

Információs rendszerek biztonságtechnikája

Vassányi István, Dávid Ákos, Smidla József,
Süle Zoltán



2014

A tananyag a TÁMOP-4.1.2.A/1-11/1-2011-0104 "A felsőfokú informatikai oktatás minőségének fejlesztése, modernizációja" c. projekt keretében a Pannon Egyetem és a Szegedi Tudományegyetem együttműködésében készült.



1. rész: Kriptográfia

Tartalomjegyzék

1. Történeti áttekintés	4
2. A modern kriptográfia alapjai	9
3. Modern blokkos kriptorendszer.....	13
4. Modern folyam-elvű kriptorendszer.....	16
5. Nyilvános kulcsú kriptorendszer és alkalmazásaik.....	19
6. Üzenetpecsétek	24
7. A kriptográfia jövője	27
Irodalom.....	33
I.MELLÉKLET: AZ AES KRIPTOGRÁFIAI ALGORITMUS	34
Bevezetés	34
Matematikai alapok	35
A kulcs kiterjesztése	39
Kódolás.....	39
Dekódolás.....	43
II. MELLÉKLET BLOKK KÓDOLÓK ÜZEMMÓDJAI.....	47
Elektronikus kódkönyv (ECB)	47
Titkositott blokkok láncolása (CBC).....	49
Kimenet visszacsatolása (OFB).....	51
Kódolt üzenet visszacsatolása (CFB)	53
Számláló mód (CTR)	55
III. MELLÉKLET AZ RSA ALGORITMUS	56
Matematikai alapok	56
Az RSA algoritmusa	61
Az RSA alkalmazása digitális aláírásra	63
IV. MELLÉKLET A SZÜLETÉSNAP-TÁMADÁS MATEMATIKAI HÁTTERE	67
V. MELLÉKLET AZ SHA-1 ÜZENETPECSÉT	70
Alapműveletek	70
Az SHA-1 működése	71
VI. MELLÉKLET A GRAIN-128 FOLYAMTITKOSÍTÓ	75
VII. MELLÉKLET A RABBIT FOLYAMTITKOSÍTÓ	78

1. Történeti áttekintés

A szteganográfia

A kriptográfia nagyon régi tudomány, melynek alkalmazásai az ókorba nyúlnak vissza. A legrégebbi időkben nem a ma megszokott, titkos vagy nyilvános kulcsokra alapozott **kriptográfiai** módszerek, hanem a **szteganográfia**, azaz a titkolni kívánt üzenet elrejtése vagy álcázása volt használatban. Néhány klasszikus példa:

- az i.e. 480-ban lezajlott szalamiszi csata, mely a perzsa hajóhad döntő vereségével végződött, illetve Xerxes egész görög hadjárata is másképp végződhetett volna, ha a hadjárat előtt egy Perzsiába száműzött görög nem figyelmezteti honfitársait a közelgő támadásra egy rejtett üzenettel, melyet egy viasszal bevont, üresnek látszó írótáblán, a viaszréteg alatt helyezett el. Az „üres” írótábla gond nélkül eljutott Perzsiából a címzettekhez.
- a diplomáciai levelezésben alkalmazták a küldönc fejére írt, lábbelijébe rejtett stb. üzeneteket, melyekről maga a küldönc sem tudott.
- elhalványuló, de valamilyen kezelésre megjelenő tintával (pl. tej, citromlé, pitypang-nedv stb.), vagy nem látható helyekre (postabélyeg alá, kemény tojás héja alá stb.) írtak üzeneteket.

A fenti szteganografikus módszerek közös jellemzője, hogy a feladónak és a címzettnek előzetesen és titokban meg kellett állapodni a rejtés módjában, és ha ezt a támadó megtudta, akkor a rejtett üzenetet könnyen el tudta olvasni, akár meg is tudta hamisítani. Mivel pedig a titkos módszerek előbb-utóbb kitudódnak, ezért a szteganográfia alkalmazása általában egyedi és erősen korlátozott. Ennek ellenére még a II. világháborúban is alkalmazta pl. a francia ellenállás. Napjainkban pedig a különféle digitális tartalmak, elsősorban kép- és hangfájlok, de akár teljes relációs adatbázisok digitális vízjelzésére (digital watermark) is alkalmazzák a szteganográfia elvét. A cél lehet a szerzői jogi információk elrejtése vagy a zajcsökkentés is. A leggyakrabban azonban az üzenet elrejtését nem önmagában, hanem valamilyen kriptográfiai módszerrel kombinálva alkalmazzák. A szteganográfia és a kriptográfia közötti átmenetre példa az üzenet betűinek összekeverése valamelyen előre megállapodott séma szerint. Néhány példa:

- A sokszögletű pálca, melyet már az ókorban is alkalmaztak. A pálcára spirálisan egy szíjat tekertek, majd az üzenetet a szíjra, a pálca oldalai mentén írták fel. Letekerés után a betűk összekeveredtek, pl. ötszög keresztmetszetű pálca esetén ötös csoportokban. A címzettnek ugyanolyan pálcával kellett rendelkeznie, mint a feladónak, melyre a szíjat feltekerve az üzenet elolvashatóvá vált.
- A Cardano-rejtjelező egy rács, melyet a négyzetrácsos cellákra osztott titkos üzenet fölé helyeztek, és a megjelölt cellák betűit olvasták össze. A rács 90 fokos elforgatásával és ismételt alkalmazásával akár hosszabb szövegek betűit is össze lehet keverni.

Az első példában a sokszögletű pálca, a másodikban a rács tekinthető egyfajta titkos kulcsnak, amely megkönnyíti az üzenetmegfejtését. A betűk összekeverésének, **permutálásának** elvét **P-doboz** néven a modern kriptográfia is alkalmazza.

A klasszikus kriptorendszerek

A történelem megmutatta, hogy a támadó általában előbb-utóbb megismeri a titkosan kommunikáló felek által használt módszert, sőt általában a titkos üzenet nyelvét és téma-ját is. Olyan módszert kellett tehát találni, melynek feltöréséhez ezeken kívül még egy titkos kulcs is szükséges. Előnyös, ha a kulcsot nehéz kitalálni, és ha a felek időről időre új kulcsokat alkalmaznak. Néhány klasszikus példa:

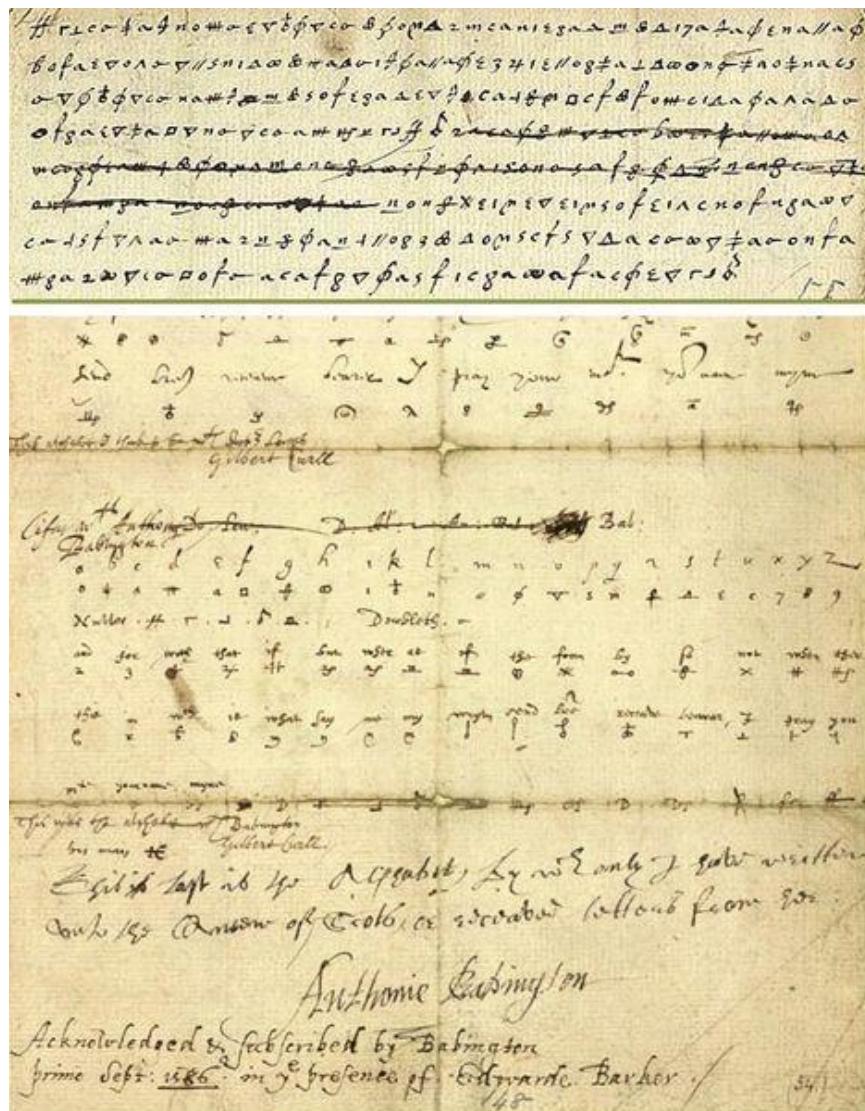
- A Caesar-kód: minden betű helyett az utána következő n-edik betűt írjuk. A lehetséges kulcsok száma így (26 jegyű ABC-t használva) mindössze 26. A Római Birodalomban alkalmazták.
- Ennek továbbfejlesztett változata minden betű szisztematikus helyettesítése egy másik betűvel. Így már 26! kulcs lehetséges, azonban a kulcs egy táblázat, amit nehezebb titkosan kezelni. Ezt a műveletet **S-doboz** néven ismeri a modern kriptográfia.

A fenti módszert **egyszerű monoalfabetikus** (betűhelyettesítéses) kódnak nevezzük. Ennek törésére a IX. századig kellett várni, amikor arab tudósok a Korán kéziratainak eredetiségét vizsgálva feltalálták a **betűgyakoriság-statisztikákra** alapozott elemzést. Az üzenet nyelvének és téma-jának ismeretében a betűpárok könnyűszerrel felderíthetők a természetes nyelvek redundanciáját felhasználva, feltéve, hogy elegendően hosszú üzenet áll rendelkezésre¹. Ennek ellenére a monoalfabetikus kriptorendszereket, időnként a virágnyelvvel kombinálva, Európa-szerte alkalmazták (és rutinszerűen törték) a diplomáciában. A támadások megnehezítésére a következő javításokat alkalmazták:

- egyes jelek betűket, mások szótárok vagy egész szavakat jelentettek
- más jelek nem jelentettek semmit (*nullitások*), vagy duplázták, esetleg törölték az előttük álló jelet

Így már olyan mértékben el lehetett bonyolítani a rendszert, hogy például az 1626 után XIII. és XIV. Lajos udvarában a Rossignolok által tervezett, diplomáciai titkok lejegyzésére használt „Nagy Kód”-ot csak 200 ével később tudták megfejteni. Azonban nem mindig vizsgázott a kód ilyen jól: egy hasonló kódöt, melyet a börtönben lévő Mária skót királynő használt az angol királyné ellen összeesküvő katolikus angol nemesekkel való kapcsolattartásra, könnyedén feltört Erzsébet angol királynő titoknoka, sőt a levelezést meghamisítva még a lázadó urak neveit is kicsalta. Valószínű, hogy a lázadók nem írták volna meg a neveket, ha nem hisznek vakon a kód biztonságában, tehát a kriptográfia alkalmazása ebben az esetben kifejezetten káros, sőt végzetes volt a számukra. Ez a jelenség számtalanszor ismétlődött már a történelem során, és a „**korhadt kilátókorlát jelenség**” néven ismeretes. A gyenge kód Mária királynő és az angol nemesek kivégzéséhez vezetett 1586-ban. Alább látható a titoknak által Mary leveléhez hamisított rész, melyben Mary kéri szövetségeit, írják meg a nevüket (felül), alatta pedig a levelezéshez használt betű- és szókódok.

¹ lásd <http://www.unsecure.co.uk/attackingmonoalphabeticciphers.asp>



ciklikus eltoltjai valahány pozícióval, például a 2-es számú tábla a B-t D-re cseréli. Ebben az esetben a lehetséges táblák száma: az eredeti ABC betűinek a száma - 1. A kulcs a felhasznált táblák sorszáma, pl. 21, 2, 5, 7, 2 egy öt hosszú kulcs esetén. A módszer továbbfejleszthető tetszőleges betű-összerendelést megengedő táblák használatával, ami a lehetséges táblák számát jelentősen megnöveli (44 jegyű ABC esetén 44!). Ekkor viszont a titkos csatornán előzetesen megosztandó titok is sokkal nagyobb méretű lesz, hiszen meg kell állapodni az egyes táblák tartalmában.

Az egyszerű betűgyakoriság-statisztikák alkalmazása ennél a módszernél azért nem működik, mert az egyes betűk pozíciójától függően változik a helyettesítési szabály. 1854-ig kellett várni, míg egy angol matematikus-filozófus, Charles Babbage, sikерrel járt. A törés alapgondolata az, hogy először a kulcs hosszát, tehát az alkalmazott táblák számát kell megállapítani, utána az egyes táblák tartalmát a szokásos betűgyakoriság-statisztikai módszerrel ki lehet deríteni. Például ha a kulcs hossza 6, akkor a második tábla felderítéséhez csak a 2., 8., 14. stb. pozíciókon álló betűkből készítünk statisztikát. Természetesen a támadónak hosszabb (jelen esetben 6-szor olyan hosszú) rejtejtő szövegre van szüksége a sikereshez, mint az egyszerű monoalfabetikus esetben. A kulcs hossza pedig abból határozható meg, hogy minden nyelvben vannak nagyon gyakori betű-kapcsolatok, például az angolban ilyen a th. Ha két th távolsága a nyílt szövegben éppen a kulcs hossza, vagy annak egész számú többszöröse, akkor a rejtejtő szövegen is ugyanaz a két betű fog tartozni hozzájuk. Tehát a rejtejtő szövegen ismétlődő betűcsoportokat keresünk, ezek távolságainak pedig megkeressük a legnagyobb közös osztóját. Valószínű, hogy ez lesz a kulcs hossza.

A polialfabetikus kriptorendszerek további fejlődésének az I. és különösen a II. világháború adott nagy lendületet. A táblázatok változatát írógép-szerű titkosítógepek végezték. Az I. világháború kimenetét alapvetően meghatározta a korábban németbarát külpolitikát folytató USA belépése Németország ellen, ami egy titkosított diplomáciai távirat (az ún. Zimmermann-féle távirat) sikeres feltörésének volt köszönhető az angol titkosszolgálat részéről. A II. világháborúra a németek kifejlesztették az **Enigmát**, egy rotoros titkosítógépet, amelynek feltörhetetlenségében a német vezetés az intő jelek ellenére vakon hitt. Azonban az angol titkosszolgálatnak A. Turing közreműködésével sikerült a törés, ezért a szövetségesek a németek tudta nélkül éveken keresztül meg tudták fejteni a német hadvezetés által kiadott parancsok és egyéb üzenetek nagy részét. Ez akkora előnyhöz juttatta a szövetségeseket, amely nélkül a világháború kimenetele egészen másmilyen is lehetett volna, de a történészek szerint legalábbis minimum 5-7 évvel tovább tartott volna a harc. Talán ez a legnagyobb történelmi jelentőségű példája a „korhadt kilátókorlátba kapaszkodás” veszélyeinek.

Az Enigma, Arthur Scherbius találmánya, valóban tökélyre fejlesztette a nagyon nagy számú kód-ABC közötti változatát. Egymás után elhelyezett forgó tárcsákon lévő huzal-darabok (a rotorok) egy áramkört zártak, amely a billentyűzeten leütött betűhöz a tárcsák pillanatnyi állása szerint meggyűjtött egy lámpát valamelyik betűnél. Tehát a nyílt szöveg begépelése során a lámpákról betűnként le lehetett olvasni a rejtejtő szöveg betűit és viszont. A tárcsák minden leütés után fordultak egyet, ezért a kód-ABC gyakorlatilag minden betű esetén más volt, betűgyakoriság-elemzést tehát nem lehetett alkalmazni. A kulcs a tárcsák induló helyzetéből, sorrendjéből és egy betűcserélő tábla (plugboard) beállításából állott, ezekkel a lehetséges kulcsok száma több, mint 10^{16} volt. Az angolok rendelkeztek az Enigma egy példányával, azonban így is több évi erőfeszítés és Alan Turing zsenialitása kellett hozzá, hogy a német

vezetés által naponta cserélt kulcsot néhány óra alatt megtalálják egy speciális gép segítségével, amely a „Turing-bomba” néven vált ismertté. Az alábbi kép egy katonai célú Enigma berendezést mutat.



Még 1918-ban szabadalmaztatták az Egyesült Államokban az ún. **Vernam-titkosítót**, amely „**one-time pad**” (eldobó kulcsú titkosítás) néven is ismert. Ez a módszer tömeges adatcserére nem alkalmas, de a diplomáciában a mai napig alkalmazzák, később részletesen tárgyaljuk.

Egyedi megoldások

A fent említett alapvető módszereken kívül még számtalan egyedi megoldást használtak a történelem során, több-kevesebb sikkerrel. Néhány példa:

- **a szótár-módszer:** minden félnél van egy hosszabb szöveg, például egy könyv (szótár) ugyanabban a kiadásban. Az üzenet minden betűjéhez keresnek a könyvben is egy olyan betűt, az üzenetbe aztán azt írják le, hogy hányadik oldalon, hányadik sorban, hányadik betűt kell venni. Ez a módszer természetesen nem alkalmas tömeges kommunikációra, viszont a szótár nélkül gyakorlatilag megfejtetlen, ezért gyakran alkalmazták a világháborúkban, sőt, még egy 1820 körül elrejtett, húszmillió dollár értékű kincs rejtekelyét is egy (feltehetőleg) ilyen módon titkosított irat őrzi, a modern kriptoanalitikusok nem kis bosszúságára.
- **a természetes nyelvek:** köztudomású, hogy egy ismeretlen és teljesen idegen nyelvnek még a szavait, hangjait sem tudjuk elkülöníteni egymástól. Ez adta az ötletet, hogy az amerikai hadseregekben a II. világháborúban navaho indiánokat alkalmazzanak titkosítási célokra a távolkeleti hadszíntéren. A navaho kódbeszélők egyszerűen elmondta a parancsot navaho nyelven, amit csak egy másik navaho kódbeszélő értett meg. Természetesen baj lett volna, ha az ellenség is szert tesz akár egyetlen navaho katonára, ez azonban nem történt meg, így mindmáig ez az egyetlen olyan, széles körben alkalmazott kriptorendszer, amit soha sem tudtak feltörni.

A kriptográfia története iránt mélyebben érdeklődő olvasóknak ajánljuk Simon Singh: Kódkönyv című, élvezetes könyvét [1].

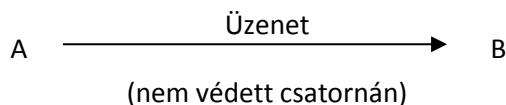
1. A modern kriptográfia alapjai

Alapfogalmak

A modern kriptográfiában használt, a korábbi gyakorlati eredményeket, módszereket rendszerező elmélet csak a XX. században született meg, és jórészt C.E. Shannon érdeme. A kriptográfia csak egy eleme az adatbiztonságnak, ami nagyon tág problémakör. Felöleli a védendő információt tartalmazó és közvetítő berendezések fizikai védelmét, a vállalat szervezeti felépítését és emberi erőforrásait, és az alkalmazott számítógépes információs rendszerek, nem utolsó sorban a kriptorendszerek biztonságát. Bármelyik területen is van hiányosság, az adatbiztonság veszélybe kerül. Ezen problémákörök közül itt a kriptorendszerek minőségi jellemzőivel, lehetséges megoldásaival foglalkozunk. A valóságból használt kriptorendszerek tervezésének alapelvei:

- a rendszer feltörésének a költsége legyen nagyobb, mint az információ aktuális maximális³ piaci értéke, vagy
- a feltöréshez a jelenlegi eszközökkel szükséges idő alatt a titok évüljön el (veszítse el az értékét).

Zárt rendszerek védelme általában megfelelő fizikai, szervezeti és humán tervezéssel megoldható. A kriptografiát érdeklő probléma az, hogy a külvilággal kommunikáló, *nyílt rendszert* hogyan lehet a rosszindulatú behatolás ellen megvédeni, illetve még inkább, hogy az ellenséges (nem védett) csatornán keresztül hogyan tud két fél, vagy két információs rendszer biztonságosan kommunikálni. Jó tervezés esetén a szervezeti, fizikai és informatikai rendszer-határ egybeesik. A kriptorendszerek alapsémája szerint az üzenetnek egy feladótól (A) kell eljutnia a címzetthez (B):



A kriptográfiai irodalomban szokás A-t Alice-nek, B-t Bobnak nevezni, a hallgatózó támadót pedig Eve-nek (az angol eavesdropper kifejezés nyomán). Az üzenetet a csatornán titkosított (rejtett) formában továbbítjuk. Az üzenetet eredeti formájában **nyílt szövegnek** nevezzük (jele X, plaintext), titkosítva **rejtett szövegnek** (jele Y, ciphertext). A rejtett szöveget a nyíltból a rejtés (encryption, E) révén kapjuk meg, melynek a nyílt szövegen kívül általában egy K **kulcs** is a paramétere. A vevőoldalon a fejtés (decryption, D) állítja vissza ugyanazon kulcs és a rejtett szöveg alapján a nyílt szöveget. Összegezve:

$$Y = E_K(X)$$
$$X = D_K(Y)$$

Feltehető, hogy a támadó fél ismeri az üzenet nyelvét és a titkosítás módját (E és D algoritmusát), de nem ismeri az éppen használt K kulcsot, ezért nem tudja kitalálni X-et. Az ilyen alapsémájú

³ „maximális” alatt azt értjük, amennyit az információ a számunkra legkellemetlenebb támadónak ér. Háborúban ez az ellenfél, az üzleti világban a konkurencia, más esetben egy napilap, vagy az adóhatóság.

kriptorendszeret **titkos kulcsú** rendszereknek nevezzük. A kriptorendszert támadó külső fél a következő alapvető módszereket használhatja:

- Hallgatózás, vagy „rejtett szöveg” típusú támadás. A támadó a csatorna forgalmának a figyelésével, az üzenet nyelvének, esetleg témajának az ismeretében próbálja megfejteni az aktuális üzenetet vagy kitalálni a K kulcsot.
- „Nyílt szöveg” típusú támadás. A támadó ismeri egy korábbi üzenet nyílt és rejtett formáját is, ebből próbálja kitalálni a kulcsot, melyet későbbi üzenetek megfejtésére használhat.
- „Választott nyílt szöveg” típusú támadás. C valahogyan rá tudta venni A-t, hogy egy általa választott nyílt szöveget rejtsen, és küldjön el B-nek. A nyílt és rejtett üzenet-párból könnyebb következtetni a kulcsra.
- „Választott rejtett szöveg” típusú támadás. C valahogyan rá tudta venni A-t (például ideiglenesen hozzáfért a fejtő berendezéshez), hogy egy általa konstruált rejtett szöveghez a nyílt szöveget állítsa elő K használatával, és ezt a nyílt szöveget is megszerezze. A rejtett szöveg ügyes konstrukcióval a nyílt és rejtett üzenet-párból könnyebb következtetni a kulcsra.

Ezek a támadások alapvetően a kulcs megszerzésére irányulnak. A támadó azonban már akkor is jelentős kárt tud okozni, ha manipulálni tudja az A-t B-vel összekötő hálózati elemeket:

- Közbeékelődés („**man in the middle**” támadás). A valójában nem B-vel, hanem C-vel van összeköttetésben., B valójában nem A-tól, hanem C-től kapja az üzenetet, de ezt A és B nem veszi észre. Egy hosszú üzenetváltás-sorozat során C hamis identitással minden üzenetet megfejt, és a megfelelő pillanatban leadja a számára előnyös üzenetet. Hasonló mesterkedés okozta Mária királynő vesztét 1586-ban.
- Átírás (**forging**). Az üzenet C-n keresztül halad, aki elfogja azt, és bár megfejteni nem tudja, egyes részeit megváltoztatja. Például korábban megfigyelt rejtett-nyílt üzenetpárok alapján egy korábbi rejtett üzenetből egy ismert értelmű darabot betesz egy rejtett üzenet megfelelő darabja helyére („**cut-and-paste**” támadás). Esetleg egy korábban megfigyelt ismert tartalmú teljes üzenetet küld el („**replay attack**”), ami C-nek előnyös reakciót vált ki B-ből.

A kriptográfia szempontjából az alaprobléma a rejtett szöveg típusú támadás. Lehet-e, és ha igen, hogyan, olyan „tökéletes” kriptorendszert készíteni, amelyben a kulcs ismerete nélkül *lehetetlen* a nyílt szöveg meghatározása a rejtett szöveg megfigyelésével?

A tökéletes titkosítás

Tökéletesnek nevezzük azt a titkosítási rendszert, amelyben a rejtett szöveg átlagosan semmi információt nem árul el a nyílt szöveggel kapcsolatban, vagyis amelyre

$$I(X, Y) = 0$$

Megmutatható, hogy ez akkor és csak akkor teljesül, ha

$$H(X | Y) = H(X) \text{ és } H(Y | X) = H(Y), \text{ azaz ha}$$

$$p_{ij} = p_i \text{ és } p_{ji} = p_j \quad \forall i, j - re$$

Itt $H(X|Y)$ a nyílt szövegnek a rejtett szövegre vonatkozó feltételes entrópiája, $p_{i|j}$ pedig az i -edik nyílt szöveg valószínűsége, feltéve, hogy a csatornán a j -edik rejtett szöveg lett elküldve. Eszerint egy adott rejtett szöveghez ugyanakkora eséllyel kell tartoznia bármelyik nyílt szövegnek (és fordítva). Mivel a kettőt a kulcs választása rendeli össze, és a lehetséges rejtett szövegek számának legalább akkorának kell lennie, mint a lehetséges nyílt szövegek számának, ezért a kölcsönös információ hiányára vonatkozó elvárásunkat úgy és csak úgy tudjuk teljesíteni, ha

- $|X|=|Y|=|K|$, tehát a lehetséges kulcsok, nyílt és rejtett szövegek száma egyenlő, és
- $P(K_i) = \frac{1}{|K|}$, tehát a kulcsok közül minden egyes üzenet rejtésekor véletlenszerűen választunk.

Ezek nehezen teljesíthető elvárásoknak tűnnék, azonban a már említett **one-time pad** (eldobó kulcsú titkosítás) esetén teljesülnek. E módszer szerint a nyílt szöveget bitenként egy kétbemenetű XOR kapura vezetjük, a másik bemenetre pedig egy, a nyílt szöveg hosszával egyező hosszságú véletlen bitsorozatot vezetünk (ez a kulcs). A rejtett szöveg a kapu kimenete. A vevőoldalon ugyanezt a műveletet hajtjuk végre a rejtett szöveggel és ugyanazzal a kulccsal, így visszakapjuk a nyílt szöveget. Látható, hogy a fenti feltételek teljesülnek a sorozatok egyenlő hosszúsága miatt.

Azt, hogy ekkor nincs a nyílt és rejtett szöveg között kölcsönös információ, a fenti tételektől függetlenül is könnyű belátni. Annak a valószínűsége ugyanis, hogy a rejtett szöveg egy bitje például „0”,

$$P(Y=0) = P(K=0)P(X=0) + P(K=1)P(X=1) = \frac{1}{2}(P(X=0) + P(X=1)) = \frac{1}{2},$$

függetlenül a nyílt szövegtől, tehát az X és Y források függetlenek.

Fontos viszont megjegyezni, hogy mihelyt egy már használt kulcsot még egyszer használunk egy másik szöveg rejtésére, tökéletes titkosításunk igen sebezhetővé válik a nyílt szöveg típusú támadással szemben, mivel a kulcs meghatározása egy rejtett-nyílt üzenetpárból triviális. Tehát minden egyes üzenethez egy, a rejtett szöveg méretével egyező méretű, új véletlen kulcsot kell generálni, és azt biztonságosan eljuttatni a vevőhöz a kommunikáció kezdete előtt⁴. Ez a gyakorlati probléma igen erősen korlátozza a módszer használhatóságát. Ezért csak különleges esetekben és elsősorban a diplomáciában alkalmazzák, mint például az amerikai és orosz elnököt összekötő telefonvonal, a „forró drót”.

Általában véve, a titkos kulcsú kriptorendszerek egyik nagy problémája a titkos kulcsok biztonságos eljuttatása az összes résztvevőhöz, amit **kulcskezelési problémának** (key management problem) neveznek. A gyakorlatban használt kriptorendszerek ezért nem tökéletesek, hanem a fejezet

⁴ Ha a kulcsokat újra felhasználják, akkor a one-time pad már rejtett szöveg típusú támadással is törhető statisztikai módszerekkel. A szovjet nagykövetségek is elkövették ezt a hibát a hidegháború éveiben, ennek következtében sok diplomáciai üzenetet meg tudott fejteni az amerikai titkosszolgálat. Az eredmény: számos szovjet ügynököt lepleztek le az USA-ban, a Rosenberg-házaszpárt pedig 1953-ban kivégezték az atomtitok szovjet kézre játszásáért.

elején ismertetett alapelvek szerint tervezik őket. A kriptografiában az elv alkalmazását a **számítási teljesítményre alapozott biztonság**nak (computational secrecy) nevezik.

A nyers erő módszere

A titkos kulcsú kriptorendszerek elleni legegyszerűbb támadás az összes lehetséges kulcs végigpróbálhatása, amit a **nyers erő** (brute force) módszerének is neveznek. Például egy 3 számjegyű bőrön-számzár esetén erre akár fél óra is elég lehet. A megfelelő kombinációt kinyilik a bőrön, illetve az üzenek nyelvének és témájának ismeretében észrevesszük, hogy egy adott K kulcs esetén a $D_K(Y)$ értelmes üzenetet eredményez. A nyers erő alkalmazása természetesen *nem praktikus*, ha a lehetséges kulcsok száma túl nagy, illetve a kulcsok számától függetlenül *értelmetlen*, ha a one-time pad módszert alkalmazták. Ekkor ugyanis az $|X|=|Y|=|K|$ feltétel miatt a támadó az összes lehetséges kulcs végigpróbálhatása során az összes lehetséges nyílt szöveget állítja elő, melyek mindegyike egyenlő valószínű, és természetesen több értelmes üzenet is van közöttük. Tehát ezek közül a támadó nem tud választani. Az alábbi példában a kulcs1 és a kulcs2 éppen ellentétes értelmű nyílt szövegeket produkál⁵:

rejtett:	PEFOGJ	PEFOGJ
kulcs1:	PLMOEZ	MAAKTG
nyílt1:	ATTACK	DEFEND (mindkettő értelmes!)

Ezért állíthatjuk azt, hogy a one-time pad a rendelkezésre álló számítástechnikai erőforrásoktól és időtől függetlenül is feltörhetetlen.

A nyelvi entrópia

Nem tökéletes kriptorendszer alkalmazása esetén, ha a nyílt és a rejtett szöveg, vagy kulcs és a rejtett szöveg egyes betűi között közvetlen kapcsolat van, mint például az S-doboz vagy az eltolás esetén, akkor betűgyakoriság-statisztikák alapján a támadó könnyen le tudja szűkíteni a valószínű kulcsok körét. Sőt, nem is kell az összes valószínű kulcsot végigpróbálni, hiszen például az egyszerű monoalfabetikus kód esetén a kulcs darabjait külön-külön is ki lehet találni. Ha például a levél első szavából már megvan annyi, hogy „ked*es”, akkor a hiányzó betű valószínűleg v, ez alapján pedig a kulcs negyedik betűje próbák nélkül is meghatározható. Ezekben a támadásokban a természetes nyelvek „beépített” redundanciáját lehet kihasználni.

A természetes nyelvek ugyanis nem tekinthetők emlékezet nélküli forrásnak, amennyiben egy forrásszimbólumnak egy betűt tekintünk, mivel egy-egy betű (és szó) előfordulási valószínűsége erősen függ a környezetétől, tehát a szöveg távolról sem véletlenszerű betűhalmaz. Ezt a jelenséget számszerűen a **nyelvi entrópiával**, vagyis a végtelen mértékben kiterjesztett forrás esetén az egy betűre jutó átlagos információmennyiséggel (entrópiával) lehet kifejezni:

$$H_L = \lim_{n \rightarrow \infty} \frac{H(A_1, A_2, \dots, A_n)}{n}$$

⁵ A példában a rejtett szöveg minden betűje a nyílt szöveg és a kulcs azonos pozíción lévő betűjének mod 26 összege, a 26 karakteres angol ABC használatával.

ahol $H(A_1, A_2, \dots, A_n)$ az n -szeresen kiterjesztett forrás, amelyben tehát egy betű- n -est tekintünk egy forrásszimbólumnak. A természetes nyelvekre a szókészlet korlátozottsága miatt minden $H(A_1, A_2, \dots, A_n) < n \cdot H(A)$. Például az angol nyelvre statisztikai vizsgálatokkal megállapított értékek:

n	$\frac{H(A_1, A_2, \dots, A_n)}{n}$
1	4 bit
2	3.56 bit
3	3.30 bit

A tényleges nyelvi entrópiát angolra 1.5 és 1 bit közé lehet tenni—az egy angol betű által ténylegesen hordozott átlagos információ tehát lényegesen kevesebb, mint a betűnkénti feldolgozás alapján várunk. Ennek az az oka, hogy a természetes nyelv zajos közegben való kommunikációra készült, és nyelvbe épült redundanciát folyamatosan használjuk a zaj által okozott hibák javítására. A nyelvi entrópia alapján definiálhatjuk a **nyelvi redundanciát**:

$$R_L = 1 - \frac{H_L}{\log |A|},$$

amely az angol nyelvre 0.75-re adódik, vagyis a szöveg körülbelül 75%-a redundáns. Ezt a redundanciát lehet a támadás során kihasználni olyan módon, hogy egyszerre *sok* lehetséges kulcsot zárunk ki egyetlen próba során, egy kulcsrészlettel megfejtett üzenet-részlet képtelensége alapján. Például szinte biztos, hogy az üzenet nem kezdődik három Q betűvel. Megmutatható, hogy a nyelvi redundancia miatt a lehetséges kulcsok száma 0-ra csökken, vagyis a titkosítás statisztikai értelemben fel van törve, ha elegendően hosszú rejtett szöveget van alkalmunk megfigyelni. A feltöréshez szükséges betűk száma:

$$n_0 = \frac{\log |K|}{R_L \log |A|},$$

amely az egyszerű módszerek esetén meglepően alacsony érték⁶. A nyelvi redundanciára alapozott támadásokat természetesen ki lehetne védeni forráskiterjesztéssel, például $n > 10$ esetén már elégé megközelíthető a nyelvi entrópia. A kiterjesztéssel azonban a szimbólumok száma (a forrás-ABC) és vele együtt a kulcs mérete is kezelhetetlenül nagyra nő. Hasonló okokból nem jó megoldás az sem, hogy a nyelv szavait, esetleg szócsoportjait tekintsük forrásszimbólumoknak.

2. Modern blokkos kriptorendszerek

Shannon javaslata nyomán olyan titkos kulcsú kriptorendszereket terveztek, melyek ugyan nem tökéletesek, mivel $|K| << |X|$, de a rejtett szöveg és a kulcs közti statisztikai kapcsolat elmosásának köszönhetően az $E_K(X)$ függvény véletlenszerű leképezésnek tekinthető a kulcs ismerete nélkül, és ha csak egyetlen bit is hibás a kipróbált kulcsban, a megfejtett szöveg tökéletesen értelmetlen lesz. Ezáltal a támadónak nem marad más választása, mint a nyers erő alkalmazása, ami idő- és energiaigényes művelet. Ha pedig a lehetséges kulcsok számát elegendően nagyra választjuk, akkor a kriptorendszer—

⁶ angol nyelvre az egyszerű monoalfabetikus betűhelyettesítéses kód esetén $n_0 = 25$.

bár nem tökéletes—gyakorlatilag nem támadható, biztonságos. Ez a számítási teljesítményre alapozott biztonság, a modern kriptográfia alapja. Kérdés azonban, hogy mekkora $|K|$ -t válasszunk? Ha a kulcstér túl nagy, lassú és drága lesz a titkosítás, ha túl kicsi, veszélybe kerül a biztonság. Természetesen mindenki lehet indulni a technika mai állásából, a jövőt azonban nehéz megjósolni.

A termodinamikai korlát

A modern kriptográfia talán legismertebb módszerét, a DES-t (Data Encryption Standard) 1977-ben vezették be⁷. A DES-re nem találtak érdemi algoritmikus törést, és úgy tűnt, hogy az 56 bites titkos kulcstér, mely 2^{56} lehetséges kulcsot jelent, örök időkre védelmet jelent majd a nyers erő típusú támadással szemben, legalábbis a polgári életben. A számítástechnika azonban a Moore-szabály⁸ szerint fejlődött, és 1998-ban egy masszív párhuzamos, 210 ezer dollárkból épített számítógépen⁹ maximum 186 óra alatt már az összes kulcsot végig lehetett próbálni. Ma pedig a DES töréséhez nincs szükség szuperszámítógépre. A DES tehát elavult. Ezért a kulcstér nagyságának a megítéléséhez a modern kriptográfia már nem a próbálhatás időszükségletét használja, hiszen ennek jövőbeli fejlődése ismeretlen, hanem az energiaszükségletből indul ki, mivel erre termodinamikai alapon alsó korlátot tudunk adni. Ha egy kulcs kipróbálásához akár egyetlen 0/1 átmenet elég is egy szuperszámítógépen, akkor is szükséges ehhez az átmenethez legalább egy energia-kvantum¹⁰. Ez alapján alsó korlátot lehet adni a nyers erő alkalmazásának az energiaszükségletére. Ha például a kulcs 192 bites (a lehetséges kulcsok száma 2^{192}), akkor a feltöréshez szükséges energia nagyobb, mint a Nap által egy év alatt kisugárzott összes energia, 256 bit esetén nagyobb, mint a Nap teljes élettartama alatt kisugárzott energia. Nem valószínű, hogy a támadónk ennyi energiával rendelkezik. A feltörési energia alsó korlátját **termodinamikai korlátnak** nevezzük.

Blokkos kriptorendszerek alapelvei

A feladat tehát az, hogy a támadót rákényszerítsük a kulcsok próbálhatására. A modern kriptorendszerök Shannon eredeti javaslata alapján ezt a következőképpen érik el:

1. A nyílt szöveget egyenlő méretű blokkokra osztják. A blokk mérete a titkos kulcs méretének a nagyságrendjébe esik.
2. A rejtést a blokkokra külön-külön végzik el, ugyanazt a titkos kulcsot használva¹¹. A vevőoldalon is blokkonként fejtik meg és rakják össze az üzenetet. Ezt a működési elvet **blokkos kriptorendszernek** nevezzük. Általában a rejtési és a fejtési algoritmus ugyanazt a kulcsot

⁷ A kriptográfia a katonai és diplomáciai alkalmazások miatt kevésbé nyilvános tudomány, ennek ellenére léteznek széles körben alkalmazott, szabványos algoritmusai és protokolljai. Ezek jó része, köztük a DES, AES, RSA, SHA stb. az amerikai National Institute of Standards and Technology (NIST) szabványa, azonban egyre erősebben jelentkeznek az EU ajánlásai is, lásd www.ecrypt.eu.org

⁸ http://en.wikipedia.org/wiki/Moore%27s_law

⁹ AWT Deep Crack, egy chipen 24 mag, 64x28 chip (43008 mag) 40 MHz-en, egy kulcsot 16 óraciklus (0.4 µs) alatt vizsgált meg, ezért maximum 186 óra (7.7 nap) alatt találta meg a kulcsot. A kulcs-teszt egyszerűen az első 3 megfejtett byte-ot ellenőrizte: ha minden a 3 alfanumerikus volt, megtaláltuk a kulcsot.

