem kérdezi meg a lenyomatot H' orákulum-szimuláltortól, akkor csak találgatni képes rá, s így $\Pr\{U|Sim \cap \overline{F}\} = 2^{-n}$. $\Pr\{Sim\} = 1 - 1/u$ összefüggés, valamint δ választásának felhasználásával

$$\frac{1}{u} \sum_{i=1}^u \Pr\{U \cap Sim \cap \{H'(m_i) \text{ alírása}\}\} \geq \varepsilon. \quad (20.10)$$

(20.10) bal oldala j egyenletes véletlen választása miatt

$$\Pr\{U \cap Sim \cap \{H'(m_j) \text{ alírása}\}\}$$

valószínűséggel egyenlő, amelynél nagyobb a

$$\Pr\{U \cap \{H'(m_j) \text{ alírása}\}\}$$

RSA-törés sikervalószínűsége, s ezzel ellentmondásra jutottunk. \square

21.

Üzenethitelesítés (MAC) biztonsága

Ha blokk-rejtjelezőt alkalmazunk, ahol a nyílt szövegek halmaza ritkított (pl. valamely formátum megkötésnek tesz eleget), akkor a rejtjeles szöveget csak elhanyagolható valószínűséggel lesz képes a támadó úgy módosítani, hogy a dekódolt szöveg a ritka nyílt szöveg halmazba essen. Ezen gyakorlati módszer ellenére kijelenthetjük, hogy a rejtjelezés nem alkalmas üzenetek biztonságos hitelesítésére. Tekintsük ugyanis a one-time-pad vagy a Goldwasser–Micali-rejtjelezést. Egy támadó képes előállítani egy korábban nem látott rejtjeles szöveget, amely egy legalíts nyílt üzenetbe dekódolódik: a rejtjeles szöveg bármely bitjének invertálása tekinthető úgy, mintha az adott bitpozícióban invertáltuk volna a nyílt szöveg bitet. A támadó ugyanis ismerheti a nyílt szövegek formátumát, amelyeknek vannak rögzített helyen megállapodott tartalmú mezői (dátum, pénzösszeg stb.), s így ezen mezők formátumát tartó módosítást végez.

Jobb megoldás a kriptográfiai ellenőrző összeg (MAC, Message Authentication Code) alkalmazása (lásd 6. fejezet). Jelen fejezet témája a biztonságos MAC konstrukció. A MAC módszer két algoritmust tartalmaz, a generáló (T) és az ellenőrző (V) algoritmust:

1. MAC generáló algoritmus: $T : K \times \{0,1\}^* \rightarrow S$, ahol K a kulcsré, egy m üzenet a $\{0,1\}^*$ halmaz eleme, továbbá S a MAC ellenőrző összegek tere.
2. MAC ellenőrző algoritmus: $V : K \times \{0,1\}^* \times S \rightarrow \{0,1\}$, ahol 1 output a sikeres ellenőrzésnek felel meg, azaz amikor

$$V_k(m, T_k(m)) = 1 \quad \forall m, k.$$

Egy Z támadó q számú kérést m_1, \dots, m_q küldhet egy MAC generáló orákulumnak, amelyre u_1, \dots, u_q választ kapja. A támadó tehát először a küldő féltől kér MAC ellenőrző összegeket általa választott üzenetekre, s ezután kiszáml egy csaló (üzenet, MAC) párt, amelyet elküld a vevő félnek (ellenőrző algoritmusának).

Megkülönböztetjük a gyenge és az erős támadót, sikervalószínűségének alábbi két definíciójával:

21.1. Definíció (gyenge MAC támadó). Legyen

$$\text{Adv}_F^w(Z) = \Pr_k \{Z \text{ outputja } (m, u) : V_k(m, u) = 1, \forall i m \neq m_i\}.$$

Egy $F = (T, V)$ MAC algoritmus (t, q, μ, ϵ) gyengén biztonságos, ha fennáll az $\text{Adv}_F^w(Z) \leq \epsilon$ egyenlőtlenség tetszőleges (t, q, μ) erőforrás-korlátú támadóra, azaz, ha

$$\text{Adv}_F^w(t, q, \mu) = \max_Z \text{Adv}_F^w(Z) \leq \epsilon.$$

(t a támadó futási idő korlátja, q az orákulum-kérések számának korlátja, μ pedig az orákulum-kérések plusz a csaló üzenet összhossza.)

21.2. Definíció (erős MAC támadó). Legyen

$$\text{Adv}_F^s(Z) = \Pr_k \{Z \text{ outputja } (m, u) : V_k(m, u) = 1, \forall i m \neq m_i, \forall j, u \neq u_i\}.$$

Egy $F = (T, V)$ MAC algoritmus (t, q, μ, ϵ) -erősen biztonságos, ha tetszőleges (t, q, μ) erőforrás-korlátú támadóra teljesül, hogy $\text{Adv}_F^s(Z) \leq \epsilon$, azaz, ha

$$\text{Adv}_F^s(t, q, \mu) = \max_Z \text{Adv}_F^s(Z) \leq \epsilon.$$

A két definíció között a különbség abban van, hogy az erős MAC támadó esetén nem megengedett, hogy egy megfigyelt ellenőrző összeghez konstruáljon új, illeszkedő üzenetet. Az erős-gyenge jelző értelmét az adja, hogy amennyiben egy támadó, azonos erőforráskorlát, azonos sikervalószínűség mellett új üzenethez új MAC-ot képes előállítani, az nehezebb feladatot old meg. Ha egy MAC algoritmus esetén nem létezik sikeres gyenge MAC-támadó, akkor nincs erős sem.

A két biztonság-definíció ekvivalens olyan MAC algoritmus esetén, melyre tetszőleges (m, u) párra $V_k(m, u) = 1$ akkor és csak akkor áll fenn, ha $T_k(m) = u$, azaz egy üzenethez nem tartozhat több, az ellenőrző algoritmus által elfogadott ellenőrző összeg. Ez esetben a két biztonság-definíció ekvivalens. Az alábbiakban feltesszük ezen ekvivalencia feltétel teljesülését, s

ennek megfelelően (t, q, μ, ϵ) -biztonságról beszélünk. Például a gyakorlatban alkalmazott determinisztikus MAC ellenőrző algoritmusok a vett m üzenetre újragenerálják az ellenőrző összeget, majd az eredményt vetik össze a vett ellenőrző összeggel, így az ekvivalencia feltétele teljesül.

Ezen definíciók nem kezelnek egy fontos támadási esetet: egy korábbi, legalis felhasználó által előállított (üzenet, MAC) pár újrafelhasználását, a visszajátszásos támadást (replay attack). Ez a fajta támadás csak akkor szűrhető ki, ha az ellenőrző algoritmus állapottal is rendelkezik, s fel tudja ismerni egy korábbi üzenet visszajátszását. Ilyen állapotfigyelést segít az üzenetek sorszámmal vagy időpecséttel ellátása a MAC számítást megelőzően.

Ha rögzített hosszúságú üzenethez kell kiszámítani a MAC-t, akkor ezt ideálisan egy megfelelő dimenziós véletlen függvényel tehetnénk biztonságosan:

21.3. Tétel. *Ha $R : X \rightarrow Y$ egy véletlen függvény, továbbá $T_R = R$, akkor $F_R = (T_R, V_R)$ MAC-algoritmus $(\infty, q, \mu, 1/|Y|)$ -biztonságú.*

Bizonyítás: (Vázlat) Mivel a támadó új, addig nem látott üzenetre kell adjon ellenőrző összeget, ugyanakkor egy véletlen függvény új inputhoz tartozó outputja is véletlen, a támadó csak véletlenszerűen találhat az ellenőrző összeg vonatkozásában, bármekkora is legyen az erőforrása. \square

A véletlen függvényt biztonságos álvéletlen függvénnel (PRF) helyettesítve kapjuk az alábbi tételt:

21.4. Tétel. *Ha $W : X \rightarrow Y$ egy (t, q, μ, ϵ) -biztonságú PRF, és $T_W = W$, akkor az $F_W = (T_W, V_W)$ MAC-algoritmus $(t, q, \mu, \epsilon + 1/|Y|)$ -biztonságos MAC.*

Bizonyítás: (Vázlat) A szokásos redukciós gondolatmenettel, indirekt bizonyítással történhet, ahol ha egy támadó töri a MAC-ot, akkor töri a PRF-t is. \square

Változó hosszúságú üzenetek esetén biztonságos MAC generálása további megfontolásokat igényel. Azaz, hogyan generáljuk az ellenőrző összeget, ha az üzenetünk a PRF input méretének többszöröse lehet. A CBC – MAC konstrukció biztonság-analízisére itt nem térünk ki, de két egyszerű példán keresztül érzékeltetjük a problémát.

21.1. Példa. ECB módon rejtelezük az üzenet blokkjait, s a rejtejes blokkat adjuk össze ($\text{mod } 2$), amely összeg legyen a MAC:

Könnyen látható, hogy ez a hitelesítési mód tökéletesen rossz: vegyük egy $m = [x, x]$ üzenetet, amely két blokk hosszúságú, s minden blokkja azonos,

n bit hosszú bináris sorozat. Nyilván a csupa zérus blokk lesz a MAC értéke. Következésképp, ha m még nem szerepelt üzenetként (az adott kulcs használata óta), a támadás a fenti definíciók alapján sikeres:

$$Adv_F(t, 0, 2n) = 1.$$
♣

21.2. Példa. A 21.1. példa MAC számítását bonyolítsuk annyival, hogy az üzenet blokkjait ugyan külön-külön rejtelezzük, de rejtelezés előtt egy üzeneten belüli blokksorszámmal láttuk el: Három MAC kérés alapján a támadó ki tud számítani egy hiteles $(m, MAC(m))$ párt egy korábban nem szerepelt m üzenetre: kérje ugyanis a MAC-ot $[x, y]$, $[x, y']$, $[x', y]$ üzenetekre, amelyeket összeadva megkapja a MAC-ot $[x', y']$ üzenetre. Következésképp

$$Adv_F(t, 3, 8n) = 1.$$
♣

A kitűzött feladatok megoldása

1. fejezet

1.1. Feladat:

1. $N \cdot \phi(N) = N^2 \prod_{p|N} (1 - 1/p)$ alapján, rendre, 312,486,240 adódik.
2. Ha $a \neq 1$, az $(a-1)x = -b \pmod{N}$ egyenletnek pontosan egy megoldása van N prímszám esetén.
3. Ha b nem osztható $(a-1, N)$ -el.

1.2. Feladat:

1. Az a kulcselem multiplikatív inverzánekként mod N kell léteznie, aminek feltétele $(a, N) = 1$. $N = 26 = 2 \cdot 13$ esetén 12 ilyen szám van az $[1, 25]$ tartományban. b kulcselem 26-féleképpen választható. Az $a = 1, b = 0$ esetet kizárvva (identitás transzformáció), a kulcstér mérete 311.
2. Nem. Ha a forrás véletlenszerűen sorsol az ábécé elemei közül, akkor az ax szorzat is ilyen, következésképp az y rejtett szöveg és b kulcselem közötti kölcsönös információ zérus.
3. Két olyan $(x, y), (x', y')$ ismert nyílt-rejtett párt elegendő, amelyre $(x - x')^{-1} \pmod{26}$ létezik.
4. A betűnkénti rejtjelezés miatt a karakter-gyakoriságok az ábécé felett permutálódnak, de az értékek nem változnak. Ezért, ha van két szignifikánsan kiemelkedő gyakoriságú karakter az ábécében, akkor ezen gyakoriságok a rejtjelezés után is ugyanúgy léteznek. Ekkor statisztikai alapú támadással

rejtett szöveg elegendő mennyiségben történő megfigyelése után végre-hajtható a támadás: a legnagyobb gyakoriságú rejtett szöveg karakterek-hez tartozó nyílt szöveg karakterekre sejtést teszünk, s minden ilyen sejtés egy újabb egyenletet ad az ismeretlen kulcselemekre. Két ilyen lineárisan független egyenlet alapján kapunk sejtést a két kulcslemre, amelyek helyességét ismert szövegpáron teszteljük.