¹⁰ legalábbis akkor, ha a ma ismert és működő digitális számítási architektúrákat vesszük figyelembe. A kvantumszámítógéppel ez természetesen változhat.

¹¹ mivel ugyanazt a titkos kulcsot több nyíltszöveg-blokk rejtés során is felhasználják, ezért a titkosítás természetesen nem tökéletes.

használja, és a rejtési és fejtési algoritmus is lényegében ugyanaz (tehát a kulcs ismételt alkalmazása a fejtés során mintegy „megszünteti” a kulcs hatását), ezért a kriptorendszert **szimmetrikusnak**¹² nevezük. Ennek előnye, hogy a szükséges hardver illetve kódmezőt fele a nem szimmetrikus megoldásénak.

3. A rejtési/fejtési műveletet több *iterációban* végzik el. Egy iterációs lépés bemenete egy szövegblokk és egy iterációs kulcs, kimenete egy szövegblokk. Az egyes iterációkhoz használt kulcsokat az eredeti titkos kulcsból állítják elő különféle keverési műveletekkel. A legelső iteráció szöveg-bemenete a nyíltszöveg-blokk, a további iterációk bemenete az előző iteráció kimenete. A legutolsó iteráció kimenete pedig a rejtetszöveg-blokk. A közbülső szövegblokkokat gyakran állapotnak (state) nevezik.
4. Egy iteráció tartalma eltolás és/vagy keverés (P-doboz) és nemlineáris függvény alkalmazása (ami általában helyettesítés, S-doboz) a bemeneti szövegblokkra az iterációs kulcsról függetlenül, majd az iterációs kulccsal való rejtés (általában XOR művelet). Az ilyen sémájú kriptorendszert gyakran **produkciós** rendszernek (product cipher) is nevezik.

A fenti elvű kriptorendszereket a DES alapjául szolgáló IBM Lucifer tervezője, Horst Feistel német származású amerikai kriptográfus után **Feistel-titkosítónak**, vagy Feistel-hálózatnak is nevezik.

A Feistel-titkosítóra jellemző, és a blokkos kriptorendszerektől elvárt az ún. **Iavinahatás**; tehát hogy a bemeneti blokk vagy a kulcs-blokk egy bitjének megváltoztatása 50% valószínűséggel változtasson meg minden bitet a kimeneti blokkban.

Bár jelenleg (2012-ben) még sok helyen alkalmazzák a DES-t, illetve ennek 2 vagy 3 kulccsal működő, 112 bitesre növelt kulcsú változatát, a TDES-t, a titkos kulcsú kriptorendszerek területén az AES nevű, 2000-ben bevezetett rendszeré a jövő, mivel flexibilisebb és hatékonyabb szoftver/hardver megvalósítást tesz lehetővé. A hatékonyságára jellemző, hogy x86-ra 457 byte-on is leprogramozták, és 700 Mbit/s feldolgozási sebességet értek el 2 GHz-en. Az AES működését és egyéb részleteit lásd az I. Mellékletben.

Minden alap-üzemmódban dolgozó blokkos titkosító támadható a rejtetszöveg-blokkok cseréjén alapuló kivágás-beillesztéses (cut-and-paste) támadással. Ennek kivédésére az egyes blokkok rejtéséhez felhasználják a korábbi rejtetszöveg-blokkokat is a titkosító láncolt üzemmódjában. A szabványos ECB, CBC, OFB, CTR üzemmódok leírását lásd a II. Mellékletben.

A véletlen kulcs

A titkos kulcsú kriptorendszerek alapkérdése a jó minőségű, tehát valóban véletlenszerű titkos kulcs készítése. Hiába elegendően nagy a kulcsréteg és hiába véletlenszerű a támadó szemszögéből az $E_k(X)$ leképezés, ha a kulcsrétegen egyes kulcsok valószínűbbek másoknál, vagy egyenesen kizáráthatók a támadó számára. A titkos kulcs tehát egy véletlen bitsorozat. „Igazi” véletlen sorozathoz azonban csak valamelyen természeti folyamat mérése révén lehet hozzájutni, mint például a radioaktív sugárzás, az ellenálláson keletkező termikus zaj, a kozmikus háttérsugárzás, a gerjedő oszcillátor, a turbulens áramlás

¹² a szakirodalomban gyakran szimmetrikusnak neveznek minden titkos kulcsú kriptorendszert, és aszimmetrikusnak minden nyilvános kulcsú rendszert.

stb. Ha gyakran és nagy mennyiségen van szükség új kulcsokra, akkor az ilyen mérések költsége túl nagy lehet (lásd a szovjet titkosszolgálat esetét a one-time paddel). Gyakran felhasználják a felhasználói interakciót (például a leütések közti szüneteket vagy az egérmozgást) és a különféle számítástechnikai rendszerparaméterek (például a nyitott fájlok száma stb.) kombinációját is, bár ezek már kevésbé véletlenszerűek. A leggyakrabban alkalmazott megoldás azonban a különféle véletlenszám-generátorok alkalmazása. A generált sorozat minőségének megítéléséhez lényeges, hogy a sorozatról el tudjuk dönten, „mennyire véletlen”. Erre a kriptografiában gyakran alkalmazzák a Samuel W. **Golomb**-tól származó következő három **kritériumot**:

1. A sorozatban az egyesek és nullák száma legyen közel egyenlő.
2. A homogén szakaszok relatív gyakorisága exponenciálisan csökkenjen a sorozat hosszával. Homogén szakasznak a csupa azonos szimbólumból álló sorozatot nevezzük, például a '011110' egy 4 hosszú 1-es sorozat, az '10001' pedig egy 3 hosszú 0-ás sorozat. A kritérium azt írja elő, hogy az n hosszú homogén szakaszok száma az egész álvéletlen bitsorozatban körülbelül a fele legyen az $n-1$ hosszú szakaszok számának.
3. A sorozat autokorreláció-függvénye minden pontban közel legyen a nullához. Az autokorreláció-függvényt az x pontban úgy képezzük, hogy a sorozatot x pozícióval ciklikusan eltoljuk, majd az eredeti és eltolt sorozat bitenkénti XOR függvényét képezzük. A XOR-olt sorozat egy adott pozícióján pontosan akkor lesz 1, ha az ezen a pozícióban lévő bitek különböznek az eredeti és az eltolt sorozatban. Az autokorreláció-függvény x -beli értéke a XOR-olt sorozatban lévő 1-esek és 0-k számának a különbsége, osztva a sorozat hosszával. Ez a kritérium azt biztosítja, hogy az álvéletlen sorozat egy részletéből ne lehessen következtetni a hiányzó részekre.

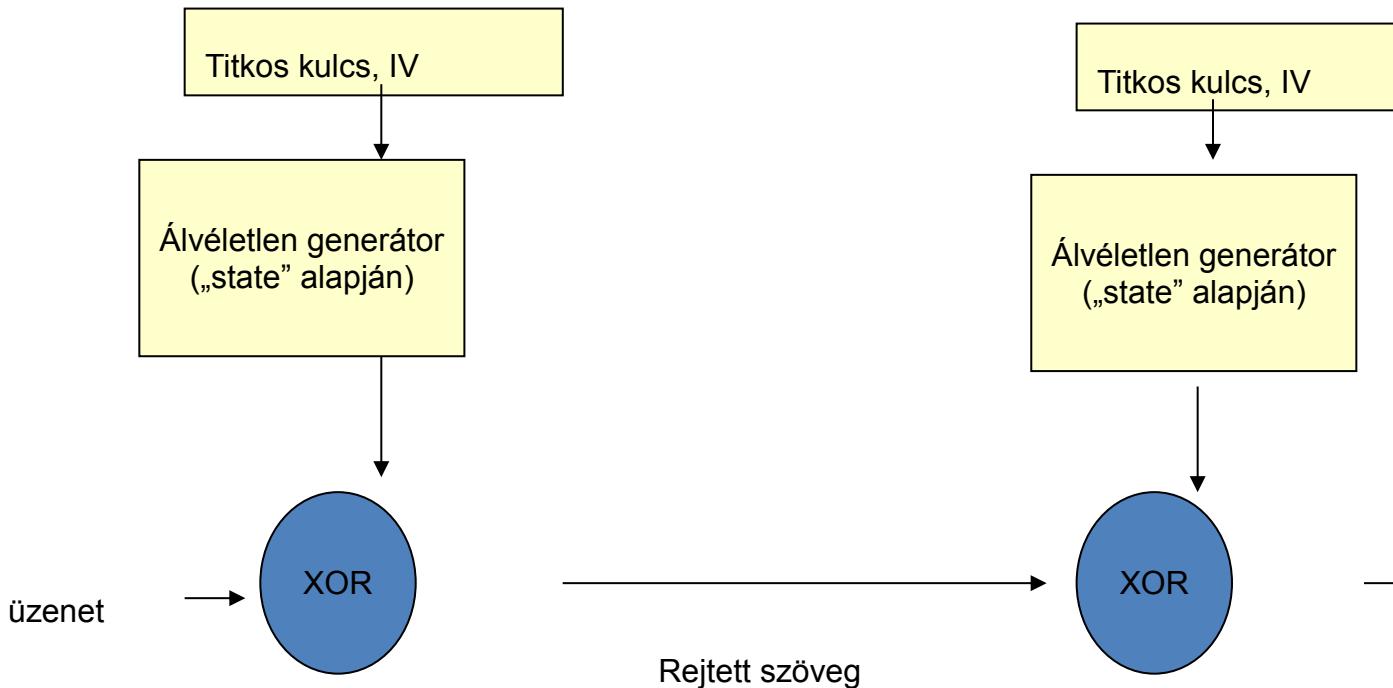
Ha sikerült egy titkos kulcsot minden résztvevőhöz eljuttatni, akkor azt csak korlátozott ideig lehet biztonságosan használni. Szoftver és hardver hibák, vírusok és egyéb támadások vagy emberi korrupció mindenhol előfordul, ezért a titkos kulcsot le kell cserélni. A gyors kulcs-cseréltetés nagyobb biztonságot, és ugyanakkor lassabb működést eredményez. Az 'igazi' titkos kulcsot nem lehet óránként fizikailag védett csatornán (páncélautóval) eljuttatni a felekhez, viszont egy titkos kulcsból sok kulcsot tud készíteni minden résztvevő:

- a titkos kulcsot egy álvéletlen generátor magjaként (seed) alkalmazva a generátor kimenete kulcsok gyakorlatilag végtelen sorát adja. Álvéletlen generátoroknál használható minden folyam- és blokktitkosító is, utóbbiak az OFB üzemmódban (lásd a II. Mellékletben).
- a titkos kulcsot lehet úgy is alkalmazni, hogy nem a valódi kommunikációra használják, hanem csak arra, hogy az egyik fél által generált rövid életű, véletlen kulcsot (session key) átvigyék. Az ilyen módon használt titkos kulcsot kulcsrejtő kulcsnak (**key encrypting key**, KEK) nevezik.

3. Modern folyam-elvű kriptorendszerek

A folyam-titkosítók előnye a blokkos titkosítókkal szemben a nagyobb működési sebesség, alacsonyabb ár és félvezető-felület, ugyanakkor a más működési elv miatt másmilyen jellegű támadásokra érzékenyek. A folyam-titkosító lényegében egy álvéletlen generátor, melyet a titkos kulccsal, mint

maggal egyszer inicializálnak¹³, utána pedig a kimeneti álvéletlen bitfolyamot úgy használják fel, mint egy gyakorlatilag végétlen hosszú one-time pad kulcsot, tehát az adónál bitenként vagy byte-onként XOR-olják a nyílt szöveggel, a vevőnél pedig a kapott rejtett szöveggel. A „blokkméret” tehát itt 1 bit (vagy 1 byte), és a véletlenszám-generátornak van egy belső állapota (state) is, amely a következő kimeneti bitet (byte-ot) és a következő állapotot is meghatározza.



A folyamtitkosítókkal szemben támasztott 3 fő követelmény:

1. Nyílt szöveg típusú támadással az álvéletlen bitfolyamot ne lehessen visszafejteni a state-re (vagy csak a kimerítő keresésnél több művelettel). Ha a state ismert, akkor a támadó is tudja generálni az álvéletlen bitfolyamot.
2. A state ismeretében a titkos kulcsot ne lehessen visszafejteni
3. A kimenetet ne lehessen megkülönböztetni egy valódi véletlen sorozattól (a Golomb-kritériumok szerint)

Az 1. követelmény kielégítése és a state méretezése szempontjából fontos az ún. Babbage-Golic támadás.

A Babbage-Golic támadás

A Babbage-Golic támadás a Hellman-féle általános idő-tárhely ekvivalencia támadás¹⁴ továbbfejlesztett változata. A támadás alapfeltevése az, hogy egy adott state után következő kimeneti bitfolyam a state

¹³ az inicializáláshoz a titkos kulcson kívül általában felhasználnak egy úgynevezett inicializációs vektort (IV) is. Ez a támadó számára is ismert lehet, gyakran egy véletlen érték, vagy például a rendszerparaméterek, rendszeridőfüggvénye. A célja az, hogy ugyanazzal a titkos kulccsal inicializálva ne kapjuk kétszer ugyanazt a state-et.

¹⁴ Ezt az elvet alkalmazzák az üzenetpecsétek elleni szivárványtáblás támadás során is, lásd később.

nem invertálható véletlenszerű függvénye. Legyen N a lehetséges state-ek száma. A támadás lépései (dőlt betűvel a paraméterek):

1. a folyam-titkosító módszer ismeretében *előfeldolgozás*: egy táblázatba feljegyzik M darab véletlenül választott state-ból indított kimeneti bitfolyam első b bitjét ($b = \log N$).
2. nyílt szöveg típusú támadás, vagyis a rejtett és a nyílt szövegből meghatározzák az álvéletlen bitfolyam egy darabját, ez alapján próbálnak következtetni a state-re. Legyen a megfigyelt bitfolyam hossza $D+b-1$, ekkor ugyanis ez tartalmaz D darab b hosszúságú rész-sorozatot, melyek mindegyike egy másik state-ból indult el. A törés sikeres, ha ezen D state közül valamelyiket ki tudjuk találni. Ezért minden rész-sorozatot megnézünk az előfeldolgozás során készített táblázatban.

Kérdés, hogy M és D milyen értékei mellett találunk meg legalább 0.5 valószínűséggel egy sorozatot a táblázatban? A születésnap-paradoxon szerint egy N elemű halmazból kivett A és B elemű részhalmazoknak várhatóan lesz közös eleme, ha $AB \geq N$. N a lehetséges state-ek száma, tehát például 128 bites state esetén $N = 2^{128}$, a részhalmazok pedig legyenek a state-ekhez tartozó b hosszúságú bitsorozatok. Várhatóan sikeres a támadás, ha $DM \geq N$. Jelölje T a próbálkozások számát, tehát a nyílt szöveg típusú támadás időszükségletét, és tegyük fel, hogy ugyanakkora energiát fordítunk az 1. és a 2. lépésre is, vagyis $T = M$. Mivel a próbálkozások száma megegyezik D -vel, ezért végül is azt kapjuk, hogy $T^2 \geq N$. Tehát a fenti előfeldolgozás esetén például a 128 bites state-tel rendelkező folyamtitkosító 2^{64} lépésben törhető. Ha ugyanennek a folyamtitkosítónak az inicializálás során felhasznált titkos kulcsa is 128 bites lenne, akkor a nyílt szöveg típusú támadás esetén a titkos kulcs teljes keresésénél (ami 2^{128} lépés) lényegesen egyszerűbb a state támadása, tehát a state mérete a titkos kulchhoz képest túl kicsi. Konklúzióképpen elmondható, hogy a folyamtitkosítók esetén a state méretét legalább a titkos kulcs méretének kétszeresére célszerű választani.

Néhány elterjedt folyam-elvű kriptorendszer

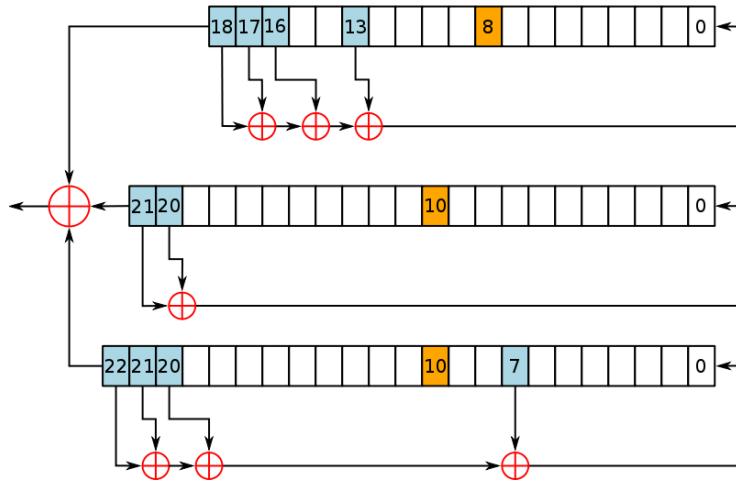
Az alábbiakban áttekintjük néhány széles körben használt folyamtitkosító jellemzőit.

- Az **RC4** (1987) 1 byte-os blokkokkal dolgozik, tízszer gyorsabb a DES-nél, az SSL egyik alap-algoritmusá. A titkos kulcs a 0..255 sorozat egy permutációja, tehát $|K| = 256!$, ami egy 1680 bites titkos kulcsnak felel meg, a periódus nagyon hosszú. minden adatbyte-nál változik a 0..255 számokat tartalmazó kulcs-tömb elemeinek a sorrendje, majd kiválasztják belőle az aktuális kulcsbyte-ot. A kulcsbyte-ot XOR-olják a nyíltszöveg byte-tal. Nem találtak érdemi törést hozzá. Nem publikus, de az RC4-gyel gyakorlatilag ekvivalens algoritmus hozzáférhető.
- A GSM rendszerek a beszélgetések titkosítására visszacsatolt léptetőregiszterekre (LFSR) alapozott folyamtitkosítókat alkalmaztak **A5/1**, később a túl könnyű (gyakorlatilag real time egy PC-n) törhetőség miatt továbbfejlesztve **A5/2** néven¹⁵. A 128 bites titkos kulcs a SIM kártyán van tárolva, az ebből challenge-response alapon generált 64 bites viszonykulcsot az LFSR-ek inicializálására használják¹⁶. Alább az A5/1 blokkvázlata a 19, 22 és 23 bites léptető

¹⁵ Ettől függetlenül még ma (2012-ben) is nagyon sok telefon alkalmazza a rendkívül gyenge A5/1-et. Az erősebb algoritmusok bevezetését elsősorban politikai okok miatt gátolják, elsősorban az USA-ban.

¹⁶ Ez a fajta megoldás a “key encrypting key” (KEK) koncepciója.

regiszterekkel. Ezekbe töltik indításkor a 64 bites titkos kulcsot. A kimeneti álvéletlen bitfolyam a 3 regiszterből kilépő 3 bit XOR-függvénye. A regiszterek középső bitjeinek (sárgán árnyékoltva) az órajelek kapuzásában van szerepe.



- A jövő igéretes folyamtitkosító algoritmusai az EU ECRYPT által ajánlott Rabbit és Grain. Rendkívül alacsony hardver költség és energiafelhasználás, magas működési sebesség, könnyű szoftver implementálhatóság és paraméterevezhetőség, ugyanakkor megfelelő biztonság jellemzi őket. Ezek implementációs részleteit lásd az V. Mellékletben.

4. Nyilvános kulcsú kriptorendszerök és alkalmazásai

Az internet térhódításával szükségessé vált olyan módszerek kifejlesztése, melyek nem igénylik titkos kulcsok védett leosztását, mivel a kommunikációban résztvevők köre előre nem ismert (például internetes boltok, szolgáltatások), vagy más ok miatt egyetlen egyszer sem lehet védett csatornát létesíteni köztük. A nyilvános kulcsú rendszerek olyan módon kerülik meg a kulcskezelési problémát, hogy a kulcs egy részét olyan formában (úgynevezett **egyirányú művelet**, trap function alkalmazásával) hozzák nyilvánosságra, hogy abból a kulcsra a jelenleg ismert módszerekkel csak nagyon nagy számítási teljesítmény felhasználásával lehet visszakövetkezteni. Megmutatható, hogy bármilyen nyilvános kulcsú kriptorendszerhez szükség van ilyen egyirányú művelet alkalmazására. Többféle egyirányú műveletet használnak, melyek hatásukban minden olyanok, mint Hamupipőke próbatétele az összekevert hamuval és liszttel: összekeverni nagyon könnyű, szétválasztani szinte lehetetlen. Néhány nevezetes egyirányú művelet:

- A diszkrét logaritmus (DLP). Ha a , x , és p ismert, akkor $y = a^x \text{ mod } p$ nagyon gyorsan számítható, de a , y , és p ismeretében x számítása nagy számok esetén nagyon komplex, idő- és energiaigényes feladat. A Diffie-Hellman kulcsmegosztási algoritmus és az Elgamal algoritmus alapja.
- Nagy prímszámok szorzatának faktorizálása (PFP). Ha p és q nagy prímszámok, akkor $n = pq$ számítása könnyen elvégezhető, de n -ből p és q meghatározása nagyon komplex, idő- és energiaigényes feladat. Az RSA algoritmus alapja.

- Véges testek felett definiált elliptikus görbék pontjainak szorzása természetes számmal (ECDLP), bővebben lásd A kriptográfia jövője alfejezetben.

Fontos megjegyezni, hogy az egyirányú műveletek inverzáneka számítása a matematika aktív kutatási területe, és elvileg bármikor születhet olyan módszer, amely gyors invertálást tesz lehetővé, miáltal az erre épülő kriptográfiai algoritmusok értelmetlennek vagy legalábbis sebezhetővé válnak. A titkos kulcsú kriptorendszer esetén sokkal kevésbé fenyeget ilyen veszély. A nyilvános kulcsú kriptorendszer másik hátránya a relatív lassúságuk és nagy erőforrás-igényük. Ezen két probléma miatt a nyilvános kulcsú rendszereket általában nem önmagukban, hanem titkos kulcsú rendszerekkel és üzenetpecsét-algoritmusokkal kombinálva, különféle protokollokba ágyazottan alkalmazzák. Az ilyen kriptorendszert **hibrid kriptorendszernek** nevezik. A hibrid rendszerben a nyilvános kulcsú kriptorendszer tipikus feladata az autentikáció és a véletlen viszonykulcs megosztása a résztvevők között. A sikeres kulcsmegosztás után a felek a kommunikáció érdemi részét már egy titkos kulcsú kriptorendszer használatával folytatják.

Hibrid kriptorendszer a HTTPS mögött álló SSL (Secure Socket Layer) protokoll, melyben a felek a session elején állapodnak meg abban, hogy melyik nyilvános és titkos kulcsú kriptorendszeret, milyen paraméterekkel fogják használni a továbbiakban. Szintén TDES-RSA alapú hibrid rendszer a mobilbankszolgáltatások alapja.

A Diffie-Hellman kulcsmegosztási protokoll

Az első nyilvános csatornán működő nyilvános kulcsú kulcsmegosztási protokoll, a Diffie-Hellman protokoll 1976-ból származik. A két kommunikáló fél célja a későbbi kommunikáció során használandó titkos viszonykulcs biztonságos megosztása. A protokoll lépései:

1. Alice és Bob nyilvánosan megállapodnak egy nagy (több ezer bites) p prímszámban és egy kis g számban, amely a p alapú véges test elemeiből álló ciklikus csoport generátora, tehát

$$\forall x \in GF[p] - re \exists k: g^k = x \text{ mod } p$$

2. Alice választ egy véletlen $a < p$ részkulcsot, és elküldi Bobnak a $k_a = g^a \text{ mod } p$ számot. Hasonlóképpen Bob is választ egy véletlen $b < p$ részkulcsot, és elküldi Alice-nak a $k_b = g^b \text{ mod } p$ számot a nyilvános csatornán. A támadónak az a és b meghatározásához meg kell oldania a diszkrét logaritmus problémát.
3. Mindkét fél kiszámítja a $K = k_b^a \text{ mod } p = k_a^b \text{ mod } p = g^{ab} \text{ mod } p$ számot. Ezt a továbbiakban egy titkos kulcsú rendszer kulcsaként használják.

A Diffie-Helmann protokoll kis módosítással alkalmas üzenet-titkosításra is. Ez a megoldás a feltalálója után **EIGamal kriptorendszer**¹⁷ néven vált ismertté (1984).

Az RSA kriptorendszer

Az RSA a legelterjedtebben használt nyilvános kulcsú, **aszimmetrikus** kriptorendszer. Az aszimmetrikus jelleg azt jelenti, hogy Alice és Bob tevékenysége és az általuk használt kulcs is különbözik egy üzenet

¹⁷ Az EIGamal pedig a DSA amerikai digitális aláírás-szabvány alapja.

rejtésekor, illetve fejtésekor. A támadó számára rendelkezésre álló kulcs-részből, az ún. **nyilvános kulcsból** a rejttet szöveg megfejtéséhez szükséges kulcs-rész, az ún. **privát kulcs** csak a PFP probléma megoldásával lehetséges. A nyilvános kulcs birtokában üzenetet rejteni bárki tud, fejteni azonban csak a publikus kulcshoz tartozó privát kulccsal lehet, melyet a címzett soha nem tesz közzé. Az RSA részletes leírása megtalálható a IV. Mellékletben, egy demo alkalmazás pedig a <http://vassanyi.ginf.hu/info/rsa/> lapon.

Az RSA már 1977 óta használatban van, ezért számos támadást dolgoztak ki ellene. Biztonságosan alkalmazni csak az ajánlások szigorú betartásával, és kellőképpen nagy kulcsmérettel lehet. Az ismert támadások:

- alapvető aritmetikai alkalmazási hiba kihasználása (n vagy x túl kicsi)
- egyéb rosszul választott paraméterekből adódó algoritmikus gyengeségek kihasználása
- választott rejttet szöveg típusú támadás (1998): megfigyelt Y alapján konstruált Y' és az ehhez tartozó X' alapján X . A címzettet (illetve annak számítógépéti) egy RSA-t használó protokoll hibáját kihasználva vették rá arra, hogy a konstruált Y' -t dekódolja és az eredményt visszaküldje. A kriptografiában az ilyen rejttet, előre tervezett, kikényszerített műveletet **oracle service**-nek hívják.
- időzítéses támadás: a fejtési művelet időszükséglete összefügg a privát kulccsal, és a fejtés időszükségletét egy protokoll gyengeségét kihasználva meg lehetett határozni. Ezáltal a támadó (az üzenet küldője) jelentősen le tudta szűkíteni a lehetséges privát kulcsok körét.

Beékelődéses támadás nyilvános kulcsú kriptorendszerek ellen

Nemcsak az RSA, hanem minden nyilvános kulcsú kriptorendszer alapproblémája, hogy éppen mivel a felek sohasem találkoznak, ezért nehéz megbizonyosodniuk egymás kilétérről. A beékelődéses (**man in the middle**) támadás jellemző lépései az RSA esetén, ha B kíván titkos üzenetet küldeni A-nak, és a támadónak (E-nek) módjában áll manipulálni az üzeneteket:

1. B elkéri A-tól A nyilvános kulcsát
2. E elfogja ezt a kérést, elkéri A nyilvános kulcsát, de nem ezt, hanem a saját nyilvános kulcsát küldi vissza B-nek
3. Ezután E minden B által írt üzenetet megfejt a saját titkos kulcsával, majd újra elrejti A nyilvános kulcsával és elküldi A-nak.

E-nek azon túl, hogy minden üzenetet el tud olvasni, módja van az üzenetek átírására, illetve B nevében írt hamis üzenet konstruálására is. A és B mindebből nem vesz észre semmit, sőt éppen a sikeresen megfejtett üzenetek győzik meg őket arról, hogy a kommunikáció titokban folyt le. Ez a támadás csak akkor védhető ki, ha A és B valamilyen külső segítséggel meg tud győzdeni egymás kilétérről, úgy ahogy a személyi igazolvány is igazolja az ismeretlen tulajdonos bizonyos adatait. Ilyen igazolást a kriptografiában a külső hatóság vagy cég által digitálisan aláírt digitális tanúsítványok (certificate, lásd alább) tudnak nyújtani.

A digitális aláírás az RSA segítségével

Az RSA, és más nyilvános kulcsú kriptorendszerek is, használhatók az üzenet-titkosításos algoritmus megfordításával is. Ekkor A, a privát kulcs birtokosa a privát kulcs használatával először rejt az üzenetet, melyet aztán közzétesz, és amelyet a nyilvános kulcs használatával bárki meg tud fejteni. Ennek gyakorlati értelme akkor van, ha nem az üzenet titkosságát, hanem annak **letagadhatatlanságát** és **sérthetetlenségét** (Non-repudiation and integrity) szeretnénk a nyilvános kulcsú rendszerrel biztosítani:

- ha a rejtett üzenetet valaki megváltoztatja, utána már nem lesz A nyilvános kulcsával (értelmesen) megfejthető, tehát ha a fejtés sikeres, az üzenet nem sérült
- mivel a nyilvános és a privát kulcs egyedi párt alkot, ezért ha a fejtés sikeres, akkor azt biztosan A rejtette el vagy legalábbis az Ő privát kulcsát használták a rejtéshez

Hatékonysági megfontolásokból nem az egész üzenetet, hanem csak egy abból készített kis méretű ún. **üzenetpecsétet** (hash) rejtenek el egy dokumentum digitális aláírása során. A hash algoritmus biztosítja, hogy ha a dokumentum változik, akkor a belőle készített hash érték (a pecsét) is változzon. A digitális aláírás gyakorlati alkalmazásának lépései:

1. A elkészíti (vagy másuktól elfogadja) az aláírandó dokumentumot, választ egy hash algoritmust, és ezzel kiszámítja a dokumentum hash értékét
2. a hash értéket és az algoritmus nevét, paramétereit a privát kulcsával rejt egy állományba, amit **digitális aláírásnak** hívnak
3. a dokumentumot és az aláírást publikálja a saját nyilvános kulcsával együtt
4. ha valaki meg akar győződni a dokumentum hiteléről, akkor a nyilvános kulccsal először is megfejti a dokumentumhoz tartozó aláírást, majd
5. az aláírásban található algoritmussal és paraméterekkel kiszámítja a dokumentum hash értékét
6. ha a kiszámított érték egyezik az aláírásban lévő hash értékkal, akkor a dokumentum nem sérült (eredeti) és azt A privát kulcsával írták alá.

Magyarországon 2001. óta a digitális aláírás egyenértékű a papír-alapú aláírással (2001. évi XXXV. törvény).

Digitális tanúsítványok és tanúsítási rendszerek

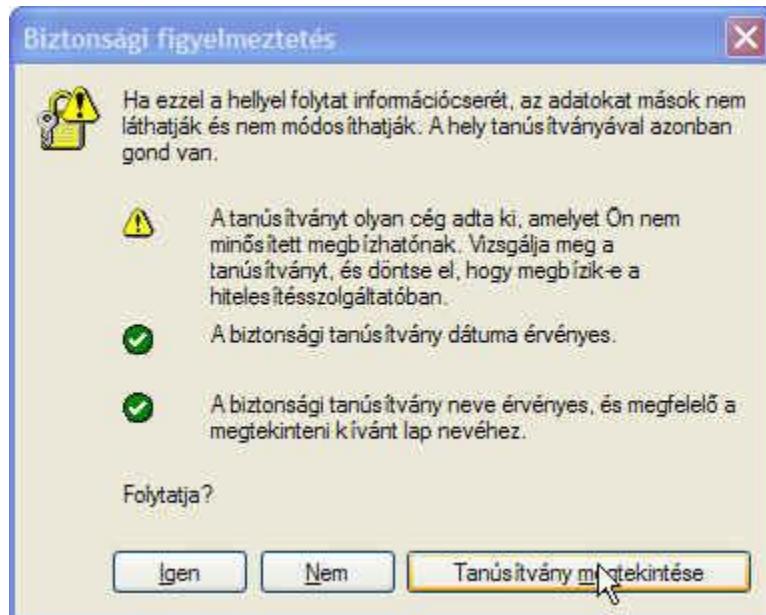
A digitális aláírás technológiáját fel lehet használni a beékelődéssel támadás kivédésére. Ehhez egy olyan dokumentumot az ún. **digitális tanúsítványt** szerkesztenek, mely A személyazonosságát és A nyilvános kulcsát együtt tartalmazza, majd ezt egy olyan szervezet, az ún. **certificate authority** (CA) látja el digitális aláírással a saját privát kulcsa használatával. A CA nyilvános kulcsa mindenki számára elérhető kell, hogy legyen. Ezután, ha bárki meg akar győződni arról, hogy A-nak tényleg az-e a nyilvános kulcsa, ami a tanúsítványon áll, akkor a CA nyilvános kulcsával megfejti az aláírást, és ellenőrzi a dokumentum sértetlenségét. A digitális tanúsítvány kötelező részei az X509 szabvány szerint:

- a tanúsítvány sorszáma és verziója
- a kibocsátó CA neve és egyéb adatai
- a tanúsítvány időbeli érvényessége (a lejárt tanúsítvány érvénytelen)
- a tulajdonos neve és egyéb adatai

- a tulajdonos nyilvános kulcs algoritmusa és a nyilvános kulcs értéke
- az aláíráshoz használt algoritmus
- a fentiekhez, mint dokumentumhoz a CA által készített digitális aláírás

A tanúsítvány használatával elkerülhető a beékelődéses támadás, mert a támadó hiába írja át a tanúsítvány dokumentum részében a nyilvános kulcsot, a CA privát kulcsa nélkül az aláírást nem tudja átírni, ezért B észreveszi a módosítást.

Természetesen felmerül a kérdés, hogy egy interneten kapott tanúsítvány ellenőrzésekor hogyan szerezzük be a CA nyilvános kulcsát. A beékelődéses támadás elkerülésére ezt a CA saját tanúsítványából kell kiolvasnunk, melyet egy CA feletti hatóság vagy szervezet írt alá, és így tovább. Az egymásra hivatkozó tanúsítványok sorozatát **hierarchikus tanúsítási láncnak** (chain of trust) nevezzük. A lánc végén egy olyan szervezetnek kell állnia, melynek nyilvános kulcsa minden résztvevő (web-böngésző program, operációs rendszer) számára eleve ismert, beégetett. Az ilyen nyilvános kulcs neve **bizalmi horgony** (trust anchor). A tanúsítványokkal biztosított kommunikáció, például egy https protokollal elérhető weblap böngészése előtt a magát igazolni kívánó fél (jelen esetben a webszerver) a teljes, horgonyig vezető tanúsítási lánc összes tanúsítványát elküldheti a másik félnek. Ha a láncot nem sikerül egy horgonyig követni, akkor a felhasználónak kell megítélnie, hogy elfogadja-e a tanúsítványt, tehát belép-e a weblapra, vagy telepíti-e a bizonytalan eredetű szoftvert. Alább egy böngésző által adott ilyen témájú figyelmeztetés.



A tanúsítványokkal is vissza lehet élni. Ahogy a személyi igazolványt, vagy a bankkártyát, a privát kulcsot is el lehet lopni, illetve a hatóság is bevonhatja például az eljárás alá vont cég tanúsítványát. Ezért a bevont vagy a tulajdonos kérésére érvénytelenített tanúsítványokról erre szakosodott szolgáltatók

bevonási listákat (**certificate revocation list**, CRL) vezetnek, melyeket a tanúsítvány végleges elfogadása előtt célszerű ellenőrizni¹⁸.

A fenti tanúsítási rendszerhez szükség van egy CA szervezetre, és különösen a magasabb biztonsági fokozatú tanúsítványok esetén egyszeri és éves költségek merülnek fel. Ez nem okoz problémát az üzleti és hivatalos világban, azonban a civil világnak nincs erre szüksége. A hierarchikus tanúsítási rendszer helyett elterjedt a hálózatos tanúsítás vagy bizalmi háló (**web of trust**), amely a személyes ismeretségre és bizalomra épül. Ilyen rendszerben működik a PGP (Pretty Good Privacy). Ki-ki elkészíti a saját privát-nyilvános kulcspárját, aztán az ismerőseivel aláírják egymás tanúsítványait. Egy tanúsítványt többen is aláírhatnak, és minél többen írják alá, annál nagyobb a hitele egy új ismerős számára. A tanúsítványokat nyilvános szerverekre töltik fel, mint amilyen a pgp.mit.edu.

A tanúsítványok tárolásához, kezeléséhez, terjesztéséhez, ellenőrzéséhez használt hardver lés szoftver elemeket, beállításokat, házirendeket és eljárásokat összefoglaló néven nyilvános kulcsú infrastruktúrának (**public key infrastructure**, PKI) nevezzük. A jelenleg biztonságosnak tekinthető PKI megoldás a kulcstároló/titkosító célhardver alkalmazása, melyből a privát kulcs sohasem lép ki. Már egy olcsó, USB-key formájú eszköz funkcionálása tartalmazza a kulcsgenerálást (fizikai véletlenszám-generálás) és kulcstárolást, X509 tanúsítványok tárolását, és az elterjedt titkosítási és aláírási algoritmusok végrehajtását.

5. Üzenetpecsétek

Üzenetpecsét (hash) algoritmusokon alapul a kriptográfiai adatintegritás-védelem. A hash függvény bemenete egy gyakorlatilag tetszőlegesen nagy méretű adatblokk, a kimenete pedig egy kis méretű, tipikusan néhány száz bites hash érték. A számítás gyorsasága nagyon fontos (lásd az alábbi táblázatot).

Módszer	Relatív sebesség	Értékelés
Hash függvény (üzenetpecsét)	3	Leggyorsabb
Folyamtitkosító	2	Gyorsabb
Blokk-titkosító	1	Gyors
Nyílt kulcsú titkosító	0.02	Nagyon lassú

A hash függvényektől elvárt tulajdonságok:

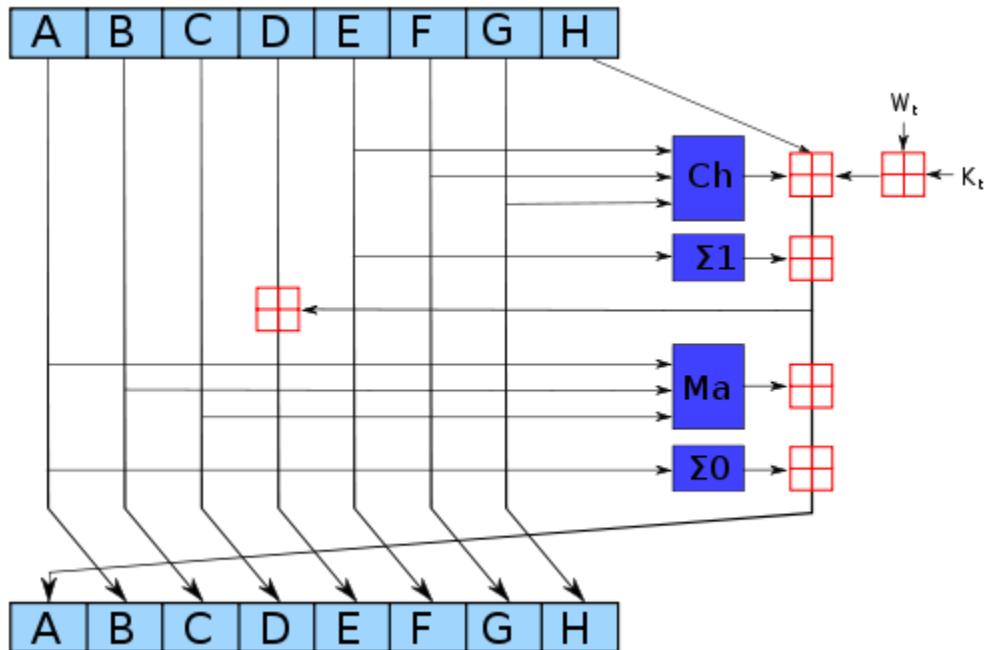
- Bármekkora x bemenetre alkalmazható legyen a függvény.
- A pecsét kicsi, fix méretű legyen.
- A pecsét számolása hatékonyan és könnyen megvalósítható legyen.
- Lavinahatás: a bemenet egy bitjének megváltoztatása 50% valószínűsséggel változtasson meg minden bitet a pecsétben.