1.3. Feladat:

Nem. Ugyanis $x = uN + v$, $y = wN + z$ jelöléssel $y = z = av + b \pmod{N}$ adódik, azaz a rejtett szöveg betűpár második karaktere csak a nyílt szöveg betűpár második karakterétől függ.

1.4. Feladat:

1. Nem. A kulcs entrópiára vonatkozó szükséges feltétel nem teljesül.
2. A támadó általa választott nyílt szövegekhez kér rejtett szöveg párt a rejtjelezőtől, majd ezen párok alapján próbálja a kulcsot megfejteni. A példában: $x = (0, 0, 0, \dots)$ esetén $y_1 = r$, ($y = r, r^2, r^3, \dots$), amellyel a valódi dekódolával azonos titok birtokában maga is dekódolni képes.
3. Az i -edik karaktertől kezdődően a dekódolt nyílt szöveg hibás lesz.

1.5. Feladat:

A pénzfeldobás sorozatot bitpárokra szeleteljük. Dobjuk el a 00 és az 11 párokat, majd $01 \rightarrow 0$ és $10 \rightarrow 1$ hozzárendeléssel képezzük az új, már egyenletes bitgyakoriságú sorozatot. A forrás bitsebessége átlagosan $1/4$ részére csökken.

1.7. Feladat:

Igen. F_r entrópia-sebessége: $-(0,25 \cdot \log_2(0,25) + 0,75 \cdot \log_2(0,75)) \cdot 5 = 4,056$ kbit/sec. F_x entrópia-sebessége: $0,5 \cdot 8$ kbit/sec = 4 kbit/sec. Mivel az F_r kulcsfolyam entrópia-sebessége nagyobb a rejtjelezendő F_x folyam entrópia-sebességénél, ezért ideális forráskódolás esetén folyamatos kódolás végezhető.

1.8. Feladat:

Nem. Például $\Pr\{Y = 1 | X = a\} = 2/5$, $\Pr\{Y = 1 | X = b\} = 1/5$, azaz X és Y nem független.

2. fejezet

2.1. Feladat:

$y = Ax + b$, ahol y, x, b n bites bináris vektorok, A egy $n \times n$ bites bináris mátrix, x a nyílt szöveg, y a rejtett szöveg, $K = [A, b]$ a kulcs. Lineáris egyenletrendszer megoldással történik a támadás. $n + 1$ párra van minimálisan szükség, amelyből n darab lineárisan független nyílt szöveghez tartozik.

2.3. Feladat:

1. 0, mivel az outputok $w = \{1, 1\}$ súlyú lineáris kombinációja (bináris összegye) konstans 0,
2. 0, mivel az outputok $w = \{0, 1\}$ súlyú lineáris kombinációja (a 2. kimenet) konstans 1.

2.9. Feladat:

Igen. A jelölésekkel egyszerűsítve, legyen $x = (a, b)$ és $y = (c, d)$ a rejtelező input, illetve output blokkja, két felével megadva. Így $c = a \oplus b \oplus k_1$, $d = b \oplus (a \oplus b \oplus k_1) \oplus k_2 = a \oplus k_1 \oplus k_2$, amelynek alapján $k_1 = a \oplus b \oplus c$, $k_2 = b \oplus c \oplus d$.

2.10. Feladat:

A DES egy rétegénék struktúrájára tekintve az igazolás nyilvánvaló: a DES S-dobozok bemenete nem változik, mivel az iterációs kulcs is és az adat input félfelblokk is komplementálva van, s XOR összegük kerül a bemenetekre, továbbá a kimeneten ehhez a másik adat input félfelblokk komplementáltja adódik, azaz $a \oplus b = \overline{\overline{a} \oplus \overline{b}}$.

1. Nem jelent könnyebbséget a támadó számára.
2. A kulcskérés ideje felére csökkenhető. Tegyük fel, hogy m üzenetre, valamint \overline{m} komplemente ismerjük a rejtett szöveget. Tekintsük (m, c_1) , $(\overline{m}, \overline{c}_2)$ nyílt szöveg-rejtett szöveg párokat, ahol $c_1 = E_k(m)$ és $\overline{c}_2 = E_k(\overline{m})$, s így a komplementálás tulajdonság miatt $c_2 = E_{\overline{k}}(m)$ összefüggés is fennáll. A kulcsok terét keressük végig, amelynek során egy k' tesztelt kulcs esetén megvizsgáljuk, hogy $E_{k'}(m)$ rejtett szöveg egyenlő-e c_1 vagy c_2 rejtett szöveggel. Ha nem a válasz, akkor ezzel nyilván leellenőriztük mind a k' , mind pedig \overline{k}' kulcsot, azaz egyidejűleg kettő kulcsot. Tehát egy kulcspár ellenőrzéséhez, egy kódolási és két komparálási lé-

pést hajtottunk végre. (Valóságos támadási köriülményeket tekintve kicsi a valószínűsége, hogy nyílt szöveg blokk és komplexitate egyidejűleg is előforduljon a kódolt blokkok között, illetve ezt célszerű elkerülni.)

2.11. Feladat:

1. Igen, végrehajtható, mivel ismerjük a rejtjeles szöveg redundáns struktúráját, amit adott esetben az egy blokktól 8 paritásbit jelent. Annak a valószínűsége, hogy egy téves kulccsal helyes paritásúra dekódolunk egy rejtett szöveg blokkot, 2^{-8} . Annak a valószínűsége, hogy például 7 rejtjeles blokk mindeneket helyes paritásúra dekódoljuk téves kulcs mellett 2^{-56} , tehát ez esetben a ki nem szűrt téves kulcsok átlagos száma a kulcs-tér teljes végigkeresése után $2^{56} \cdot 2^{-56} = 1$ lenne. Ezért például, 10 rejtjeles blokk megfigyelése elegendő lenne a gyakorlatilag egyértelmű kulcsazonosításhoz.
2. Nem. $1/2$ annak a valószínűsége, hogy egy téves kulccsal helyes paritásúra dekódolunk egy rejtett szöveg blokkot, így annak a valószínűsége, hogy 50 rejtjeles blokk mindeneket helyes paritásúra dekódoljuk egy téves kulcs mellett, 2^{-50} . A szűrésen átment kulcsok átlagos száma a kulcs-tér teljes végigkeresése után $2^{56} \cdot 2^{-50} = 64 \gg 1$.

2.12. Feladat:

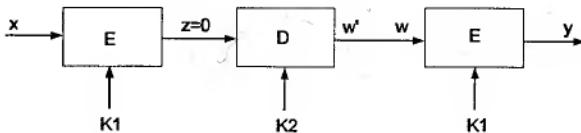
1. Középen találkozás támadást alkalmazva egy (x, y) nyílt-rejtett szöveg-pár felhasználásával az összes kulccsal kódoljuk az x nyílt szöveget, s az eredmény z rejtett szöveget az alkalmazott kulcs indexével együtt tároljuk (z, k) párokban, z szerint sorrendezve. Ez $2^{56} \cdot 16$ byte méretű tárterület erőforrásigényt jelent. Ezután az y rejtett szöveget dekódoljuk egy k' kulccsal, s megnézzük, hogy a kapott w dekódolt szöveg szerepel-e a táblázatban. Ha szerepel, s mellette k kulcs állt, akkor (k, k') kulcspárra juttattunk. Ezen kulcspár helyességét egy további (x', y') nyílt-rejtett szöveg-páron teszteljük. Ha ezen teszten megbukik a párok, akkor folytatjuk a dekódolási lépésnél a még meg nem vizsgált kulcsok vonatkozásában. Ugyanis annak a valószínűsége, hogy egy téves kulccsal dekódolt szöveg beletálljon a táblázatba – a táblázat mérete és az összes lehetőségek aránya kapcsán – $2^{56}/2^{64} = 2^{-8}$, következetesképpen kipróbálva az összes dekódolási kulcsot, átlagosan $2^{56} \cdot 2^{-8} = 2^{48}$ számú téves kulcspár keletkezik. A második nyílt-rejtett szövegpáron történő tesztelésnél, mivel egy téves kulcspár $1/2^{64}$ valószínűséggel rendeli x' nyílt szöveghez az y' rejtett szö-

veget, ezért 2^{48} számú téves kulcspárt tekintve legfeljebb $2^{48}/2^{64} = 2^{-16}$ lesz annak a valószínűsége, hogy egy téves kulcspárra dönt végül is az algoritmus. (A tényleges kulcspár – természetesen – minden teszten.)

2. A naív kimerítő kereséses támadás erőforrásigénye 2^{112} számítási egység és elhanyagolható memóriaigény.
3. A k kulcs r bitjét rögzítetten tartjuk, s ekkor az algoritmusbeli táblázat 2^{56-r} méretű lesz, ugyanakkor le kell az algoritmust futtatni 2^r táblázatra a különböző r bites rögzítések miatt. A memóriaigény 2^{56-r} (az egyes beírások $64 + 56 - r$ bit méretűek), míg a műveletigény $2^r 2^{56-r} 2^{56} = 2^{112}$, azaz az idő- és memóriaigény szorzata nem változik.
4. Nem sikerült. Ugyanis egy kódolási lépésként kezelve a két belső kódolást: $E_{k_1}(E_{k_1}(x)) = E_{k_1}^*(x)$, az $y = E_{k_2}(E_{k_1}^*(x))$ kétszeres kódolásra jutunk, amely ellen a fentiekben elemzett támadás némi erőforrásigény növekedés mellett ismét végrehajtható. \square

2.13. Feladat:

Az első kódoló z kimenetét rögzítik zérus blokkra. Ezt úgy érhetjük el, hogy az x nyílt szöveg blokkot $x = D_{k_1}(0)$ alapján választjuk, ahol k_1 végigfut a 2^{56} méretű kulcstéren. minden egyes ilyen x nyílt szövegre kérjük az y rejtett szöveget a háromszoros kódolótól, majd ezen y rejtett szöveget az x szöveget előállító k_1 kulccsal dekódoljuk: $w = D_{k_1}(y)$ eredménnyel. A $z = 0$ blokkot dekódoljuk sorra véve k_2 kulcs értékeit, lépésekkel: $w' = D_{k_2}(0)$ eredménnyel. A $w = w'$ egyenlőséget keressük.



21.1. ábra. Választott nyílt szövegű támadás a kétkulcsos háromszoros kódolás ellen

Az erőforrásigény $3 \cdot 2^{56}$ nagyságrendű rejtjelezés művelet, 2^{56} nagyságrendű $64 + 56$ bites $((w, k_1))$ elemekből álló memóriaigény.

3. fejezet

3.1. Feladat:

e számítása: $N = 55 \rightarrow \varphi(N) = 4 \cdot 10 = 40 = 2 \cdot 2 \cdot 5 \rightarrow e = 3$. d számítása:
 $40 = 13 \cdot 3 + 1 \rightarrow 1 \cdot 40 + (-13) \cdot 3 = 1 \rightarrow d = -13 = 27 \text{ mod } 40$. $D(x) = 2^{27} = 18 \text{ mod } 55$.