¹⁸ Például a Mozilla Firefox böngészőben a CRL-szerverek listája telepítéskor üres, a felhasználónak kell konfigurálni.

- Álljon ellen a támadásoknak:
 - egy adott pecsétből egy ilyen pecsétet adó üzenetet nehéz meghatározni (az erre irányuló kísérlet a **preimage** támadás)
 - egy adott üzenethez nehéz egy ugyanolyan pecsétet adó másik üzenetet találni (**second preimage** támadás)
 - nehéz két, ugyanolyan pecsétet adó üzenetet találni (ütközés, **collision** támadás)

A jelenleg széles körben használt algoritmusok az MD5, az SHA-1 és az SHA-2. Az MD5 gyakorlatilag törtnek tekinthető, alkalmazása nem javasolt. Az SHA-1 algoritmus részletes leírását lásd a III. Mellékletben. Mivel az SHA-1-ben 2005-ben sebezhetőségeket találtak, ezért SHA-2 néven továbbfejlesztették. Ennek jellemzői:

- 224, 256, 384 vagy 512 bites hash méret
- a 256 bites változatban 64 iteráció a bemenet 512 bites blokkjain, melyeket 8 db. 32 bites state regiszterben tárolnak
- eddig nem találtak rá érdemi törést.



A fenti ábrán az SHA-2 szerkezete látható. A kék hátterű dobozok AND, XOR, OR, rotálás és shiftelés műveleteket tartalmaznak, W_t a bemenetből az adott iterációra kiterjesztett darab, K_t az iterációhoz tartozó konstans.

A születésnap-támadás

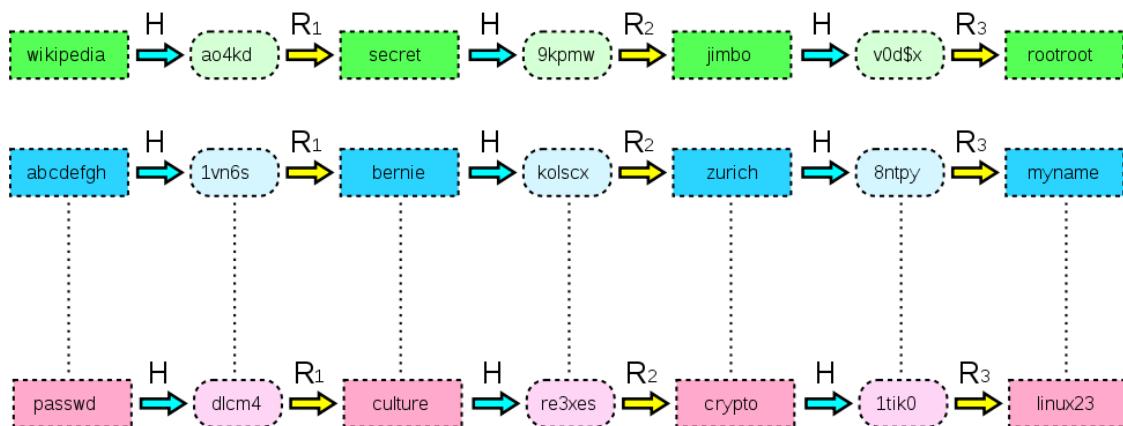
Minden üzenetpecsétre alapuló módszer támadható az ún. születésnap-támadással, ami lényegében egy ütközéses támadás. A digitális aláírás elleni támadás alapgondolata az, hogy bár egy konkrét üzenetpecsétet adó dokumentumot nagyon nehéz találni, ennél a születésnap-paradoxonnak

köszönhetően lényegesen könnyebb két *azonos* üzenetpecsétet adó dokumentumot találni. A születésnap-paradoxon matematikai hátterét lásd a IV. Mellékletben. Tehát a támadó az aláírandó dokumentumnak, például egy szerződésnek eleve két példányát készíti el: az egyik tartalma a számára kedvezőbb. Ezután minden dokumentum-változaton lényegtelen változtatásokat hajt végre, mint például a szóközök beszúrása üres sorokba stb., és közben minden kiszámolja a két változat hash értékét. Ha a két hash egyezik, aláíratja a dokumentumot, majd lecseréli a számára kedvezőbb változatra. A hash egyezése (az ütközés) biztosítja az aláírások egyezését a két változatra. A születésnap-támadás kivédhető az elegendően nagy üzenetpecsét-mérettel (ami 2012-ben 256 bit vagy annál több). De a fenti példában van egy egyszerűbb megoldás is: nyomjunk még egy szóközt aláírás előtt a dokumentum egy üres sorába...

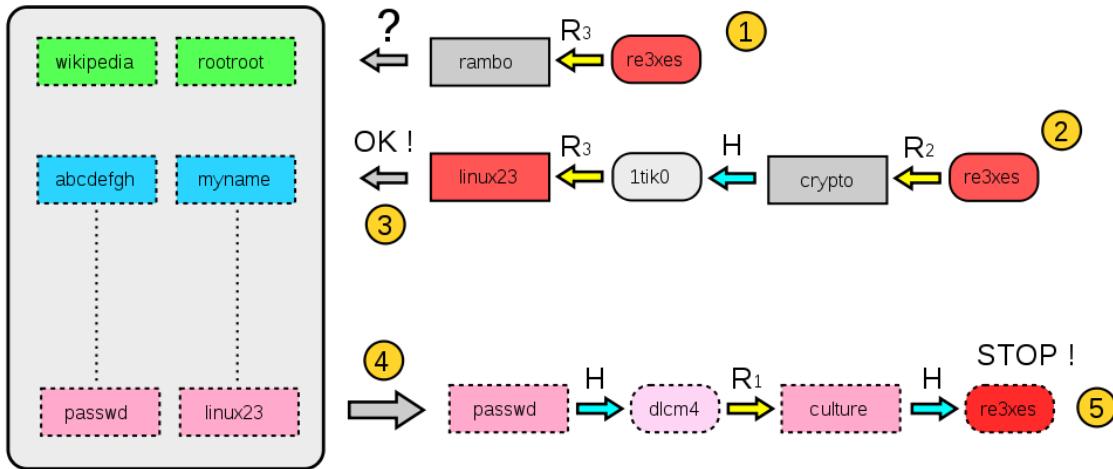
A szivárványtáblás támadás

A szivárványtáblás támadás az általános idő-tárhely ekvivalencia támadás hash függvényekre, különösen jelszó-hash visszafejtésre kifejlesztett változata. A támadás alapfeltevése, hogy a hash érték a jelszó-string nem invertálható random függvénye. A támadás lépései:

- előfeldolgozás, a szivárványtáblák feltöltése. A támadó feltételez egy jelszóhosszat és egy karakterkészletet, melyet a feltörni kívánt hash-hoz tartozó jelszó használt. Ezután készít egy olyan ún. redukciós függvényt, mely egy hash értékből a megfelelő hosszúságú és karakterkészletű stringet állít elő. Ezután egy véletlen jelszóból indulva kiszámítja az ehhez tartozó hash értéket, majd a hash-re alkalmazza a redukciós függvényt, ennek eredményére ismét a hash értéket számít és így tovább. A lánc hossza a tábla paramétere. A szivárványtábla egy rekordjába csak a lánc első és utolsó elemét menti el.



- a szivárványtáblák keresése. A támadó most betölti a táblát a memóriába, és a feltörni kívánt hash értékre alkalmazza a redukciós függvényt, majd megnézi, szerepel-e az így kapott string a tábla valamelyik rekordjában, mint utolsó érték. Ha nem, alkalmazza a stringre a hash függvényt, majd a redukciós függvényt, újra keres a táblában és így tovább. Ha megtalálja stringet, akkor az illető rekord feljegyzett első elemétől indulva megtalálható az a jelszó, amely a kérdéses hash értéket adja.



Mivel a sikeres támadáshoz szükséges táblák mérete a jelszó hosszával és a karakterkészlet számoságával gyors ütemben nő, ezért a szivárványtáblás támadás ellen hatékonyan lehet védekezni a jelszó méretének megnövelésével. A jelszóhoz a hash számítása előtt hozzáfűznek egy felhasználó-specifikus stringet, miáltal az előre elkészített táblák használhatatlanná válnak. A műveletet **sózásnak** (salting) nevezik.

6. A kriptográfia jövője

A tömeges alkalmazású kriptográfia gerincét jelenleg (2012-ben) az alábbi elemek alkotják:

- titkos kulcsú kommunikáció: AES (128 bites)
- kulcsmegosztás és autentikáció: RSA
- dokumentum-integritás: SHA-2

Ezen algoritmusok, módszerek részletes leírása megtalálható a Mellékletekben. A kriptográfia jelentősége azonban az exponenciális ütemben gyűlő információtömeggel egyre nő, és a jövőben új megoldások elterjedése várható. Ebben a fejezetben néhány ilyen igényes módszert mutatunk be.

Az elliptikus görbükre alapuló kriptográfia (ECC)

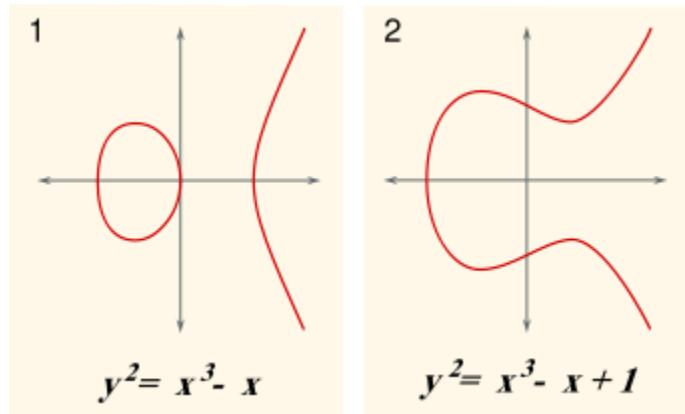
Mint említettük, a nyílt kulcsú kriptorendszerök biztonsága erősen függ egyrészt az egyirányú függvény invertálására irányuló gyors matematikai módszerek fejlődésétől, másrészt a támadó számára rendelkezésre álló, egyre nagyobb számítástechnikai kapacitástól. Ennek gyakorlati következménye, hogy a például a leginkább elterjedt kulcsmegosztási és autentikációs protokoll, az RSA évről évre egyre nagyobb kulcsmérettel üzemeltethető csak, ha egy adott biztonsági szintet meg kívánunk tartani. Az alábbi táblázat az ECRYPT 2010-es kulcsméret-ajánlásait tartalmazza.

<i>Blokkos kriptorendszer kulcsmérete</i>	<i>Biztonsági szint</i>	<i>RSA kulcsméret</i>	<i>ECC kulcsméret</i>
72	Rövid idő alatt, egyszerű módszerekkel törhető	1008	144
80	“elméletileg” megfelelő	1248	160

96	Az ajánlható „abszolút minimum”	1776	192
112	Az ajánlható „megfelelő minimum”	2432	224
128	Megfelelő, kivéve a szigorúan titkos dokumentumokat	3248	256
256	A szigorúan titkos dokumentumok számára is megfelelő	15424	512

Látható, hogy az RSA számára ajánlott „megfelelő minimum” a több ezer bites kulcsméret. Ennek támogatása természetesen nem jelent gondot egy asztali számítógépnek, azonban egyre inkább terjednek azok a smart card (chipkártya) alapú alkalmazások, melyek rendkívül korlátozott energiabelhasználással és memóriával kénytelenek működni. Ezeken a növekvő kulcsméret támogatása egyre nagyobb nehézségekbe ütközik. Az RSA algoritmikus sebezhetőségei mellett ezért is lényeges a kisebb kulccsal dolgozó, gyengébb hardvert igénylő alternatívák kutatása. Mint a fenti táblázatból látszik, az ECC ugyanazt a biztonsági szintet lényegesen kisebb kulcsmérettel tudja garantálni, ezért a jövőben valószínűleg le fogja váltani az RSA-t.

Az **elliptikus görbe** azon pontok halmaza, melyek kielégítik az $y^2 = x^3 + ax + b$ egyenletet, ahol a többszörös gyöök kizárása érdekében feltesszük, hogy $4a^3 + 27b \neq 0$. Az alábbi ábra két elliptikus görbét mutat.



Az ellitikus görbe két pontjának „összegét” úgy definiáljuk, hogy megkeressük azt a pontot a görbén, ahol a két pontot összekötő egyenes metszi a görbét, majd ezt tükrözük az x tengelyre. A tükrökép is a görbe pontja (1), hiszen a görbe szimmetrikus az x tengelyre. Természetesen igaz, hogy $P + Q = Q + P$, tehát a művelet kommutatív (2). Bebizonyítható, hogy az összeadási művelet asszociatív, tehát $P + (Q + R) = (P + Q) + R$ fennáll (3). Jelöljük 0-val a görbe azon pontját, melyre $y = \infty$. Bármely ponthoz ezt a 0-t adva, az összeadási szabály szerint P-be jutunk vissza, tehát $P + 0 = 0 + P$ fennáll (4). Továbbá, a $P + -P = 0$ feltételt kielégítő $-P$ pont létezik: ez a P x tengelyre vett tükröképe, P additív inverze (5), mivel a két pontot összekötő egyenes párhuzamos az y tengellyel. A fenti (1)-(5) öt tulajdonság megléte biztosítja, hogy a görbe pontjai az összeadás műveettel Abel-csoportot alkotnak. Ha pedig a P pontot saját magával adjuk össze, akkor az egyenes a P pontban húzott érintő. Egy pont többszöröseit tehát könnyen kiszámíthatjuk ismételt összeadásokkal, így értelmezhetjük kP -t, ahol k természetes szám.

A kriptográfiai alkalmazáshoz az ellitpikus görbét diszkretizáljuk olyan módon, hogy a pontok koordinátái a valós számok helyett egy p prímszám alapú F_p véges test elemei (a p -nél kisebb természetes számok) legyenek, az aritmetikai számítások megfelelő (modulo p) módosításával. A véges test feletti E görbét $E(F_p)$ -vel jelöljük. $E(F_p)$ pontjainak számára ismeretes Hasse eredménye (1933):

$$p + 1 - 2\sqrt{q} \leq \text{card}(E(F_p)) \leq p + 1 + 2\sqrt{q}$$

Az $E(F_p)$ pontjai által alkotott csoport ciklikus. Egy P pontot generátornak nevezünk, ha annak többszörösei ($P, 2P, 3P, \dots$) a görbe összes pontját előállítják.

Az elliptikus görbe feletti diszkrét logaritmus problémát ekkor (ECDLP) úgy definiálhatjuk, mint a k szorzótényező meghatározását a P és a $Q = kP$ pontokból, egy adott diszkretizált $E(F_p)$ görbüre. Ha p nagy, legalább 200 bites prímszám, akkor a feladat a diszkrét logaritmushoz hasonlóan nagyon komplex, idő- és energiaigényes. A Diffie-Hellman kulcsmegosztási algoritmust könnyű úgy módosítani, hogy az ECDLP-t használja:

1. A felek nyilvánosan megegyeznek egy $E(F_p)$ diszkretizált görbüben, és a görbe egy P pontjában (ezekre például a NIST ajánlásokat tesz közzé).
2. Alice választ egy véletlen k_a részkulcsot ($1 < k_a < p-1$), és elküldi Bobnak a $k_a \cdot P$ pont koordinátáit. Hasonlóképpen Bob is választ egy véletlen k_b részkulcsot ($1 < k_b < p-1$), és elküldi Alice-nak a $k_b \cdot P$ pont koordinátáit.
3. Mindketten kiszámítják a $k_b \cdot k_a \cdot P = k_a \cdot k_b \cdot P$ pont koordinátáit. A titkos viszonykulcs ezen pont x koordinátája lesz.

Például a $p = 61$ feletti $y^2 = x^3 + 2x + 4$ görbe kiszemelt pontja legyen a $P = (7, 19)$. Ha $k_a = 14$, akkor az Alice által küldött pont a $(47, 22)$, a Bob által küldött pont pedig $k_b = 50$ esetén a $(8, 31)$. A közösen kiszámított $k_b \cdot k_a \cdot P = k_a \cdot k_b \cdot P$ pont a $(23, 54)$, tehát a 23 lesz a titkos viszonykulcs.

Kvantum-kriptográfia

A „kvantum-kriptográfia” kifejezést általában olyan módszerre értik, mellyel a csatornát a kulcsmegosztás idejére kvantumfizikai törvényeket kihasználva biztosítják, ezáltal elérők, hogy ne kelljen nyilvános kulcsú kulcsmegosztást alkalmazni (Quantum Key Distribution Networks, QKD).

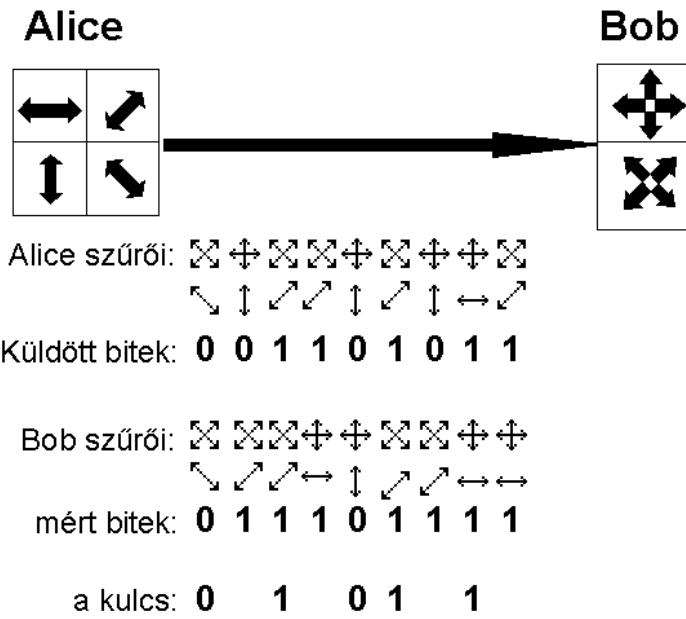
A makroszkopikus világban, ha egy folyamatot elindítunk - például leejtünk egy követ -, akkor a klasszikus fizika törvényei segítségével kiszámíthatjuk, hogy melyik időpillanatban mi történik (milyen gyorsan zuhan a kő, hol tart éppen), és ami nagyon fontos: a zuhanó kő helyét anélkül megállapíthatjuk, hogy megzavarnánk az esésben.

Mi történik az atomok szintjén? Képzeljünk el egy részecskét, amiről tudjuk, hogy észak-dél irányban rezeg. Ha meg akarjuk mérni, hogy a részecske kelet-nyugati vagy észak-dél irányban rezeg-e, akkor a mérés eredménye az, hogy észak-dél irányban. Mi van azonban, ha azt mérjük meg, hogy északkelet-délnyugat, vagy északnyugat-délkelet irányban rezeg-e? Azt várnánk, hogy a mérés eredménye az, hogy egyik irányban sem. Valójában, ha egymás utáni, kezdetben észak-dél irányban rezgő részecskéken

végeznénk el ezt a mérést, akkor az esetek felében azt kapjuk, hogy északkelet-délnyugat, máskor meg, hogy a másik irányban, méghozzá teljesen véletlenszerűen. Sőt, a mérés után a részecske valóban a mért irányban fog tovább rezegni! A kvantummechanika kimondja, hogy nem lehet megmérni egy fizikai mennyiséget anélkül, hogy hatással ne lennénk a mérendő mennyiségre. A makroszkopikus mérésnél is tapasztalunk hasonló jelenséget (például áramerősségi mérésekor a mérőműszer beleavatkozik az áram folyásába), ám ez azért van, mert a műszereink nem ideálisak. A kvantummechanika szerint viszont, még akkor is beleavatkoznánk a fenti részecske rezgésébe, ha ideális műszerünk lenne: a részecske „észreveszi”, hogy őt most mérik, ezért gyorsan beállítja a rezgését egyik, vagy a másik irányba. A kvantumkriptográfia ezt a furcsa viselkedést felhasználva elméletileg is feltörhetetlen titkosítási módszert ad a kezünkbe.

1984-ben Charles Bennett és Gilles Brassard kidolgozták a BB84 protokollt, amely a kvantummechanikán alapulva megoldást nyújt a fenti problémára, vagyis hogy hogyan cseréljen egymás között két személy, Alice és Bob titkos kulcsot anélkül, hogy az illetéktelen – Eve – kezébe jutna. A módszer mikroszkopikus részecskéket használ, a gyakorlatban a fotonok a legkönnyebben alkalmazhatóak. Képzeljünk el egy 3 dimenziós, derékszögű koordináta rendszert. A fotonok a Z tengely mentén haladnak, ám van egy, a haladás irányára merőleges rezgésük az XY síkban, amit polarizáltságnak hívnak. A hagyományos izzó mindenféle polarizáltságú fotont bocsát ki magából. Polárszűrő segítségével azonban ki tudjuk választani, például az Y irányban polarizált fotonokat, hogy csak azokat küldjük tovább.

Alice és Bob két fajta szűrőkészlettel rendelkezik: Az egyik a *retilineáris*, a másik a *diagonális*. A retilineáris készlet egy X és Y irányú szűrőből áll: \leftrightarrow és $\uparrow\downarrow$. A diagonális készlet szűrői ehhez képest 45 fokkal el vannak forgatva: $\swarrow\uparrow$ és $\nwarrow\downarrow$. A retilineáris szűrőkészlet jele +, míg a diagonális \times . A bináris 1-es a következőképpen polarizált fotonoknak felel meg: \leftrightarrow vagy $\swarrow\uparrow$. A 0 pedig: $\uparrow\downarrow$ vagy $\nwarrow\downarrow$. Alice véletlenszerűen küldi az 1-eseket és 0-kat jelképező fotonokat úgy, hogy a szűrőkészleteit is véletlenszerűen váltogatja. A másik oldalon Bob fogadja Alice fotonjait, és megpróbálja detektálni azok polarizáltságát: ő is véletlenszerűen használja a szűrőkészletet. Akkor állapítja meg helyesen az érkező fotonok polarizáltságát, ha az adott fotont ugyanazzal a szűrőkészlettel méri meg, mint amit Alice használt arra a fotonra. Ha rosszat választ, akkor $\frac{1}{2}$ a valószínűsége annak, hogy ugyanazt a bitet kapja, mint amit Alice küldött. Bob gondosan feljegyzi, hogy mikor milyen szűrőt alkalmazott, és milyen biteket kapott. Miután Alice elküldte a biteket, egy nyilvános csatornán (amit bárki lehallgathat) közli Bobbal, hogy mikor milyen szűrőt használt. Bob megnézi a feljegyzéseit, és ekkor már tudja, hogy azokat a biteket detektálta helyesen, ahol ő is azt a szűrőt használta, amit Alice. A többi bitet eldobja. Ekkor Bob közli Alice-al szintén egy nyilvános csatornán, hogy mely sorszámu咬 biteket találta el. Ennél a pontnál Alice tudja, hogy Bob mely biteket tartotta meg, és ezt a bitsorozatot használják a titkos kulcsként (lásd az alábbi ábrát).



Nézzük meg, mi történik, ha Eve beiktat egy saját szűrkészlet Alice és Bob közé. Ő is ugyanúgy tudja csak detektálni Alice fotonjait, ahogy Bob. Az esetek egy részében ő is rossz szűrőt használ. Viszont, hogy ne keltsen gyanút, a fotonokat továbbküldi Bob felé. Azonban, ha nem megfelelő szűrőt alkalmaz, például Alice $\uparrow\downarrow$ -t küld, ami 0, és Eve a \times szűrőt használja, akkor véletlenszerűen \leftarrow -t vagy \rightarrow -t mér, azaz 1-est vagy 0-t. Sajnos nem tudja az eredeti, detektálás előtti fotont továbbküldeni, hanem a tévesen detektáltat továbbítja Bobnak¹⁹. Tehát a támadó az esetek $\frac{1}{4}$ -ében elrontja az elküldött bitet. Ekkor, ha Bob ugyanazt a szűrőt használja, mint amit Alice, akkor nem jól állapítja meg a foton polarizáltságát, vagyis a küldött bit értékét, hiszen Eve csavart egyet a fotonon. Miután az adás végén Alice és Bob a nyilvános csatornán egyeztetik a használt szűrőket, és a Bob által elvileg jól detektált bitek sorszámát, a kapott kulcs egy részét elküldik egymásnak szintén a nyílt csatornán. Ha azt látják, hogy ez a kis részlet nem egyezik, akkor tudják, hogy Eve hallgatózott (és ezzel elrontotta a kulcsot). Ekkor átkapcsolnak egy másik csatornára. Ha az összehasonlítás után azt látják hogy nem volt hallgatózás, akkor a kulcs maradék részét fogják alkalmazni. Ebből adódik a módszer hátránya is: Ha nagy távolságra akarunk kulcsot küldeni, akkor nem alkalmazhatunk jelerősítőket, hiszen az olyan, mint mikor Eve hallgatózik. Az egyes eseteket az alábbi táblázat foglalja össze.

Alice véletlen bitsorozata	0	1	1	0	1	0	0	1
Alice véletlen szűrői	+	+	\times	+	\times	\times	\times	+
Az Alice által küldött polarizációk	\uparrow	\rightarrow	\downarrow	\uparrow	\downarrow	\nearrow	\nearrow	\rightarrow
Eve véletlen szűrői	+	\times	+	+	\times	+	\times	+

¹⁹tehát a fotont nem lehet másolni ("no cloning theorem")

Eve által mért és továbbküldött polarizációk	\uparrow	\nearrow	\rightarrow	\uparrow	\searrow	\rightarrow	\nearrow	\rightarrow
Bob véletlen szűrői	+	\times	\times	\times	+	\times	+	+
A Bob által mért polarizációk	\uparrow	\nearrow	\nearrow	\searrow	\rightarrow	\nearrow	\uparrow	\rightarrow
A szűrők nyilvános egyeztetése								
A megosztott titkos kulcs	0		0			0		1
Hibák a kulcsban	✓		\times			✓		✓

Ha a felek n bitet egyeztetnek, akkor a hallgatózás felfedezésének a valószínűsége $P = 1 - \left(\frac{3}{4}\right)^n$, tetszőlegesen közelíthető 1-hez. Arra az esetre, ha a felek a felfedezett hallgatózás ellenére ezt a csatornát használják tovább, kidolgoztak olyan protokollokat, mellyel bit-blokkonként paritást számolva nyilvános csatornán tetszőlegesen kis mértékig csökkenthetik a két oldal titkos kulcsa közti különbséget (information reconciliation).

A QKDN elleni lehetséges támadások:

- Hallgatózás (észrevehető, lásd fent)
- Beékelődés. A megbízható, tanúsítvány alapú autentikációt nem pótolja a kvantumkriptográfia sem.
- Fotonhasítás. Ha a fotonforrás nem ideális, csak átlagosan 1 fotont ad ki, időnként kettőt is, akkor a dupla foton egyik felét a támadó kvantum-memoriában tárolhatja.
- Hacking a véletlenszám-generátor és a protokoll hibái ellen
- DoS: elvágják a biztonságos optikai kábelt

Számos helyen, több éve stabilan működnek QKDN rendszerek. 2008-tól Ausztriában Bécsben és Sankt Pöltenben már sikerült létrehozni egy hat helyszínt összekötő hálózatot, 200 kilométernyi optikai kábellel és 69 km fizikai átmérővel, ami ezt a protokollt használja. A szükséges hardver azonban jelenleg (2012-ben) még rendkívül drága²⁰. Amennyiben ez a módszer bárki számára elérhető lesz, úgy biztosak lehetünk abban, hogy a két végpont között senki nem tudja majd lehallgatni a kommunikációt.

²⁰kvantumkritográfiai eszközökkel foglalkozik például a MagiQ és id Quantique: www.magiqtech.com, www.idquantique.com.

Irodalom

- [1] Simon Singh: Kódkönyv. Park, 2001.
- [2] Virrasztó Tamás: Titkosítás és adatrejtés. Netacademia, 2004.
- [3] Richard B. Wells: Applied Coding and Information Theory for Engineers, Prentice Hall, 1999
- [4] R.E. Smith: Internet security. Addison-Wesley, 1997.
- [5] Andrew S. Tannenbaum: Számítógép-hálózatok, Panem, 2004

I.MELLÉKLET: AZ AES KRIPTOGRÁFIAI ALGORITMUS

Bevezetés

Korábban a DES (Data Encryption Standard) egy szabványos, 56 bites blokk kódoló algoritmus volt, amelyet az IBM-nél fejlesztettek ki. Az algoritmust és változatait évtizedeken át alkalmazták, ám számos kritika érte a kis kulcsméret miatt, valamint sokan bizalmatlanok is voltak vele szemben. Az amerikai kormányzati szervek nem szerették volna, ha olyan titkosítási módszer jut a lakosság – és így a bűnözők – kezébe, amelyet a hatóságok nem képesek megfejteni. Ám nyilván a civil szférának szüksége volt egy biztonságos kódoló eljárásra, elsősorban az üzleti életben. A kormányzat úgy döntött, hogy a DES 56 bites kulcsokat használhat, ugyanis az akkori vélekedés szerint a civilek, bűnözők nem rendelkeztek olyan számítási kapacitással, amellyel egy ilyen kulccsal kódolt üzenetet meg lehetett fejteni, viszont a hatóságok igen. Ez természetesen az összeesküvés-elmélet gyártók számára remek táptalajt adott. Továbbá a DES több s-dobozt is alkalmazott, amelyeket az IBM szerint véletlenszerűen generáltak. Sokan feltételezték, hogy valójában valamilyen hátsó ajtó van elrejtve ezekben az s-dobozokban, így aki megfelelően használja őket, az könnyen feltörhet bármilyen üzenetet. Eddig azonban még nem sikerült igazolni ezt a sejtést.

A 90-es évek végére nyilvánvalóvá vált, hogy a DES elavult, így szükség van egy új titkosítási szabványra. Szerették volna azt is elkerülni, hogy a közzélemezű bizalmatlan legyen a szabvánnyal. 1997-ben az NIST (National Institute of Standards and Technology) kiírt egy kriptográfiai versenyt, amelyben öt szempontnak megfelelő algoritmusokat vártak. A szempontok az alábbiak voltak:

- Szimmetrikus blokk-kódoló legyen
- Az algoritmus minden részlete nyilvános
- Támogassa a 128, 192 és 256 bites kulcsokat
- Hardveresen és szoftveresen nagy hatékonysággal implementálható legyen
- Szabadon felhasználható legyen

A versenyre számos pályázat érkezett, a nyertes a belga szerzők által megalkotott Rijndael algoritmus lett. A módszer neve a szerzők nevéből származik: Joan Daemen és Vincent Rijmen. Az algoritmus mindegyik követelménynek kiválóan eleget tesz, a harmadiknál még túl is tett: Amellett, hogy alkalmazható 128, 160, 192, 224 és 256 bites kulcsokkal, a kódolható blokk mérete szintén 128, 160, 192, 225 és 256 bit lehetett. A hivatalos szabványba viszont csak a 128 bites blokkméret, és a verseny követelményeinek megfelelő 128, 192 és 256 bites kulcsok kerültek be. Az új szabvány neve AES lett, azaz Advanced Encryption Standard. Ettől fogva bizonyos források a DES rövidítést már Data Encryption System-nek tüntetik fel.

Az alábbiakban először ismertetjük az algoritmushoz szükséges matematikai alapműveleteket, majd bemutatjuk a kódolás, illetve dekódolás részleteit.

Matematikai alapok

Mint a legtöbb modern kriptográfiai algoritmus, az AES is nagyban támaszkodik a bitenkénti kizárá vagy, azaz XOR műveletre. A műveletnek eredménye akkor 1, ha a két operandus értéke eltér egymástól, egyébként az eredmény 0, ahogy az 1. táblázat is mutatja.

1. táblázat Az XOR művelet igazságtablája

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

A működéséből adódik egy fontos tulajdonsága, amely miatt a kriptografiában előszeretettel alkalmazzák:

$$(A \text{ XOR } B) \text{ XOR } B = A$$

Ennek igazolása látható a 2. táblázatban. Ezt a tulajdonságot arra használják, hogy egy kulcs bitsorozat és egy kódolandó üzenetet-et az XOR művelettel adjanak össze. A dekódoláskor pedig elég annyit tenni, hogy a korábban kapott kódolt üzenethez újra hozzáadjuk a kulcsot az XOR segítségével, így előáll az eredeti üzenet. Ennek alkalmazására a legegyszerűbb példa a korábban már bemutatott OTP algoritmus, de az AES egyes iterációiban is elvégezzük ezt a műveletet.

A	B	A XOR B = C	C XOR B = A
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Az XOR művelet hardveresen rendkívül egyszerűen megvalósítható, akár két tranzisztor is elég hozzá. Továbbá szinte minden mai általános célú processzor támogatja.

Az AES főbb lépései polinomokkal való összeadásra és szorzásra épülnek. Először ismertetjük az összeadást, majd arra alapozva bemutatjuk a szorzást.

Összeadás

Két polinomot a $GF(2^8)$ algebra szerint adunk össze, valamint az összeadást az \oplus -al jelöljük. A polinomok együtthatói a $\{0,1\}$ halmzból kerülnek ki. Amennyiben a két polinom i . tagjának együtthatója 1, az eredmény együtthatója 0 lesz, és ez nincsen hatással az $i+1$. együtthatóra, ahogy ezt az alábbi példában láthatjuk:

$$(x^6 + x^4 + x + 1) \oplus (x^7 + x^5 + x^3 + x + 1) = x^7 + x^6 + x^5 + x^4 + x^3$$

Mivel a polinomok együtthatói a $\{0,1\}$ halmazba tartoznak, ezért a polinomokat egyszerűen ábrázolhatjuk bitsorozatokkal. Az $x^6 + x^4 + x + 1$ polinomot a 01010011 bitsorozat reprezentálja: Az i . bit megegyezik az i . együttható értékével. Ez alapján látható, hogy a \oplus művelet könnyen megvalósítható az XOR segítségével. A fenti példa tehát a következőképpen néz ki bitsorozatokkal:

$$\begin{array}{rcl}
 01010011 & x^6 + x^4 + x + 1 \\
 10101011 & x^7 + x^5 + x^3 + x + 1 \\
 \hline
 11111000 & x^7 + x^6 + x^5 + x^4 + x^3
 \end{array}$$

Ennek köszönhetően az összeadást egyetlen processzorutasítással is elvégezhetjük. Megjegyezzük, hogy az AES esetében a kivonás művelet azonos az összeadással, erre a szorzásnál lesz szükségünk.

Szorzás

A szorzást szintén olyan polinomok között értelmezzük, mint az összeadásnál, a művelet jele pedig: \circ . Az eredmény polinomban továbbra is a 0, illetve 1 együtthatók szerepelhetnek. Fontos, hogy a műveletet modulo aritmetikával végezzük el, amelyhez a következő polinomot használjuk:

$$x^8 + x^4 + x^3 + x + 1$$

Ennek következtében az eredménye az, hogy az eredmény polinom legfeljebb 7-ed fokú lehet, így az elfér egy byte-ban. A szorzás két fő lépésből áll: Először összeszorozzuk a két polinomot, majd elvégezzük a maradékos osztást. A szorzás hasonlóan működik, mint ahogy azt korábban megtanultuk: Az első polinom minden tagját szorozzuk a második minden tagjával, így rész polinomokat kapunk. Ezeket a rész polinomokat a \oplus műveettel összeadjuk, ahogy az alábbi példában is láthatjuk:

$$\begin{aligned}
 (x^6 + x^4 + x + 1)^\circ (x^7 + x^5 + x^3 + x + 1) &= [(x^{13} + x^{11} + x^8 + x^7) \oplus (x^{11} + x^9 + x^6 + x^5)] \\
 &\quad \oplus [(x^9 + x^7 + x^4 + x^3) \oplus (x^7 + x^5 + x^2 + x)] \oplus (x^6 + x^4 + x + 1) = \\
 (x^{13} + x^9 + x^8 + x^7 + x^6 + x^5) &\oplus (x^9 + x^5 + x^4 + x^3 + x^2 + x) \oplus (x^6 + x^4 + x + 1) = \\
 (x^{13} + x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + x) &\oplus (x^6 + x^4 + x + 1) = \\
 x^{13} + x^8 + x^7 + x^3 + x^2 + 1
 \end{aligned}$$

A kapott polinomot maradékosan osztani kell a $x^8 + x^4 + x^3 + x + 1$ polinommal, vagyis az osztás az

$$\begin{aligned}
 (x^6 + x^4 + x + 1)^\circ (x^7 + x^5 + x^3 + x + 1) \bmod (x^8 + x^4 + x^3 + x + 1) &= \\
 x^{13} + x^8 + x^7 + x^3 + x^2 + 1 \bmod (x^8 + x^4 + x^3 + x + 1)
 \end{aligned}$$

művelet elvégzését jelenti. A maradékos osztás ugyanazon az elven működik, mint számok esetében, visszavezethető a kivonásra: Addig vonjuk ki az osztandóból az osztót, amíg ez utóbbi nem kisebb, mint az osztandó. Ehhez az osztó polinomot meg kell szorozni x olyan hatványával, hogy a szorzat

legmagasabb fokú tagja megegyezzen az osztandó legmagasabb fokú tagjával. A kapott szorzatot és az osztandót pedig az \oplus művelettel összeadjuk. Ezt addig ismételjük, amíg van az eredmény polinomban 7-nél magasabb fokú tag. Az algoritmust a fenti példán a következő módon hajtjuk végre:

Az osztandó polinom: $x^{13} + x^8 + x^7 + x^3 + x^2 + 1$

Az osztó polinom: $x^8 + x^4 + x^3 + x + 1$

Az osztandó polinom legmagasabb fokú tagjának foka 13, ezért x^5 -el beszorozzuk az osztót, majd hozzáadjuk az osztandóhoz:

$$(x^{13} + x^8 + x^7 + x^3 + x^2 + 1) \oplus (x^{13} + x^9 + x^8 + x^6 + x^5) = x^9 + x^7 + x^6 + x^5 + x^3 + x^2 + 1$$

Az eredmény legmagasabb fokú tagjának foka 9, tehát még nem végeztünk, legyen ez az osztandó. Az osztót most x -el kell szoroznunk, majd a szorzatot hozzáadjuk az osztandóhoz:

$$(x^9 + x^7 + x^6 + x^5 + x^3 + x^2 + 1) \oplus (x^9 + x^5 + x^4 + x^2 + x) = x^7 + x^6 + x^4 + x^3 + x + 1$$

A kapott eredményben már nincs olyan tag, amely foka nagyobb 7-nél, tehát megkaptuk a végeredményt.

Ahogy az összeadást, úgy a szorzást is bitsorozatokon végzett műveletek segítségével implementálják. Vegyük észre, hogy ha egy polinomot megszorzunk x^i -vel, akkor az azt reprezentáló bitsorozatot balra toljuk i bittel, az új bitek pedig a 0 értéket veszik fel. Ezen felül még az XOR-ra lesz szükségünk. Jelöljük a polinomokat reprezentáló bitsorozatokat A-val és B-val. A polinomoknál első lépésként az egyik polinom minden tagjával megszoroztuk a másik polinomot, és így rész polinomokat kaptunk. Ezt bitekkel úgy végezzük el, hogy feljegyezzük, hogy B-ben mely $0 \leq i \leq 7$ pozícióban van 1. Legyen ezen 1-es bitek száma n . Előállítunk n darab új bitsorozatot úgy, hogy A-t rendre eltoljuk a feljegyzett i értékekkel. A kapott n sorozatot pedig XOR segítségével összeadjuk, ahogy az alábbi példán láthatjuk:

Számítsuk ki a $01010011 \cdot 10101011 \bmod 100011011$ értékét. Legyen A = 01010011, valamint B = 10101011. B-ben a következő pozíciókon van 1: 7, 5, 3, 1 és 0. Tehát ezen értékekkel kell A-t eltolnunk:

$\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}$	Eltolva 7 bittel
$\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}$	Eltolva 5 bittel
$\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}$	Eltolva 3 bittel
$\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}$	Eltolva 1 bittel
$\begin{array}{cccccccc} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}$	Eltolva 0 bittel

XOR -----

0	1	0	0	0	1	1	0	0	0	1	1	0
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

A következő lépés pedig a maradékos osztás elvégzése, mivel a kapott eredményben a 13. pozíció 1-es bit található. Az osztó bitsorozatot rendre eltoljuk annyira, hogy annak legmagasabb helyi értéken lévő 1-es bitje fedje az osztandó bitsorozat legmagasabb helyi értékű 1-es bitjét. A két sorozatot XOR-al

összeadjuk. Amennyiben az összegben van olyan 1-es bit, amely 7-nél magasabb pozícióban található, úgy a fenti lépést megismételjük.

$ \begin{array}{r} 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1 \\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0 \\ \hline \text{XOR} \quad \text{-----} \\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1 \\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0 \end{array} $	Osztandó bitsorozat Osztó bitsorozat eltolva 5 bittel Legmagasabb 1-es bit: 9, folytatjuk Osztó bitsorozat eltolva 1 bittel Legmagasabb 1-es bit: 7, vége
$ \begin{array}{r} 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1 \\ \hline \text{XOR} \quad \text{-----} \end{array} $	
$ \begin{array}{r} 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1 \end{array} $	

A kapott 11011011 bitsorozat pedig az $x^7 + x^6 + x^4 + x^3 + x + 1$ polinomot reprezentálja.

Multiplikatív inverz

Az AES-ben multiplikatív egység elemeknek tekintjük a 00000001 bitsorozat által reprezentált polinomot.

Egy A polinom multiplikatív inverze az az A^{-1} polinom, amivel megszorozva az egység elemet kapjuk: $AA^{-1} \equiv 1 \pmod{M}$, ahol $M = x^8 + x^4 + x^3 + x + 1$. Az S-doboz megalkotásához szükségünk lesz minden lehetséges polinom multiplikatív inverzére. Mivel összesen 256 polinomot használunk, akár brute force módszerrel is megkereshetünk minden multiplikatív inverzet, majd azokat egy táblázatban eltárolhatjuk, de használhatjuk akár a kiterjesztett euklideszi algoritmust (lásd az RSA-nál). Vegyük észre, hogy a 00000000 bitsorozatnak a fenti definíció szerint nem létezik a multiplikatív inverze, ezért ebben az esetben a 00000000-t választjuk invernek.

2. táblázat Byte-ok multiplikatív inverzei az AES-ben

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	00	01	8D	F6	CB	52	7B	D1	E8	4F	29	C0	B0	E1	E5	C7
	1	74	B4	AA	4B	99	2B	60	5F	58	3F	FD	CC	FF	40	EE	B2
	2	3A	6E	5A	F1	55	4D	A8	C9	C1	0A	98	15	30	44	A2	C2
	3	2C	45	92	6C	F3	39	66	42	F2	35	20	6F	77	BB	59	19
	4	1D	FE	37	67	2D	31	F5	69	A7	64	AB	13	54	25	E9	09
	5	ED	5C	05	CA	4C	24	87	BF	18	3E	22	F0	51	EC	61	17
	6	16	5E	AF	D3	49	A6	36	43	F4	47	91	DF	33	93	21	3B
	7	79	B7	97	85	10	B5	BA	3C	B6	70	D0	06	A1	FA	81	82
	8	83	7E	7F	80	96	73	BE	56	9B	9E	95	D9	F7	02	B9	A4
	9	DE	6A	32	6D	D8	8A	84	72	2A	14	9F	88	F9	DC	89	9A
	a	FB	7C	2E	C3	8F	B8	65	48	26	C8	12	4A	CE	E7	D2	62
	b	0C	E0	1F	EF	11	75	78	71	A5	8E	76	3D	BD	BC	86	57
	c	0B	28	2F	A3	DA	D4	E4	0F	A9	27	53	04	1B	FC	AC	E6
	d	7A	07	AE	63	C5	DB	E2	EA	94	8B	C4	D5	9D	F8	90	6B
	e	B1	0D	D6	EB	C6	0E	CF	AD	08	4E	D7	E3	5D	50	1E	B3
	f	5B	23	38	34	68	46	03	8C	DD	9C	7D	A0	CD	1A	41	1C

A kulcs kiterjesztése

Az alábbiakban az AES 128 bites kulcsos változatát ismertetjük. Az algoritmus a kódolandó blokkot először egy 4*4-es méretű állapot táblába másolja, majd több iteráción keresztül transzformálja ezt, az állapotot. Egy transzformáció egyik lépése az, hogy a kezdeti kulcsból előállított byte-okat XOR műveettel hozzáadja az állapothoz. Ennek érdekében a 128 bites kulcsot ki kell terjeszteni, hogy minden transzformációs lépésben mást adhassunk hozzá az állapothoz. A kulcs kiterjesztést elég akkor elvégezni, ha a kódolónknak, illetve dekódolónknak új kulcsot adunk meg, azaz nem szükséges ezt minden blokk kódolásánál végrehajtani.

Jelöljük a kulcs i . ($0 \leq i \leq 15$) byte-ját k_i -vel. Ez alapján a kulcs byte-okat egy 4*4-es táblázatba rendezzük az alábbi módon:

w[0]	w[1]	w[2]	w[3]
k_0	k_4	k_8	k_{12}
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

A táblázat oszlopait a w tömbben tároljuk, a tömb legkisebb indexe a 0. Ezt a táblázatot kell kiterjeszteni 44 oszlopra az első 4 alapján. Az új oszlopokat egymást követően számítjuk ki az alábbi algoritmussal:

Ciklus $i = 4$ -től 43-ig

1. Legyen temp = $w[i - 1]$
2. Ha i osztható 4-vel, akkor
 - i. temp byte-jait forgatjuk: $\text{temp} = [w_0, w_1, w_2, w_3] \rightarrow [w_1, w_2, w_3, w_0]$
 - ii. temp minden byte-ját kicseréljük az S-doboz alapján
 - iii. temp új első byte-jához hozzáadjuk az $x^{i/4-1} \bmod(x^8 + x^4 + x^3 + x + 1)$ polinomot reprezentáló byte-ot
3. elágazás vége
4. $w[i] = w[i - 4] \text{ XOR temp}$

Ciklus vége

Kódolás

A bemenet 16 byte-ját első lépésként egy 4*4-es állapot táblázatba másoljuk, majd ezt az állapotot transzformáljuk 10 iteráción keresztül. Jelöljük a bemenet i . ($0 \leq i \leq 15$) byte-ját a_i -vel. Ez alapján a bemenet byte-okat egy 4*4-es táblázatba rendezzük az alábbi módon:

a_0	a_4	a_8	a_{12}
a_1	a_5	a_9	a_{13}

a_2	a_6	a_{10}	a_{14}
a_3	a_7	a_{11}	a_{15}

Az állapot táblázaton 4 különböző transzformációt hajthatunk végre:

1. Kulcs hozzáadás
2. Byte helyettesítés
3. Sorforgatás
4. Oszlopkeverés

Az alábbiakban sorra ismertetjük ezeket a lépéseket, majd bemutatjuk, hogy ezek segítségével hogyan történik a kódolás.

Kulcs hozzáadás

Ebben a lépésben az állapot táblázat minden byte-jához XOR művelettel hozzáadunk egy másik byte-ot a kiterjesztett kulcs táblázatból. Az első kulcs hozzáadáskor a táblázat első 4 oszlopát használjuk, a következő alkalommal pedig a következő 4 oszlopot. Az AES iterációs lépései előtt előkészítésként elvégezzük ezt a műveletet, így az i . iterációban a kulcs táblázat $[4*i, (i+1) * 4 - 1]$ oszlopait használjuk fel, ahol $1 \leq i \leq 10$. A műveletet az alábbi ábrán mutatjuk be:

$$\begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|} \hline w_{0,i} & w_{0,i+1} & w_{0,i+2} & w_{0,i+3} \\ \hline w_{1,i} & w_{1,i+1} & w_{1,i+2} & w_{1,i+3} \\ \hline w_{2,i} & w_{2,i+1} & w_{2,i+2} & w_{2,i+3} \\ \hline w_{3,i} & w_{3,i+1} & w_{3,i+2} & w_{3,i+3} \\ \hline \end{array} =$$

$$\begin{array}{|c|c|c|c|} \hline s_{0,0} \oplus w_{0,i} & s_{0,1} \oplus w_{0,i+1} & s_{0,2} \oplus w_{0,i+2} & s_{0,3} \oplus w_{0,i+3} \\ \hline s_{1,0} \oplus w_{1,i} & s_{1,1} \oplus w_{1,i+1} & s_{1,2} \oplus w_{1,i+2} & s_{1,3} \oplus w_{1,i+3} \\ \hline s_{2,0} \oplus w_{2,i} & s_{2,1} \oplus w_{2,i+1} & s_{2,2} \oplus w_{2,i+2} & s_{2,3} \oplus w_{2,i+3} \\ \hline s_{3,0} \oplus w_{3,i} & s_{3,1} \oplus w_{3,i+1} & s_{3,2} \oplus w_{3,i+2} & s_{3,3} \oplus w_{3,i+3} \\ \hline \end{array}$$

Byte behelyettesítés

Az AES-ben kettő S-dobozt használunk, az egyiket a kódolásnál, a másikat a dekódolásnál, ez utóbbi az első inverze. A kódolásnál alkalmazottra csak, mint S-dobozra hivatkozunk, a másikat inverz S-doboznak nevezzük. Egy xy alakú, hexadecimálisan leírható byte-ot az S-doboz x . sorában lévő y . cellában található értékre cserélünk ki. Ahogy a bevezetőben említettük, az S-doboz nem egy véletlenszerűen generált táblázat. Egy tetszőleges a byte helyettesítő eleme a következő módon számolható ki:

1. Legyen b = az a byte multiplikatív inverze, amennyiben $a = 0$, úgy $b = 0$.
2. Jelölje b byte i . bitjét b_i .

Legyen $\bar{b}_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$, ahol $c = 01100011$. Ezt a lépést mátrixos alakban is felírhatjuk:

$$\begin{bmatrix} \bar{b}_0 \\ \bar{b}_1 \\ \bar{b}_2 \\ \bar{b}_3 \\ \bar{b}_4 \\ \bar{b}_5 \\ \bar{b}_6 \\ \bar{b}_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

A 2. lépéstre a szakirodalomban mint affin-transzformációra is hivatkoznak. A 3. táblázatban megjelenítettük az így generálható helyettesítő byte-okat hexadecimális formában.

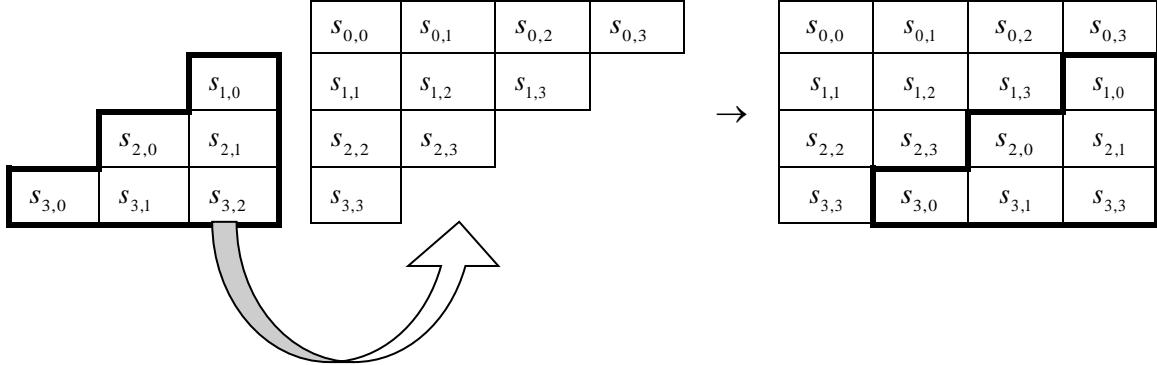
3. táblázat Az AES-ben használt S-doboz

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Sorforgatás

A sorforgatást és az azt követő oszlopkeverést együttesen diffúziós rétegnek is nevezik. Céljuk az, hogy e két lépés segítségével a bemenet minden byte-ja hatással legyen a kimenet minden byte-jára. A forgatás során az i . sor elemeit ($0 \leq i \leq 3$) i -vel balra forgatjuk, ahogy azt az 1. ábra Sorforgatás a kódolásnál láthatjuk:

1. ábra Sorforgatás a kódolásnál

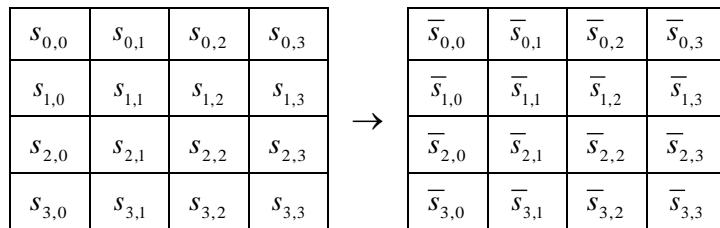


Oszlopkeverés

Ez a transzformáció az állapot tábla oszlopain belül keveri meg a byte-okat. Az oszlopok között nincsen kapcsolat, minden oszlopra ugyanazt a műveletsort végezzük el, a többitől függetlenül: Kiválasztunk egy oszlopot, ennek indexe legyen c , majd az oszalon belüli cellákat a következő képletek alapján változtatjuk meg:

- $\bar{s}_{0,c} = (0x02^\circ s_{0,c}) \oplus (0x03^\circ s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$
- $\bar{s}_{1,c} = s_{0,c} \oplus (0x02^\circ s_{1,c}) \oplus (0x03^\circ s_{2,c}) \oplus s_{3,c}$
- $\bar{s}_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (0x02^\circ s_{2,c}) \oplus (0x03^\circ s_{3,c})$
- $\bar{s}_{3,c} = (0x03^\circ s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (0x02^\circ s_{2,c})$

A kiszámolt byte-okat pedig visszaírjuk az állapot táblázatba:



Az oszalon belüli keverést a sorok eltolása után hajtjuk végre. A sor eltolást és oszlopkeverést háromszor egymás után elvégezve elérhetjük, hogy minden bemeneti byte hatással legyen a kimenet minden byte-jára. Az AES esetében ez okozza a lavinahatást: A bemenet egy bitjének megváltoztatása 50% valószínűséggel változtat meg minden bitet a kimenetben.

Kódoló iterációk

Feltételezzük, hogy a bemenetről beolvastuk a kódolandó üzenetet az állapot táblázatba. Innentől a kódolás menete a következő:

1. Az állapothoz hozzáadjuk a kulcs táblázat [0,3] intervallumba eső oszlopait.
2. Ciklus $i := 1$ -től 9-ig:
 - a. Byte csere
 - b. Sorok forgatása
 - c. Oszlopok keverése
 - d. Az állapothoz hozzáadjuk a kulcs táblázat [$i*4, (i+1)*4-1$] oszlopait
3. Byte csere
4. Sorok forgatása
5. Az állapothoz hozzáadjuk a kulcs táblázat [40, 43] oszlopait

Az utolsó iterációban tehát elhagyjuk az oszlopok keverését. Az eljárás kimenete pedig az állapot táblázat tartalma, ahol a byte-okat oszlop folytonosan kell kiolvasni.

Dekódolás

A dekódolási folyamat a kódolás ismeretében roppant egyszerű, csupán „visszafele” kell csinálni minden lépést, amelyet a kódolás során végrehajtottunk. A dekóolandó üzenetet beolvassuk az állapot táblázatba, hasonlóan, mint a kódoláskor. Tudjuk, hogy a kódolás utolsó lépése az volt, hogy az állapothoz hozzáadtuk a kulcs tábla [40-43]-as oszlopait. Ezt a lépést semlegesíthetjük ennek a lépéseknek a megismétléssel, hiszen tudjuk, hogy ha XOR-al kétszer adunk hozzá egy kulcsot valamilyen értékhez, akkor visszakapjuk az eredeti értéket. A kódolás többi lépését is visszavonhatjuk, ha az egyes műveletek inverzeit alkalmazzuk, így az eljárás végén megkaphatjuk az eredeti üzenetet. Az alábbiakban bemutatjuk az egyes inverz műveleteket.

Inverz byte helyettesítés

Ebben a lépésben az eredeti S-doboz inverzét használjuk. A 4. táblázatban megfigyelhető, hogy az inverz S-doboz x. sorában és y. oszlopában olyan ij érték szerepel, hogy az S-doboz i. sorában és y. oszlopában xy található. Az inverz S-doboz előre eltárolhatjuk, vagy menet közben is kiszámíthatjuk az egyes byte-ok helyettesítő értékeit az alábbi módon:

1. Az S-doboz előállításánál használt affin-transzformáció inverzét alkalmazzuk. A B byte bitjeit a következő módon transzformáljuk:

$$\bar{b}_i = b_{(i+2)\bmod 8} \oplus b_{(i+5)\bmod 8} \oplus b_{(i+7)\bmod 8} + d_i$$

$$\begin{bmatrix} \bar{b}_0 \\ \bar{b}_1 \\ \bar{b}_2 \\ \bar{b}_3 \\ \bar{b}_4 \\ \bar{b}_5 \\ \bar{b}_6 \\ \bar{b}_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

2. A kapott \bar{B} értéknek megkeressük a multiplikatív inverzét, az eredmény lesz a B helyettesítő értéke.

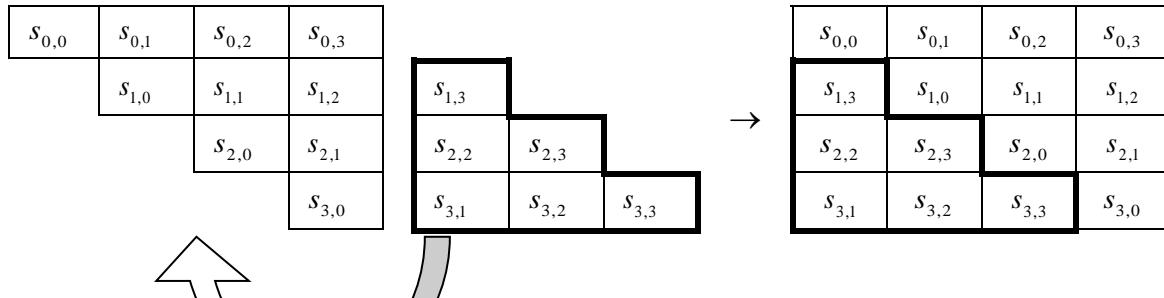
4. táblázat Az AES-ben használt inverz S-doboz

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Inverz sorforgatás

A kódolásnál balra forgattuk a sorokat, ennek az inverze pedig a jobbra forgatás, ahogy azt a 2. ábra láthatjuk. minden sorban annyi pozícióval kell forgatni, amennyivel a kódolásnál is, csupán az irány változik.

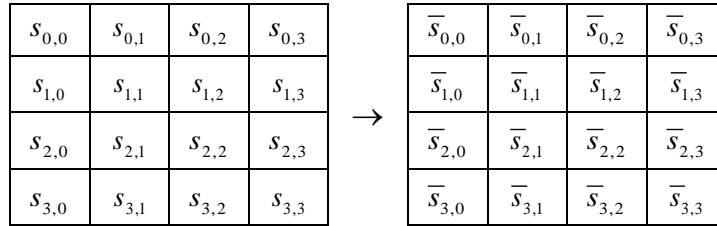
2. ábra Sorforgatás a dekódolásnál



Inverz oszlopkeverés

Ebben a lépésben minden egyes oszlopra elvégzünk egy transzformációt. Az eljárás működési elve hasonló a kódolás oszlopkeveréséhez, csupán az egyes szorzatok szorzói különböznek:

- $\bar{s}_{0,c} = (0x0E^\circ s_{0,c}) \oplus (0x0B^\circ s_{1,c}) \oplus (0x0D^\circ s_{2,c}) \oplus (0x09^\circ s_{3,c})$
- $\bar{s}_{1,c} = (0x09^\circ s_{0,c}) \oplus (0x0E^\circ s_{1,c}) \oplus (0x0B^\circ s_{2,c}) \oplus (0x0D^\circ s_{3,c})$
- $\bar{s}_{2,c} = (0x0D^\circ s_{0,c}) \oplus (0x09^\circ s_{1,c}) \oplus (0x0E^\circ s_{2,c}) \oplus (0x0B^\circ s_{3,c})$
- $\bar{s}_{3,c} = (0x0B^\circ s_{0,c}) \oplus (0x0D^\circ s_{1,c}) \oplus (0x09^\circ s_{2,c}) \oplus (0x0E^\circ s_{3,c})$



Dekódoló iterációk

Feltételezzük, hogy a bemenetről beolvastuk a dekódolandó üzenetet az állapot táblázatba. Innentől a dekódolás menete a következő:

1. Az állapothoz hozzáadjuk a kulcs táblázat [40,43] intervallumba eső oszlopait.
2. Ciklus $i := 9-től 1-ig$:
 - a. Inverz sorforgatás
 - b. Inverz byte-csere
 - c. Az állapothoz hozzáadjuk a kulcs táblázat [$i*4, (i+1)*4-1$] oszlopait
 - d. Inverz oszlopkeverés
3. Inverz sorforgatás
4. Inverz byte-csere

5. Az állapothoz hozzáadjuk a kulcs táblázat [0, 3] oszlopait

Az utolsó iterációban tehát elhagyjuk az oszlopok inverz keverését. Az eljárás kimenete pedig az állapot táblázat tartalma, ahol a byte-okat oszlop folytonosan kell kiolvasni.

II. MELLÉKLET

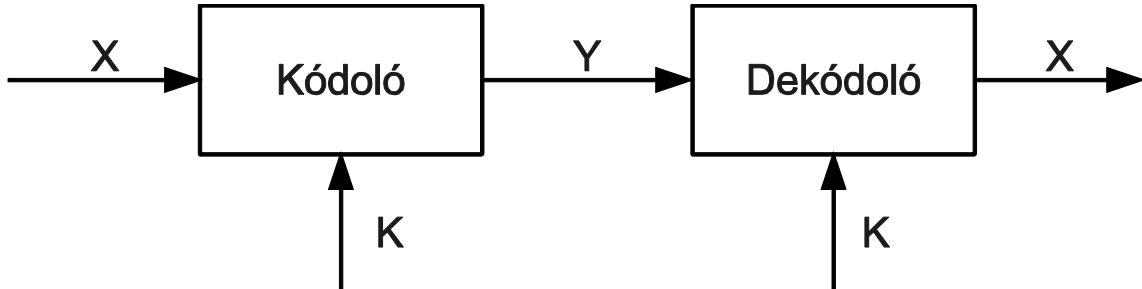
BLOKK KÓDOLÓK ÜZEMMÓDJAI

Elektronikus kódkönyv (ECB)

Az elektronikus kódkönyv üzemmód (Electronic Code Book, azaz ECB) a blokk kódolók legegyszerűbb felhasználási módja. A következő jelöléseket fogjuk használni:

- X : Kódolatlan szöveg
- Y : Kódolt szöveg
- K : Kulcs
- $Y = E_K(X)$, azaz E_k függvény végzi el a kódolást a K kulcs alapján.
- $X = D_K(Y)$, azaz D_k függvény végzi el a dekódolást a K kulcs alapján.

A kódolandó bemenetet azonos bithosszúságú blokkokra bontjuk, majd azokat egyenként kódoljuk. A kódolt blokkokat elküldjük a fogadó oldalra, ahol azokat dekódolják, és a beérkezett, majd megfejtett blokkokból összerakják a teljes üzenetet. A lényeg tehát az, hogy a blokkokat egymástól függetlenül kódoljuk, illetve dekódoljuk, ahogy azt a 3. ábra is mutatja.



3. ábra Elektronikus kódkönyv üzemmód működése

Ennek az üzemmódnak több előnye is van. Egynélzett ez a legegyszerűbb. Amennyiben nagy adatmennyiséget szeretnénk kódolni, úgy egyszerűen párhuzamosíthatunk. Végül, ha az üzenet átvitele során egy blokk sérül, akkor a dekódoláskor szintén csak egy blokkot kapunk vissza hibásan, a többi helyes marad. Itt azonban ki is merülnek az előnyök, lássuk, milyen problémák adódnak.

Tegyük fel, hogy Alice-nak és Bobnak van 1-1 saját bankjuk. A két bank időnként pénzt utal át egymásnak, mégízzá valamelyen számítógépes hálózaton keresztül. Egy átutalást a következő szerkezetű adatcsomag átküldésével bonyolítanak le:

1	2	3	4	5
Küldő bank azonosítója	Küldő bankszámlaszám	Fogadó bank azonosítója	Fogadó bankszámlaszám	Pénzösszeg

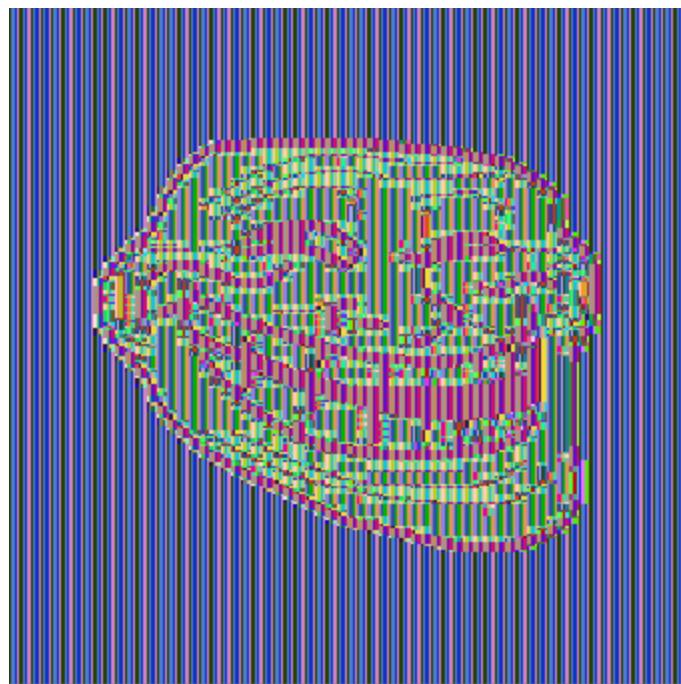
Természetesen ezt az adatcsomagot valamelyen blokk kódoló segítségével titkosítják, és úgy küldik el a hálózaton. Tegyük fel, hogy az adatcsomag egyes szeletei akkorák, amekkorák az általunk használt blokk kódoló által igényelt blokkméret. Továbbá a Alice és Bob bankjai csupán hetente változtatják meg a felhasznált kulcsot. Eve pedig ezt kihasználva úgy dönt, hogy megpróbál a rendszer gyengeségeit kihasználva gyorsan meggazdagodni. Az üzeneteket nem képes visszafejteni, viszont más módja is van célja elérésének. Eve a következőt fogja tenni: Először nyit magának 1-1 bankszámlát Alice-nál és Bobnál. A következő lépében csekély összegeket utal magának Alice bankjából Bob bankjába. Eközben figyeli, hogy a bankok között milyen adatcsomagok mozognak. Észreveszi, hogy van 5 blokk, amelyek ismétlődnek az utalásoknál. Lementi az 1. 3. és 4. blokkot, hogy ezeket később majd felhasználja. Sejti, hogy az 1. blokk jelöli a bank azonosítóját, a 3. tartalmazza azt a bankot, ahová a pénz megy, és végül a 4. blokk tárolja azt a számlaszámot, ahol a pénzt el kell helyezni. Ezek után figyeli, hogy a gyanútlan ügyfelek utalásai hatására minden adatcsomagok indulnak meg a hálózaton. Eve számára azok a csomagok érdekesek, amelyek arról szólnak, hogy Alice-éból Bob bankjába kell pénzt küldeni. Ezeket Eve könnyen kiszűrheti, ugyanis rendelkezik olyan kódolt blokkokkal, amely biztosan Alice, illetve Bob bankjának azonosítóját jelöli. Tehát amelyik elfogott csomag 1. és 3. blokkjában a megfelelő értéket látja, abban kicseréli a 4. blokkot arra, amitől korábban elmentett. Ugyanis a behelyettesített blokk Eve számlaszámának azonosítóját tartalmazza. A csere után tovább küldi az üzenetet, amely hatására Bob bankjában Eve számlájára íródik valamekkora pénzösszeg. Mivel a bankok között rendkívül gyakran utaznak ilyen adatcsomagok, Eve számláján hamar igen nagymennyiségű pénzösszeg halmozódik fel.

Egy másik példában fogtunk egy 32 bites színmélységű tömörítetlen bitképet, és a fejléc kivételével egy 128 bites kulcsot használó AES-el titkosítottuk a tartalmát. Az eredeti képet a 4. ábra, a kódolt változatot pedig az 5. ábra ábrázolja. Látható, hogy a titkosított képen még mindig jól felismerhető az eredeti ábra. A problémán az se segít, ha erősebb kulcsot használunk.

A bemutatott két probléma oka az, hogy az ECB üzemmód determinisztikus. Azaz ha egy adott blokkot többször egymás után kódolva ugyanazt az eredményt kapjuk. Azaz lényegében egy behelyettesítő kódolást kapunk, amelyről közismert, hogy rendkívül egyszerűen feltörhető. Az üzemmód innen is kapta a nevét: elektronikus kódkönyv. Régen gyakran használtak un. kódkönyveket, ahol minden szimbólumhoz leírtak 1-1 helyettesítő szimbólumot. A később ismertetésre kerülő üzemmódok ezt a gyengeséget védi ki más-más megközelítéssel.



4. ábra Az eredeti, kódolatlan kép

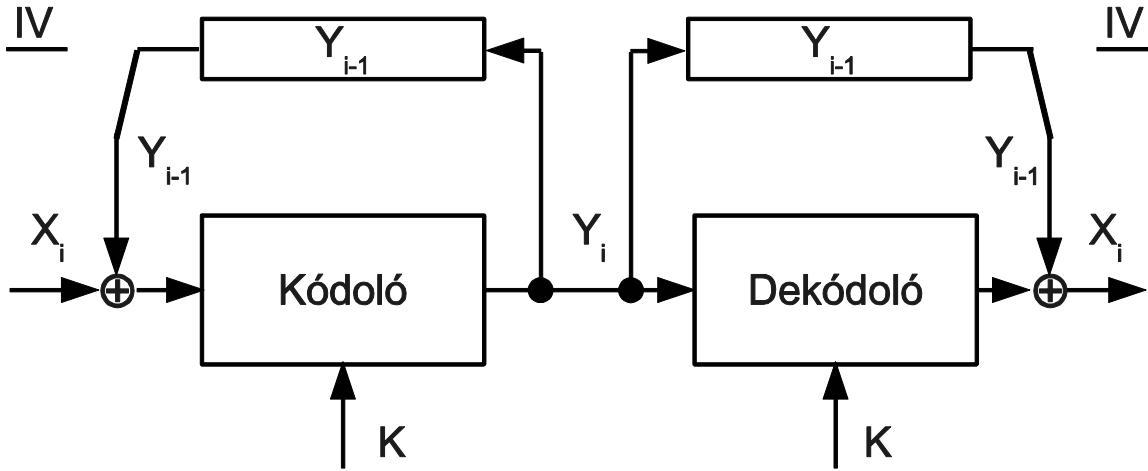


5. ábra 128 bites AES kulccsal, ECB módban titkosított kép

Titkosított blokkok láncolása (CBC)

Az ECB gyengeségének kijavítására alkalmazhatjuk azt a módszert, hogy a kódolás kimenetére nem csak a kódolandó blokk van hatással, hanem az előzőleg kódolt blokk is. Mivel az első kódolandó blokk előtt

nem volt semmi, ezért szükségünk van egy IV-vel jelölt inicializáló vektorra. Az üzemmód működését a 6. ábra láthatjuk.



6. ábra A CBC mód működése

Az első blokk kódolása előtt bitenkénti XOR művelettel hozzáadjuk az IV-t, majd az így módosított blokkot kódoljuk. Az eredményt tovább küldjük a fogadó oldalra, valamint a kódolás helyén betöljtük egy regiszterbe. A következő blokhoz már az ebbe a regiszterbe betöltött adatsort adjuk hozzá. A fogadó oldalon az első blokk fogadásakor elvégezzük a dekódolást, majd az eredményhez újra hozzáadjuk az IV-t. Az XOR tulajdonságainak köszönhetően ekkor visszakapjuk az eredeti blokkot. A fogadó oldalon a dekódolás előtt elmentjük a beérkezett titkosított blokkot egy regiszterbe, hogy azt felhasználhassuk a következő blokk dekódolásához. Tehát az egyes blokkokat egymásra láncoljuk, innen jön az üzemmód neve is: Cipher Block Chaining, azaz CBC. Fontos, hogy az inicializáló vektor azonos legyen minden oldalon. Formálisan a következőképpen néz ki a folyamat:

- Első blokk kódolása: $Y_1 = E_K(IV \oplus X_1)$
- Első blokk dekódolása: $X_1 = D_K(Y_1) \oplus IV$
- Következő (i .) blokkok kódolása: $Y_i = E_K(Y_{i-1} \oplus X_i)$
- Következő (i .) blokkok dekódolása: $X_i = D_K(Y_i) \oplus Y_{i-1}$

Gondoljuk végig, hogy az ECB-nél ismertetett bankos problémát megoldja-e a CBC. Amennyiben üzenetenként más IV vektort használnak, Eve nem képes mintákat felfedezni az üzenetekben. Továbbá, ha az egymás utáni üzeneteket folyamatosan egymás után láncolják, akkor szintén megoldódik a probléma. Tegyük fel azonban, hogy üzenetenként újra inicializáljuk a fogadó és küldő oldalt, azaz minden üzenet elején újra elővesszük ugyanazt az IV vektort. Ekkor Eve újra elfogja az egyes üzeneteket, majd kicséri a 4. blokkot és az üzenetet tovább küldi. A vevő fogadja az üzenetet és visszafejti azt. Eve pechére a fogadó bank rosszul dekódolja a 4. és 5. blokkot. Ugyanis a 4. blokkra hatással van az első három is, ahol a 2. blokk üzenetenként más és más. Mivel a fogadó oldalon tévesen dekódolják a 4. és 5.

blokkot, ezért a pénz nem fog megérkezni Eve számlájára. A beavatkozás észlelése egy másik probléma, ezzel részletesebben foglalkozunk majd a digitális aláírás témakörén belül.

Nézzük, hogy ha a korábban bemutatott képet CBC módban kódoljuk, akkor milyen ábrát kapunk. Az eredményt a 7. ábra láthatjuk. Ezen már felismerhetetlen az eredeti kép, úgy tűnik, mintha egy véletlenszerűen generált ábrát látnánk.



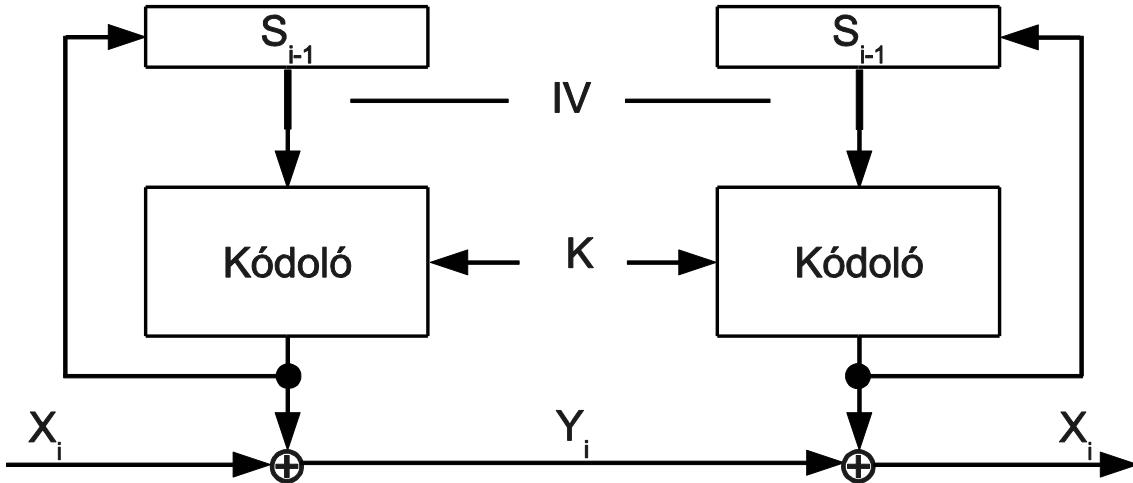
7. ábra 128 bites AES-el kódolt kép CBC üzemmódban

Látható tehát, hogy a CBC megszűnteti az ECB determinisztikusságát, így kivédi annak a gyengeségét. Az IV vektorban valamilyen módon meg kell egyezniük a kommunikáló feleknek, ezt akár szimmetrikus kódolóval is megtehetik. Az üzemmódnak van azonban egy komoly hátránya: Amennyiben nem garantálható, hogy minden blokk hibamentesen átjut a hálózaton, akkor ha csak egy blokknak egyetlen bitje hibás lesz, akkor a lavina hatás miatt az összes ezt követő blokk hibásan lesz dekódolva. A következő üzemmód ezt küszöböli ki.

Kimenet visszacsatolása (OFB)

Ennél az üzemmódnál szintén szükségünk van egy IV vektorra. Itt a kódoló nem kapja meg magát az X blokkot. Az első blokk elküldésekor a kódoló egy K kulccsal titkosítja az IV vektort. Az eredményt egyrészt elmenti egy S regiszterbe, másrészt XOR-al hozzáadja a titkosítandó X blokkhöz. Az eredményt pedig elküldi a fogadó oldalra, ahol szintén egy kódolóval és az IV vektorral előállítunk egy bitsorozatot, azt XOR-al hozzáadjuk a beérkező üzenethez, és megkapjuk az eredetit. A következő blokk elküldése előtt már nem az IV-t, hanem az S regiszter tartalmát kell kódolni. A küldő és fogadó oldalon tehát pontosan ugyanzt kell végrehajtani. Mivel itt a kódoló kimenetét csatoljuk vissza annak a

bemenetére, az üzemmódot Output Feedback-nek, azaz OFB-nek nevezzük. A módszer működésének vázlatát a 8. ábra láthatjuk.