3.2. Feladat:

1. $\varphi(N) = 22 \cdot 70 = 1540$.
2. $P = 0.057$ a következő alapján:

$$\begin{aligned} P &= \frac{m - \Phi(m)}{m} = 1 - \frac{(p_1 - 1)(p_2 - 1)}{p_1 p_2} \\ &= 1 - \frac{(p_1 p_2 - p_1 - p_2 + 1)}{p_1 p_2} = \frac{1}{p_1} + \frac{1}{p_2} - \frac{1}{p_1 p_2}. \end{aligned}$$

3.3. Feladat:

1. $1 = (3999996, 379) = a \cdot 3999996 + b \cdot 379$. Az euklideszi algoritmus felhasználásával: $1 = (139) \cdot 3999996 + (-1467017) \cdot 379$, ahonnan $d = -1467017 = 2532979$.
2. $\varphi(N) = (p - 1)(q - 1) = pq - (p + q) + 1 = N - (p + N/p) + 1$ alapján másodfokú egyenletre jutunk p -ben, ahonnan $p = 2003$, $q = 1999$.

3.4. Feladat:

A támadó elkészíti a lehetséges üzenetek kódolt megfelelőjét a nyilvános kódoló kulcs birtokában, s tárolja a $\{\text{nyílt üzenet}, \text{rejtett üzenet}\}$ párok halmazát. Megfigyelve a csatornában az aktuális rejtett üzenetet, a tárolt párok halmaza alapján rekonstruálni képes az üzenetet. A védekezés módja a nyílt üzenet friss véletlen elemmel bővítése a rejtjelezést megelőzően.

3.6. Feladat:

„Ismételt négyzetre emelés és szorzás” algoritmussal történő hatványozás okán. $e = 2^t + 1$ kivevő 1 moduláris szorzást és t négyzetre emelést igényel, azaz $t + 1$ a szükséges szorzási műveletek száma. Például $e = 3$ esetén 1 szorzás és 1 négyzetre emelés, $e = 2^{16} + 1$ esetén 1 szorzás és 16 négyzetre emelés szükséges.

3.7. Feladat:

1. Nem, $5^{86} = 25 \pmod{87}$.
2. Igen, $32^{32} = (-1)^{32} = 1 \pmod{33}$.
3. $b = n - 1$, ahol n páratlan választás esetén mindenig igaz, hogy $b^{n-1} = (n-1)^{n-1} = (-1)^{n-1} = 1 \pmod{n}$. $n-1$ nem alkalmas választásbázis céljára.

3.8. Feladat:

$e^2 = 1 \pmod{24}$, minden szóba jöhető e értékre, $(e, 24) = 1$, $0 < e < 24$: mivel $24 = 2^4 \cdot 3$, ezért a szóba jövő e értékek, $5, 7, 11, 13, 17, 19, 23$, amelyek mindegyikére $24|(e-1)(e+1)$ fennáll.

3.9. Feladat:

Azért tud fejteni, mivel az egész számként ábrázolható üzenetek köbének mérete is kisebb, mint a modulus, s így a rejtett szövegek köbgyökét kereszik az egészek között, ami ismert könnyű feladat. A tanulság az, hogy rövid üzenetek esetén a blokkhosszra kiegészítést az alsó helyéértékeken célszerű elvégezni, s ha egészen rövidek az üzeneteink, akkor véletlen elemekkel történő kiegészítés is szükséges.

A „kicsi kódoló kulcsok problémája” is kapcsolódik a feladathoz, amikor egy adott üzenetet ugyanazon kicsi kódoló kitevővel (e) kódolva küldünk el több partnerhez, különböző (relatív prím) modulusokkal. Ekkor a kínai maradéktétel segít ahhoz, hogy az egyszerű gyökvonásig eljusson a támadó.

3.10. Feladat:

Nem alkalmazhatók ezek a módok. Az RSA ugyan blokk kódolás, de nyilvános kulcsú, s a CFB illetve OFB mód esetén mind az adási, mind a vételi oldalon csak kódoló transzformációt alkalmazunk, s így a nyilvános kódoló kulcs miatt a támadó által is elvégezhető a dekódolás.

3.11. Feladat:

Igen. Az RSA dekódolás Euler–Fermat-tételen alapuló szokásos bizonyítása működik ezen z modulus esetére is. Mivel $z|\varphi(N)$, ezért z használata vezethet kisebb d dekódoló kulcsra, ami gyorsabb dekódolást tehet lehetővé.

3.12. Feladat:

(2. → 1.) Triviális.

(1. → 2.) Tegyük fel, hogy ki tudjuk számítani a d dekódoló kulcsot. Mivel $ed = 1 \pmod{\varphi(N)}$, ezért létezik t , amelyre $ed - 1 = t\varphi(N)$. Az Euler-Fermat-tétel alapján $b^{\varphi(N)} = 1 \pmod{N}$, tetszőleges b , $(N, b) = 1$ esetén. Egy $x^2 = 1 \pmod{N}$ alakú egyenletnek a négy lehetséges megoldása közül kettő nyilvánvaló: $x = \pm 1 \pmod{N}$. Továbbá $x = b^{t\varphi(N)/2} \pmod{N}$ is megoldása ezen másodfokú egyenletnek tetszőleges b , $(N, b) = 1$ esetén ($\varphi(N)$ páros). Véletlenszerűen választva egy b értéket, legalább $1/2$ annak a valószínűsége, hogy $b^{t\varphi(N)/2} \neq \pm 1 \pmod{N}$, ezért $(b^{t\varphi(N)/2} \pm 1, N) = p$ vagy q . A támadó tehát véletlenszerűen választ egy b értéket, majd ellenőrzi ezen l.n.k.o.-t, s ha egynél nagyobb szám adódik, az N nemtriviális faktorja. Mivel ezzel megoldanánk a faktorizálás feladatát, ami nehéz feladat, ezért a dekódoló kulcs kiszámítása az ismert adatokból hasonlóan nehéz feladat kell legyen. Megjegyezzük, hogy ha d bitmérete legfeljebb negyede az $N = pq$ modulus bitméretének, akkor létezik hatékony algoritmus a d kiszámítására (N, e) publikus adatokból. (d kisebb méretre választásának praktikus oka az lehet, hogy gyorsabb dekódolást tesz lehetővé, s ez a szempont nem mellékess kis számítási kapacitású (pl. chipkártya) eszközöknél, illetve olyan eszköz esetén, amelynél sokszor kell ezt a feladatot végrehajtani (pl. egy szerverben)).

3.13. Feladat:

A 2. megoldás hibás, mivel a láncolás során csak a második blokk tartalmát befolyásolja a véletlen blokk, tehát a támadó az első rejtjeles blokkot továbbra is azonosítani tudja (feltéve, hogy az IV inicializáló vektor ismert a támadó által).

3.14. Feladat:

A támadó felfedi az eredeti x tartalmat azáltal, hogy beszorozza y rejtjeles szöveget az üzenetek teréből választott r véletlen elem $r^e \pmod{N}$ rejtjelezettjével, s az $r^e y \pmod{N}$ szorzatot nyújtja be dekódolásra. A dekódolás eredménye $z = rx \pmod{N}$ lesz. Amennyiben r invertálható mod N (gyakorlatilag biztos esemény), akkor a támadó könnyen kiszámítja az x üzenetet. Ezen támadás ugyanakkor könnyen kivéhető, ha a támadott csak olyan dekódolt üzeneteket ad ki, amelyek előre rögzített formátumnak megfelelők.

3.15. Feladat:

Igen, megadható ilyen formátum, sőt, figyelmetlenül könnyen választhatnánk is ezt. Legyen a formátummegkötés az, hogy az érvényes üzenetek t számú lsb bitje zérus.

3.17. Feladat:

A legális felhasználó bármelyike indíthat sikeres támadást, ugyanis tetszőleges (e_i, d_i) nyilvános-rejtett kulcs pár ismeretében az N modulus faktorizálható. A p és q prímek, valamint tetszőleges e_j ismeretében d_j kiszámítható, azaz kiszámíthatja bármelyik felhasználó dekódoló kulcsát. Rendszeren kívüli támadó, ha azonos x üzenetblokk rejtelezettjeit hallgatja le, amelyet két különböző rendszerbeli felhasználónak küld a szerver, dekódolni képes az x üzenetblokkot.

3.18. Feladat:

A kérdés tehát az, hogy milyen feltételek esetén áll fenn az $x^e = x \pmod{N}$ egyenlőség. Ez triviálisan teljesül az $x = 0, x = 1, x = N - 1$ esetekben. Az $x^{e-1}x = x \pmod{pq}$, azaz az $(x^{e-1} - 1)x = kpq$ alakból azt kapjuk, hogy a keresett x üzenetek összes száma $[1 + (e-1, p-1)][1 + (e-1, q-1)]$, azaz legalább 9 ilyen tulajdonságú üzenet van ($e > 2$). Ebből a szempontból $e = 3$ egy optimális választás.

3.19. Feladat:

Igen, felelőtlennél jár el. Veszünk egy $u, 1 < u < N$ számot és $b = u^2 \pmod{N}$ számot, adjuk gyökvonás céljából A -nak. Ekkor A $1/2$ valószínűséggel $v \neq \pm u$ gyököt ad vissza. De $(\pm u)^2 - (\pm v)^2 = b - b = 0 \pmod{N}$, azaz $(u+v)(u-v) = rpq$, tehát ha v gyököt kapjuk vissza, p és q faktoroknak van közös osztója $(\pm u)^2 - (\pm v)^2$ különbséggel.

3.20. Feladat:

1. Egy támadó eltávolíthatná az aláírást, s maga írna alá. Bár nem tudná esetleg, hogy mit is írt alá, elképzelhető olyan szituáció, amikor ez kihasználható.
2. Két esetet érdemes meggondolni. $D_A(M)$ bináris blokkot bináris számként tekintve: Ha a legmagasabb helyiértéken 1 áll, akkor $D_A(M) < m_A$ miatt a legmagasabb helyiértékű 1 bitet k darab 0 bit kell kövesse. Véletlen blok-

ként modellezve az aláírást, a k darab 0 bit követelmény valószínűsége $(1/2)^k$. Ha a legmagasabb helyiértéken 0 áll, akkor nyilván kisebb, mint bármely modulus.

4. fejezet

4.2. Feladat:

Ha az iterációs függvény nem CRHF tulajdonságú, akkor hatékonyan előálítható M_1, M'_1 pár, amelyre $f(M_1, H_0) = f(M'_1, H_0)$. Az iterációs eljárás miatt $H([M_1, m], H_0) = H([M'_1, m], H_0)$ tetszőleges m üzenet esetén szintén fennáll, ezért $[M_1, m]$, $[M'_1, m]$ szintén egy ütköző üzenetpár H hash függvényhez.

4.3. Feladat:

$H([M_1, M_2], H_0) = f(M_2, f(M_1, H_0)) = H(M_2, H'_0)$, ahol $H'_0 = f(M_1, H_0)$.
(Megjegyzés: MD-kiegészítés esetén ez a támadás nem végrehajtható).