8. ábra Az OFB mód működése

Formálisan a következő módon írhatjuk fel a módszer működését:

- Első blokk kódolása: $S_1 = E_K(IV)$, $Y_1 = X_1 \oplus S_1$
 - Első blokk dekódolása: $S_1 = E_K(IV)$, $X_1 = Y_1 \oplus S_1$
 - Következő (i . $.)$ blokkok kódolása: $S_i = E_K(S_{i-1})$, $Y_i = X_i \oplus S_i$
 - Következő (i . $.)$ blokkok dekódolása: $S_i = E_K(S_{i-1})$, $X_i = Y_i \oplus S_i$

A kódoló kimenete tehát teljesen független a titkosítandó szövegtől. A kódoló kimenetét kulcsfolyamnak is nevezhetjük, ezt akár előre is kiszámolhatjuk. Az átviteli hibákra ez a módszer a legkevésbé érzékeny, ugyanis ha a hálózaton átvitt üzenetben 1 bit sérül meg, akkor az eredményben is csak 1 bit hiba keletkezik. Fontos, hogy a vevő oldalon is kódolót kell alkalmazni a kulcsfolyam előállítására!

Azonban oda kell figyelnünk arra, hogy a kulcs folyamot ne ismételjük meg. Tegyük fel, hogy a P_1 és P_2 szöveget ugyanazzal a K kulcsfolyammal titkosítjuk. Ekkor a keletkezett kódolt üzenetek:

$$C_1 = P_1 \oplus K \text{ és } C_2 = P_2 \oplus K.$$

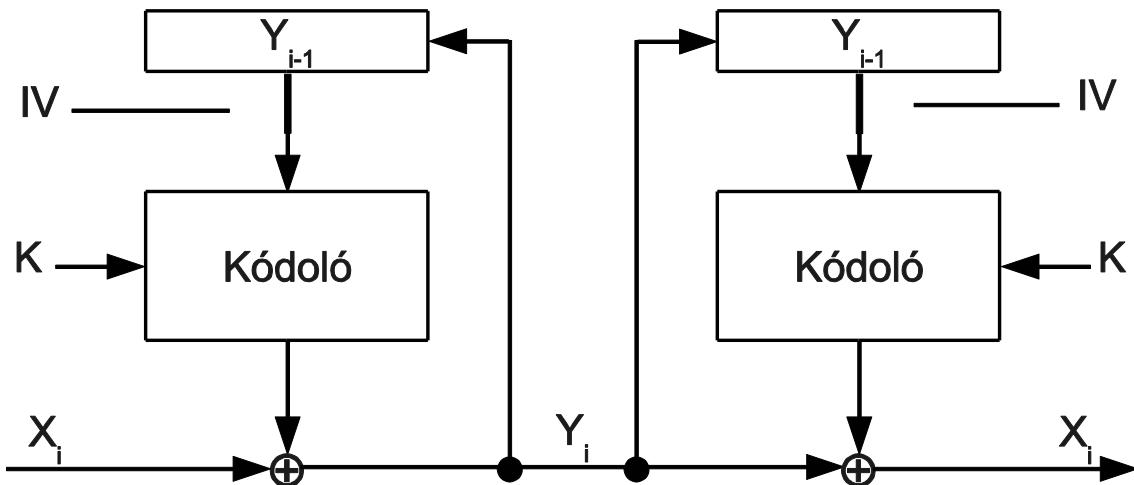
Eve elfogja C_1 -et és C_2 -t, majd XOR-al összeadja őket:

$$C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = P_1 \oplus P_2$$

Amennyiben Eve ismeri P_1 -et, vagy P_2 -t, úgy a másik egy XOR segítségével könnyen meghatározható. Ellenkező esetben pedig $P_1 \oplus P_2$ gyakoriságelemzéssel törhető, ugyanis az egyes szimbólumpároknak megfeleltethető egy helyettesítő szimbólum.

Kódolt üzenet visszacsatolása (CFB)

Az alábbi üzemmódban szintén kódolót használunk a küldő és fogadó oldalon. A módszer annyiban különbözik az OFB-től, hogy a kódolóra nem annak kimenetét, hanem magát a kódolt üzenetet csatoljuk vissza, ezért ennek a neve Cipher Feedback, azaz CFB.



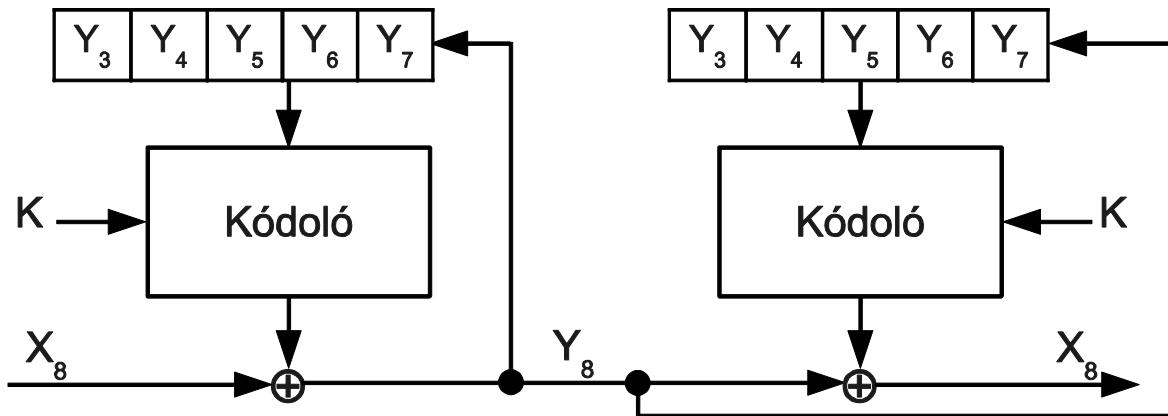
9. ábra A CFB mód működése

Formálisan a következőképpen néz ki az algoritmus:

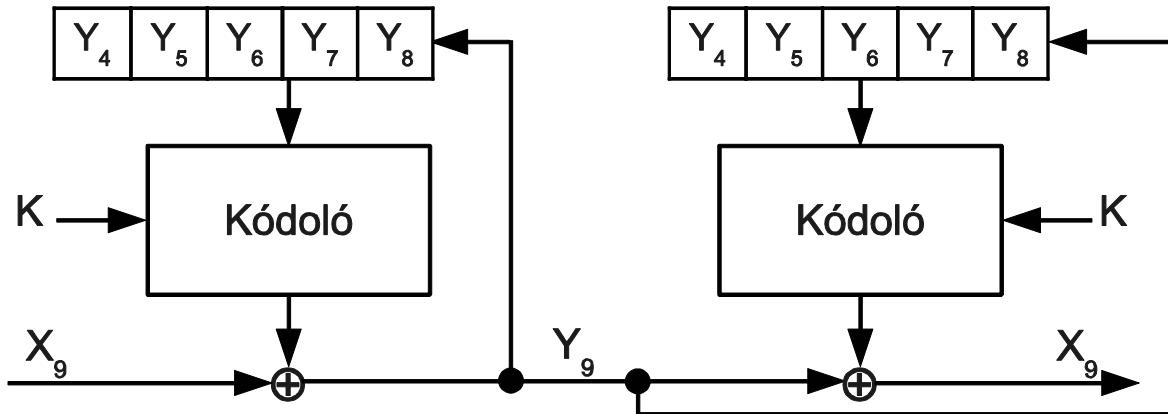
- Első blokk kódolása: $Y_1 = X_1 \oplus E_K(IV)$
- Első blokk kódolása: $X_1 = Y_1 \oplus E_K(IV)$
- Következő (i .) blokkok kódolása: $Y_i = X_i \oplus E_K(Y_{i-1})$
- Következő (i .) blokkok dekódolása: $X_i = Y_i \oplus E_K(Y_{i-1})$

Eddig látszólag ez a módszer nem nagy előrelépés az előzőekhez képest. Tegyük fel azonban, hogy olyan üzeneteket kell kódolni, amelyek jóval rövidebbek, mint a blokk kódolónk által igényelt blokk méret. Például ha a felhasználó gépel, akkor lehet, hogy a szoftvernek nincs ideje megvárni, míg összegyűlik a megfelelő mennyiségi karakter, hogy azokat majd egyben kódolva elküldje. Az egyik megoldás az lehet, hogy az egyes karaktereket kiegészítjük a megfelelő blokk méretre, majd elvégezzük a kódolást. Könnyen látható, hogy ekkor azonban pazarlóan bánunk a hálózat sávszélességével. A megoldás a CFB mód megfelelő használatában rejlik.

Tegyük fel, hogy a kódolók bemenetére egy shift-regisztert kötünk. A regiszterbe betölthetünk egy új byte-ot, ennek hatására a regiszterben lévő legrégebbi byte kiesik. A regiszter kezdetben az IV byte-jait tartalmazza. Az új üzenet küldésekor a shift-regisztert kódoljuk. Az elküldendő byte-ot XOR-al összeadjuk a kódoló kimenetének meghatározott indexű byte-jával, majd az eredmény byte-ot visszatöljtük a shift-regiszterbe, valamint elküldjük a hálózaton keresztül a fogadónak. A fogadó oldalon a küldőhöz hasonlóan előállítanak egy kulcsot, amelynek a megfelelő byte-jával XOR kapcsolatba hozzák a beérkező byte-ot, és megkapják a dekódolt byte-ot. Továbbá, a hálózaton érkező kódolt byte-ot betölzik a shift-regiszterbe. A 10. ábra és 11. ábra látható, ahogy az újabb elküldött, illetve fogadott byte-ok hatására a shift-regiszterek tartalma változik. A módszer hátránya is ebből fakad: Amennyiben az egyik byte átvitele során hiba lép fel, akkor amíg a hibás byte a fogadó oldal shift-regiszterében van, addig a beérkező byte-okat hibásan dekódolják.



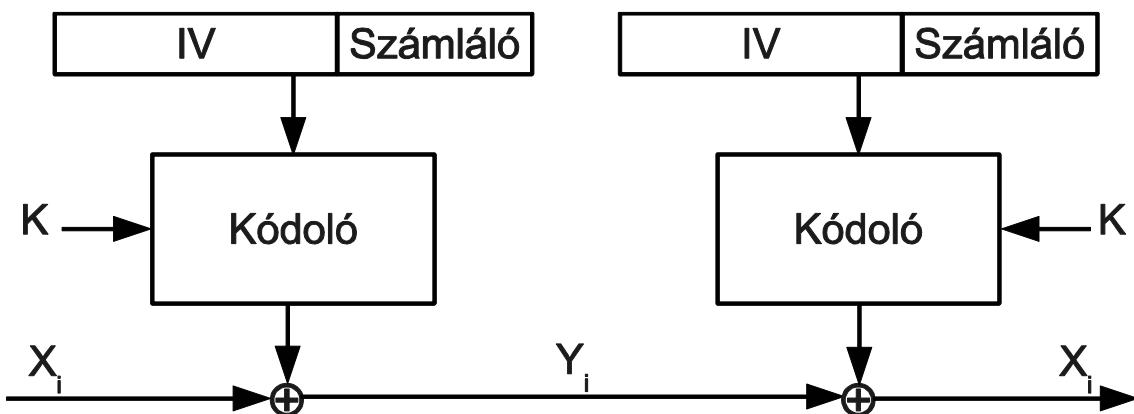
10. ábra A shift-regiszterek tartalma a 8. byte küldésekor



11. ábra A shift-regiszterek tartalma a 9. byte küldésekor

Számláló mód (CTR)

A CBC, OFB és CFB üzemmódok nagy hátránya az, hogy ha rendelkezünk egy nagy mennyiségű kódolt adathalmazzal, például egy merevlemez tartalmával, és szeretnénk ezen adathalmaz egy adott szakaszához hozzáférni, akkor az előtte lévő kódolt adatokat is fel kell dolgozni. Tehát mivel az egymás utáni blokkok szoros kapcsolatban állnak egymással, a direkt hozzáférés meglehetősen problémás lehet. Ennek a megoldására dolgozták ki a számláló módot (Counter mode, azaz CTR). Itt is kódolót alkalmaznak a küldő és fogadó oldalon. A kódolók bemenetét úgy kapjuk meg, hogy összefűzünk egy inicializáló vektort és egy számlálót. A számláló értéke kezdetben nulla. Az első blokk elküldésekor az IV-t és a számlálót összefűzve kódoljuk, majd az eredményt XOR-al hozzáadjuk a küldendő blokkhoz. A fogadó oldalon hasonlóképpen cselekszünk a kódolóval, majd annak kimenetével megfejtjük a hálózaton beérkezett titkosított üzenetet. Az üzenet elküldése illetve fogadása után minden két oldalon 1-el növeljük a számlálót. Lényegében újfent egy kulcsfolyamot állítunk elő, amelyre nincsenek hatással sem a múltban elküldött blokkok, sem a múltban generált kulcsfolyam-szeletek.



12. ábra A CTR mód működése

Formálisan a következő módon működik a CTR mód, ha CTR_e jelöli a küldő, CTR_d a fogadó számlálóját:

- Kezdetben $CTR_e = CTR_d$.
- Az i . blokk kódolása: $Y_i = X_i \oplus E_K(IV | CTR_e)$, majd CTR_e -t 1-el megnöveljük.
- Az i . blokk dekódolása: $X_i = Y_i \oplus E_K(IV | CTR_d)$, majd CTR_d -t 1-el megnöveljük.

Ez az üzemmód kifejezetten hasznos többek között merevlemezek titkosításakor. Ekkor a számláló jelölheti az aktuális merevlemez szektor sorszámát.

III. MELLÉKLET

AZ RSA ALGORITMUS

Az RSA szó a fejleztőinek neveiből származik: Ron Rivest, Adi Shamir és Len Adleman. Az algoritmust 1976-ban publikálták. 1997-ben nyilvánosságra hozták, hogy Clifford Cocks 1973-ban már kidolgozta az eljárást, a brit GCHQ (Government Communications Headquarters) nevű szervezetnél, ám több mint két évtizedig nem engedélyezték, hogy ezt publikálja.

Matematikai alapok

Az RSA számelméleti eredményeken alapul, ezért először ezeket ismertetjük. A bemutatásra kerülő számelméleti tételek helyességének bizonyításától eltekintünk.

Legnagyobb közös osztó

Definíció: Egy A és B egész szám legnagyobb közös osztójának nevezük azt a legnagyobb egész számot, amely A-nak és B-nek is osztója.

A jegyzetben A és B legnagyobb közös osztóját $\text{lko}(A, B)$ -vel jelöljük. Kiszámítására több módszer is van, próbáljuk meghatározni $\text{lko}(84, 30)$ -at. Bontsuk a két számot prímtényezőkre:

- $84 = 2 * 2 * 3 * 7$ és
- $30 = 2 * 3 * 5$

A két szám osztói közül a 2 és 3 a közös, tehát $\text{lko}(84, 30) = 2 * 3 = 6$. Kis számoknak nem okoz nehézséget az előbbi módszer, ám nagy számoknál ez az algoritmus használhatatlanná válik. Szerencsére létezik ennél jóval gyorsabb módszer is, mégpedig az euklideszi algoritmus. A bemenet az A és B egész számok, tegyük fel, hogy $A > B$.

1. $R := A \bmod B$
2. Ha $R = 0$ akkor végeztünk, a legnagyobb közös osztó := B
3. $A := B$
4. $B := R$
5. Vissza az 1. pontra

A fenti algoritmusban a mod a maradékos osztást jelöli. Az $\text{lko}(84, 30)$ a következő módon számolható ezzel az algoritmussal:

Lépés	A	B	R
1	84	30	24
2	30	24	6
3	24	6	0

Multiplikatív inverz

Az RSA-ban moduláris aritmetikát használunk, ezért definiáljuk a multiplikatív inverzet:

Definíció: Egy A egész szám multiplikatív inverze az a B szám modulo M-ben, amire igaz, hogy:

$$AB \equiv 1 \pmod{M}$$

Például legyen $A = 23$, $M = 120$. Ekkor, ha B-t 47-nek választjuk, akkor

$$AB = 1081$$

$$1081 \pmod{120} = 1$$

Tehát 23 multiplikatív inverze modulo 120 esetén 47. Mivel az RSA kulcsválasztási lépésekben multiplikatív inverzet is kell számolni, ezért szükségünk van egy hatékony algoritmusra ennek a meghatározásához. Erre alkalmas a kiterjesztett euklideszi algoritmus.

A kiterjesztett euklideszi algoritmus a következő egyenlet megoldását teszi lehetővé:

$$ax + by = \text{lko}(a, b),$$

ahol a és b egészek adottak, x és y egészeket kell meghatározni. Tegyük fel, hogy x az a multiplikatív inverze modulo m-ben:

$$ax \equiv 1 \pmod{m}$$

Tehát m osztója $ax - 1$ -nek, az osztás eredményét jelöljük q-val:

$$ax - 1 = qm$$

Ezt átrendezve azt kapjuk, hogy

$$ax - qm = 1$$

Amennyiben $\text{lko}(a, m) = 1$, és a kiterjesztett euklideszi algoritmussal meghatározzuk x-et és q-t, akkor x lesz a keresett multiplikatív inverz. Az $\text{lko}(a, m) = 1$ feltétel az RSA-hoz elegendő, hiszen ott erre az esetre lesz csak szükségünk.

Ahogy a neve is utal rá, az algoritmus hasonlóan működik, mint az euklideszi algoritmus, csupán azt kell kiegészíteni néhány utasítással. A könnyebb tárgyalhatóság kedvéért határozzuk meg az $\text{lko}(a = 120, b = 23)$ -at:

Osztandó	Osztó	Hányados (q)	Maradék (r)
120	23	5	5
23	5	4	3
5	3	1	2
3	2	1	1
2	1	2	0

A maradékokat jelöljük r_i -vel. Az algoritmust az $ax + by = \text{lko}(a, b)$ egyenlet megoldására használjuk, tehát az ismeretleneket x-el és y-al jelöljük. Fejezzük ki az r_i -t a és b alapján:

$$r_i = ax_i + by_i$$

A táblázat alapján könnyen látszik, hogy a táblázat egy sorában lévő maradék értéke 1 sorral lejjebb szintén megjelenik, mint osztó, 2 sorral lejjebb, pedig mint osztandó. Tehát ha $i > 2$ akkor:

$$r_i = r_{i-2} - q_i r_{i-1}$$

Az előző kettő összefüggésből a következőt kapjuk:

$$r_i = (ax_{i-2} + by_{i-2}) - q_i(ax_{i-1} + by_{i-1})$$

Emeljük ki a -t és b -t:

$$r_i = a(x_{i-2} - q_i x_{i-1}) + b(y_{i-2} - q_i y_{i-1})$$

Az algoritmus során az ismeretlen x és y értékét fokozatosan megközelítjük. Tegyük fel, hogy

- $x_1 = 1, y_1 = 0$ és
- $x_2 = 0, y_2 = 1$

Valamint, ha $i > 2$, akkor

- $x_i = x_{i-2} - q_i x_{i-1}$
- $y_i = y_{i-2} - q_i y_{i-1}$

Definiáljuk továbbá r_1 -et és r_2 -t:

- $r_1 = ax_1 + by_1 = a$
- $r_2 = ax_2 + by_2 = b$

A fentiek alapján az algoritmus a következő:

1. $r_1 = a, r_2 = b$
2. $x_1 = 1, y_1 = 0, x_2 = 0, y_2 = 1$
3. $i := 3$
4. $q_i := r_{i-2} / r_{i-1}$
5. $r_i := r_{i-2} \text{ mod } r_{i-1}$
6. Ha $r_i = 1$ akkor vége
7. $x_i := x_{i-2} - q_i x_{i-1}$
8. $y_i := y_{i-2} - q_i y_{i-1}$

9. $i := i + 1$

10. Ugrás a 4. pontra

A fenti algoritmusban az euklideszi algoritmussal ellentétben akkor állunk meg, ha a maradék 1. Mivel nekünk az RSA miatt van szükségünk erre az algoritmusra, a fenti megközelítés elegendő lesz.

Az alábbi példában megkeressük x és y értékét, ha $a = 120$ és $b = 23$. Korábban meghatároztuk, hogy $\text{Inko}(120, 23) = 1$.

i	q_i	r_i	x_i	y_i	$r_i = ax_i + by_i$
1		120	1	0	$120 = 120 * \underline{1} + 23 * \underline{0}$
2		23	0	1	$23 = 120 * \underline{0} + 23 * \underline{1}$
3	5	5	1	-5	$5 = 120 * \underline{1} + 23 * \underline{(-5)}$
4	4	3	-4	21	$3 = 120 * \underline{(-4)} + 23 * \underline{21}$
5	1	2	5	-26	$2 = 120 * \underline{5} + 23 * \underline{(-26)}$
6	1	1	-9	47	$1 = 120 * \underline{(-9)} + 23 * \underline{47}$

Tehát $1 = 120 * (-9) + 23 * 47$, azaz $x = -9$ és $y = 47$. A korábbi megállapítások alapján az alábbi egyenleteket írhatjuk fel:

$$47 * 23 \equiv 1 \pmod{120}$$

$$120 * (-9) \equiv 26 * 38 \equiv 1 \pmod{47}$$

$$120 * (-9) \equiv 14 * 5 \equiv 1 \pmod{23}$$

Azaz 47 és 23 egymás multiplikatív inverzei modulo 120, vagy 26 és 38 egymás modulo inverzei modulo 47, továbbá ugyanez igaz 14-re és 5-re modulo 23-ban.

Euler-függvény

Az Euler-függvény definíciójához először szükségünk van a relatív prímek fogalmára:

Definíció: A és B egész számok relatív prímek, ha $\text{Inko}(A, B) = 1$.

Ez alapján az Euler-függvény definíciója:

$$\varphi(m) = |\{a : \text{Inko}(m, a) = 1 \text{ és } a < m\}|$$

Azaz az Euler-függvény egy adott m pozitív egész szám esetén megadja, hogy hány darab m-nél kisebb és m-el relatív prím létezik.

Számoljuk ki $\varphi(6)$ értékét!

- $\text{Inko}(6, 1) = 1 \checkmark$

- $\lnko(6, 2) = 2$
- $\lnko(6, 3) = 3$
- $\lnko(6, 4) = 2$
- $\lnko(6, 5) = 1 \checkmark$

Tehát $\varphi(6) = 2$. Bár egy számpárra az euklideszi algoritmussal hatékonyan tudjuk ellenőrizni a relatív prím definíciójának való megfelelést, egy nagy m számra $m-1$ ellenőrzésre van szükség, ami elfogadhatatlan mennyiségű számítást tesz szükségessé.

Amennyiben ismerjük m prímtényezős felbontását, úgy $\varphi(m)$ értékét hatékonyan meghatározhatjuk. Legyen

$$m = p_1^{e_1} p_2^{e_2} \cdots p_n^{e_n},$$

ahol p_i prímszámok. Ez alapján az Euler-függvény értékét így határozhatjuk meg:

$$\varphi(m) = \prod_{i=1}^n (p_i^{e_i} - p_i^{e_i-1})$$

Határozzuk meg $\varphi(240)$ értékét a fenti képlet alapján!

$$m = 16 * 15 = 2^4 * 3^1 * 5^1 = p_1^{e_1} p_2^{e_2} p_3^{e_3}, n = 3$$

$$\varphi(240) = (2^4 - 2^3)(3^1 - 3^0)(5^1 - 5^0) = 8 * 2 * 4 = 64$$

Ennek a módszernek előnye a hatékonyság, hátránya viszont az, hogy ismerni kell hozzá m prímtényezős felbontását. Nagy számoknál ehhez rengeteg számításra van szükség, viszont az RSA-nál ezt a tulajdonságot használjuk ki.

Vegyük észre, hogy ha m két prímszám, nevezetesen p és q szorzata, akkor

$$\varphi(m) = (p-1)(q-1)$$

A kis Fermat-tétel

A következő összefüggés Pierre de Fermat-tól származik, bár ő ezt csupán sejtésként fogalmazta meg. A kis jelzőt azért kapta, hogy megkülönböztessük a nagy Fermat-tételtől, amely hosszú évszázadokig, egészen 1994-ig csupán sejtést volt. A kis Fermat-sejtést Fermat 1636-ban fogalmazta meg, és Leibniz viszonylag hamar, 1683-ban közölte, hogy ő már ismert egy bizonyítást rá.

A téTEL a következőt mondja ki: Legyen p prímszám, továbbá a bármely egész szám, ekkor igaz az, hogy

$$a^p \equiv a \pmod{p}$$

Az RSA megértéséhez a fenti összefüggésnek egy másik változatára lesz szükségünk. Legyen p továbbra is prímszám, ám a olyan egész, amely p -hez relatív prím. Ekkor igaz a következő:

$$a^{p-1} \equiv 1 \pmod{p}$$

Legyen $a = 34$, $p = 37$. Ekkor

$$34^{37} = 462082935536608083712617840903502214718703588813721042944$$

$$4620829355366080-218672808090350221471870-706153575042944 \bmod 37 = 34$$

A tételek igaz akkor is, ha $p < a$, legyen $a = 34$ és $p = 7$.

$$34^7 = 52523350144$$

$$52523350144 \equiv 6 \equiv 34 \pmod{7}$$

Az RSA algoritmus

Az RSA azt használja ki, hogy két nagy prímszámot könnyen összeszorozhatunk, viszont pusztán a szorzat ismeretében jelenlegi tudásunk szerint nem tudjuk elfogadható időn belül visszafejteni a prímszámokat.

Az algoritmus előkészítő lépései mindenki generál magának privát és publikus kulcsokat, majd a publikus kulcsokat nyilvánosságra hozzák. A kulcsgenerálás lépései a következők:

1. Legyen p és q elegendően nagy prímszámok, és $p \neq q$
2. $N = pq$
3. $\varphi(N) = (p-1)(q-1)$
4. Legyen e olyan szám, amelyre igaz, hogy $\text{Inko}(e, \varphi(N)) = 1$ és $e \geq 3$
5. Legye d olyan szám, amelyre igaz, hogy $ed \equiv 1 \pmod{\varphi(N)}$
6. A nyilvános kulcs legyen: (N, e) , a privát kulcs pedig d

A nyilvános kulcsból e -t az euklideszi algoritmussal keressük meg. Ez a szám lehet kicsi is, így az is megfelelő, ha 3-tól kezdve egyesével teszteljük az egész számokat, hogy megfelelőek-e. Rendszerint 11-nél tovább ritkán kell elmenni. A d meghatározásához pedig a kiterjesztett euklideszi algoritmusra van szükség. Jelöljük x -el a kódolandó üzenetet, y -al pedig a kódolt változatát. Ekkor a kódolás a következő módon zajlik:

$$y = x^e \pmod{N}$$

A dekódolás menete pedig a következő:

$$x = y^d \pmod{N}$$

A támadónak szüksége van d értékére ahhoz, hogy az üzenetet meg tudja fejteni. Ehhez a nyilvános e mellett szüksége van $\varphi(N)$ értékére is, amit elvileg a szintén nyilvános N -nől meg lehet határozni. Korábban viszont láttuk, hogy egy megfelelően nagy szám esetén annak prímtényezős felbontása túl számításigényes, gyakorlatilag kivitelezhetetlen, így a támadó N alapján nem képes $\varphi(N)$ -t, és így d -t meghatározni.

Az alábbiakban bemutatjuk az RSA működését egy egyszerű példán.

1. Legyen $p = 73$ és $q = 151$
2. $N = pq = 11023$
3. $\varphi(N) = (p-1)(q-1) = 10800$
4. e legyen 11, mert $\text{lcm}(10800, 11) = 1$
5. d értéke 5891, mert $ed \equiv 1 \pmod{10800}$
6. A nyilvános kulcs tehát: $(11023, 11)$, míg a privát kulcs 5891
7. Titkosítsuk az $x = 17$ üzenetet!
8. $17^{11} \pmod{11023} = 1782$
9. Dekódolás: $1782^{5891} \pmod{11023} = 17$

Az ajánlásokban általában minimum 1024 bites kulcsméret szerepel, de katonai alkalmazásoknál 2048, vagy ennél nagyobb kulcsméretek is előfordulnak. A tapasztalatok azt mutatják, hogy az RSA, hasonlóan a többi nyilvános kulcsú kódoló eljáráshoz, nagyságrendekkel lassabb, mint a blokk-kódolók. Emiatt az RSA-t az üzenetváltásnak csupán bizonyos lépéseinél, például a kulcs megosztásnál alkalmazzák.

Az alábbiakban megmutatjuk, hogy az RSA helyesen működik, azaz bizonyítsuk az alábbi állítást:

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

Tudjuk, hogy d az e -nek mod $\varphi(N)$ -ben multiplikatív inverze, azaz

$$ed \equiv 1 \pmod{\varphi(N)}$$

Tehát ez azt jelenti, hogy ed egy olyan egész szám, amit $\varphi(N)$ -el elosztva 1-et kapunk maradékul, az osztás egész részét pedig jelöljük v -vel:

$$ed = v\varphi(N) + 1$$

Az eredeti állítást a következő módon írhatjuk át:

$$x^{v\varphi(N)+1} \equiv x \pmod{N}$$

Az állítás igazolásához először egy segédállítást kell belátnunk. Bizonyítsuk be, hogy tetszőleges x és s értékek, és tetszőleges u prímszám esetén igaz a következő:

$$x^{s(u-1)+1} \equiv x \pmod{u}$$

Két esetet kell megvizsgálnunk, az első az, mikor x -et nem osztja u , ekkor a kis Fermat-tétel segítségével az állítás belátható:

$$x^{u-1} \equiv 1 \pmod{u} \rightarrow (x^{u-1})^s \equiv 1 \pmod{u} \rightarrow x(x^{u-1})^s \equiv x \pmod{u}$$

A másik eset viszont az, mikor x -et osztja u . Ekkor u és x nyilván nem relatív prímek, tehát ekkor a kis Fermat-tételt nem alkalmazhatjuk, tehát az előző eset bizonyítása ide nem jó. Mivel x -et osztja u , ezért az osztás maradéka 0, azaz:

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

Tehát a segédállításunk igaz. Ezek után már minden adott az eredeti állítás bizonyításához. Ezt szeretnénk belátni:

$$x^{v\varphi(N)+1} \equiv x \pmod{N}$$

Tudjuk, hogy

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

ha u prím. Valamint

$$ed = v\varphi(N) + 1 = v(p-1)(q-1) + 1.$$

Legyen $s_1 = v(p-1)$, és $s_2 = v(q-1)$. Ezek alapján igazak a következők:

$$x^{s_1(q-1)+1} \equiv x \pmod{q}$$

$$x^{s_2(p-1)+1} \equiv x \pmod{p}$$

Vagyis q osztja $x^{s_1(q-1)+1} - x$ -et és p osztja $x^{s_2(p-1)+1} - x$ -et, tehát $pq = N$ osztja $x^{v\varphi(N)+1} - x$ -et. Ebből következik, hogy $x^{v\varphi(N)+1} \equiv x \pmod{N}$, ami pedig az eredeti állítással ekvivalens. \square

Ez alapján igaz az is, hogy $(x^d)^e \equiv x \pmod{N}$, erre majd az üzenethitelesítésnél lesz szükségünk.

Az RSA alkalmazása digitális aláírásra

Tegyük fel, hogy Bob az x üzenetet szeretné elküldeni Alice-nak. Bob privát kulcsa legyen $k_{pr} = d$, nyilvános kulcsa pedig $k_{pub} = (n, e)$. Bob eljuttatja (n, e) -t Alice-nak. Az üzenet elküldése előtt Bob kiszámolja x -re a digitális aláírást: $s \equiv x^d \pmod{n}$. Bob elküldi Alice-nak (x, s) -t. Alice fogadja az

üzenetet, és ellenőrzi a kapott aláírást, mégpedig úgy, hogy dekódolja s -t: $\bar{x} \equiv s^e \pmod{n}$. Amennyiben Alice azt tapasztalja, hogy $x = \bar{x}$, akkor biztos lehet vele, hogy az aláírást Bob privát kulcsával készítették, tehát az aláírás helyes. Mivel csak Bob ismeri d értékét, ezért más nem képes a megfelelő s -t előállítani, tehát biztos, hogy Bob küldte az üzenetet. Abban az esetben, ha Eve elfogja az (x, s) párost, megváltoztatja x -et, a megfelelő s -t is ki kellene számolnia, de erre nem képes, hiszen nem rendelkezik Bob privát kulcsával. Lássunk erre egy konkrét szám példát!

Legyen az elküldendő üzenet $x = 4$. Generálunk megfelelő RSA privát és publikus kulcsot, ahol a lépések a következők:

1. Legyen $p = 3$ és $q = 11$
2. $n = pg = 33$
3. $\varphi(n) = (3-1)(11-1) = 20$
4. Legyen $e = 3$
5. $d \equiv e^{-1} \equiv 7 \pmod{20}$

Bob elküldi Alice-nak a saját nyilvános kulcsát, mégpedig a $(33, 3)$ -at. Az üzenet elküldésekor pedig kiszámolja az $x = 4$ üzenet digitális aláírását:

$$s = x^d \equiv 4^7 \equiv 16 \pmod{33}$$

Bob ezek után elküldi $(x, s) = (4, 16)$ -ot Alice-nak. Alice ellenőrzi az üzenet hitelességét, kiszámolja \bar{x} -et:

$$\bar{x} = s^e \equiv 16^3 \equiv 4 \pmod{33}.$$

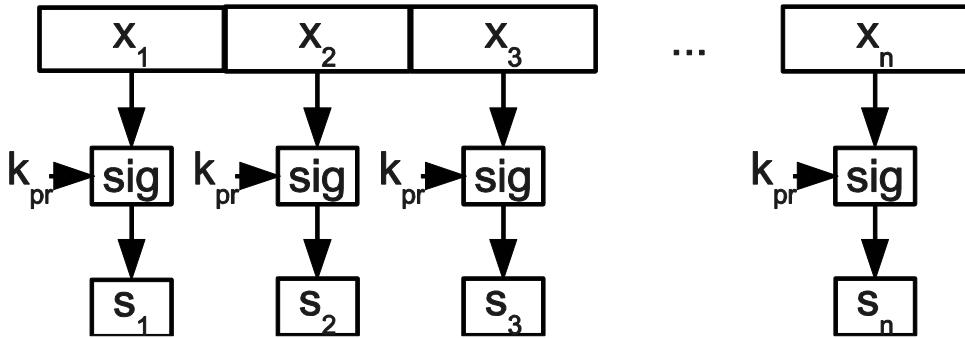
Mivel $x = \bar{x}$, ezért biztos, hogy az aláírás helyes.

Ez a módszer egyszerű, ám van egy hátránya: RSA esetében könnyen látható, hogy az x nem lehet nagyobb, mint n . Sajnos hasonló korlátozás érvényes más nyilvános kulcsú kódolóra is, ezért bonyolultabb aláírási módszert kell alkalmazni.

Bontsuk az x üzenetet n darab szeletre, ahol minden x_i ($1 \leq I \leq n$) szeletre igaz, hogy $x_i < n$. A fenti módszerrel állítsuk elő a privát kulcs segítségével minden x_i szelet aláírását:

$$s_i = x_i^d \pmod{n}.$$

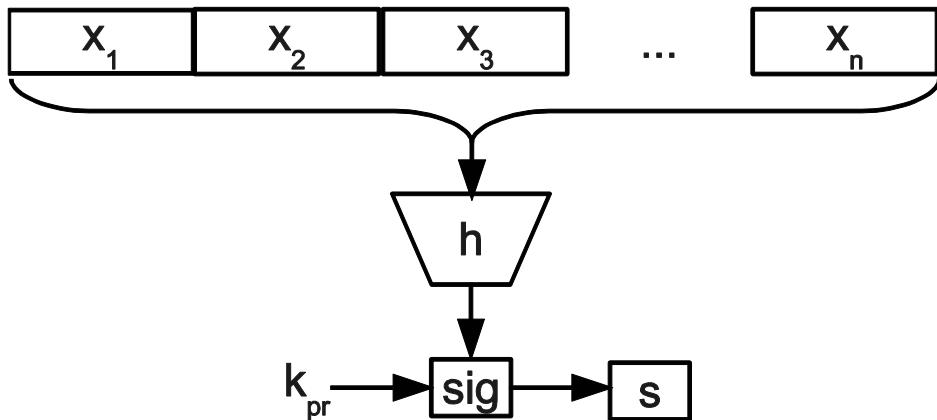
A kapott s_i -et pedig összefűzve megkapjuk a teljes x üzenet digitális aláírását.



13. ábra Naiv módszer a digitális aláírás előállítására

Az aláírást továbbá sem képes más reprodukálni. A módszer azonban egyszerűtlen, másrészt nem is biztonságos. A nyilvános kulcsú kódolók közismerten lassúak, tehát nagy üzenet esetén rendkívül sokáig tartana előállítani az egyes üzenetpecsét szekrényeket. A másik hátrány, hogy a kapott aláírás mérete nagyságrendileg akkora, mint az eredeti üzenet, tehát az elküldendő adatmennyisége közel megkétszereződik, ez pedig elfogadhatatlan tárolás, illetve hálózati átvitel szempontjából. A legsúlyosabb ellenérő pedig a módszer támadhatósága: Eve a nélküli átrendezheti a továbbítandó (x_i, s_i) párokat, hogy arról Alice vagy Bob tudomást szerezne.

A gyakorlatban ténylegesen alkalmazott megoldás az, hogy először számolunk ki egy rövid hash-t a teljes üzenetre, ezt később h -val jelöljük, majd csupán ez utóbbit kódoljuk nyilvános kulcsú kódolóval.



14. ábra A helyes aláírás generálás

Ez a módszer kijavítja az előző hibáit. A hash függvényt úgy kell megadni, hogy annak kimenetére alkalmazható legyen az nyilvános kulcsú kódoló. Mivel csak egy rövid bitsorozatra alkalmazzuk a kódolást, ezért a kapott aláírás mérete kicsi lesz, valamint az gyorsan is számolható. Továbbá a támadó nem tud mit átrendezni az elküldött (x, s) -ben anélkül, hogy ez a fogadónak fel ne tűnne. Lássuk

formálisan, hogyan néz ki a fenti módszer. Az elküldendő üzenet továbbra is x . Bob nyílt kulcsa: k_{pub} , a privát pedig k_{pr} .

1. Bob először kiszámolja az x -re a hash-t: $z = h(x)$
2. Bob z-ből kiszámolja a digitális aláírást: $s = \text{sig}_{k_{pr}}(z)$
3. Bob elküldi (x, s) -t Alice-nak, aki elvégzi az ellenőrzést. Ő is kiszámolja a hash függvény értékét: $\bar{z} = h(x)$. Végül ellenőrzi, hogy ha dekódolja s -t, akkor ugyanazt a hash értéket kapja-e, mint amit ő kiszámolt. Tehát, ha $\bar{z} = k_{pub}(s)$, akkor a digitális aláírás helyes, egyébként pedig helytelen. Most lássuk, hogy Eve hogyan támadhatja a digitális aláírást. Nyilván Eve, ahogy bárki más, Bobnak csak a nyilvános kulcsát ismeri, a privátat nem. Eve elfogja Bob üzenetét, és megváltoztatja az x üzenetet \bar{x} -re. Ekkor nyilván új digitális aláírást kell mellékelni a megváltoztatott üzenethez, ha ez sikerülne, akkor Alice nem venné észre, hogy átverés áldozata lett. Eve \bar{x} hash értékét nyilván ki tudja számolni: $\bar{z} = h(\bar{x})$. A következő lépés pedig az, hogy Bob privát kulcsával kódolja \bar{z} -t, ám erre nem képes, hiszen nem rendelkezik a megfelelő kulccsal.