4.3. Feladat:

Tekintünk az m_1, m_2, \dots, m_t bemeneteket, s vezessük be az X_{ij} indikátort, amely 0, ha $H(m_i) \neq H(m_j)$, egyébként pedig 1 értékű. A valószínűségi modellünk alapján az $X_{ij} = 1$ esemény valószínűsége 2^{-n} , következésképpen $E[X_{ij}] = 2^{-n}$. Az összes különböző párok vonatkozásában számítva az indikátor összegek (X) várható értékét

$$E[X] = \sum_{i < j} E[X_{ij}] = t(t-1)/2^{n+1}.$$

Így az $E[X] \approx 1$ értékhez $t = 2^{n/2}$ választás adódik.

4.4. Feladat:

Lássuk be, hogy $h(x)$ ütközés-ellenálló. Nem lehet a bemenet pár minden tagjának mérete n , ami a $h(x)$ alakjából triviális. Nem lehet az sem, hogy a bemenet pár minden tagjának mérete n -től különböző, mivel $g(x)$ ütközés-ellenálló. Ugyanakkor, ha a bemenet mérete n , akkor 1 bittel kezdődő lenyomatunk van, ellenkező esetben 0 bittel kezdődő.

$h(x)$ nyilván nem OWHF, hiszen az 1 bittel kezdődő lenyomatok minden egyikéhez triviálisan ismert az ōskép (azaz az 1 bitet követő bitsorozat).

4.5. Feladat:

Vegyük észre, hogy az üzenetblokk ismeretében $H_{i-1} = D_{M_i}(H_i)$ leképezés-sel visszafelé is iterálhatunk. Tekintsünk egy tetszős szerinti – például csalárd – üzenetet (azaz üzenetblokkok sorozatát). Vágjuk két részre az üzenetet, s minden két résznek állítsuk elő $2^{n/2}$ számú – csalárd – variációját, ahol n a lenyomat mérete (például nem nyomtatható karakterek beszúrásával). H_0 -tól előre iterálva állítsuk elő az első részhez tartozó hash értéket. Másfelől – felhasználva az E kódoló transzformáció invertálhatóságát – lenyomattól viszszafelé iterálva állítsunk elő ugyancsak $2^{n/2}$ számú blokkot a második rész variációira. Nagyobb, mint 50 százalék az esélye annak, hogy a két iterációs irány illeszkedik valamely variensen.

4.6. Feladat:

Nem, mivel a kapott hash függvény ellen $O(2^{n'/2})$ -nél kisebb számításigényű születésnapi ütközéses támadás továbbra is végrehajtható. Elegendő ugyanis csak magára az n bites H_{t-1} -re végrehajtani ezt a támadást, mert, ha az $m = [M_1, M_2, \dots, M_{t-1}]$, $m' = [M'_1, M'_2, \dots, M'_{t-1}]$ üzenetpárra ütközés áll elő H_{t-1} -re, akkor M_t blokkal meghosszabbítva m és m' üzeneteket, előáll az ütközés H_t -re is, s így $[H_{t-1}, H_t]$ -re is. Ezért a nevezett támadás $O(2^{n'/4})$ számításigényű maradt, tehát ez a dimenzióövelés nem hatásos.

4.7. Feladat:

1. Indirekt. Ha lenne m, m' üzenetpár, amelyre ütközés állna elő a H' hash függvény kimenetén, akkor az nyilván ütközést jelentene a komponens $H1, H2$ hash függvények vonatkozásában is, ami e feltételeknek ellentmondana.
2. Ahhoz, hogy a kimeneten ütközés álljon elő két különböző m, m' bemenettel, az kell, hogy $g(H1(m), H2(m))$ és $g(H1(m'), H2(m'))$ megegyezzenek. Ha minden két komponens vonatkozásában ugyanazon m, m' párra tudnánk ütközést előállítani, akkor ez teljesülne. Heurisztikusan gondolkodva két „független” leképezés vonatkozásában, ezen egyszerre történő ütköztetés összeszorozná a komponensek köré ütköztetés feladat sikervalósínűségét (véletlenül sorsolva a párokat a bemeneten). Így egy erősebb függvényt kombinálhatunk.

4.8. Feladat:

- Sajnos „könnyen” támadható. Ha ugyanis M üzenethez $H([k, M])$ lenyomatot megfigyeljük, ahol k a kulcsblokk prefix, akkor egy tetszőleges m blokkal kiegészített $[M, m]$ üzenethez tartozó lenyomat az iterációs eljárás miatt nyilván $H([H([k, M]), m])$. Sőt az sem segít, ha az eljárás az MD kiegészítés trükköt is alkalmazza, mivel ekkor a bithossz is az üzenet részeként tekinthető, s a fenti generálás megismételhető.
- Sajnos ez esetben pedig a születésnapi támadás alkalmazható. A támadó $2^{n/2}$ komplexitással előállít egy M, M' üzenetpárt, amelyek lenyomataira ütközést állít elő (n bites a lenyomat). Ehhez nem szükséges a titkos k suffix ismerete. Ezután legálisan megszerez az M üzenetre egy, a feladat szerinti generálású $H([M, k])$ lenyomatot. Az iterációs lenyomat-készítés miatt képes előállítani helyes lenyomatot M' üzenetre is, hiszen $H([M', k]) = H([M, k])$.
- Ha a k blokkot alkalmazzuk mind prefixként, mind suffixként, akkor a két mondott támadást ötletében megakadályozzuk.

4.9. Feladat:

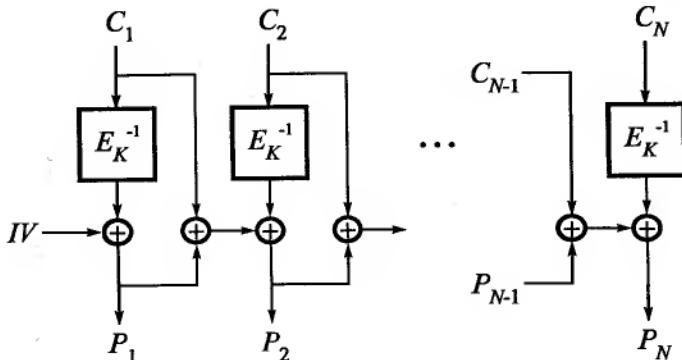
- Nem. Ezen kódolás védtelen az üzenetblokkok teszőleges permutációjára, sőt arra is, ha páros számú tetszés szerinti blokkot beillesztünk.
- Igen, lehet. Ugyanis, ha elegendően erős a rejtelezés, a támadó nem képes a rejtett szöveg módosítását úgy végrehajtani, hogy ahhoz illesszen megfelelő lenyomatot.

5. fejezet**5.1. Feladat:**

- A PCBC dekódoló sémáját a 21.2. ábrán vázoltuk.
- Tekintsük a PCBC dekódoló sémáját, és jelöljük X_i -vel azt az értéket, amit $E_K^{-1}(C_i)$ -hez XOR-olunk, hogy megkapjuk P_i -t ($i = 1, 2, \dots, N$ és $X_1 = IV$). Ekkor könnyen látható hogy

$$X_{i+2} = X_i \oplus E_K^{-1}(C_i) \oplus C_i \oplus E_K^{-1}(C_{i+1}) \oplus C_{i+1}$$

minden $i = 1, 2, \dots, N - 2$ esetén. Ez azt jelenti, hogy C_i és C_{i+1} cseréjének nincs hatása X_{i+2} -re, és így a P_{i+2}, P_{i+3}, \dots blokkokra sem.



21.2. ábra. Dekódolás PCBC módban

3. Nem önszinkronizáló. Tegyük fel, hogy az i -edik rejtett blokkban egy bit hiba keletkezik. Ez elrontja a visszaállított i -edik nyílt blokkot, amit P'_i -vel jelölünk. Mivel $C_i \oplus P_i = C'_i \oplus P'_i$ elenyészően kicsi valószínűséggel áll fenn, ezért az $(i+1)$ -edik visszaállított nyílt blokk is hibás lesz. Mivel $C_{i+1} \oplus P_{i+1} \neq C'_{i+1} \oplus P'_{i+1}$, ezért az $(i+2)$ -edik visszaállított nyílt blokk is hibás, és így tovább.

5.2. Feladat:

- Legyen $u = 2^{64}$. Az összes permutációk száma $u!$. Használjuk fel azt a tételelt, miszerint ha az összes permutációt felsoroljuk, s a b hosszú ciklusaiat tekintjük, azok összhossza b értékétől függetlenül $u!$. Ezért, ha a véletlen permutációval modellezett kódolót bármely w állapotból elindítjuk, a ciklushossz egy véletlenül választott érték lesz az $\{1, 2, \dots, u\}$ halmazból. Következésképp a ciklushossz várható értéke $(u+1)/2 = 2^{63}(+0.5)$.
- Egy visszacsatoltan alkalmazott $z_{i+1} = f(z_i)$ véletlen leképezés modellt tekintünk a 64 bites állapotok halmazán. A születésnapi paradoxon miatt átlagosan $2^{64/2} = 2^{32}$ lépést követően ugyanazt a kimenetet állítja elő, s ezzel egy ciklust ír le.

A fentiek alapján levonhatjuk azt a következtetést, hogy törekedni kell a teljes blokkméret visszacsatolására OFB mód esetén.

5.3. Feladat:

- Nem. A CBC mód hibaterjedési tulajdonsága szerint egy véletlen hiba esetén a hibázás utáni első dekódolt blokk bitjeinek átlagosan fele hibás lesz (továbbá még a rákövetkező dekódolt blokk egy bitje is). Nagymértekű meghibásodást csak igen költséges, komplex javító kóddal tudnánk eliminálni. A helyes megoldás a rejtelezés utáni hibajavító kódolás, s fejtés előtti hibajavító dekódolás alkalmazása.
- Nincs hibaterjedés a kulcsfolyamatos típusú rejtelezésmód miatt. Ez esetben alkalmazhatjuk a hibajavítást a rejtelezést megelőzően is.

6. fejezet

6.1. Feladat:

Egy rövid példán szemléltetjük a megoldás ötletét. A párbeszéd egy részlete, ahol Feri és Sanyi beszélgetnek, az alábbi (a dölt betűs sorok az utólag beszúrtak):

-
- | | |
|----|---|
| A: | Halló, Feri vagyok. |
| A: | <i>Halló, itt Jóska.</i> |
| B: | Szia, Laci. |
| B: | Szia, itt Sanyi. |
| A: | <i>Jövő hétfőn továbbmegyek Rómába.</i> |
| A: | Holnap indulok Athénba. |
| B: | Menjek én is? |
| B: | <i>Sajnos én nem tudok menni?</i> |
| A: | <i>Mindenképp gyere.</i> |
| A: | Semmiképp ne gyere. |
-

Nos, kik és hol találkoznak, vagy nem találkoznak? Az egyes sorok végére kulcsos hitelesítő ellenőrzőösszeget helyezünk, az eredeti sorok végére korrektül, a döltbetűs sorok végére szándékasan hibásat. A kulcs ismeretében a legális partnerek ki tudják szűrni a helyes sorokat, míg a támadó erre nem képes. A megoldás hátrányául két megjegyzést teszünk. Jelentős plusz kommunikáció-igény lép fel, továbbá a zavaró sorok értelmének illeszkednie kell a zavart környezethez, ahhoz, hogy ne lehessen értelme alapján nyilvánvalóan detektálni azokat (következésképp nehéz a megoldást automatizálni).