IV. MELLÉKLET

A SZÜLETÉSNAP-TÁMADÁS MATEMATIKAI HÁTTERE

Az alábbi feladat a születésnap paradoxon nevet viseli. A kérdés az, hogy hány embernek kell lennie egy szobában ahhoz, hogy legalább 0.5 legyen annak a valószínűsége, hogy van két ember, akiknek azonos napon van a születésnapjuk? Tegyük fel, hogy nincsenek a szobában ikrek és szökőév sincsen. Jellemzően az első válasz, ami az esetek többségében elhangzik, az, hogy $365 / 2$ emberre van szükség, holott valójában elég csupán 23. A feladat maga tehát nem paradoxon, csupán azért alkalmazzák rá ezt a megjelölést, mert első hallásra talán furcsának hangzik a helyes megfejtés.

Ahhoz, hogy megértsük, miért elég 23 ember, vegyük észre, hogy a feladat nem azt kérdezi, hogy egy kiválasztott személyhez találunk-e olyan párt, akinek ugyanazon a napon van a születésnapja. Itt tetszőleges párokat keresünk. 23 személy esetén pedig $\binom{23}{2} = \frac{23 * 22}{2} = 253$ pár alkotható. Annak a valószínűsége, hogy mindenki másik napon született:

$$P(\text{mindenki másik napon született}) = \frac{364}{365} \frac{363}{365} \dots \frac{365 - n + 1}{365}$$

Ennek ellenkezőjének valószínűsége pedig:

$$P(\text{legalább két embernek azonos napon van a születésnapja}) = 1 - \frac{364}{365} \frac{363}{365} \dots \frac{365 - n + 1}{365}$$

Ez utóbbi valószínűség $n = 23$ esetben pedig 0.507. Ezt a gondolatmenetet alkalmazhatjuk az üzenetpecsétekre is. Határozzuk meg, hogy ha $\lambda = 0.5$ valószínűsséggel lehet találni két olyan hash függvény bemenetet, amelyekre a kimenet azonos, akkor egy n bites kimenettel rendelkező hash függvénynél mennyi próbálkozásra van szükség?

A fenti valószínűség képletet az alábbi módon írhatjuk át:

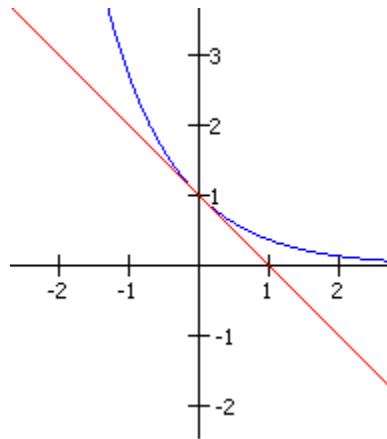
$$P(\text{nincs ütközés}) = \left(1 - \frac{1}{2^n}\right) \left(1 - \frac{2}{2^n}\right) \dots \left(1 - \frac{t-1}{2^n}\right) = \prod_{i=1}^{t-1} \left(1 - \frac{i}{2^n}\right)$$

Itt a t jelöli azt, hogy mennyiszer érdemes próbálkozni, míg megfelelő bemeneteket találunk. A folytatáshoz szükségünk lesz egy közelítő képletre. Az 15. ábra láthatjuk pirossal az $1 - x$, kékkel pedig az e^{-x} függvényt. Látható, hogy amennyiben x nagyon közel van az 1-hez, akkor

$$e^{-x} \approx 1 - x.$$

Mivel i/e^n rendkívül közel van 0-hoz, ezért a fenti közelítést alkalmazhatjuk a valószínűség kiszámítására:

$$P(\text{nincs ütközés}) \approx \prod_{i=1}^{t-1} e^{-\frac{i}{2^n}} \approx e^{-\frac{1+2+3+\dots+t-1}{2^n}}$$



15. ábra Az e^{-x} és az $1-x$ függvények

A képlet jobb oldalán látható számlálót helyettesíthetjük az alábbi módon:

$$1 + 2 + 3 + \dots + t - 1 = \frac{t(t-1)}{2}$$

Ezt felhasználva a valószínűség képletét a következő módon közelíthetjük:

$$P(\text{nincs ütközés}) \approx e^{-\frac{t(t-1)}{2^*2^n}} \approx e^{-\frac{t(t-1)}{2^{n+1}}}$$

Ahogy korábban is írtuk, λ jelöli annak a valószínűségét, hogy legalább egy ütköző bemenetpárt találunk:

$$\lambda \approx 1 - e^{-\frac{t(t-1)}{2^{n+1}}}$$

Mivel arra vagyunk kíváncsiak, hogy mennyi t próbálkozásra van szükség ahhoz, hogy λ valószínűséggel ütközést találunk, rendezzük a fenti egyenletet t -re:

$$\ln(1-\lambda) \approx -\frac{t(t-1)}{2^{n+1}}$$

$$t(t-1) \approx 2^{n+1} \ln\left(\frac{1}{1-\lambda}\right)$$

Mivel t egy elegendően nagy szám, ezért a fenti egyenlet bal oldalát így közelíthetjük:

$$t(t-1) \approx t^2$$

Ezek után az egyenlet a következőképpen rendezhető:

$$t \approx \sqrt{2^{n+1} \ln\left(\frac{1}{1-\lambda}\right)} \approx 2^{(n+1)/2} \sqrt{\ln\left(\frac{1}{1-\lambda}\right)}$$

A fenti képlet ismeretében határozzuk meg, hogy $n = 80$ bites hash függvény kimenetnél mennyire érdemes t -t, választani, azaz mennyi próbálkozásra van szükség ahhoz, hogy 0.5 valószínűséggel találunk ütközést:

$$t = 2^{81/2} \sqrt{\ln\left(\frac{1}{1-0.5}\right)} \approx 2^{40.2}$$

A fenti eredmény ismerete nélkül azt gondolnánk, hogy ha egy hash algoritmus kimenete 80 bites, akkor a támadónak 2^{79} üzenetet kell átnéznie, mire megfelelőt talál. Ám valójában elég 2^{40} darabot átnéznie, ez pedig a ma használatos számítógépek teljesítményét figyelembe véve rendkívül kicsi mennyiség, akár egy egyszerű laptop segítségével is találhatunk egyező kimenettel rendelkező bemeneteket.

V.MELLÉKLET

AZ SHA-1 ÜZENETPECSÉT

Alapműveletek

Mielőtt belemerülnénk az SHA-1 algoritmus részleteibe, bemutatjuk a felhasznált alapműveleteket.

Bitenkénti XOR

A művelet megegyezik az AES kódolónál ismertetettel, azaz a művelet két operandusa 1-1 bitsorozat, az eredmény bitsorozatban azon a helyiértéken van 1, ahol a két operandusban a bitek különböztek egymástól, formálisan, ha A és B bitsorozatok:

$$A \oplus B = C, \text{ ahol } C_i = 1 \text{ akkor és csak akkor, ha } A_i \neq B_i.$$

Példa: $01101 \oplus 01010 = 00111$.

Bitenkénti AND

A bitenkénti AND, vagy ÉS műveletben a kimenet i . bitje akkor és csak akkor 1, ha a bemenetek i . bitjei szintén 1-ek:

$$A \wedge B = C, \text{ ahol } C_i = 1 \text{ akkor és csak akkor, ha } A_i = B_i = 1.$$

Példa: $01101 \wedge 01010 = 01000$.

Bitenkénti OR

A bitenkénti AND, vagy ÉS műveletben a kimenet i . bitje akkor és csak akkor 1, ha a bemenetek i . bitjei közül legalább az egyik 1.

$$A \vee B = C, \text{ ahol } C_i = 1 \text{ akkor és csak akkor, ha } A_i = 1 \text{ vagy } B_i = 1.$$

Példa: $01101 \vee 01010 = 01111$.

Balra_forgatás

Adott egy x bitsorozat, ezt balra forgatjuk n pozícióval, és feltételezzük, hogy n kisebb, mint x hossza. A forgatás során n -el balra toljuk a biteket, és az eredeti sorozat n darab legmagasabb helyiértéken lévő bitet a sorozat végéhez illesztjük.

Példa: balra_forgatás(110010, 2) = 001011.

Összeadás mod 2^{32} -ben

Adottak A és B 32 bites előjel nélküli egész változók. Az összeadást az alábbi módon végezzük el:

$$A + B \equiv C \pmod{2^{32}}$$

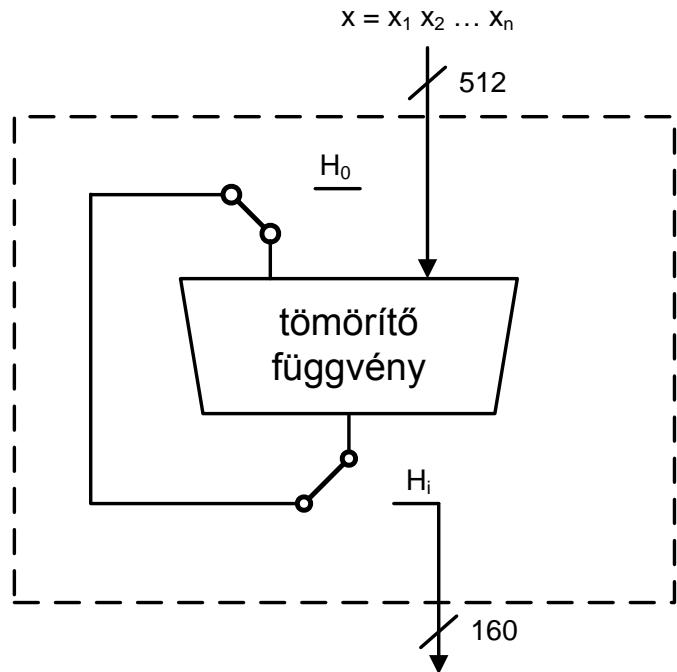
Példa: adjunk össze két számot, amelyeket hexadecimális formában írunk fel:

$$0xCA62C1D6 + 0x8F1BBCDC \equiv 0x597EEB2 \text{ mod } 0x100000000$$

Megjegyezzük, hogy ezt a műveletet C/C++-ban rendkívül egyszerűen implementálhatjuk, ugyanis ezen nyelvekben a 32 bites előjel nélküli típusok összeadás művelete pontosan így működik.

Az SHA-1 működése

A legtöbb hash függvény úgy működik, hogy valamilyen előkészítő lépés után azonos hosszúságú blokkokra bontja a bemenetet. Ezen blokkokon egyenként végrehajt valamilyen tömörítő eljárást, azaz az l bit hosszú blokkot leképezi egy rövidebb, m bites blokkra. A tömörítőnek kettő bemenete van, az egyik az előzőleg leképezett m bites blokk, a másik pedig a következő blokk. Miután elfogytak az l bites blokkokat, a tömörítő legutolsó kimenete lesz a hash függvény kimenete is, ezt a működési folyamatot reprezentálja a 16. ábra. Az SHA-1 függvény kimenete 160 bites, és a bemenetet 512 bit hosszúságú blokkokra bontja.



16. ábra Az SHA-1 működése

A fent vázolt algoritmus előtt azonban ki kell egészíteni a bemenetet, ugyanis nem biztos, hogy a bemenet hossza osztható 512-vel. Első lépésként a bemenetet ki kell egészíteni egy 64 bites információval, amely leírja, hogy az eredeti bemenet hány bit hosszú. Amennyiben szükséges, kiegészítő 0 biteket fűzünk a 64 bites információ elé, hogy kijöjenek az 512 bit hosszú blokkok. A lépések a következők:

1. Az x bitsorozat után egy darab 1-es bitet fűzünk.
2. Kiszámoljuk, hogy hány darab 0-val kell kibővíteni a sorozatot:

$$k = 512 - 64 - 1 - l = 448 - (l + 1) \text{ mod } 512$$

3. Az 1-es bit után fűzünk k darab 0-t az x bitsorozatban.
4. Végül az x-hez hozzáfűzzük a 64 bites előjel nélküli egész értéket, mégpedig úgy, hogy ez l-t, azaz x eredeti bithosszát tárolja. Fontos, hogy ez a 64 bites érték big endian kódolásban kerüljön a sorozat végére!

Ebből látszik, hogy az SHA-1 algoritmus legfeljebb $2^{64}-1$ bit hosszúságú adathalmazt képes feldolgozni. Lássunk egy példát a bemenet kiegészítésére: Legyen a bemenet a következő string: x = „abc”.

01100001 01100010 01100011
a b c

Az üzenet után egy 1-es bitet fűzünk, ekkor a következő sorozatot kapjuk:

01100001 01100010 01100011 1

Tudjuk, hogy l = 24, ezért k = 448-(24+1) mod 512 = 423 darab 0-t is a sorozat végére kell fűznünk:

01100001 01100010 01100011 1 0000...000
 423 db 0

Végül az l = 24-et egy 64 bites számként a sorozat végéhez illesztjük:

01100001 01100010 01100011 1 0000...000 0000...11000
 423 db 0 64 bites egész

A blokkok feldolgozása

Az alábbiakban bemutatjuk, hogy milyen transzformációkat végez a tömörítő függvény egy blokkon. Az egyszerűség kedvéért ezek után jelöljük x -el magát a blokkot. Az SHA-1 rendelkezik 5 darab 32 bites belső változóval, amelyek végül a kimenetet tárolják majd, ám ezeknek a legelső blokk feldolgozása előtt beállítunk bizonyos kezdőértékeket (a többi blokk előtt már nem):

- $H_0 = 0x67452301$
- $H_1 = 0xEFCDAB89$
- $H_2 = 0x98BADCCE$
- $H_3 = 0x10325476$
- $H_4 = 0xC3D2E1F0$

A blokk feldolgozása két főbb lépésből áll. Először fel kell töltenünk egy 80 elemű, w nevű tömböt, amely 32 bites egészeket tárol. A tömböt 0-tól kezdjük indexelni. Az x -et, vagyis a blokkot pedig 16 darab 32 bites szóra bontjuk: $x = x_0, x_1, \dots, x_{15}$. A w tömböt a következő módon töltjük ki:

- Ha $0 \leq i \leq 15$, akkor $w[i] = x_i$
- Egyébként $w[i] = \text{balra_forgatás}(w[i-3] \oplus w[i-8] \oplus w[i-14] \oplus w[i-16], 1)$.

A blokk feldolgozásához szükségünk lesz 5 darab 32 bites ideiglenes változóra is, ezeket jelöljük A, B, C, D, E-vel. A blokk feldolgozása előtt inicializáljuk ezeket a változókat:

1. $A = H_0$
2. $B = H_1$
3. $C = H_2$
4. $D = H_3$
5. $E = H_4$

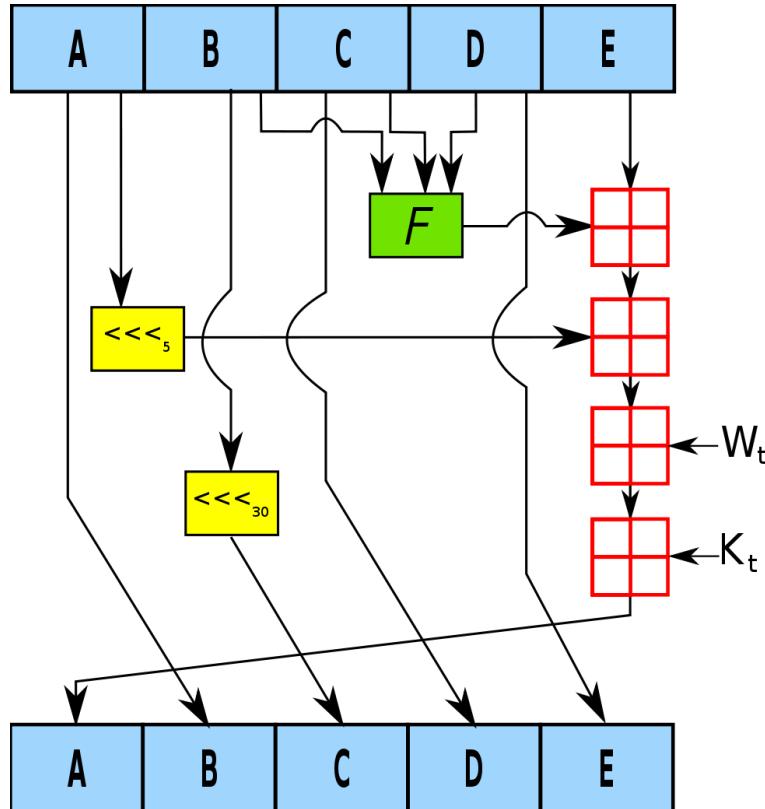
A tömörítő eljárás 4*20 iterációt hajt végre ezeken az ideiglenes változókon, majd ezen iterációk után elvégezzük az alábbi műveleteket:

1. $H_0 = A + H_0 \pmod{2^{32}}$
2. $H_1 = B + H_1 \pmod{2^{32}}$
3. $H_2 = C + H_2 \pmod{2^{32}}$
4. $H_3 = D + H_3 \pmod{2^{32}}$
5. $H_4 = E + H_4 \pmod{2^{32}}$

A következő blokk feldolgozásakor pedig ezen H_i értékekből indulunk ki. Végül lássuk, miből áll a 80 iteráció, amit az A, B, C, D és E változókon végre kell hajtani:

1. Ciklus $i := 0$ -tól 79-ig
 - a. TEMP = balra_forvatás(A, 5) + f(B, C, D) + E + K + w[i]
 - b. E = D
 - c. D = C
 - d. C = balra_forvatás(B, 30)
 - e. B = A
 - f. A = TEMP
2. Ciklus vége

Az alábbi ábra a fenti blokk-feldolgozás lépésein mutatja, az F-doboz a fenti f műveletet jelöli.



A fenti algoritmusban szereplő K konstans és f függvény az i ciklusváltozó függvénye, ezt tartalmazza az 5 táblázat.

5. táblázat Az SHA-1 tömörítő algoritmusa által használt konstansok és függvények

i	f	K
$0 \leq i \leq 19$	$f = D \oplus (B \wedge (C \oplus D))$	0x5A827999
$20 \leq i \leq 39$	$f = B \oplus C \oplus D$	0x6ED9EBA1
$40 \leq i \leq 59$	$f = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	0x8F1BBCDC
$60 \leq i \leq 79$	$f = B \oplus C \oplus D$	0xCA62C1D6

Az SHA-1 tehát képes 512 bitnél rövidebb bemenetre is kimenetet számolni. Jól ismert tesztmondat a következő: „The quick brown fox jumps over the lazy dog”. Az SHA-1 a következő hexadecimális kimenetet adja erre a mondatra: 2FD4E1C6 7A2D28FC ED849EE1 BB76E739 1B93EB12. Hasonlóan értelmezhető az üres string is mint bemenet, ennek a kimenete: DA39A3EE 5E6B4B0D 3255BFEF 95601890 AFD80709.

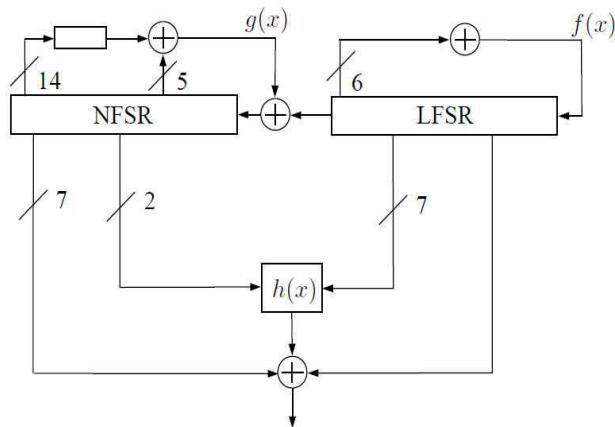
VI. MELLÉKLET

A GRAIN-128 FOLYAMTITKOSÍTÓ

Jellemzők:

- Svéd-svájci tervezés
- 128 bites titkos kulcs + 96 bites IV
- LFSR alapú
- Nagyon alacsony hardver igény
- Kevés plusz hardverrel gyorsítható, akár 32x
- „eddig” ellenáll a támadásnak
- <http://www.ecrypt.eu.org/stream/grainp3.html>

Áttekintés: egy lineárisan és egy nemlineárisan visszacsatolt léptető regiszter (LFSR és NFSR) kombinációja.



A visszacsatolások egyenletei:

$$s_{i+128} = s_i + s_{i+7} + s_{i+38} + s_{i+70} + s_{i+81} + s_{i+96}.$$

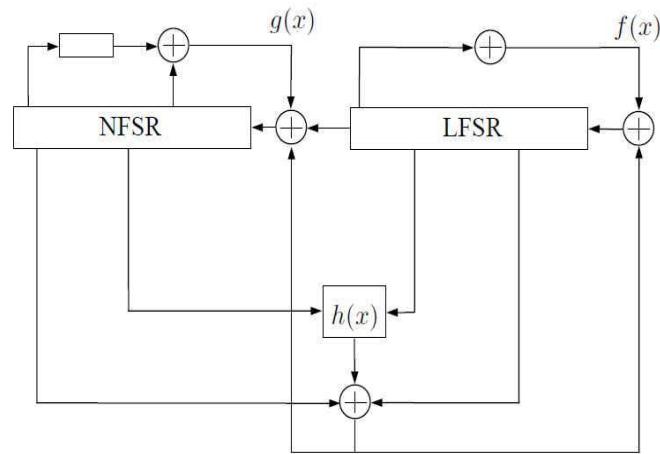
$$\begin{aligned} b_{i+128} &= s_i + b_i + b_{i+26} + b_{i+56} + b_{i+91} + b_{i+96} + \\ &+ b_{i+3}b_{i+67} + b_{i+11}b_{i+13} + b_{i+17}b_{i+18} + \\ &+ b_{i+27}b_{i+59} + b_{i+40}b_{i+48} + b_{i+61}b_{i+65} + \\ &+ b_{i+68}b_{i+84}. \end{aligned}$$

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8$$

$$z_i = \sum_{j \in A} b_{i+j} + h(x) + s_{i+93}$$

ahol $A = \{2, 15, 36, 45, 64, 73, 89\}$

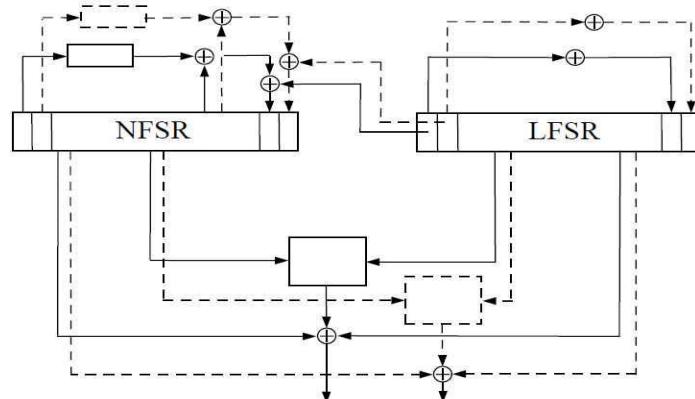
Inicializálás:



Az inicializálás lépései:

- Titkos kulcs (128 bit) >> NFSR
- IV (96 bit) >> LFSR alja (teteje 1-esekkel)
- 256 órajelig járatva

Gyorsítás a visszacsatolások többszörözésével lehetséges. A lényeg, hogy a legnagyobb területet elfoglaló léptetőregisztereket nem kell többszörözni:



A Grain hardver igénye (gate count) különböző gyorsítások esetén, ha a NAND2=1 kapu, NAND3=1.5 kapu, XOR2=2.5 kapu, a D-FF pedig 8 kapu:

<i>Building Block</i>	<i>Speed Increase</i>					
	1x	2x	4x	8x	16x	32x
LFSR	1024	1024	1024	1024	1024	1024
NFSR	1024	1024	1024	1024	1024	1024
$f(\cdot)$	12.5	25	50	100	200	400
$g(\cdot)$	37	74	148	296	592	1184
Output func	35.5	71	142	284	568	1136
Total	2133	2218	2388	2728	3408	4768

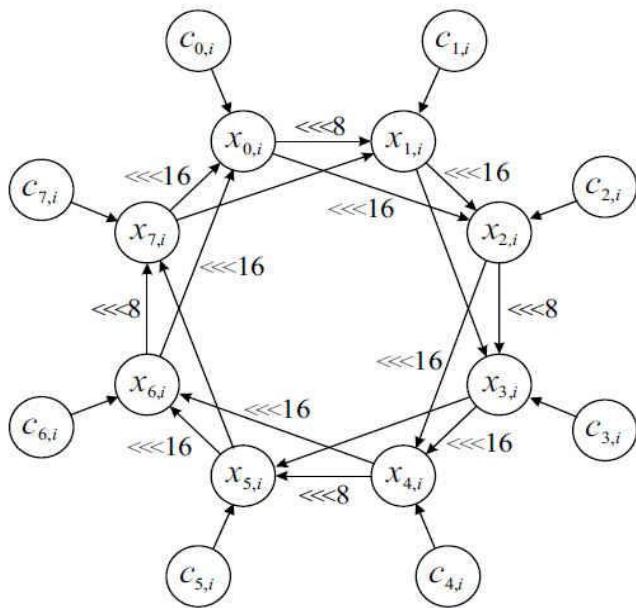
VII. MELLÉKLET

A RABBIT FOLYAMTITKOSÍTÓ

Jellemzők:

- Dán tervezés
- 128 bites titkos kulcs, 64 bites IV
- 8 db. 32 bites állapot regiszter, 8 db. 32 bites számláló és egy carry bit: összesen 513 bites state
- Az állapotváltozók nemlineárisan függnek egymástól (nincs S-doboz és LFSR!!)
- Gyors szoftver megvalósítás: byte-orientált, és az egész state elfér a regiszterekben
- „eddig” ellenáll a támadásnak
- http://www.ecrypt.eu.org/stream/p3ciphers/rabbit/rabbit_p3.pdf

Áttekintés:



Az $x_{0,i}$ a 0-dik állapotregisztert, a $c_{0,i}$ a 0-dik számlálót jelöli az i -edik ütemben. A $<<$ művelet a megadott számú rotálást jelenti.

Az állapotregiszterek számításának módja:

$$\begin{aligned}
 x_{0,i+1} &= g_{0,i} + (g_{7,i} \lll 16) + (g_{6,i} \lll 16) \\
 x_{1,i+1} &= g_{1,i} + (g_{0,i} \lll 8) + g_{7,i} \\
 x_{2,i+1} &= g_{2,i} + (g_{1,i} \lll 16) + (g_{0,i} \lll 16) \\
 x_{3,i+1} &= g_{3,i} + (g_{2,i} \lll 8) + g_{1,i} \\
 x_{4,i+1} &= g_{4,i} + (g_{3,i} \lll 16) + (g_{2,i} \lll 16) \\
 x_{5,i+1} &= g_{5,i} + (g_{4,i} \lll 8) + g_{3,i} \\
 x_{6,i+1} &= g_{6,i} + (g_{5,i} \lll 16) + (g_{4,i} \lll 16) \\
 x_{7,i+1} &= g_{7,i} + (g_{6,i} \lll 8) + g_{5,i} \\
 g_{j,i} &= ((x_{j,i} + c_{j,i+1})^2 \oplus ((x_{j,i} + c_{j,i+1})^2 \gg 32)) \bmod 2^{32}
 \end{aligned}$$

A \gg művelet a jobbra shiftelést jelenti a megadott számú pozícióval. Látható, hogy az alkalmazott nemlineáris művelet a négyzetre emelés. A hatékonyságot növeli, hogy a byte-os eltolások megvalósíthatók egyszerű byte-cserével a regiszteren belül.

A számlálók léptetésének egyenletei:

$$\begin{aligned}
 c_{0,i+1} &= c_{0,i} + a_0 + \phi_{7,i} \bmod 2^{32} & a_0 &= 0x4D34D34D & a_1 &= 0xD34D34D3 \\
 c_{1,i+1} &= c_{1,i} + a_1 + \phi_{0,i+1} \bmod 2^{32} & a_2 &= 0x34D34D34 & a_3 &= 0x4D34D34D \\
 c_{2,i+1} &= c_{2,i} + a_2 + \phi_{1,i+1} \bmod 2^{32} & a_4 &= 0xD34D34D3 & a_5 &= 0x34D34D34 \\
 c_{3,i+1} &= c_{3,i} + a_3 + \phi_{2,i+1} \bmod 2^{32} & a_6 &= 0x4D34D34D & a_7 &= 0xD34D34D3. \\
 c_{4,i+1} &= c_{4,i} + a_4 + \phi_{3,i+1} \bmod 2^{32} \\
 c_{5,i+1} &= c_{5,i} + a_5 + \phi_{4,i+1} \bmod 2^{32} \\
 c_{6,i+1} &= c_{6,i} + a_6 + \phi_{5,i+1} \bmod 2^{32} \\
 c_{7,i+1} &= c_{7,i} + a_7 + \phi_{6,i+1} \bmod 2^{32}
 \end{aligned}$$

, az a_i konstansok:

A carry bit:

$$\phi_{j,i+1} = \begin{cases} 1 & \text{if } c_{0,i} + a_0 + \phi_{7,i} \geq 2^{32} \wedge j = 0 \\ 1 & \text{if } c_{j,i} + a_j + \phi_{j-1,i+1} \geq 2^{32} \wedge j > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Az álvéletlen bitfolyamot 128 bites darabokban állítják elő az állapotregiszterekből, az alábbi egyenletekkel:

$$\begin{array}{ll} s_i^{[15..0]} = x_{0,i}^{[15..0]} \oplus x_{5,i}^{[31..16]} & s_i^{[31..16]} = x_{0,i}^{[31..16]} \oplus x_{3,i}^{[15..0]} \\ s_i^{[47..32]} = x_{2,i}^{[15..0]} \oplus x_{7,i}^{[31..16]} & s_i^{[63..48]} = x_{2,i}^{[31..16]} \oplus x_{5,i}^{[15..0]} \\ s_i^{[79..64]} = x_{4,i}^{[15..0]} \oplus x_{1,i}^{[31..16]} & s_i^{[95..80]} = x_{4,i}^{[31..16]} \oplus x_{7,i}^{[15..0]} \\ s_i^{[111..96]} = x_{6,i}^{[15..0]} \oplus x_{3,i}^{[31..16]} & s_i^{[127..112]} = x_{6,i}^{[31..16]} \oplus x_{1,i}^{[15..0]} \end{array}$$

A state inicializálásához az alábbi kezdő állapotot állítják be (K a 128 bites titkos kulcs, $k_0 = K^{[15..0]}$, $k_1 = K^{[31..16]}$, stb):

$$x_{j,0} = \begin{cases} k_{(j+1 \bmod 8)} \diamond k_j & \text{for } j \text{ even} \\ k_{(j+5 \bmod 8)} \diamond k_{(j+4 \bmod 8)} & \text{for } j \text{ odd} \end{cases}$$

$$c_{j,0} = \begin{cases} k_{(j+4 \bmod 8)} \diamond k_{(j+5 \bmod 8)} & \text{for } j \text{ even} \\ k_j \diamond k_{(j+1 \bmod 8)} & \text{for } j \text{ odd.} \end{cases}$$

Ezzez a kezdő állapottal 4 ciklusig járatják a titkosítót, majd a 64 bites IV-t is belekombinálják (XOR műveettel) a számlálók állapotába.

A Rabbit teljesítménye szoftver megvalósítás esetén, 8 KB adat titkosításakor az alábbi táblázatban látható, különféle (gyenge) processzorokra. Az első oszlop az órajelciklusok száma, a második a kód mérete byte-okban, a harmadik pedig a felhasznált memória, szintén byte-okban. A mezőkben lévő 3 érték a rejtesi iteráció / kulcsinitializálás / IV inicializálás adatát mutatja. Az 513 bites state egy 68 byte-os struktúrában van tárolva.

Processor	Performance	Code size	Memory
Pentium III	3.7/278/253	440/617/720	40/36/44
Pentium 4	5.1/486/648	698/516/762	16/36/28
PowerPC	3.8/405/298	440/512/444	72/72/72
ARM7	9.6/610/624	368/436/408	48/80/80
MIPS 4Kc	10.9/749/749	892/856/816	40/32/32

Célhardveres megvalósítás esetén 0.18 um-es CMOS technológiával 4100 kapuval valósítható meg a Rabbit, 0.048 mm^2 felületen, és ezzel 500 MBit/s működési sebesség érhető el. Ez a sebesség több hardver felhasználásával növelhető, a 32 bites négyzetre emelés pipeline-os gyorsításával (3 db 16×16 bites szorzás és egy összeadás). Az elérhető sebességek:

Pipelines	Gate count	Die area	Performance
1	28K	0.32 mm ²	3.7 GBit/s
2	35K	0.40 mm ²	6.2 GBit/s
4	57K	0.66 mm ²	9.3 GBit/s
8	100K	1.16 mm ²	12.4 GBit/s

2. rész: Hálózati biztonság

Tartalomjegyzék

1.	Behatolás-érzékelő és -megelőző rendszerek.....	5
	Mi az a behatolás-érzékelő rendszer?	5
	Mi az a behatolás-megelőző rendszer?	6
	Az IDS és az IPS kombinált használata	7
	Az IDS és IPS rendszerek típusai.....	7
	A HIPS és a NIPS összehasonlítása	9
	NIPS tulajdonságok	9
	Szignatúra alapú IDS és IPS	10
	Házirend alapú IDS és IPS.....	11
	Anomália alapú IDS és IPS.....	11
	Mézescsupor alapú rendszerek	12
2.	Virtuális magánhálózatok biztonsága	13
	Mi az a virtuális magánhálózat?	13
	VPN topológiák.....	14
	A biztonságos VPN sajátosságai	16
	A VPN biztonsága: az IPsec és a GRE.....	17
	Alagutazás telephelyek közötti VPN használatával.....	17
	Alagutazás távoli elérésű VPN használatával.....	18
	A hitelesítés kérdése	18
	Az IPsec biztonsági funkciói	18
	Az IPsec protokolljai	19
	Az IPsec fejlécei.....	19
	Az IKE protokoll.....	20
	Az ESP és az AH protokoll.....	21
	Az AH által biztosított hitelesség és sérzetlenség	23
	Az ESP protokoll	23
	A telephelyek közötti VPN működése.....	24
	A telephelyek közötti IPsec VPN beállítása	26

A GRE protokoll.....	27
Biztonságos GRE alagutak	28
A GRE alagutak biztonságossá tétele az IPsec segítségével.....	28
GRE az IPsec fölött	28
OpenVPN – alternatíva virtuális magánhálózat megvalósításra.....	28
3. Hálózatbiztonsági architektúrák (terheléselosztás, azonnali helyreállítás)	29
A tűzfal célja	29
Megvalósítás	29
Hardveres megvalósítás	29
Szoftveres megvalósítás.....	30
Tűzfalak csoportosítása.....	31
Csomagszűrő tűzfalak	31
Nem állapottartó működés	31
Állapottartó működés	31
Proxy tűzfalak.....	32
Helyreállítás áramszünet után	32
Tűzfal-architektúrák	33
Egyedülálló tűzfal	33
Kettős (szendvics) tűzfal.....	34
Tűzfalak tartalékolása, hibatűrő elosztott tűzfalak.....	34
A tűzfalak ellenőrzése	35
A hálózati forgalmi adatok és a hálózati hozzáférés auditálása	36
Netflow a hálózati audit segítésére	36
4. Vezeték nélküli hálózatok biztonsága	38
Betelekintés a vezeték nélküli technológia alapjaiba	38
Kik kommunikálnak?	38
Rádióhullámok és az interferencia.....	38
Topológiák.....	39
Pehelysúlyú hozzáférési pont	41
A kontroller	41
Roaming, az állomások vándorlása	42
A vezeték nélküli hálózat forgalmának titkosítása.....	42

WEP	42
WPA és WPA2	43
Előre kiosztott kulcsok, WPA-PSK mód	44
WPA-802.1x mód	44
Mit tanácsos, és mi nem tanácsos?	44
5. Irodalomjegyzék.....	44

Bevezetés

A számítógépes hálózatok óriási fejlődésen mentek keresztül az elmúlt években, évtizedekben. Nem csupán a hálózatok fizikai alkotóelemei fejlődtek az egyre modernebb gyártástechnológiának köszönhetően, hanem az eszközök együttműködését lehetővé tevő és szabályozó protokollok is. Ennek eredményeképpen ma már nincs az életnek olyan szegmense, ahol eredményesen, hatékonyan lehetne boldogulni megfelelő hálózati támogatás nélkül. Az egyik vezető hálózati eszközöket gyártó cég vezetője foglalta össze találóan, hogy a hálózat megváltoztatta mindenzt, ahogy dolgozunk, élünk, játszunk és tanulunk. A változás különösen szembetűnő a webes felületeken, illetve a közösségi oldalakon, hisz a fiatal generáció tagjai ennek segítségével beszélük meg az iskolai felkészülést, szervezik meg a közös programokat, majd ide töltik fel az ott készült fotókat és videókat. Jelenleg is óriási mennyiségű adat halad az információs szupersztrádán, és ez a mennyiség a videokommunikáció folyamatos előretörésével exponenciálisan emelkedik. 2015-re várhatóan a hálózati forgalom 91%-át videoanyagok továbbítása teszi ki, továbbá kb. 30 millió munkavállaló legalább heti egy napot otthonról fog dolgozni.

A hálózati infrastruktúra általánosságban készen áll a megnövekedett adatforgalom kiszolgálására, de mi a helyzet a hálózat és az azon keresztülhaladó adatok védelmével? Szerencsére egyre nagyobb hangsúlyt kap a megfelelő hálózatbiztonság kialakítása kis- és nagyvállalatoknál egyaránt. Ennek eredményeképpen használunk jogosultságkezelést, tűzfalat, demilitarizált zónát (DMZ), behatolás-érzékelő (IDS) vagy -megelőző (IPS) rendszereket.

A számítógépes bűnözés fejlődésével viszont a hálózat elleni támadások is egyre kifinomultabbá váltak. Már nem feltétlenül igényelnek felhasználói interakciót, inkább a hálózat alsóbb rétegeit célozzák. Az itt működő irányító- és szállítási protokollok felelnek az összes áthaladó adat megfelelő irányba történő, legjobb szándékú továbbításáért. Ha egy támadás sikerrel jár, akkor az összes áthaladó információ eltéríthető.

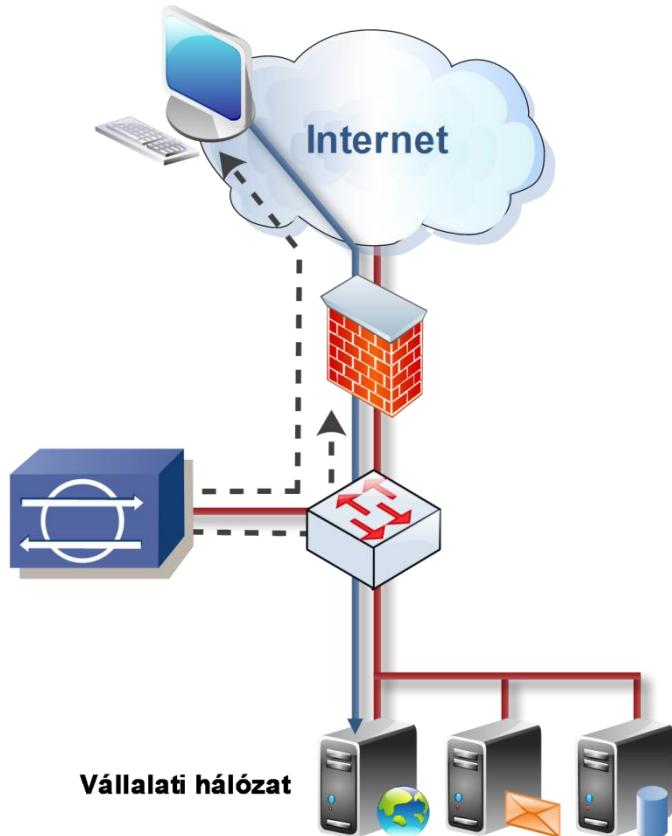
Az alábbi alfejezetek a hálózati biztonság négy kiemelkedő területével foglalkoznak. Az 7. fejezet a behatolás-érzékelő és -megelőző rendszerekkel foglalkozik. A 8. fejezet a VPN technológiákat tekinti át. A 9. fejezet a hálózatbiztonsági architektúrákat veszi számba, különös tekintettel a tűzfal megoldásokra. A 10. fejezetben pedig a napjainkban igencsak népszerű vezeték nélküli hálózatok biztonsági kérdéseit taglalja. A lista természetesen nem teljes körű, de ez terjedelmi korlátok miatt nem is lehetett cél. Az olvasóról feltételezzük, hogy bizonyos szintű informatikai és hálózati ismeretekkel rendelkezik. Az itt esetlegesen nem részletezett hálózati alapfogalmak leírása megtalálható az [1]-ben.