6.2. Feladat:

Vegyük egy hash függvényt, s készítsünk lenyomatot a forráskódról azon alkalmakkor, amikor a központból az illetékes személy ellenőrzésre szólít

fel telefonon. A lenyomatképzés a telefonálás időhosszához képest „pillanatok alatt” elvégezhető egy megfelelő program lefuttatásával. A lenyomatot hexadecimális ábrázolásban tekintjük, majd sorban beolvassunk ezen hexa karakterekből – a biztonsági méretezéstől függően – néhányat. A központi illetékes a hangunk alapján, s az adott napra feltett aktuális kérdéssel ellenőrizheti a személyünket, a beszélgetés friss voltát. (Itt hallgatólagosan azt is feltételeztük, hogy az ellenőrzött gépen a lenyomatkészítő program sértetlen: gondoljunk például arra az esetre, ha egy átírt „lenyomatkészítő program” egyszerűen nem tesz másat, mint azt, hogy a parancssorban ellenőrzésre kért szoftver nevéhez egy előre elkészített táblázatból kiolvassa a helyes szoftverhez tartozó helyes lenyomatot, miközben ténylegesen a szoftver már nem az eredeti.)

6.3. Feladat:

A CBC mód lépésein végiggondolva közvetlenül látható, hogy a (6.7) kódolás eredménye $E_k(m \oplus IV)|E_k(0)$ az üzenettől függetlenül, azaz semmiféle integritásvédelmet nem kapunk. A tanulság az, hogy a (6.7) alakú kódolásnál mindenkorábban eltérő kulcsot kell alkalmazni a kétféle funkcióra. Például DES alkalmazása esetére k' kulcsra egy szokásos választás a k kulcs minden második félbájtjának komplementálása.

6.4. Feladat:

1. Nem. Kimerítő kulcskereséssel igen: ugyan az üzenetblokkok véletlenek, de a kulcs tesztelésénél támaszkodhatunk a redundáns blokkra.
2. Igen. A módszer az $1, 2, \dots, m$ rejtett blokkok sorrendcseréjére, valamint páros számú ilyen blokk beszűrására érzéketlen.

6.5. Feladat:

Legyen μ tetszőlegesen választott MAC. Előállítható olyan üzenet, amelynek lenyomata μ . Tekintsünk egy $m = M_1|M_2|\dots|M_r$ üzenetet, azaz M_i üzenetblokkok r hosszúságú sorozatát, amelyre $MAC_k(m)$ CBC-MAC blokkot kaptuk. Ha az $(r+1)$ -edik üzenetblokkot $M_{r+1} = D_k(\mu) \oplus MAC_k(m)$ szerint választjuk, ellenőrizhető, hogy $m' = M_1|M_2|\dots|M_{r+1}$ üzenetre az előírt μ MAC értéket kapjuk. Ugyanakkor vegyük azt is észre, hogy bár a konstruált sorozat első r blokkját tetszszerinti csalárd tartalmára beállíthatjuk, az $(r+1)$ -edik blokk tartalma már véletlenszerűen alakul.

6.6. Feladat:

A támadó kér egy-egy MAC lenyomatot egy m_1 és egy m_2 , egy blokk méretű üzenetre. Ezután előállítja az $m = m_1|z$, $z = MAC_k(m_1) \oplus m_2$, két blokk méretű üzenetet, amelyre $MAC_k(m) = E_k(E_k(m_1) \oplus z) = E_k(E_k(m_1) \oplus E_k(m_1) \oplus m_2) = MAC_k(m_2)$. Tehát a támadó a k kulcs ismerete nélkül elő tudott állítani MAC lenyomatot egy új m üzenethez. Ugyanakkor vegyük újra észre, hogy m második blokkja a támadó által gyakorlatilag nem befolyásolható, véletlen blokk.

6.7. Feladat:

Legyen m_1, m_2 , és m_3 három darab, blokkméretű üzenet. A támadó kér MAC lenyomatot az m_1 és m_2 üzenetekre, azaz kap $A_i = E_k(E_k(m_i) \oplus \mathbf{1})$, $i = 1, 2$ lenyomatokat, ahol $\mathbf{1} = 0|...|0|1$ blokk az üzenet blokkban kifejezetten hosszát tartalmazza. Eztán lenyomatot kér az $m = m_1|\mathbf{1}|m_3$ három blokk hosszú üzenetre, azaz kap egy

$$y = E_k(E_k(E_k(m_1) \oplus \mathbf{1}) \oplus m_3) \oplus \mathbf{3} = E_k(E_k(A_1 \oplus m_3) \oplus \mathbf{3}),$$

ahol $\mathbf{3} = 0|...|0|1|1$. Látható azonban, hogy egy $m' = m_2|\mathbf{1}|z$ üzenetre, ahol $z = A_1 \oplus A_2 \oplus m_3$, szintén y lenyomatot kap a támadó:

$$E_k(E_k(E_k(m_2) \oplus \mathbf{1}) \oplus z) \oplus \mathbf{3} = E_k(E_k(A_2 \oplus A_1 \oplus A_2 \oplus m_3) \oplus \mathbf{3}) = y.$$

6.8. Feladat:

MAC kód képzése előtt az előző csomagvételi lépésekben kapott véletlen elemet, valamint egy friss véletlen elemet illesztenek az üzenetcsomaghoz. Ez a kriptográfiai szokásos láncolási technika egy megvalósítása. A kommunikáló felek a következő csomag vételekor csak olyan csomagot fogadnak el, amelynél helyes a MAC kód, valamint a megfelelő véletlen elem, az általuk az előző csomag adásakor generált elem.

6.9. Feladat:

- Nem. Ha a támadó ismeri a nyílt szöveg struktúráját, akkor célzott bitpozíciókban invertálva a biteket közvetlenül manipulálhatja a nyílt szöveget.
- Nem. A rejtejes blokkok sorrendje átrendezhető.
- Nem. A támadó tetszőleges manipulációja észrevétlen marad bármielőtt a rejtelezés alkalmazása mellett, hiszen a dekódolás eredménye tetszőleges nyílt szöveg lehet. A tanulság az, hogy rejtelezés felhasználásával

történő integritásvédelem esetén „valamiféle” redundancia léte minden- képp szükséges.

6.10. Feladat:

Igen. Első esetben a kódolás védtelen az üzenetblokkok tetszőleges permutációjára, sőt arra is, ha páros számú, tetszés szerinti blokkot szűrtunk be. Második esetben elegendően erős a rejtjelezés esetén, a támadó nem képes a rejtett szöveg módosítását úgy végrehajtani, hogy ahhoz illesszen megfelelő CRC-t.

6.11. Feladat:

Ütközés-ellenállónak kell lennie, hiszen előállítva (m, m') MDC-ütközésre vezető üzenetpárt, m „nem gyanús” üzenetre kérve a kódolást,

$$E_k(MDC(m)) = E_k(MDC(m'))$$

egyenlőség miatt m' üzenetre is ismert lesz a MAC.

8. fejezet

8.1. Feladat:

Jelölje (PK_A, SK_A) , illetve (PK_B, SK_B) A, illetve B (nyilvános, titkos) kulcs-páját. minden egyes kommunikációs kapcsolat során A véletlenszerűen generál egy, csak az adott kapcsolat idejéig élő (nyilvános, titkos) kulcspárt, amit jelöljön (PK'_A, SK'_A) . Ezen kulcspár PK'_A nyilvános elemét SK_A kulcsával aláírva A átküldi B-nek. Ezután B a PK'_A kulccsal rejtjelezett üzenetet küldhet A számára. A kommunikáció befejeztével A megsemmisíti az SK'_A titkos kulcsot. (A-tól B felé történő üzenetküldés hasonló elven történhet.) Ha a kommunikáció végén (vagy több további kapcsolat után) C megpróbálná megszerezni a rejtjelezett üzenetek dekódolásához szükséges titkos kulcsot, (itt kulcsokat) nyilván nem járhat sikerrel (meg vannak semmisítve). Ha elfeledkezünk a nyilvános kulcsú módszer követelményről, miért ne tehetnék meg, hogy az „alap” kulcspárra $((PK_A, SK_A), (PK_B, SK_B))$ támaszkodva beszél meg a két fél közös szimmetrikus kulcsot, mondjuk egy DES számára. A probléma ez esetben nyilván az, hogy C megtalálva SK_A, SK_B kulcsokat a szimmetrikus kulcsokat rekonstruálhatja, majd ennek alapján a rejtett üzeneteket is. (Megjegyezzük, hogy ha ezen szimmetrikus kulcscsereit a kapcsolat idejéig élő kulcspárra támaszkodva teszik $((PK'_A, SK'_A))$, akkor megmarad a

védettség, s a gyorsabb rejtjelezést is lehetővé tesszük.) A módszert nevezhetjük egyszer használatos nyilvános kulcsú rejtjelezésnek. Hátránya, hogy a véletlen kulcspár gyakori előállítása sok véletlen bitet kíván.

8.2. Feladat:

1. Feltevések:

$$A \equiv (A \xleftarrow{k} B)$$

$$A \equiv \#(k)$$

$$B \equiv (\xrightarrow{PuA} A)$$

$$B \equiv (\xrightarrow{PuB} B)$$

$$B \equiv \#(T_A)$$

$$B \equiv (A \Rightarrow (A \xleftarrow{k} B))$$

$$B \equiv (A \Rightarrow \#(k))$$

2. Célok:

$$A \equiv (A \xleftarrow{k} B) : \text{implicit kulcs hitelesítés } A \text{ számára (feltevés)}$$

$$A \equiv \#(k) : \text{kulcs frissesség } A \text{ számára (feltevés)}$$

$$B \equiv (A \xleftarrow{k} B) : \text{implicit kulcs hitelesítés } B \text{ számára}$$

$$B \equiv \#(k) : \text{kulcs frissesség } B \text{ számára}$$

3. Levezetés:

$$B \triangleleft \{A|k|T_A|\{A \xleftarrow{k} B|\#(k)|T_A\}_{PrA}\}_{PuB}$$

$$\underline{B \equiv (\xrightarrow{PuB} B)}$$

$$B \triangleleft (A|k|T_A|\{A \xleftarrow{k} B|\#(k)|T_A\}_{PrA})$$

$$B \triangleleft \{A \xleftarrow{k} B|\#(k)|T_A\}_{PrA}$$

$$B \triangleleft \{A \xleftarrow{k} B|\#(k)|T_A\}_{PrA}$$

$$\underline{B \equiv (\xrightarrow{PuA} A)}$$

$$B \equiv (A \succ (A \xleftarrow{k} B|\#(k)|T_A))$$

$$\underline{B \equiv \#(T_A)}$$

$$B \equiv \#(A \xleftarrow{k} B|\#(k)|T_A)$$

$$B \equiv (A \succ (A \xleftarrow{k} B|\#(k)|T_A))$$

$$\underline{B \equiv \#(A \xleftarrow{k} B|\#(k)|T_A)}$$

$$B \equiv (A \equiv (A \xleftarrow{k} B|\#(k)|T_A))$$

$$\begin{array}{l}
 B \equiv (A \equiv (A \xrightarrow{k} B | \#(k) | T_A)) \\
 B \equiv (A \mapsto (A \xrightarrow{k} B)) \\
 \hline
 B \equiv (A \mapsto \#(k)) \\
 \hline
 B \equiv (A \xrightarrow{k} B) \\
 B \equiv \#(k)
 \end{array}$$

9. fejezet

9.1. Feladat:

Legyen pw a jelszó. B választ egy jelszó bitméretű r véletlen számot:

$$\begin{array}{ll}
 (1) & B \rightarrow A : r \\
 (2) & A \rightarrow B : pw \oplus r
 \end{array}$$

9.2. Feladat:

Igen, a jelmondat. Biztonságos lenyomatképző felhasználásával szokásos jelszóhosszra tömöríthetünk hosszabb, értelmes jelmondatot. A jelmondatok számossága nyilván könnyen felülmúlja a lehetséges jelszóméretű fűzérek számát, így a teljes jelszóteret kihasználhatjuk. A megoldás hátránya az, hogy a jelmondat begépelése viszonylagosan hosszadalmas lehet a szokásos jelszavakhoz képest. A jelmondat hosszára alsó korlátot a nyelv redundanciája és a biztonságos jelszóhossz alapján kaphatunk.