7. Behatolás-érzékelő és -megelőző rendszerek

Mi az a behatolás-érzékelő rendszer?

A behatolás-érzékelő rendszer (IDS, Intrusion Detection System) olyan hardveres, illetve szoftveres megoldás, amely a hálózati forgalomba történő beavatkozás nélkül, válogatás nélkül figyeli az áthaladó csomagokat. A részletes definíció, sok más hasznos információval együtt megtalálható a [2]-ben és a [3]-ban. Passzív működési jellege miatt az IDS csak korlátozott válaszadási képességekkel rendelkezik. Ilyen például, ha rosszindulatú forgalmat érzékel, akkor egy figyelmeztetést küld a hozzá kapcsolódó

felügyeleti állomásnak. Habár adott esetben arra is biztosít lehetőséget, hogy megakadályozza további rosszindulatú forgalom áthaladását a hálózati eszközök (pl.: útválasztók) beállításainak aktív megváltoztatásával, a riasztás kiváltó rosszindulatú forgalom addigra már áthaladt a hálózaton, akár el is érhette tervezett célját. Problémás TCP-kapcsolat esetében az IDS küldhet TCP-reset kérést a végállomásnak, amely ezáltal bontja a TCP-kapcsolatot. A behatolás-érzékelő rendszer általános működése az 1. ábrán látható.

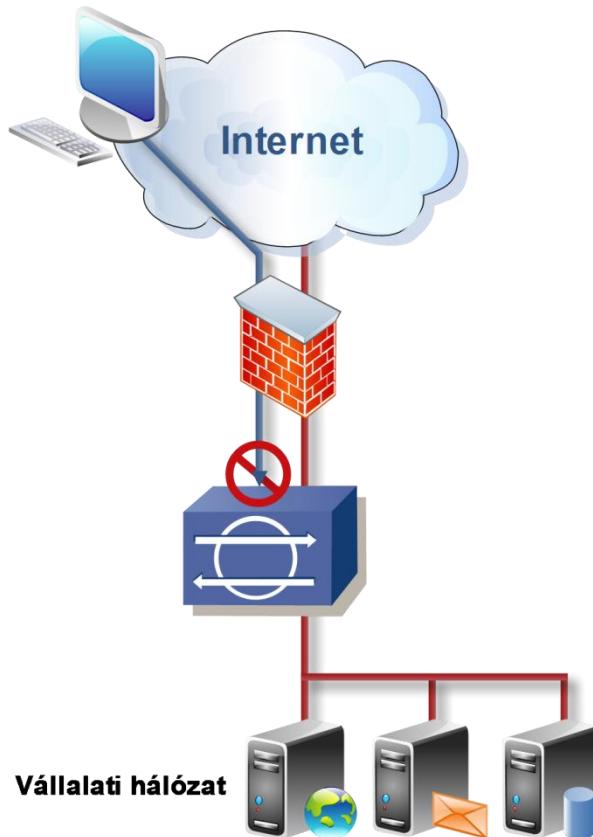


1. ábra: Általános behatolás-érzékelő rendszer

Mi az a behatolás-megelőző rendszer?

A behatolás-megelőző rendszer (IPS, Intrusion Prevention System) olyan aktív, a hálózat forgalmát áteresztő és vizsgáló eszköz, amely a hálózatba érkező csomagokat átengedi vagy eldobja. Elhelyezésénél ügyelni kell arra, hogy lehetőség szerint minden forgalom áthaladjon az IPS-en.

Amennyiben az IPS rosszindulatú forgalmat érzékel, lehetősége van az azonnali beavatkozásra. Riasztást küld a hozzá kapcsolódó felügyeleti állomásnak, majd a beállítások megváltoztatásával azonnal blokkolja a rosszindulatú forgalmat. Működése ezáltal proaktív, mivel a riasztást kiváltó, majd az ezt követő forgalmat egyaránt képes blokkolni. Működése a 2. ábrán látható.



2. ábra: Általános behatolás-megelőző rendszer

Az egyre kifinomultabb támadási módszerek ezt meg is kívánják, a kártékony kódok, illetve a sebezhetőségek elleni harcban.

Az IDS és az IPS kombinált használata

A látszattal ellentétben az IDS és az IPS egymással jól megférő technológiák, ezért nem ritka, hogy vállalati környezetben mindenkorral párhuzamosan alkalmazzák. Az IPS aktív működése révén blokkolja a nemkívánatos forgalmat, ezáltal egyfajta túzfal-rendszernek is tekinthető. Ezért úgy érdemes beállítani, hogy kizárolag az ismert rosszindulatú forgalmat szűrje ki a kapcsolódási problémák elkerülése érdekében. Az IDS ezáltal ellenőri a rosszindulatú forgalmat, ugyanakkor riasztást küldhet a „szürke zónába” sorolható forgalomról. Ide soroljuk minden adatot, amelyek nem egyértelműen rosszindulatúak, de legitimnek sem tekinthetők. Ha az IPS blokkolja az ilyen típusú forgalmat, fennáll a kockázata, hogy a szabályos forgalmat is megszakítja. Rosszindulatú forgalom esetében viszont az IDS által küldött riasztás értékes információt szolgáltat a lehetséges problémákkal, illetve a támadási módszerrel kapcsolatban.

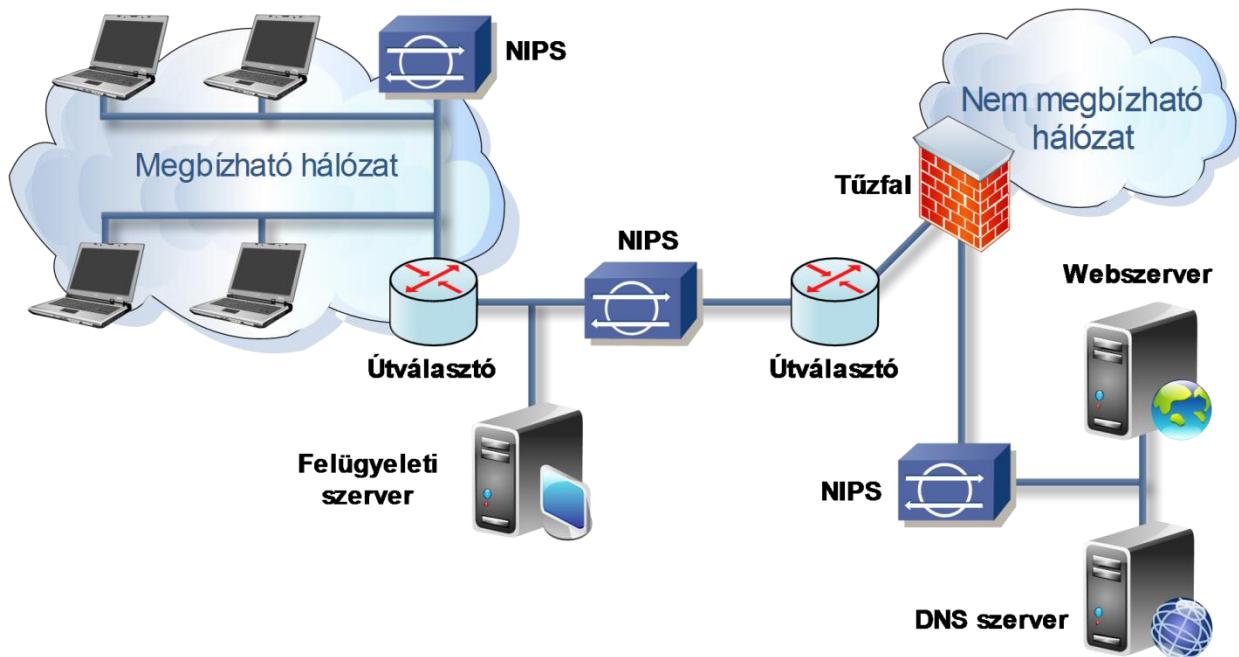
Az IDS és IPS rendszerek típusai

Az IDS és IPS megoldások típusainak csoportosítása a hálózatban elfoglalt helyük, valamint a rosszindulatú forgalom azonosítására használt módszerük alapján lehetséges. Az előbbi alapján beszélhetünk hálózat alapú, illetve állomás alapú rendszerekről, az utóbbi alapján pedig négy technikát különböztetünk meg:

- szignatúra alapú
- házirend (policy) alapú
- anomália alapú
- „mezescsupor” alapú

Az állomás alapú IPS (HIPS) minden egyes számítógép és állomás tevékenységét külön vizsgálja. Teljes hozzáféréssel rendelkezik a végberendezés belső adataihoz, ezáltal a bejövő forgalmat az állomás tevékenységeinek viszonylatában vizsgálja. VPN környezetben, ahol az adat titkosítva halad át a hálózaton, a HIPS az egyetlen módja, hogy a célállomáson a valódi forgalmat megvizsgáljuk. Hátránya, hogy jellemzően egy adott operációs rendszert támogat, és nem védi az alacsonyabb szintű – az OSI rétegmodell első és harmadik rétegét érintő – támadások ellen. További hátránya, hogy kellő felderítés után a támadó tudni fog az állomás létezéséről, sőt arra is rájöhet, hogy az állomást HIPS védi.

A hálózat alapú IPS (NIPS) a hálózaton áthaladó minden egyes csomagot analizál, ezáltal olyan rosszindulatú csomagok felismerésére is alkalmas, amelyek egy tűzfal egyszerű szűrési szabályain átjutnak. A NIPS hálózatban történő elhelyezésénél ügyelni kell arra, hogy lehetőség szerint a teljes, de legalább a kritikus forgalom vizsgálható legyen. A NIPS képes kiszűrni az alacsonyabb szintű támadásokat, de a szenzoron áthaladó titkosított forgalmat nem tudja vizsgálni. A NIPS a támadásokat kizárolag a hálózat szemszögéből, kontextus nélkül analizálja, ezért előfordulhat, hogy az amúgy ártalmatlan forgalmat is támadásnak véli. Ebbéli hiányosságai miatt mindenkorral kell a NIPS következetetet kezelni. A NIPS működési elve a 3. ábrán látható.



3. ábra: A NIPS működési elve

A fenti csoportosítás analóg módon az IDS rendszerekre is alkalmazható.

A HIPS és a NIPS összehasonlítása

Mivel a NIPS nem az állomások szintjén vizsgálja az áthaladó forgalmat rosszindulatú tevékenység után kutatva, kizárálag a jellemzők (szignatúrák) alapján dönti el, hogy a csomagokat továbbengedi vagy blokkolja. Rendkívül nehéz, de még inkább lehetetlen, hogy a hálózat alapú IPS felmérje, hogy egy adott támadás sikkerel járt-e. Az ilyen rendszerek kizárálag a rosszindulatú tevékenység jelenlétét érzékelik.

A HIPS ellenben a helyi állomást, illetve operációs rendszert vizsgálja. Lehet összetett, amely tényleges rendszerhívásokat vizsgál, vagy egyszerű, amely minden össze a rendszernaplózást és a naplóállományok elemzését teszi lehetővé az állomásokon. Van olyan HIPS, amelyik a támadást még azelőtt elhárítja, hogy az végbement volna, más rendszerek viszont csak jelentik, ha valami már megtörtént.

Általánosságban elmondható, hogy a HIPS a puffer-túlcordulások, webkiszolgálók elleni támadások, hálózati felderítések, valamint elárasztásos, más néven szolgáltatás-megtagadásos (DoS) támadások detektálására alkalmas, míg a HIPS az alkalmazások és állomások által használt erőforrásokat óvja.

Komoly előny a HIPS javára, hogy tudja monitorozni az operációs rendszer folyamatait és védeni a kritikus rendszererőforrásokat. A HIPS a viselkedés alapú elemzést a szignatúra alapú szűrésekkel kombinálja, ezáltal a vírusirtók, valamint a hálózati és alkalmazás rétegbeli tűzfalak legjobb tulajdonságait ötvözi egy csomagban.

NIPS tulajdonságok

A NIPS a hálózat különböző pontjain található eszközök és szenzorok monitorozásával képes a forgalom elfogására és elemzésére. A szenzorok valós időben detektálják a rosszindulatú vagy jogosulatlan tevékenységeket, és szükség esetén beavatkozhatnak. Mivel a szenzorok a hálózat meghatározott pontjain helyezkednek el, bármilyen esemény kapcsán a támadás céljától függetlenül figyelemmel kísérhető a hálózati forgalom alakulása.

Mivel a NIPS szenzor feladata a behatolások elemzése, az alatta futó operációs rendszerről el kell távolítani az összes szükségtelen szolgáltatást, a működéshez nélkülözhetetlen szolgáltatásokat viszont be kell biztosítani. A NIPS szenzor hardvere az alábbiakból áll:

- hálózati adapter, amellyel a NIPS csatlakoztatható bármilyen (leggyakrabban Ethernet jellegű) hálózatra
- processzor, amely elegendő számítási teljesítményt nyújt a behatolás-érzékeléshez szükséges protokollok, illetve mintaegyezések vizsgálatához.
- memória, melynek elegendő mennyisége teszi lehetővé a támadások hatékony és pontos detektálását

A NIPS kiváló skálázhatóságot biztosít egy védett hálózat számára, hisz új állomások hozzáadásával nem szükséges további szenzorok kihelyezése. Új alhálózatok esetében is csak akkor szükséges további szenzorok kihelyezése, ha a meglévő szenzor(ok) vizsgálati kapacitását meghaladja a megnövekedett forgalom, teljesítménye nem éri el a kívánt szintet vagy a biztonsági házirend felülvizsgálata indokolja azt.

Az NIDS és az NIPS esetében a szenzorok helye a hálózatban kulcsfontosságú, általában a hálózat – kritikus szegmenseit védő – belépései pontjainál kell elhelyezni azokat.

A hálózat alapú IPS és IDS előnyei közé sorolható, hogy könnyedén érzékeli a teljes hálózat ellen irányuló támadásokat, segítségével világosan látszik, hogy milyen mértékű a hálózatot ért támadás. További előnye, hogy mivel kizárolag a hálózati forgalmat vizsgálja, nem szükséges a hálózati állomásokon használt különböző típusú operációs rendszereket támogatnia.

Hátrányai közé sorolható, hogy a titkosított adatfolyammal nem tud mit kezdeni a szenzor, de rendkívül nehéz problémát okoz számára a töredezett forgalom helyreállítása is. A legnagyobb hátrány azonban az egyre nagyobb méretű hálózatokból adódik; egyre nehezebb úgy elhelyezni egy szenzort, hogy az lehetőleg az összes csomag elfogását lehetővé tegye. A problémát ugyan megoldja további szenzorok kihelyezése, de ez jelentősen megnövelte a költségeket.

Szignatúra alapú IDS és IPS

A szignatúra alapú megközelítésmód viszonylag merev, ellenben egyszerűen alkalmazható. A mintaegyezséhez előre meghatározott, fix bájtszekvenciákat keres a csomagok fejlécében és adattartalmában. A legtöbb esetben csak akkor beszélhetünk mintaegyezsérről, ha a gyanús csomag bizonyos szolgáltatásokhoz (még inkább konkrét portokhoz) van társítva. Ezzel a módszerrel csökkenthető a vizsgálatból adódó hálózati terhelés, ugyanakkor lényegesen nehezebbé válik az alkalmazása olyan rendszerekben, amelyek nem a jól ismert portokhoz társított protokollokat használnak.

Eleinte viszonylag nagy számban előfordulhatnak hamis riasztások, de a rendszer behangolását követően ez a szám kevesebb lesz, mint a házirend alapú megközelítésmód esetében.

A szignatúra bizonyos környezetben (kontextusban) előforduló bájtsorozat. Ilyen környezet lehet a szekvencia adatfolyamban elfoglalt pozíciója vagy egy alkalmazásrétegbeli protokoll érvényes parancsnak részlete.

Íme néhány példa:

- A webkiszolgálók ellen irányuló támadások jellemzően speciálisan összeállított URL-eket használnak, így az IDS és az IPS olyan szignatúrákat keres az adatfolyam elején, amelyek kliens-oldali HTTP kéréssel kezdődnek.
- Az SMTP kiszolgálók ellen irányuló támadások jellemzően puffer-túlcordulást próbálnak előidézni az SMTP menet MAIL FROM parancsnak segítségével, ezért az IDS és az IPS az SMTP menetek MAIL FROM parancssal kezdődő sorában keres egy bizonyos támadási mintát.
- A levelezőkliensek ellen irányuló támadások jellemzően a tényleges üzenet MIME fejlécébe rejtett puffer-túlcordulásra építenek, ezért az IDS és az IPS olyan bájtszekvenciákat keres, amelyek azonosítják az üzenetbe rejtett új MIME részeket, és puffer-túlcordulást idéznek elő az üzenet olvasását követően.

A fenti példák is jól illusztrálják, hogy a szignatúra alapú IDS és IPS kizárolag olyan támadások detektálására képes, amelyek már előzőleg rendelkezésre állnak egy – a gyártó által biztosított vagy a rendszergazda által karbantartott – mintaadatbázisban. A szignatúra alapú IDS és IPS nem képes a még nem ismert és nem jelentett, ún. nulladik napi támadások detektálására, ezáltal nagyobb terhet ró a rendszergazdára, akinek folyamatosan ügyelnie kell a mintaadatbázis naprakészen tartására, amennyiben a gyártó ezt nem teszi meg. További információ a szignatúra alapú rendszerekről a [4]-ben található.

Házirend alapú IDS és IPS

A házirend alapú megközelítésmód roppant egyszerűen működik; a házirend megsértése esetén az IDS és az IPS blokkolhatja a forgalmat vagy riasztást küldhet az eseményről. A riasztás szükségességéről egy algoritmus alapján dönt. A módszer azért is rendkívül népszerű, mert képes a még nem ismert támadásokat is detektálni.

A házirend alapú IDS és IPS esetében minden pontosan tisztázni kell, hogy a házirend milyen célt szolgál, és pontosan mit takar. Amennyiben a hálózati hozzáférést akarjuk házirend segítségével szabályozni, pontosan meg kell adni az engedélyeket, mely hálózatok érhetik el egymást, és milyen protokollok használatával.

Példaként vegyük azt az esetet, amikor portpásztázás (port sweep) jellegű tevékenységet akarunk detektálni. Ekkor a házirendben meg kell határozni egy küszöbértéket, hogy egy bizonyos számítógép vagy eszköz esetében hány egyedi port szkennelése lehetséges. A házirend további korlátozásokat is tartalmazhat, így meghatározhatja a házirend szempontjából „érdekes” (pl.: SYN) csomagokat, de azt is rögzítheti, hogy minden kérésnek azonos forrásból kell származnia. A megfelelő küszöbértékek meghatározása sokszor nem egyszerű feladat, minden figyelembe kell venni a vizsgált hálózat forgalmi sajátosságait.

Léteznek olyan biztonsági házirendek, amelyek nagyon nehezen építhetők be az IDS és az IPS rendszerekbe. Ha például nem engedélyezett a „felnőtt” vagy warez tartalmak böngészése, kapcsolatot kell biztosítani egy ún. feketelistát működtető adatbázishoz, amely alapján eldönthető, hogy megsértették-e a házirendet.

Anomália alapú IDS és IPS

Az anomália alapú rendszerek általában a „normáltól” eltérő hálózati forgalmat keresik. Ebből adódik a fő problémájuk is: mit tekintünk „normálnak”? Anomáliának tekintjük például bizonyos típusú forgalom szokatlan mértékű növekedését, a vizsgált hálózaton jellemzően nem előforduló típusú forgalom megjelenését, de akár egy ismert protokoll deformált üzenetét. Léteznek olyan rendszerek, amelyekbe bele van kódolva a normál forgalom mintája, míg más rendszerek megtanulják, hogy mi számít normál forgalomnak. Ez utóbbinál felmerül annak a lehetősége, hogy nem megfelelő csoportosítással a normáltól eltérő forgalmat is normálnak tekinti. Ennek eredményeképp kisebb méretű környezetben relatíve jól használható, de nagyobb – főleg vállalati – hálózatok esetében alkalmazása nehézkes, nem váltja be a hozzá fűzött reményeket.

Az anomália alapú IDS és IPS két típusát különböztetjük meg:

Statisztikai alapú anomáliadetektálásról akkor beszélünk, ha a rendszer bizonyos idő alatt megtanulja a vizsgált hálózat „profilját”, vagyis az azon áthaladó forgalom mintáját. Ezt követően a vizsgált forgalom statisztikai vizsgálatával dönti el, hogy az kellően eltér-e a szokásostól. Amennyiben igen, riasztást küld.

Nem statisztikai alapú anomáliadetektálás esetében az ismert, normál viselkedés jellemzői előre rögzítve vannak, bármilyen ettől eltérő forgalmi minta riasztást vált ki.

Az alábbiakban található néhány példa a rosszindulatú, nem statisztikai alapon detektálható anomáliákra:

- IPX kommunikáció előfordulása két olyan eszköz között, amelyek kizárolag TCP/IP protokollt használó hálózatban működnek
- Felhasználói eszközről származó útvonalfrissítés előfordulása
- Szórási vihar (broadcast storm), illetve a hálózat végigpásztázása
- Olyan ellentmondásos csomag, amelyben az összes TCP jelölő bit be van állítva, esetleg megegyezik a TCP szegmens forrás és cél IP-címe vagy a TCP forrás- és célportha

Mézescsupor alapú rendszerek

A mézescsupor alapú megközelítésmód azon az ötleten alapul, hogy a támadást minél messzebbre kell terelni a valódi hálózati eszközöktől. A mézescsupor tulajdonképpen nem más, mint egy speciálisan erre a célra kialakított – az éles hálózat többi eszközétől teljesen elszigetelt – eszköz, amely irányított körülmények között lehetővé teszi a bejövő támadások és rosszindulatú forgalmi minták elemzését, és elegendő időt biztosít a felkészülésre, mielőtt a forgalom elérné a valódi eszközöket. Fontos, hogy soha ne bízzunk meg a mézescsuporként funkcionáló eszközökben, hisz előfordulhat, hogy a tudomásunk nélkül már feltörték azt, és ugródeszkaként használják az éles hálózat ellen indított támadásokhoz!

Kétféle módon foghatunk mézescsupor építéséhez:

Készíthetünk olyan mézescsuprot, amely bizonyos fokig valóban sebezhető a támadásokkal szemben. Így a mézescsupor ellen indított támadások általában sikerrel járnak, de a rendszergazdának időt és lehetőséget ad, hogy naplózza és nyomon kövesse a támadó minden lépését, anélkül hogy az éles rendszerek veszélybe kerülnének.

Ugyanakkor a mézescsupor lehet olyan érdekesnek tűnő célpont, amely megfelelően fel van vértezve a támadásokkal szemben, ugyanakkor sokkal sebezhetőbbnek és áthatolhatóbbnak tűnik a támadó számára. Íme két trükk, amellyel elérhetjük, hogy rendszereink sebezhetőbbnek tűnjeneck:

- Több kapcsolódási lehetőség (vagy kevésbé védett hozzáférés) biztosítása a mézescsuporhoz
- A ténylegesen használt alkalmazások verziószámanak megváltoztatása, ezáltal a támadó egy korábbi sérülékenység kihasználásával próbálkozhat

Néhány példa:

- A klasszikus mézescsupor egy olyan UNIX rendszer, amely például gyenge jelszavak használatával engedi, hogy a támadó bejelentkezzen valamiféle hamis környezetbe, ahol a rendszergazda nyomon követheti minden lépését.
- A levélszemét elleni harcban sem ismeretlen a mézescsupor fogalma. Itt jelenthet olyan levelező-kiszolgálót, amely nyílt relének tűnik, de valójában odavonzza és összegyűjt a levélszemetet küldő címeket, majd eldobja ezeket a levelekét. Az összegyűjtött címek vagy a küldő IP-címe ezután felkerül egy ún. szürke- vagy feketelistára, amelyek használatával a levelező-kiszolgálók tovább szűkíthatik a levélszemét mozgásterét.

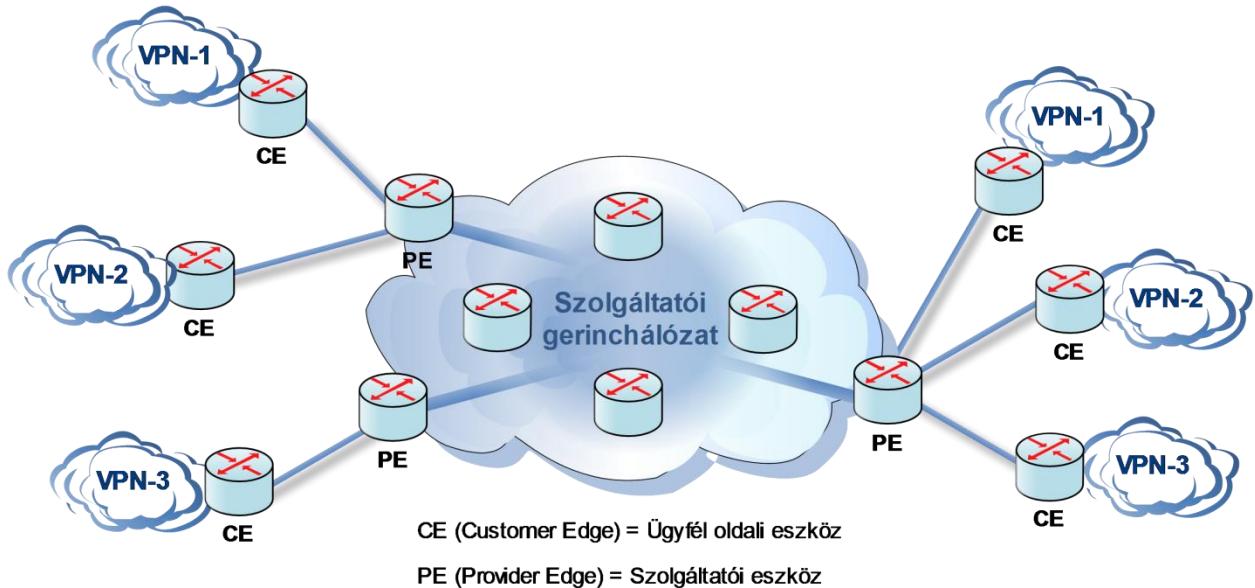
További információ a mézescsupor alapú rendszerekről az [5]-ben található.

8. Virtuális magánhálózatok biztonsága

Mi az a virtuális magánhálózat?

A virtuális magánhálózat (VPN, Virtual Private Network) fejlett titkosítási és alagúttechnikák segítségével lehetővé teszi, hogy biztonságos, végponttól végpontig terjedő hálózati kapcsolatot hozunk létre harmadik fél által üzemeltetett – alapvetően nem megbízható – hálózatokon (pl.: az interneten) keresztül. Jelen fejezet főként a VPN elméleti háttérét tárgyalja, a VPN megoldások gyakorlati oldalát a [6] mutatja be részletesen. A biztonságos kapcsolat azt jelenti, hogy a küldő hiteles azonosítása mellett, az üzenet sérültlensége (integritása) és bizalmas kezelése is ellenőrizhető. A VPN beágyazással, titkosítással vagy a kettővel együtt gondoskodik az adatok védelméről. Azt sem árt leszögezni, hogy beágyazás (más néven alagutazás) alatt az adatok transzparens módon történő átvitelét értjük megosztott hálózatok fölött. A VPN megoldások az OSI rétegmodell második (L2), harmadik (L3), illetve negyedik (L4) rétegében implementálhatók.

Alapvetően kétféle VPN modell létezik: átlapolódó (overlay) és egyenrangú (peer-to-peer) rendszerekről beszélhetünk. Az átlapolódó VPN a 4. ábrán látható.



4. ábra: Az átlapolódó VPN struktúrája

A szolgáltatók a legtöbb esetben átlapolódó VPN modellt használnak, ahol még a tényleges forgalom megkezdése előtt megtörténik a virtuális áramkörök (VC) megtervezése és kiosztása a gerinchálózaton keresztül. Ez IP-hálózat esetében azt jelenti, hogy bár az alapvető technológia kapcsolat nélküli, a szolgáltatás nyújtásához kapcsolatorientált megközelítésmód szükséges. A módszer nem túl skálázható, mivel jelentős mennyiségi áramkört és alagutat kell felügyelni és kiosztani a felhasználói berendezések között. Az átlapolódó modell L2 és L3 rétegbeli VPN-eket is tartalmaz:

Az L2 rétegbeli átlapolódó VPN független az ügyfél által használt hálózati protokolltól, így a VPN nincs kizárolag IP-forgalom továbbítására korlátozva.

Az L3 rétegbeli átlapolódó VPN leggyakrabban az „IP az IP-ben” alagútsémát használja PPTP (Point to Point Tunneling Protocol), L2TP (Layer 2 Tunneling Protocol), illetve IPsec technológiák alkalmazásával.

VPN topológiák

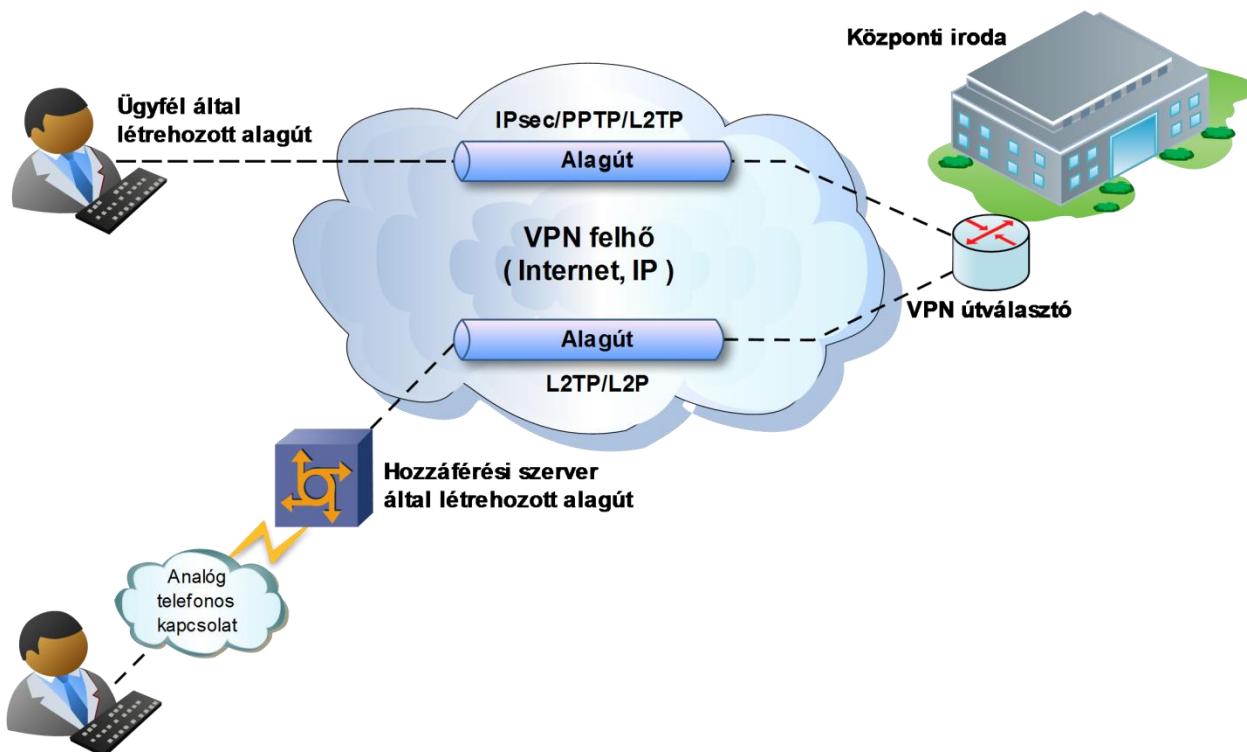
A VPN megoldások topológia szerint három csoportba sorolhatók:

- Távoli elérésű VPN-t elsősorban az otthon vagy folyamatos mozgásban lévő dolgozók használják, akik megosztott infrastruktúrán keresztül (DSL, ISDN, mobil- vagy kábelnet segítségével) férnek hozzá a vállalati intranethez vagy extranethez. A távoli elérésű VPN-hez mindenkor egy VPN átjáró szükséges. A biztonságos kapcsolatot kezdeményező félnek VPN kliensszoftver segítségével kell a VPN átjáróhoz kapcsolódnia. A VPN kliens teszi lehetővé, hogy a központi hálózathoz csatlakozva elérhesse az – akár különböző helyszíneken található – erőforrásokat (pl.: adatközpontokat). Az alagutak létrehozásához az IPsec, a PPTP (pont-pont alagútprotokoll), az L2TP (második rétegbeli alagútprotokoll), esetleg az L2F (második rétegbeli továbbítás) használható.

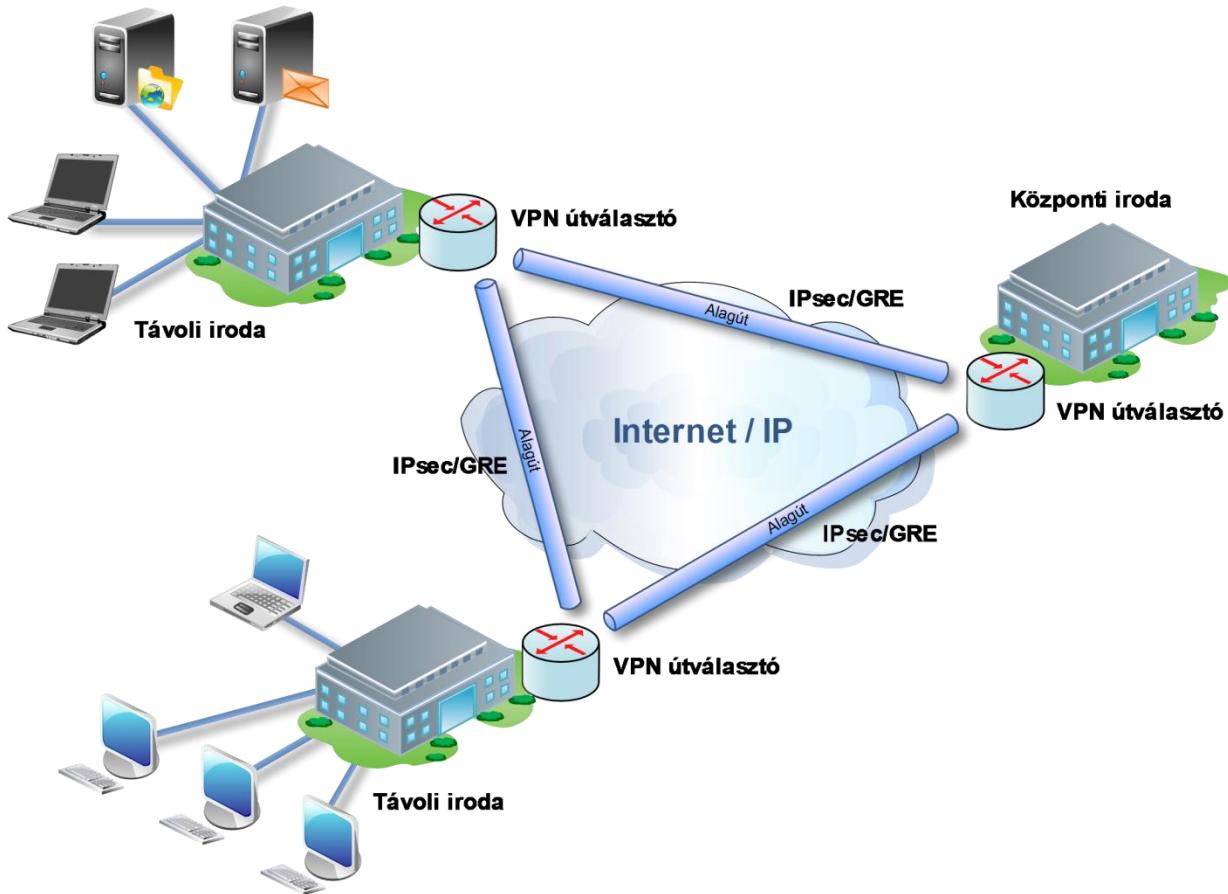
Előnyei: Nem kell magas hívásköltségekkel kalkulálni, mint a betárcsázós esetben. Javítja a

termelékenységet, hisz a dolgozó a tényleges helyzetétől függetlenül végezhet érdemi munkát. Példa a távoli elérésű VPN-re az 5. ábrán látható.

- A telephelyek közötti intranet VPN a vállalati központok, telephelyek, kihelyezett irodák és a belső hálózat között biztosít dedikált, állandó kapcsolatot, megosztott infrastruktúrán keresztül. Az intranet VPN és az extranet VPN közötti fő különbség, hogy előbbi kizárolag a megbízható munkatársak részére biztosít hozzáférést. Az intranet VPN esetében egyazon vállalat különböző földrajzi helyszínei között – az internet segítségével – alakítanak ki biztonságos alagutakat, a felhasználók felé pedig úgy tűnik, mintha mindenki ugyanabból a belső hálózatból csatlakoznának. Az ilyen hálózatokkal szemben az erős titkosításon túl szigorú elvárások vannak a teljesítménnyel és sávszélességgel kapcsolatban is. Az alagutak létrehozásához IPsec vagy IPsec/GRE protokoll használatos. Példa a telephelyek közötti intranet VPN-re a 6. ábrán látható.
Előnyei: Komoly költségek takaríthatók meg a hagyományos bérelt vonalakkal szemben.
- A telephelyek közötti extranet VPN külső ügyfeleket, beszállítókat össze a vállalati ügyfélhálózattal. Általában tűzfalakkal egészül ki az alagutak használata, hogy a külső felhasználók csak bizonyos információkhoz és erőforrásokhoz férhessenek hozzá.
Előnyei: A teljes partnerhálózat azonos házirend, biztonsági és QoS beállítások szerint üzemelhet.