9.3. Feladat:

Egy PW jelszót – a rendszeren kívül – kicsérél a két fél (ha azonosítási irányt is meg kívánunk különböztetni, akkor két különböző jelszót használunk). Azonosításkor átküldjük a $[T, Hash(T, PW)]$ üzenetet, ahol T jelöli az időt. A másik fél ellenőrizheti az időbeli frissességet, majd ezután ő is előállítja $Hash(T, PW)$ elemet. A támadó nem képes sem a PW megismerésére, sem replay, sem pedig pre-play támadásra. Hátránya, hogy klasszikus számítógépes környezetben ez megkívánja, hogy a szerver oldalon nyíltan rendelkezésre álljon a jelszó.

9.4. Feladat:

1. A támadás menete a következő:

$$\begin{array}{ll}
 X \rightarrow B: X \\
 X_A \rightarrow B: A \\
 B \rightarrow X: N_{BX}
 \end{array}$$

$$\begin{aligned}
 B &\rightarrow X_A: N_{BA} \\
 X \rightarrow B: & \{N_{BA}\}_{K_{XS}} \\
 X_A \rightarrow B: & \{N_{BA}\}_{K_{XS}} \\
 B \rightarrow S: & \{X \mid \{N_{BA}\}_{K_{XS}}\}_{K_{BS}} \\
 B \rightarrow S: & \{A \mid \{N_{BA}\}_{K_{XS}}\}_{K_{BS}} \\
 S \rightarrow B: & \{N_{BA}\}_{K_{BS}} \\
 S \rightarrow B: & \{\ast\}_{K_{BS}},
 \end{aligned}$$

ahol \ast jelöli azt a pénzfeldobás sorozatot, amit S kap, amikor $\{N_{BA}\}_{K_{XS}}$ -et dekódolja a K_{AS} kulccsal. B a szervertől kapott válaszokat helytelenül rendeli hozzá az éppen futó protokollpéldányokhoz. Ez azért van, mert csak a szervertől visszakapott kihíváselem értékét tudja ellenőrizni, és mivel az megegyezik azzal az N_{BA} -val, amit B A -nak küldött, ezért azt hiszi, a protokoll A -val futott le sikeresen, és valaki megpróbálta megszemélyesíteni X -et. Valójában azonban A egyáltalán nincs jelen a támadás alatt.

2. Egy lehetséges javítás a következő:

Javított Woo–Lam-protokoll

-
- | | |
|-----|---|
| (1) | $A \rightarrow B: A$ |
| (2) | $B \rightarrow A: N_B$ |
| (3) | $A \rightarrow B: \{B, N_B\}_{K_{AS}}$ |
| (4) | $B \rightarrow S: A, \{B, N_B\}_{K_{AS}}$ |
| (5) | $S \rightarrow B: \{A, B, N_B\}_{K_{BS}}$ |
-

Figyeljük meg az azonosítók explicit használatát az üzenetekben!

9.5. Feladat:

1. A protokoll lehet a következő:

-
- | | | |
|-----|---|---|
| (1) | $A \rightarrow B: w = g^r \bmod p$ | $(r$ véletlenszám mod $p - 1$) |
| (2) | $B \rightarrow A: b$ | $(b \in \{0, 1\}$ egy véletlen kihívás) |
| (3) | $A \rightarrow B: z = r + bx \bmod (p - 1)$ | |
-

Az utolsó lépésben B ellenőri, hogy g^z megegyezik-e $w y^b$ -vel mod p . Ezen protokollnak t egymás utáni alkalommal sikeresen kell lefutnia ahoz, hogy B 2^{-t} valószínűséggel elfogadja A állítását.

2. A titkot mod $(p - 1)$ egészket kifejezzük, majd a g primitív elemre hatványozzuk, s a hatvány eredményét átküldjük B -nek, aki számára bizonyítani kívánjuk a titok birtoklását. Ezután lezajlik a fenti bizonyítási protokoll.

13. fejezet

13.2. Feladat:

L = összetett szám. x egy egész szám. $M(x) = 1$, ha M összetett szám bemenetre dönt, $M(x) = 0$, ha M prímszám bemenetre dönt. $\Pr\{M(x) = 1\} = 3/4$, ha $x \in L$, illetve $\Pr\{M(x) = 0\} = 1$, ha $x \notin L$.

13.3. Feladat:

M' futtasson m példányt M gépből, azonos x bemenettel. A kimenetek alapján végezzen majoritásos döntést. Használjuk a Csebisev-egyenlőtlenséget. Legyen y_i az i -edik gép kimenete:

$$\begin{aligned}\Pr\left\{\frac{1}{m} \sum_{i=1}^m y_i > 1/2\right\} &= \Pr\left\{\frac{1}{m} \sum_{i=1}^m y_i - E(y_i) > 1/2 - E(y_i)\right\} \\ &= 1 - \Pr\left\{\frac{1}{m} \sum_{i=1}^m y_i - E(y_i) \leq \delta\right\} \\ &\geq 1 - \Pr\left\{\left|\frac{1}{m} \sum_{i=1}^m y_i - E(y_i)\right| \leq |\delta|\right\},\end{aligned}$$

ahol $\delta = 1/2 - E(y_i) = 1/2 - (1/2 + 1/p(n)) = -1/p(n) < 0$. A Csebisev-egyenlőtlenség felhasználásával

$$\begin{aligned}1 - \Pr\left\{\left|\frac{1}{m} \sum_{i=1}^m y_i - E(y_i)\right| \leq |\delta|\right\} &> 1 - \frac{\sigma_{y_i}^2}{\delta^2 \cdot m} \\ &= 1 - \frac{(1/2 + 1/p(|x|))(1/2 - 1/p(|x|))}{\delta^2 \cdot m} \\ &= 1 - \frac{1}{m} \left(\frac{p(|x|)^2}{4} - 1 \right).\end{aligned}$$

Így $m = p(n)^2$ választással $2/3$ feletti valószínűség adódik.

13.4. Feladat:

- Igen. Mivel $|f_{len}(x)| = \log_2 |x|$, ezért nem lenne olyan algoritmus, amely invertálni lenne képes $f_{len}(x)$ függvényt $p(|f_{len}(x)|)$ időben, egyszerűen azért, mert nem lenne elég ideje arra, hogy kiírja a bemenet bitjeit.
- Nem. Triviálisan invertálható $p(|x|)$ időben.

13.5. Feladat:

1. $1/2^n$.
2. 1.

13.6. Feladat:

Nem igaz, hiszen annak valószínűsége, hogy véletlenszerűen választott r $\log_2 |x|$ számú zérussal kezdődik, $2^{-\log_2 |x|} = 1/|x|$, így g legalább $1 - 1/n$ valószínűsséggel invertálható.

13.7. Feladat:

$$b(c, x) = c$$

13.8. Feladat:

Ha b nem kiegyenlített, s mondjuk 1 értéket nem elhanyagolhatóan gyakrabban veszi fel, akkor Z' keménybit kiszámító algoritmus kimenete legyen konstans 1, azaz $Z'(f(x)) = 1$, tetszőleges x esetén. Ezen Z' algoritmus nem elhanyagolható valószínűsséggel kiszámolja a keménybitet.

13.9. Feladat:

$$f(x) = 0^{|x|}$$

13.10. Feladat:

Legyen Z' az f függvényt invertáló algoritmus, amelynek bemenete egy véletlen r_1 értékre számított $y = f(r_1)$. Z' kisorsol egy véletlen r_2 értéket, majd meghívja Z algoritmust $(y, f(r_2))$ bemenettel. Ezt m alkalommal végezve $> 1 - (3/4)^m$ valószínűsséggel Z' kiszámít egy $f^{-1}(y)$ inverzet.

13.11. Feladat:

1. $g(w) = w$.
2. $f \circ g$ függvény nem egyirányú. Például legyen $g(w) = 0$ tetszőleges w esetén. $g \circ f$ függvény viszont egyirányú. Ha ugyanis $g \circ f$ nem lenne egyirányú, akkor g leképezés lehetne azon Z algoritmus első lépése, amely algoritmus f inverzet nem elhanyagolható valószínűsséggel számolna.
3. Nem, például $f = g$.
4. $f'(w) = [f(w), w_1 + \dots + w_{|w|}]$, ahol $+ \bmod 2$ összeadást jelöl.

13.12. Feladat:

Indirekt bizonyításul tegyük fel, hogy f' nem egyirányú. Ekkor létezik Z algoritmus, amelyhez létezik $p(n)$ polinom, hogy legalább $2^{2n}/p(n)$ számú különböző ($2n$ bites) bemenethez tartozó kimenet invertálható (végtelen sok n értékre). Vegyük észre, hogy ebből következik, hogy az f függvény vonatkozásában legalább $2^n/p(n)$ számú (n bites) bemenethez tartozó kimenet invertálható, ami ellentmond f egyirányúságának.

13.17. Feladat:

Nem. Az adott leképezés akkor és csak akkor invertálható, ha $(u, \phi(N)) = 1$. Ugyanis $x^u \equiv y \pmod{N}$ kongruencia ekvivalens az $x^u \equiv y \pmod{p}$ és $x^u \equiv y \pmod{q}$ kongruenciák rendszerével, s az utóbbiaknak csak akkor van megoldása, ha $(u, p-1) = 1$ és $(u, q-1) = 1$ fennáll, azaz ha $(u, \phi(N)) = 1$.

13.18. Feladat:

Legyen p Carmichael-szám. Mivel

$$x^{1+k\phi(N)} \equiv x (x^{p-1})^{k(q-1)} \equiv x \cdot 1 \equiv x \pmod{p},$$

ezért a dekódolás tetszőleges x nyílt szöveg blokkra helyesen megtörténik. Ha viszont p összetett szám nem Carmichael-szám, akkor a kis Fermat-tétel alapú tesztnél látottak szerint $x^{p-1} \equiv 1 \pmod{p}$ legalább a nyílt szöveg blokkok felére nem áll fenn, azaz nem működik a dekódolás.

13.19. Feladat:

Igen. Ha ugyanis a kérdéses rend r , azaz $e^r \equiv 1 \pmod{\phi(N)}$, akkor $y^{e^r} \equiv y \pmod{N}$, ahonnan $x^{e^{r-1}} \equiv x \pmod{N}$, azaz $e^{r-1} \pmod{\phi(N)}$ dekódoló kitevőként használható.

14. fejezet**14.1. Feladat:**

1. Legyen $Z = 0$, ha $0 \leq x < 2^{n-1}$ és $Z = 1$ egyébként.
2. Nem lehet nagyobb (lásd 2. Tulajdonság következményét).

14.2. Feladat:

Az analízis eredményeképp látható, hogy az n paraméterben polinomiálisan kicsi az eltérés a kétféle fejdobás valószínűsége között, ezért polinomiális számú dobással tetszőlegesen jó megkülönböztetési hibavalószínűség érhető el.

14.3. Feladat:

Először lássuk be, hogy ha létezik Z algoritmus t erőforrással, amely δ előnyvel megkülönbözteti $L(D)$ és $L(D')$ eloszlásokat, akkor létezik Z' algoritmus, amely $t+t'$ erőforrással, s ugyancsak δ előnyvel megkülönbözteti D és D' eloszlásokat. Ennek belátásához Z' bemenetét képezzük le L algoritmussal, majd az eredmény legyen Z bemenete. Z' megkülönböztető ereje nyilván megegyezik Z erejével. Ha pedig az összes lehetséges $t+t'$ erőforrással rendelkező algoritmusokat tekintjük, futtatva a megkülönböztető erő csak nőhet.

14.4. Feladat:

$d_t(D, D')$ definíciójából és a feltételes valószínűség fogalmából kiindulva elemi átalakításokkal közvetlenül adódik.

14.6. Feladat:

Indirekt: Ha az állítás nem lenne igaz, akkor létezne hatékony Z' megkülönböztető algoritmus X', Y' párra, de akkor $Z = Z' \circ L$ hatékony megkülönböztető algoritmus lenne X, Y párra.

14.7. Feladat:

Ha két eloszlás atomjainak azonos részhalmazát öszevonjuk, a variációs távolság nem változik. Tetszőleges függvény hatása egymás utáni ilyen összevonási műveletekkel képezhető, végiglépve a függvény értékkészletének elemein.

14.8. Feladat:

1. Ha a feltétel fennáll, akkor variációs távolságban közelsgé fennáll: $S^* = \{x : P_D(x) > P_{D'}(x)\}$ jelöléssel $\sum_{x \in S^*} P_D(x) - P_{D'}(x) = V(D, D')$.

2. Ha variációs távolságban közelség fennáll, akkor a feltétel fennáll: mivel $S = S^1 \cup S^2$, $S^1 \subseteq S^*$, $S^2 \subseteq \overline{S^*}$, így

$$\begin{aligned} |P_D(S) - P_{D'}(S)| &= |P_D(S^1) - P_{D'}(S^1) + P_D(S^2) - P_{D'}(S^2)| \\ &\leq |P_D(S^1) - P_{D'}(S^1)| + |P_D(S^2) - P_{D'}(S^2)| \\ &\leq |P_D(S^*) - P_{D'}(S^*)| + |P_D(\overline{S^*}) - P_{D'}(\overline{S^*})| = 2 \cdot V(D, D'). \end{aligned}$$

15. fejezet

15.1. Feladat:

1. Mutassuk meg, hogy ha létezik hatékony prediktor, akkor annak felhasználásával konstruálható hatékony megkülönböztető algoritmus is az álvéletlen sorozatra. Ha Z algoritmus egy sikeres prediktor lenne, akkor konstruálunk Z' algoritmust, amely sikerkel megkülönbözteti az álvéletlen sorozatot a véletlentől: ha Z predikciója helyes, akkor $Z' = 1$, egyébként a $Z' = 0$. Továbbá használjuk fel, hogy Z $1/2$ valószínűsséggel képes csak helyesen predikálni véletlen sorozat esetén.

2. Az indirekt bizonyítás fő lépései:

- (a) Legyen a két extrém hibrid a véletlen, illetve az álvéletlen elem. Ha megkülönböztethetők, akkor – hibrid technikával igazolva – létezik két megkülönböztethető szomszédos hibrid is, azaz létezik olyan $l(n)$ bites bináris valószínűségi változó pár, amelyek közül az egyiknek k bites prefixe, a másiknak $k - 1$ bites prefixe az álvéletlen sorozat megfelelő prefixe, míg a suffixek péntfeldobás bittel vannak feltöltve.
- (b) Ezen két hibridből kiindulva, igazoljuk, hogy az álvéletlen prefix k -adik bitje predikálható. A bizonyításnak ezen része használhatja az $1 - 1$ értelmű OWF függvényre alapuló 1 bittel hossznövelő standard bizonyítás technikai lépéseit. \square

15.2. Feladat:

Igazoljuk, hogy a PRG kimenetének eloszlása és az egyenletes eloszlás statisztikai távolsága egy pozitív konstansnál nagyobb, n értékétől függetlenül:

$$\begin{aligned} \sum_z |\Pr\{U_{l(n)} = z\} - \Pr\{G(U_n) = z\}| \\ \geq \sum_{z \notin \{G(s) : s \in \{0,1\}^n\}} (\Pr\{U_{l(n)} = z\} - \Pr\{G(U_n) = z\}) \\ = (2^{l(n)} - 2^n) \cdot (2^{-l(n)} - 0) \geq 1/2. \end{aligned}$$

15.3. Feladat:

1. Tekintsük a Blum–Micali–Yao-konstrukciót, azaz egy $F : \{0, 1\}^n \rightarrow \{0, 1\}^n$ biztonságos, egyirányú permutációval képezzük le a véletlen magot, majd hosszabbítjuk meg F keménybitjével.
2. Legyen $G' : \{0, 1\}^{n-1} \rightarrow \{0, 1\}^{n+1}$ egy biztonságos PRG. Definiáljuk a következő $G' : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$ leképezést: $G(x, b) = G'(x)$, ahol b egy bit. Így $(x, 0)$ és $(x, 1)$ ugyanabba a $G'(x)$ kimenetbe képződik, tehát G nem $1 - 1$ értelmű. Azt kell még belátni, hogy G is egy PRG. Indulunk ki a PRG definíciójából és alkalmazzuk a feltételes valószínűség technikát $b = 0$ illetve $b = 1$ feltételekkel.

15.4. Feladat:

Indirekt: Ha az állítás nem lenne igaz, létezne hosszmérést végző Z megkülönböztető algoritmus: Z bemenete w , amely vagy a generátor egy kimenete, vagy egy $h(n)$ hosszú véletlen sorozat. Z algoritmus $p(n)$ alkalommal kisorsol véletlen magot, majd generáltatja hozzájuk G -vel a kimenetet. Méri a kimenetek hosszának minimumát. Ha ez a minimum kisebb, mint $|w|$, akkor $Z(w) = 1$ (arra dönt, hogy w a generátor egy kimenete volt), egyébként a $Z(w) = b$, ahol b pénzfeldobás bit. Formálisan analizáljuk Z megkülönböztető erejét.

15.5. Feladat:

Tekintsük a PRG definícióját, s vegyük figyelembe, hogy a sorrendcsere egy polinomiális lépés.

15.6. Feladat:

- G' : $h(s)$ eloszlása egyenletes, ha s eloszlása egyenletes, továbbá G' polinomiális időben kiértékelhető marad.
- G'' : A PRG definícióját tekintsük: a $h(U_{l(n)})$ és az $U_{l(n)}$ eloszlása azonos (egyenletes), így

$$\begin{aligned} & |\Pr\{Z[h(G(s))] = 1\} - \Pr\{Z[h(U_{l(n)})] = 1\}| \\ &= |\Pr\{Z'(G(s)) = 1\} - \Pr\{Z'(U_{l(n)}) = 1\}| \end{aligned}$$

különbséget vesszük, ahol $Z' = Z \circ h$. Ha tehát Z megkülönböztető algoritmus lenne G'' PRG esetén, akkor Z' megkülönböztető algoritmus lenne G

PRG esetén is. Továbbá G'' polinomiálisan kiszámítható. (Vegyük észre, hogy itt nem használtuk ki, hogy h permutáció.)

15.7. Feladat:

Teljes indukció: Tegyük fel, hogy $G^{(i)}(U_n)$ álvéletlen eloszlású, ami $i = 1$ esetén teljesül. Ekkor $G^{(i+1)}(s) = G(C(G^{(i)}(s)))$ is álvéletlen eloszlású, ellenkező esetben $G(C(\cdot))$ leképezés megkülönböztető lenne, azaz megkülönböztetné a $G^{(i)}(U_n)$ és az U_{n+1} eloszlást.

15.8. Feladat:

Tekintsük a $\Pr\{Z(U_{2n}) = 1\}$ valószínűséget, amely a PRG definíciója alapján a fenti két valószínűség egyikétől sem lehet „távol”. Alkalmazzunk bebővíést és háromszög-egyenlőtlenséget.

15.9. Feladat:

Ha létezik egyirányú függvény, akkor biztonságos $G : \{0,1\}^{n/2} \rightarrow \{0,1\}^n$ PRG is konstruálható. Legyen $B(1^n)$ kimenet véletlen, s így eloszlása statisztikailag egyenletes, továbbá legyen $C(1^n)$ kimenet a G által generált. G definíciója szerint a két eloszlás megkülönböztethetetlen, továbbá az eloszlások tartói metszetének mérete nyilván $\leq 2^{n/2}$.

Tegyük fel, hogy a két eloszlás megkülönböztethetetlen, s tartóik a fenti értelemben lényegében diszjunktak. Legyen $B(1^n)$ ismét a véletlen leképezés. Ha $C(1^n)$ algoritmus m bites r véletlen elemet használ egy kimenete kiszámításához, definiálunk egy $f : \{0,1\}^m \rightarrow \{0,1\}^n$ leképezést $f(r) = C(1^n)$ módon. Legyen továbbá I egy invertáló algoritmus az f leképezéshez, továbbá egy Z algoritmus az alábbi:

Legyen u az f képterének (C kimenet terének) egy eleme. $Z(u) = 1$, ha $f(I(u)) = u$, azaz ha I algoritmus ki tudta számítani u egy ōsképet, egyébként legyen $Z(u) = 0$. Tegyük fel, hogy $\Pr\{Z(C(1^n)) = 1\} = \Pr\{f(I(f(r))) = f(r)\} \leq \varepsilon$, ahol ε nem elhanýagolható. Mivel $B(1^n)$ véletlen leképezés, ezért $\Pr\{Z(B(1^n)) = 1\} < 1/2^{n/2}$, ugyanis Z a $B(1^n)$ kimeneti halmazának csak azon töredékén belül invertálhat sikeresen, amely metsződik $C(1^n)$ kimeneti halmzával. Innen $\Pr\{Z(C(1^n)) = 1\} - \Pr\{Z(B(1^n)) = 1\} > \varepsilon - 1/2^{n/2}$, azaz $B(1^n)$ és $C(1^n)$ eloszlások megkülönböztethetők ($\varepsilon - 1/2^{n/2}$ nem elhanýagolható), amivel ellentmondásra jutottunk az állítás feltételével. Következésképp f biztonságos OWF.

16. fejezet

16.1. Feladat:

Legyen F'_n egy PRF, amelyből ellenpélda F_n úgy képezhető kis módosítással, hogy F_n a 0^n bemenetet konstans kimenetbe képezze le, míg a többi bemenetet az F'_n leképezésnek megfelelően. F_n nyilván nem PRF, de rendelkezik a fenti tulajdonsággal.

16.2. Feladat:

Igazoljuk először, hogy ha létezik PRG, akkor létezik olyan is, amelyre teljesül, hogy $G(0^n) = 0^{2^n}$. Ugyanis indirekcióval könnyen látható, hogy a PRG tulajdonság nem változik akkor, ha egy PRG két különböző bemenetéhez tartozó kimenetet megcseréljük. Ha ezt már beláttuk, akkor ezen PRG esetén a feladatbeli konstrukció az s magtól függetlenül zéró bemenetre zéró kimenetet ad, azaz nem lehet PRF.

16.3. Feladat:

1. $W = F_k(N)$, ahol F_k PRF. A biztonsági tárigény a k kulcs mérete, azaz konstans, s nem nő az eladott zárak darabszámaival. Vegyük észre, F_k nem kell, hogy permutáció, azaz rejtjelező leképezés legyen.
2. Ekkor egy H egyenletes eloszlású (azaz „valódi”) véletlen függvény szükséges: $W = H(N)$. Ekkor függetlenül sorsolt ’jelszavak’ adják a kombinációkat.

16.4. Feladat:

A bizonyítást hibrid bizonyítás-technikával végezhetjük el, ahol a két extém hibrid $DES_{H_n}^3$ és $DES_{H_n}^4$, míg a többi három hibridet úgy kapjuk, hogy a 16.3. ábra struktúrájában először $H_n^{(1)}$ leképezést cseréljük le $F_n^{(1)}$ leképezésre, majd a $H_n^{(2)}$ leképezést is lecseréljük $F_n^{(2)}$ leképezésre, s legvégül a $H_n^{(3)}$ leképezést cseréljük le $F_n^{(3)}$ leképezésre. Z megkülönböztető algoritmus bemenetére $\alpha = (\alpha_1, \alpha_2, \alpha_3)$, $\alpha_i \in \{0, 1\}^{2^n}$ vektor érkezik, ahol $\alpha^{(1)} = (F_n^{(1)}(U_n^{(1)}), F_n^{(2)}(U_n^{(2)}), F_n^{(3)}(U_n^{(3)}))$ vagy $\alpha^{(0)} = (H_n^{(1)}, H_n^{(2)}, H_n^{(3)})$ $1/2-1/2$ valószínűséggel.

16.5. Feladat:

1. (G', F', I') csapda permutáció (k, pk) nyilvános és sk csapda kulccsal:

Ha egy Z algoritmus (k, pk) ismeretében $z = F'((k, pk), x)$ bemenetből ki tudná számítani x ōsképet, akkor tudnánk egy Z' algoritmust konstruálni, amely invertálni képes $F(pk, \cdot)$ leképezést is: egy $y = F(pk, x)$ bemenet esetén Z' véletlenszerűen választ egy k kulcsot, majd $F'((k, pk), x) (= E(k, y))$ bemenettel meghívja a Z algoritmust, amely kiszámítja y ōsképet az sk titkos csapda kulcs nélkül, s ez ellentmondana annak, hogy F csapda permutáció.

2. (G', F', I') álvéletlen permutáció:

Ha egy Z algoritmus meg tudná különböztetni F' permutációt egy véletlen permutációtól, akkor tudnánk egy olyan Z' algoritmust konstruálni, amely képes megkülönböztetni E permutációt egy (egyenletes eloszlású) véletlen permutációtól: Z' hozzáfér az O , O^{-1} orákulumokhoz (amely vagy az E , D , vagy pedig a véletlen permutáció és inverze, amit jelöljön π illetve π^{-1}). Z' szimulálja Z környezetét, az F' és I' orákulumokat. Z' meghívja G publikusan elérhető kulcsgenerátort, s generál (pk, sk) véletlen kulcs-párt.

O -kérés: amikor Z egy x kérésre $F'((k, pk), x)$ választ várja, Z' x bemenettel előbb kiszámítja $v = F(pk, x)$ értéket, majd ezt mint kérést elküldi O orákulumnak, s annak $O(v)$ válaszát küldi Z' válaszként Z kérésére.

O^{-1} kérés: amikor Z egy z kérésre $I'(sk, z)$ választ várja, Z' z bemenettel előbb kiszámítja $w = I(sk, z)$ értéket, majd ezt mint kérést elküldi O^{-1} orákulumnak, s annak $O^{-1}(w)$ válaszát küldi Z' válaszként Z kérésére.

Amikor $(O, O^{-1}) = (E, D)$, akkor Z az F', I' leképezéseknek megfelelő válaszokat kapja, míg amikor $(O, O^{-1}) = (\pi, \pi^{-1})$, akkor Z egy véletlen permutációval és inverzával megfelelő válaszokat kapja (ti. F egy permutáció, így $\pi \circ F, \pi^{-1} \circ I$ is egy (egyenletes eloszlású) véletlen permutáció és annak inverze).

18. fejezet

18.1. Feladat:

Indirekt bizonyítást alkalmazhatunk minden részfeladat esetén: ha könnyű lenne a feladat, nem lenne a rejtelező *ind-cpa*-biztonságú.

A. függelék

Kapcsolódó szabványok

USA ANSI szabványok

ANSI #	Tárgy
X9.17	kulcsgondozás és véletlenszám-generálás
X9.30-1	digitális aláírás algoritmus (DSA)
X9.30-2	SHA hash függvény a DSA számára
X9.31-1	RSA aláíró algoritmus
X9.31-2	hash függvények az RSA számára
X9.42	Diffi e-Hellman kulccscsere
X9.45	attribútum-tanúsítványok
X9.52	3DES
X9.55	tanúsítványok és visszavonási listák
X9.57	tanúsítvány menedzsment

USA FIPS szabványok

FIPS #	Tárgy
FIPS 46-2	a DES leírása
FIPS 74	irányelvek a DES használatához
FIPS 81	a DES működési módjai
FIPS 112	jelszavak használata
FIPS 113	adat hitelesítés (CBC-MAC)
FIPS 140-1	kiptomodulokkal kapcsolatos biztonsági követelmények
FIPS 171	kulcsgenerálás
FIPS 180-1	a SHA-1 hash függvény
FIPS 185	kulcs-escrow (Clipper & SKIPJACK)
FIPS 186	digitális aláírás szabvány (DSA és ECDSA)
FIPS 196	partner hitelesítés

Nemzetközi ISO szabványok	
ISO #	Tárgy
7498-2	OSI biztonsági architektúra
8372	64 bites blokkrejtjelezők működési módjai
9594-8	hitelesítési keretrendszer (X.509)
9796	digitális aláírás üzenet visszaállítással (pl. RSA)
9797	integritásvédelmi mechanizmusok (MAC)
9798-1	partnerhitelesítés – bevezető
9798-2	– szimmetrikus rejtjelezéssel
9798-3	– nyilvános kulcsú technikákkal
9798-4	– kulcsolt egyirányú függvényekkel
9798-5	– zero-knowledge technikák felhasználásával
9979	criptográfiai algoritmus nyilvántartás
10116	<i>n</i> bites blokkrejtjelezők működési módjai
10118-1	hash függvények – bevezető
10118-2	– blokkrejtjelezőkből konstruált
10118-3	– dedikált algoritmusok
10118-4	– moduláris aritmetikára épülő
11770-1	kulcsgondozás – bevezető
11770-2	– szimmetrikus kulcsú technikák
11770-3	– aszimmetrikus kulcsú technikák
13888-1	letagadhatatlanság – bevezető
13888-2	– szimmetrikus kulcsú technikák
13888-3	– aszimmetrikus kulcsú technikák
14888-1	digitális aláírás (hash-and-sign technikák) – bevezető
14888-2	– identitás alapú mechanizmusok
14888-3	– tanúsítvány alapú mechanizmusok
15946	elliptikus görbékre épülő kriptográfiai technikák

PKCS (Public-Key Cryptography Standards) szabványok

PKCS #	Cím
PKCS 1	RSA encryption standard
PKCS 3	Diffi e-Hellman key-agreement standard
PKCS 5	Password-based encryption standard
PKCS 6	Extended-certificate syntax standard
PKCS 7	Cryptographic message syntax standard
PKCS 8	Private-key information syntax standard
PKCS 9	Selected attribute types
PKCS 10	Certification request syntax standard
PKCS 11	Cryptographic token interface standard

Néhány kapcsolódó Internet RFC

RFC #	Tárgy
RFC 2104	a HMAC algoritmus leírása
RFC 2246	TLS (Transport Layer Security) protokoll (SSL 3.1)
RFC 2401	az IPSec biztonági architektúra áttekintése
RFC 2402	– az AH protokoll leírása
RFC 2406	– az ESP protokoll leírása
RFC 2408	– az ISAKMP protokoll leírása (kulccscsere)
RFC 2409	– az IKE protokoll leírása (kulccscsere)

Néhány kapcsolódó GSM specifikáció

3GPP TS #	Tárgy
55.205	példák az A3 és az A5 algoritmusokra (GSM MILENAGE)
55.216	az A5/3 és a GEA3 (GPRS) rejtjelező algoritmusok leírása

Irodalom

- [1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. In *IEEE Transactions on Software Engineering*, 22(1), 1996.
- [2] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the First Annual ACM Conference on Computer and Communications Security*, 1993.
- [3] M. Bellare and P. Rogaway. Optimal asymmetric encryption – How to encrypt with RSA. In *Advances in Cryptology – EUROCRYPT'94*, Lecture Notes in Computer Science 950, Springer-Verlag, 1995.
- [4] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *Advances in Cryptology - EUROCRYPT'96*, Lecture Notes in Computer Science 1070, Springer-Verlag, 1996.
- [5] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, New York, 1993.
- [6] D. Bleichenbacher. A chosen ciphertext attack against protocols based on the RSA encryption standard RSA PKCS #1. In *Advances in Cryptology – CRYPTO'98*, Lecture Notes in Computer Science 1462:1–12, Springer-Verlag, 1998.
- [7] M. Blum and S. Micali. How to generate cryptographically strong pseudo-random bits. *SIAM Journal of Computing*, 13(4):850–863, November 1984.

- [8] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203-213, 1999.
- [9] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *Proceedings of the Royal Society*, December 1989.
- [10] R. Cramer and V. Shoup. A practical public key cryptosystem probably secure against adaptive chosen ciphertext attack. In *Advances of Cryptology – Crypto'98*, Lecture Notes in Computer Science 1462, Springer-Verlag, 1998.
- [11] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag, 2001.
- [12] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):210–217, 1986.
- [13] O. Goldreich and L. Levin. A hard predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, 1989.
- [14] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer-Verlag, Algorithms and Combinatorics, Vol. 17, 1998.
- [15] A. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, April 1984.
- [16] N. Koblitz. *A Course in Number Theory and Cryptography*. Graduate Texts in Mathematics, No. 114, Springer-Verlag, New York, 2nd edition, 1994.
- [17] N. Koblitz. *Algebraic Aspects of Cryptography*. Algorithms and Computation in Mathematics, Vol. 3, Springer-Verlag, New York, 1998.
- [18] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal of Computation*, 17(2), April 1988.
- [19] A. Mehrotra and L. Golding. Mobility and security management in the GSM system and some proposed future improvements, In *Proceedings of the IEEE*, 86(7):1480–1496, July 1998.
- [20] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1996.
- [21] Nemetz T. és Vajda I., Algoritmusos adatvédelem. Akadémiai Kiadó, 1991.

- [22] D. O'Mahony, M. Peirce, and H. Tewari. *Electronic payment systems*. Artech House Inc, Boston, 1997.
- [23] B. Preneel. Analysis and Design of Cryptographic Hash Functions. Doctoral Dissertation, Katholieke Universiteit Leuven, 1993.
- [24] B. Schneier. *Applied Cryptography*. Wiley, 1996.
- [25] D. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995.
- [26] S. Vaudenay. Security flaws induced by CBC padding – Applications to SSL, IPSEC, WTLS, ... In *Advances in Cryptology – EUROCRYPT'02*, Lecture Notes in Computer Science 2332, Springer-Verlag, 2002.
- [27] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the 2nd Usenix Workshop on Electronic Commerce*, 1996.
- [28] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.