5. ábra: Távoli elérésű VPN



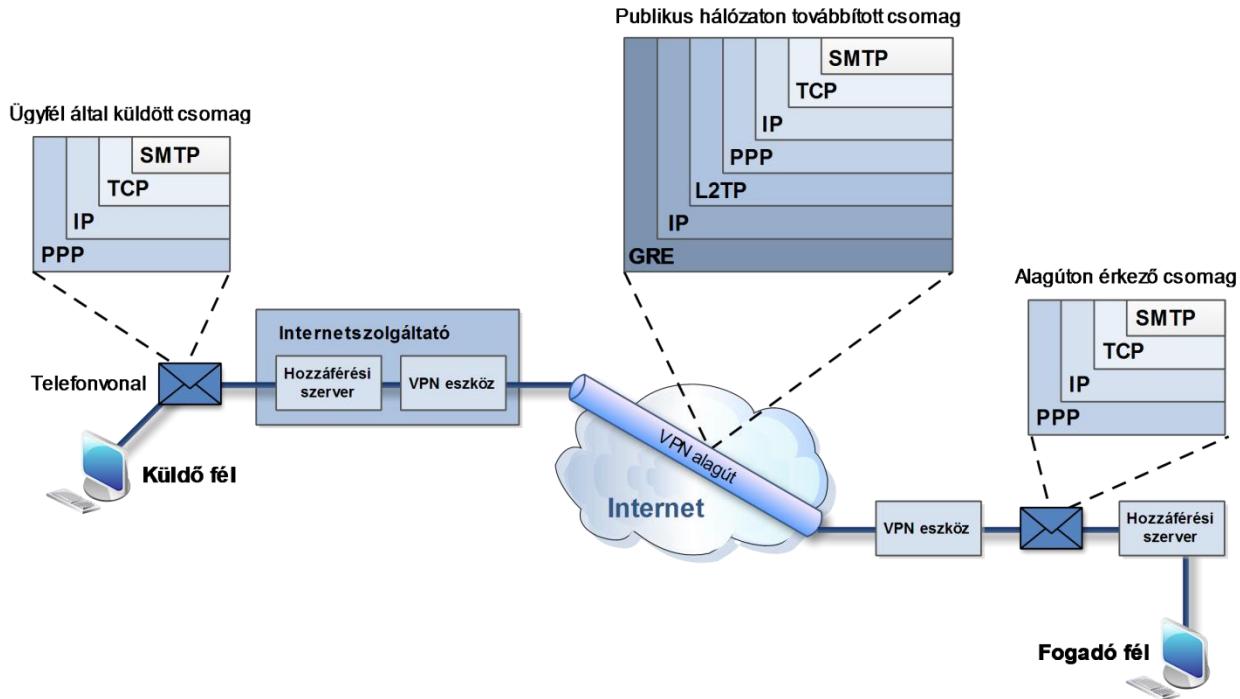
6. ábra: Telephelyek közötti intranet VPN

A biztonságos VPN sajátosságai

A VPN-t használó adattovábbítás megfelelő biztonságáról egyrészt a beágyazás, másrészről a titkosítás gondoskodik. Beágyazásnak (más néven alagutazásnak) nevezük azt a folyamatot, amelynek során a teljes csomagot egy másikba beágyazva az összetett csomagot továbbítjuk egy (akár publikus) hálózaton. Az alagutazáshoz az alábbi protokollok szükségesek:

- Átviteli (carrier) protokoll, ami az információt szállítja.
- Beágyazási protokoll (pl.: GRE, IPsec, L2F, PPTP, L2TP), amellyel az eredeti adat becsomagolásra kerül. Nem minden protokoll képes ugyanazzal a biztonságot garantálni.
- Utasprotokoll (passenger protocol), amely tulajdonképpen az eredeti adat (IPX, AppleTalk, IPv4, IPv6).

A beágyazás folyamata a 7. ábrán látható.



7. ábra: A beágyazás (alagutazás) folyamata

A VPN biztonsága: az IPsec és a GRE

Az alagútprotokollok jelentős mértékben különbözhettek az átvitt adatok számára kínált szolgáltatások, a kezelhető problémák és a biztonsági szint tükrében.

Az IPsec önmagában megbízható magánhálózatot kínál, de csak egyedi címzésű (unicast) IP-csomagok számára. Amennyiben az IPsec és a GRE protokollt kombináljuk, lehetőség nyílik a csoportcímzésű (multicast) IP-csomagok, a dinamikus IGP irányítóprotokollok és az IP-től eltérő szállítási protokoll használatára is.

Az IPsec két titkosítási módja: az alagútmód és a szállítási mód.

Alagútmódban mind a fejléc, mind pedig az adattartalom titkosítva lesz, szállítási módban viszont kizárolag az adattartalom titkosítása történik meg. Fontos, hogy a szállítási módhoz teljesen IPsec-kompatibilis rendszer szükséges. Emellett elengedhetetlen a közös kulcs és a tűzfalak nagyon hasonló házirenden alapuló beállítása is. Az IPsec többféle eszköz (útválasztó-útválasztó, tűzfal-útválasztó, PC-útválasztó, PC-kiszolgáló) között képes az adatokat titkosítani.

A GRE becsomagolja a csomagok IP-fejlécét és adattartalmát, és kiegészíti azt a GRE-beágyazás fejlécével. A hálózattervezők előszeretettel használják ezt a módszert a csomagok IP-fejlécének elrejtésére, méghozzá a GRE-beágyazás adattartalmi részébe.

Alagutazás telephelyek közötti VPN használatával

A telephelyek közötti VPN esetében a GRE felel azért, hogy az utasprotokoll alkalmassá tegye az átviteli (jellemzően IP-alapú) protokoll feletti szállításra.

Alagutazás távoli elérésű VPN használatával

A távoli elérésű VPN esetében az alagutazáshoz használt protokoll jellemzően a PPP és társai. Amikor létrejön a hálózati kapcsolat a kliens számítógép és a távoli elérésű rendszer között, a PPP lesz az átviteli protokoll. A távoli elérésű VPN esetében is használhatók az alábbi – a PPP alapstruktúráját használó – protokollok: L2F, PPTP, L2TP.

A hitelesítés kérdése

Amennyiben hálózaton keresztül dolgozunk vagy bonyolítunk üzletet, nagyon fontos, hogy tudjuk ki van a vonal, email vagy fax másik végén. Ugyanez igaz VPN használata esetén, tehát a VPN alagút túlsó oldalán lévő eszközöt hitelesíteni kell, mielőtt a létrejövő kapcsolatot megbízhatónak tartanánk. Az alábbi módszerek állnak rendelkezésre, hogy a felek meggyőződjenek arról, hogy a megfelelő partnerhez kapcsolódnak-e:

- Felhasználói név és jelszó
- Egyszeri jelszó (OTP, One Time Password)
- Biometrikus azonosítás
- Digitális tanúsítvány
- Publikus/privát kulcspár
- Közös titok (shared secret)

Távoli elérésű VPN környezetben sokkal biztonságosabb hozzáférést tesz lehetővé az ún. AAA szerverek használata. Az AAA betűszó az angol hitelesítés (authentication), jogosultság- (authorization) és fiókkezelés (accounting) szavak rövidítése. Egy kliensoldalról kezdeményezett kapcsolat esetében a kérés automatikusan egy AAA szerverhez kerül továbbításra, amely ellenőrzi, hogy ki az ügyfél, milyen jogosultságokkal rendelkezik, majd naplózza a felhasználó minden tevékenységét. Ez utóbbi különösen hasznos egy esetleges biztonsági probléma felmerülése során.

Az IPsec biztonsági funkciói

Az IPsec egy szabvány, amely az IP-hálózaton történő biztonságos adatátvitel menetét határozza meg, biztosítja az adatok bizalmasságát, sérтetlenségét és a nem megbízható hálózatok feletti kommunikáció hitelességét. Az IPsec egy olyan protokollkészlet, amely egyik titkosítási vagy hitelesítési algoritmushoz, kulcsgenerálási technikához vagy biztonsági társításhoz (SA, security association) sem kötődik. Az IPsec tulajdonképpen csak biztosítja a szabályokat, míg a létező algoritmusok adják a titkosítást, hitelesítést vagy kulcskezelést.

Az adatok megbízhatóságáról az IPsec titkosítással gondoskodik, amely megakadályozza a nyilvános vagy vezeték nélküli hálózatokon átvitt adatok elolvasását vagy lehallgatását, mivel az elfogott csomagok nem dekódolhatók. A titkosításhoz többek között az alábbi algoritmusok használhatók: DES, 3DES, és AES.

Az adatok sérтetlenségét az IPsec hash segítségével biztosítja. A hash pusztán redundancia-ellenőrzés, melynek során az IPsec összeadja az üzenet összetevőit (jellemzően a bájtok számát), majd eltárolja az összeget. A megérkezett csomagon az IPsec megvizsgálja az ellenőrzőösszeget, és összehasonlítja az eredeti, hitelesített értékkal. Amennyiben az értékek megegyeznek, biztosak lehetünk benne, hogy a kérdéses adatot nem manipulálták. Az adatok sérтetlenségét a HMAC (hash-alapú üzenethitelesítési kód,

Hash-based Message Authentication Code) függvény biztosítja, amely az alábbi algoritmusokat támogatja: MD5, illetve SHA-1.

Az IPsec adatok forrásának hitelességét mindig a fogadó fél ellenőrizheti. Az IPsec felhasználók és eszközök hitelesítésére egyaránt alkalmas. Az adatforrás hitelesítésének minőségét az adatok sérteletlenségét biztosító szolgáltatás határozza meg.

A visszajátszás elleni (anti-replay) védelem ellenőrzi, hogy minden egyes csomag egyedi, nem pedig duplikált. Az IPsec úgy gondoskodik a csomagok védelméről, hogy minden beérkező csomag sorszáma összehasonlíta a meglévőkkel, valamint csúszóablakot (sliding window) is alkalmaz. Ha egy csomag sorszáma a csúszóablak értékénél korábbi, azt későnek tekinti. A duplikált, illetve késő csomagok eldobásra kerülnek.

Az IPsec protokolljai

Az IPsec szabvány hitelesítési és adatvédelmi módszert kínál a biztonságos adatátvitelben résztvevő (akár több) partnerek számára. Az IPsec része az IKE (Internet Key Exchange) kulcsok cseréjére használt protokoll, valamint két IP-alapú protokoll: az ESP (Encapsulating Security Payload) és az AH (Authentication Header).

Az IPsec három fő protokollja biztosítja a biztonságos keretet az alábbiakhoz:

- Az IKE felelős a biztonsági paraméterek egyeztetéséért, valamint a hitelesített kulcsok létrehozásáért. Az IPsec szimmetrikus titkosítási algoritmusokkal valósítja meg az adatok védelmét, amelyek sokkal hatékonyabbak és könnyebben alkalmazhatók hardveres környezetben, mint az egyéb típusú algoritmusok. Az IKE biztosítja az ezen algoritmusok számára szükséges biztonságos módszert a kulccseréhez.
- Az AH, vagyis az IP hitelesítési fejléce biztosítja az IP-adatcsomagok számára a kapcsolat nélküli sérteletlenséget, illetve az adatforrás hitelesítést, valamint opcionális védelmet a visszajátszások ellen. Az AH-t a védeni kívánt adatba kell beágyazni, de mára szinte teljesen felváltotta az ESP.
- Az ESP felelős az adatok titkosításáért, hitelesítéséért és védelméért. Az ESP nem csupán adatvédelmi szolgáltatásokat vagy opcionális adathitelesítést, hanem visszajátszás elleni szolgáltatásokat is kínál. A védeni kívánt adatokat az ESP csomagolja be, a legtöbb IPsec megvalósítás ezt használja.

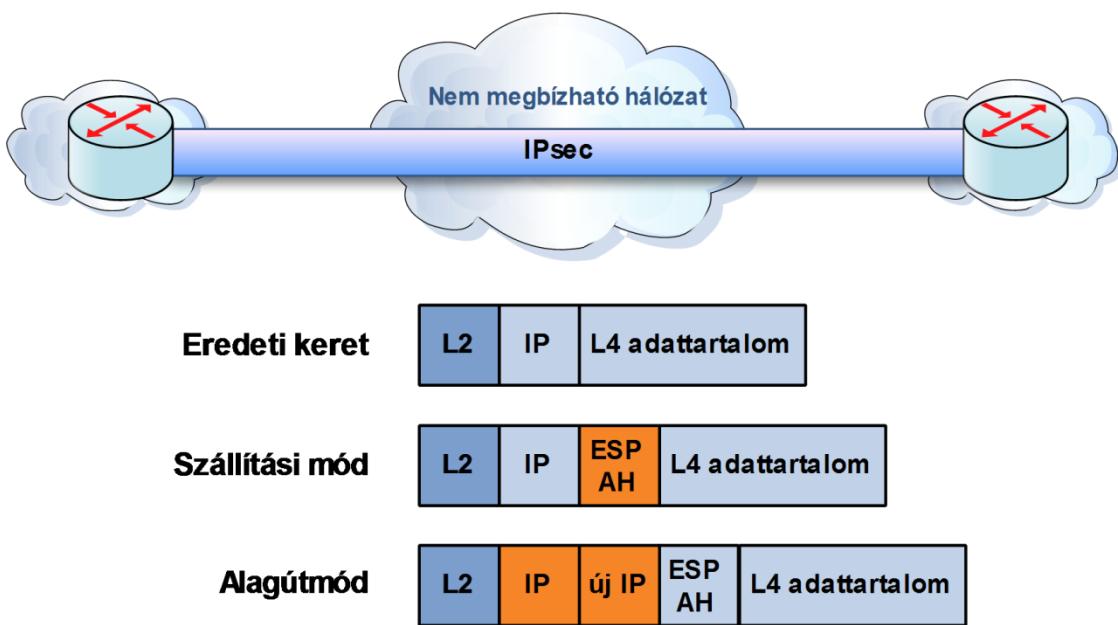
Az IPsec fejlécei

Az IPsec a hitelesítésről, az adatok sérteletlenségéről, valamint a titkosításról az IP-adatcsomagba beszúrt AH vagy ESP fejléccel, esetleg minden kettővel gondoskodik.

Az AH lehetőséget biztosít az IP-adatcsomag hitelességének, illetve sérteletlenségének ellenőrzésére, az ESP ezen felül információt tartalmaz az adattartalom titkosításáról is. Az AH és az ESP két állomás (pl.: végberendezés vagy átvállaló) között használható.

Az AH és az ESP megoldások szabvány alapú módszert kívánnak meg az adatok manipulációja és illetéktelen olvasása elleni védelemhez. Az IPsec az alábbi, különböző erősségű titkosításokat támogatja: DES (Data Encryption Standard), 3DES (Triple Data Encryption Standard) és az AES (Advanced Encryption Standard). Az IPsec számos, különböző erősségű hash módszert is támogat: HMAC (Hash-based Message Authentication Code), MD5 (Message Digest 5) és az SHA-1 (Secure Hash Algorithm 1).

Az IPsec által használt fejlécek a 8. ábrán láthatóak.



8. ábra: IPsec fejlécek

Az IKE protokoll

A titkosított VPN megoldásoknál rendszeres időközönként szükséges válik a titkosítási kulcsok cseréje. Ennek elmaradása esetén a hálózat kiszolgáltatottá válhat a kipörgetéses (brute-force) támadásokkal szemben. A probléma kiküszöbölésére az IPsec az IKE protokoll alkalmazza, amely további protokollok (pl.: DH kulcscsere) segítségével biztosítja a résztvevő felek hitelesítését, illetve a kulcsok generálását. Az IKE az 500-as UDP portot használja.

Az IPsec az IKE protokoll használatával az alábbi funkciókat biztosítja:

- A biztonsági társítások jellemzőinek egyeztetése
- Automatikus kulcsgenerálás
- Automatikus kulcsfrissítés
- Felügyelhető (menedzselhető) kézi beállítások

A biztonsági társításhoz az alábbiak szükségesek:

- Az ISAKMP (Internet Security Association and Key Management Protocol) egy olyan protokoll-környezet, amely megadja a kulcscseréhez használt protokoll, valamint a biztonsági házirend egyeztetésének menetét. Az ISAKMP bármelyik szállítási protokoll felett alkalmazható.
- A SKEME egy olyan kulcscseréhez használt protokoll, amely megadja, hogy gyors kulcsfrissítéssel hogyan származtathatók hitelesített kulcsok.
- Az OAKLEY egy olyan kulcscseréhez használt protokoll, amely megadja hogyan szerezhetők be hitelesített kulcsok. Az OAKLEY alapértelmezés szerint a DH kulcscsere algoritmust használja.

Az IKE automatikusan egyezteti az IPsec biztonsági társításait, és költséges előkészítés nélkül teszi lehetővé a biztonságos IPsec kommunikációt. Az IKE az alábbi tulajdonságokkal rendelkezik:

- Szükségtelenné teszi az IPsec összes biztonsági paraméterének megadását minden oldalon (félnél).
- Meghatározhatja az IPsec biztonsági társításainak élettartamát.
- Lehetővé teszi az IPsec kapcsolat ideje alatti kulcscserét.
- Lehetővé teszi a visszajátszás elleni védelmet az IPsec számára.
- Biztosítja a hitelesítés-szolgáltató (tanúsítvány-kibocsátó) támogatását a felügyelhető, skálázható IPsec implementációkhoz.
- Lehetővé teszi a résztvevők dinamikus hitelesítését.

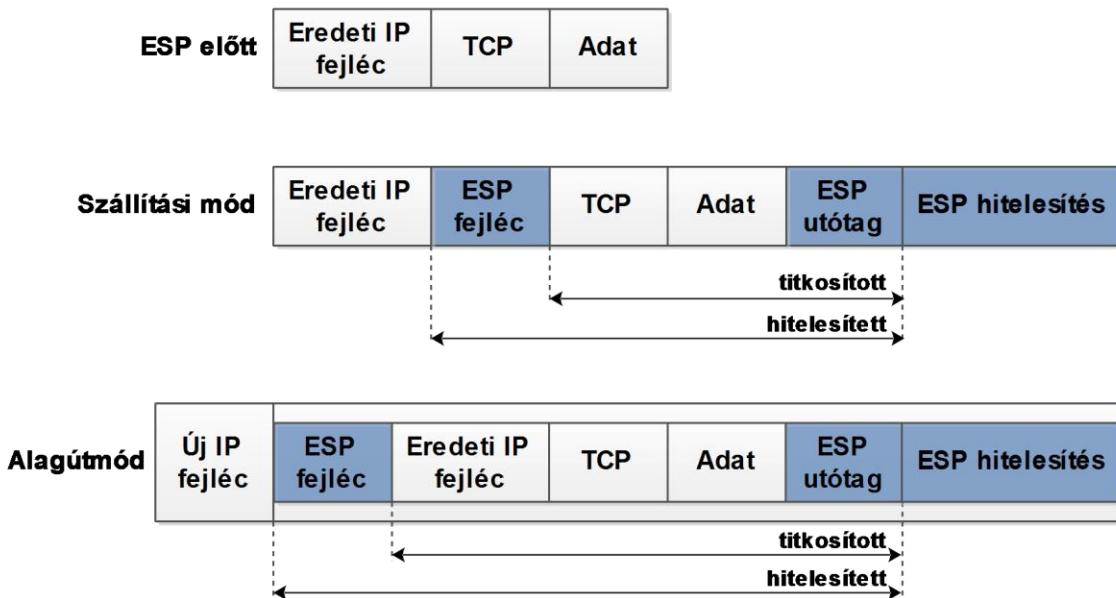
Az IKE lépései részletes leírása a telephelyek közötti VPN működésénél található.

Az ESP és az AH protokoll

Az IPsec protokoll magját az ESP fejléc adja, amely megfelelő titkosítással és transzformációs készlettel gondoskodik az adatok visszafejthetetlenségéről. Az ESP kizárálag a csomag adatrészét védi, opcionálisan gondoskodhat a védett adatok hitelesítéséről is.

Az IPsec másik fontos eleme, az AH protokoll nem a hagyományos értelemben – az adatokat elrejtve – védi a kommunikációt, hanem egyfajta pecséttel látja el az adatcsomagokat. Így tulajdonképpen az IP-fejléc mezőit – ideértve a címmezőket – is védi. Az adatok megbízhatóságát önmagában azonban nem képes garantálni.

Az IPsec két módban továbbíthatja az adatokat a hálózaton keresztül: alagútmódban, illetve szállítási módban. A két mód nem csupán a használatukban, hanem az „utazó” csomaghoz adott többlet mennyiségében is eltér egymástól. Az ESP működési elvét a 9. ábra illusztrálja.



9. ábra: Az ESP működési elve

Az alagútmód a teljes IP-csomagot becsomagolja és védi. Mivel az alagútmód becsomagolja vagy elrejti az IP-csomag címét, a sikeres továbbításhoz a csomagnak egy új – kb. 20 bájtos – fejlécet kell kapnia. Alagútmódban egyaránt használható az ESP és az AH, vagy a kettő kombinációja.

Teljesítmény szempontjából a jellemzően kisméretű csomagokat használó átvitel költségesebb, mint ha ugyanazt az adatmennyiséget nagyobb csomagok szállítanák, ezért lehetőség van az ún. szállítási mód használatára is. Az IPsec szállítási módja az ESP fejlécet az IP-fejléc és a csomag szállítási rétege közé szúrja be, ezáltal a biztonságos adatforgalmat bonyolító minden hálózati csomópont címe látható marad. Ez ugyan kevésbé biztonságos megoldás, viszont nem keletkezik új IP-fejléc, ezért a méret sem nő. Szállítási módban egyaránt használható az ESP és az AH, vagy a kettő kombinációja. A szállítási mód különösen jól tud együttműködni a GRE protokollal, amely egy IP-fejléc beszúrásával már eleve elrejti a végberendezések címeit.

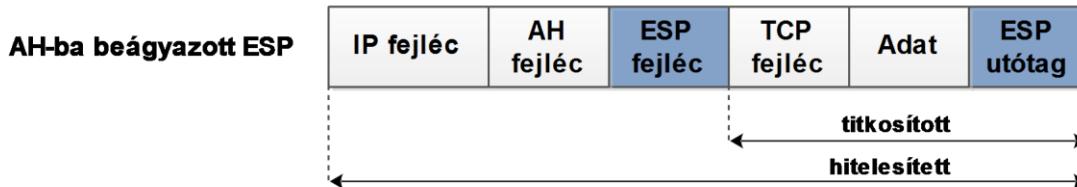
Az ESP titkosítási algoritmusai önmagukban nem tudják garantálni az adatok hitelességét vagy sérhetetlenségét. Az ESP a problémát az adatok hitelességét és sérhetetlenségét biztosító szolgáltatásokkal kiegészítve, kétféleképpen oldhatja meg:

- Hitelesített ESP formátummal
- Az AH-ba beágyazott ESP-vel

A hitelesített ESP esetében az IPsec először szimmetrikus kulcs segítségével titkosítja az adattartalmat, majd egy második szimmetrikus kulcs, valamint a HMAC-SHA1 vagy a HMAC-MD5 használatával kiszámol egy hitelesítési értéket a titkosított adatra. Ezt a hitelesítési értéket a csomag végéhez fűzi. A fogadó fél először kiszámítja a titkosított csomaghoz tartozó hitelesítési értéket a második szimmetrikus kulcs és

ugyanazon algoritmus használatával. Amennyiben a kiszámolt érték megegyezik a csomaghoz kapott hitelesítési értékkel, akkor az első szimmetrikus kulcs segítségével kikódolja az eredeti adatot.

A másik megoldás, hogy az ESP csomag beágyazható egy AH csomagba is. Először az adattartalom kerül titkosításra, majd a titkosított adatokra kell ráereszteni egy hash-függvényt (pl.: MD5 vagy SHA-1). A továbbiakban a hash biztosítja a forrás hitelességét, valamint az adattartalom sérzetlenségét. Erre példa a 10. ábrán látható.



10. ábra: AH-ba beágyazott ESP

Az AH által biztosított hitelesség és sérzetlenség

Az AH függvény a teljes adatcsomagra alkalmazandó, kivéve az IP-fejléc olyan mezőit, amelyek az átvitel során módosulhatnak. Ennek tipikus példája az útválasztók által folyamatosan módosított TTL mező.

Az AH működése az alábbi lépésekkel áll:

1. Az IP-fejléc és az adattartalom hash-értékének kiszámítása
2. Az AH fejléc felépítése a hash alapján, amely az eredeti csomaghoz lesz hozzáfűzve.
3. Az „új” csomag továbbítása az IPsec partner útválasztója felé.
4. A partner útválasztó kiszámítja a kapott IP-fejléc és az adattartalom hash-értékét.
5. A partner útválasztó az AH fejlécből kivonja a kapott hash-értéket.
6. A partner útválasztó összehasonlítja a két hash-értéket, amelyeknek pontosan meg kell egyezniük.

Az ESP protokoll

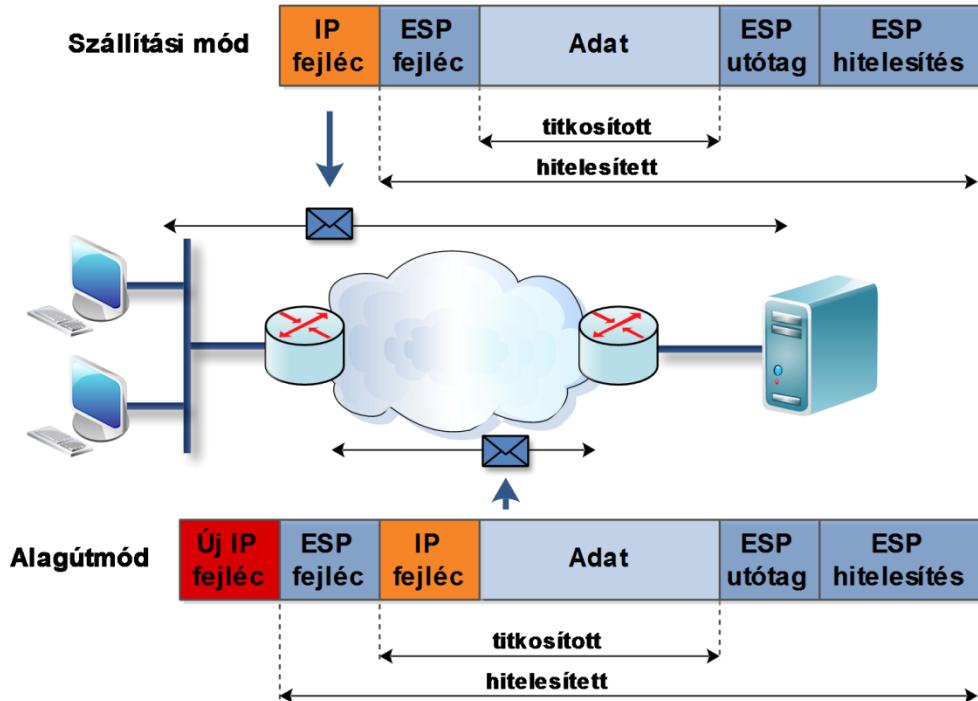
Amikor az ESP végzi a hitelesítést és a titkosítást egyaránt, először a titkosítás történik meg. Ezt a sorrendet az indokolja, hogy a fogadó fél így gyorsabban tudja detektálni és eldobni az ismételt vagy hamis csomagokat. Mivel a fogadó fél még a kikódolás előtt ellenőrizheti a bejövő csomagok hitelességét, csökkenheti egy esetleges elárasztásos (DoS) támadás káros hatásait.

Az IPsec alapértelmezés szerint a szállítási módot használja, amely kizárolag a csomag adattartalmát, valamint a magasabb rétegbeli protokollokat védi, de az eredeti IP-címet védtelenül hagyja. Ez teszi lehetővé, hogy a célhoz vezető útvonalat a csomag az eredeti IP-címe alapján találja meg. Az ESP szállítási módja mindenkorán két állomás között használható.

Az IPsec alagútmódjának használata esetén minden csomagot titkosítva lesz. Az alagútmód biztosítja a teljes IP-csomag védelmét, amelyet AH vagy ESP adattartalomként kezel.

(Tulajdonképpen a teljes IP-csomag kap egy AH vagy ESP fejlécet, majd a beágyazott csomag egy újabb IP-fejlécet.) Az ESP alagútmód általában egy állomás és egy biztonsági átjáró, esetleg két biztonsági átjáró között használható. Távoli elérésű hozzáférésnél általában az ESP alagútmód használata jellemző.

Az ESP szállítási, illetve alagútmódját a 11. ábra illusztrálja.



11. ábra: Az ESP működési módjai

A telephelyek közötti VPN működése

Az IPsec működése öt pontban foglalható össze, amelyet a 12. ábra is illusztrál:

1. Ha a VPN eszköz védelmet igénylő forgalmat érzékel, az „érdekes” forgalom elindítja az IPsec folyamatot.
2. Az IKE első fázisa, amelynek során az IKE hitelesíti az IPsec partnereket (előre megosztott kulcsok, RSA tanúsítványok vagy RSA-val titkosított, egyszer használatos kulcsok segítségével), majd egyezteti az IKE biztonsági társításait, megteremtve ezzel a következő fázishoz szükséges biztonságos kommunikációs csatornát. Az IKE első fázisa kétféle módban lehetséges: normál (main) vagy agresszív módban. Normál módban a felek között három kétirányú üzenetváltás zajlik. Az elsőben a két fél egyezteti, hogy milyen algoritmust és hash-t használnak az IKE kommunikáció biztonságossá tételehez. A másodikban egy Diffie-Hellmann által generált megosztott kulcs, illetve egyszer használatos kulcs (nonce) segítségével ellenőrzik egymás kilétét. Miután létrejön a megosztott kulcs, ezzel generáljuk az összes további titkosítási és hitelesítési kulcsot. A harmadikban mindegyik fél

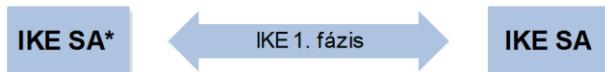
hitelesítéssel ellenőrzi a másik fél kilétét. Agresszív módban kevesebb üzenetváltás, ezáltal kevesebb csomag szükséges, szinte minden az első fázisban történik, a végeredmény ugyanaz.

3. Az IKE második fázisa, amelynek során az IKE egyezteti az IPsec biztonsági társítások paramétereit, majd beállítja az egymással megegyező IPsec biztonsági társításokat a partnereknél. Ezek a paraméterek határozzák meg az átvitt adatok és üzenetek védelmét a végpontok között. Két pont közötti biztonságos kapcsolat létrehozásakor szükséges a biztonsági protokoll által használt algoritmusok ismertetése. Ezeket nem egyenként, hanem ún. transzformációs készletekbe szervezve lehet összevetni. A transzformációs készlet tartalmazza a titkosítási algoritmust, a hitelesítési algoritmust, módot, valamint a kulcs hosszát is. Amennyiben nincs egyezés a partnerek transzformációs készletei között, az alagút megszűnik. Pont-pont környezetben elég, ha egyetlen IKE házirendet határoz meg minden végpont. Küllős (hub-and-spoke) környezetben viszont a központi szerepet betöltő helyszínhez több IKE házirendet is szükséges lehet megadni, hogy az összes partner igényeinek meg tudjon felelni. Mivel a biztonsági társítás élettartama véges, ezért szükség lehet annak (és persze a kulcsok) megújítására.
4. Adatátvitel, a biztonsági társítás biztonsági házirend-adatbázisban (SPD, Security Policy Database) tárolt paraméterei és kulcsai alapján történik.
5. Az IPsec alagút megszüntetése törlés vagy időtúllépés miatt. Időtúllépést eredményezhet bizonyos mennyiségű idő eltelése vagy bizonyos adatmennyiség átvitele. Ha a folyamatos átvitel biztosításához új IPsec biztonsági társítás szükséges, az IKE második – vagy szükség esetén az első – fázisa hajtódik végre, jellemzően még az érvényben lévő biztonsági társítás lejárta előtt.

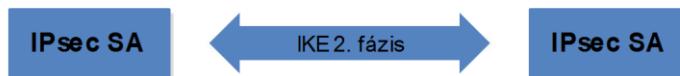
1. Az "A" munkaállomás "érdekes" adatforgalmat küld a "B" munkaállomás felé.



2. Az "A" és "B" útválasztó elvégzi az IKE első fázisához tartozó lépéseket.



3. Az "A" és "B" útválasztó elvégzi az IKE második fázisához tartozó lépéseket.



4. Információcsere történik az IPsec alagúton keresztül.



5. Az IPsec alagút megszűnik.

*SA (Security Association) = Biztonsági társítás

12. ábra: A telephelyek közötti IPsec VPN működésének lépései

A telephelyek közötti IPsec VPN beállítása

A telephelyek közötti VPN beállítása az alábbi lépések szerint történik:

1. Az IKE alagút létrehozásához szükséges ISAKMP házirend beállítása, amely kitérhet a kulcs terjesztési módjára, a titkosításhoz használt algoritmusra, a hash algoritmusra, hitelesítési módra, a kulcscsere menetére, valamint az IKE társítás időtartamára.
2. A transzformációs készlet (transform set) megadása, amely rögzíti az IPsec alagút paramétereit (pl.: a titkosítást, illetve az adatok sértetlenséget biztosító algoritmusokat)
3. Egy kripto hozzáférési lista (ACL) létrehozása, amely meghatározza az IPsec alagúton áthaladó forgalmat. A szabályra nem illeszkedő csomagok természetesen nem kerülnek eldobásra, hanem titkosítás nélkül, az irányítóprotokoll normál működésének megfelelően kerül továbbításra.
4. Kripto-leképezés létrehozása, amely az előzőekben beállított paramétereket kombinálja, illetve megadja az IPsec partnerek között (peer). A bejegyzések az alábbiakra terjedhetnek ki:
 - Mely forgalmat kell az IPsec-kel védeni (a kripto-ACL alapján)?
 - Milyen a védett adatfolyam finomsága (a biztonsági társítások alapján)?
 - Hova kell az IPsec által védett forgalmat továbbítani (vagyis ki az IPsec partner)?

- Mi az IPsec forgalomhoz használt lokális cím (opcionális)?
 - Milyen IPsec biztonságot válasszunk a kérdéses forgalomhoz (a transzformációs készletből kiválasztva)?
5. A kripto-leképezés alkalmazása a VPN eszköz kimenő interfészére.
 6. Az ACL létrehozása és interfészre történő alkalmazása. A határ-útválasztók jellemzően korlátozó ACL-eket használnak, amelyek akaratlanul is blokkolhatják az IKE és IPsec protokollt.

Tippek, trükkök:

- Amennyiben az interfészen dinamikus irányítóprotokoll működik, annak forgalmát is engedélyeznünk kell.
- Amennyiben az IPsec forgalom áthalad cím- vagy portfordítást (NAT, illetve PAT) használó eszközökön, szükség van az IPsec NAT-T (Network Address Translation Traversal) funkciójára, amely az IPsec csomagot egy UDP fejléccel kiegészítve csomagolja be. A megfelelő működéshez szükség lehet a túzfalszabályok további módosítására (kiegészítésére).

A GRE protokoll

A GRE (Generic Routing Encapsulation) egy olyan alagútprotokoll, amely sokféle protokoll- és csomagtípus beágyazását teszi lehetővé az IP-alagutakon belül, egyfajta virtuális pont-pont kapcsolatot létrehozva az IP-hálózaton két útválasztó között. Ezáltal a GRE-t alkalmazó IP-alagúttechnika lehetővé teszi a hálózat bővülését minden össze egyetlen protokollt támogató gerinchálózatok fölött is. Így az alagútban használt irányítóprotokollok is küldhetnek és fogadhatnak útvonal-frissítési információkat a virtuális hálózatban. Ehhez minden össze annyi szükséges, hogy az adattartalmi rész és az alagutazáshoz használt IP-fejléc közé beszúrunk egy GRE fejlécet is. A GRE fejléc tartalmaz egy protokolltípus (protocol type) mezőt, amely bármilyen L3 rétegbeli protokoll beágyazását támogatja. A GRE nem állapottartó, és nem biztosít különösebben erős biztonsági mechanizmust sem az adattartalom védelmének érdekében. Az eredeti, alagúton átutazó csomaghoz képest legalább 24 bájtnyi többletet jelent a GRE, illetve az alagutazáshoz használt fejléc.

A GRE fejlécében előforduló alagútkulcs (tunnel key) kétféle célra használható:

Egyszerű, titkosítás nélküli szövegként hitelesíthető vele minden áthaladó csomag a GRE végpontok között, ugyanakkor a csomagok útvonalán bárki könnyen megtekintheti a kulcsot, és meghamisíthatja az alagútcsomagokat is.

Sokkal elterjedtebb a kulcs használata olyan környezetben, ahol két útválasztó között ugyanarról az IP-címről induló alagutat kell kialakítani. Ekkor a különböző alagutakhoz tartozó GRE csomagok a kulcs segítségével válnak megkülönböztethetővé.

Biztonságos GRE alagutak

A GRE legfőbb előnye az erőteljes, ugyanakkor egyszerű alagúttechnika biztosítása. A GRE bármilyen L3 rétegbeli protokollt elfogad adattartalomként, és azok számára virtuális pont-pont összeköttetést biztosít. A GRE segítségével az irányítóprotokollok használata is lehetővé válik az alagút felett.

A GRE fő hiányossága, hogy szegényes a biztonsági készlete. Kizárolag egyszerű, titkosítás nélküli szöveges kulcsot használ a hitelesítéshez, amely nem tekinthető biztonságosnak. A csomagok útvonalán gyakorlatilag bárki könnyen megtekintheti a kulcsot, és meghamisíthatja az alagútcsomagokat is. A biztonságos VPN által megkövetelt alábbi feltételeknek a GRE önmagában nem felel meg.

Erős titkosítás:

- az adatforrás hitelesítése, amelyet nem lehet kijátszani beékelődéses (man-in-the-middle) támadással
- az adat sérтetlenségének biztosítása úgy, hogy ne lehessen kijátszani beékelődéses támadással

A GRE alagutak biztonságossá tétele az IPsec segítségével

Az IPsec gyakorlatilag biztosítja a GRE-ből hiányzó összes alagútjellemzőt:

- szimmetrikus algoritmust (pl.: 3DES-t vagy AES-t) használó titkosítás
- Az adatforrás hitelesítése hash-alapú üzenethitelesítési kódok (HMAC) (pl.: MD5 és SHA-1) segítségével
- Az adat sérтetlenségének ellenőrzése HMAC segítségével

Az IPsec fenti szolgáltatásai viszont eredetileg kizárolag IP-forgalomhoz készültek. Több protokoll egyidejű támogatásához mindig szükség van további alagútpotokollra.

GRE az IPsec fölött

A pont-pont alapú „GRE az IPsec fölött” legtöbb megvalósítása a küllős topológiát használja, mivel a VPN helyszínek közötti teljes kapcsolat létrehozásához ez igényli a legkevesebb alagutat. A küllős topológia minimalizálja az IPsec alagutak karbantartásához szükséges felügyeleti felesleget (többletet).

OpenVPN – alternatívá virtuális magánhálózat megvalósításra

Az OpenVPN egy GNU GPL alatt kiadott multiplatformos virtuális magánhálózatot megvalósító szoftver. Elérhető többek közt Linux, BSD, Windows, Solaris, Android rendszerekre. Alkalmas telephelyek közti és távoli elérésű VPN kiépítésére is. Egy bináris openvpn programot tartalmaz, ami kliensként és szerverként is tud viselkedni. Konfigurációját egy szöveges állományon keresztül lehet beállítani. Egyes rendszerekre elérhető hozzá grafikus konfiguráló segédprogram, de tudja kezelni a Linux világban népszerű Network-manager is. Tud működni inicializálás után felhasználói térben (userspace) rendszergazdai jogosultság nélkül is. Az alagút titkosítását az OpenSSL szoftver valósítja meg teljes egészében, így használható bármilyen titkosító algoritmussal, amit az OpenSSL ismer és használni képes. Egy virtuális hálózati kártyát hoz létre, melynek két típusa lehet: TUN vagy TAP. TUN típus esetén IP-forgalom szállítására lesz használható. TAP típus esetén bármely ethernet forgalom továbbítható lesz rajta keresztül. Ennek köszönhetően működhet TAP felett minden olyan protokoll is, ami szórási címre

küldött csomagokat is használ a működéséhez. Ilyen például a Windows rendszereken a fájl- és nyomtatómegosztásra használt SMB (server message block) protokoll. A felek azonosítása történhet előre kiosztott kulcspárok és tanúsítványok felhasználásával, de az hitelesítési motorja kiegészíthető beépülő modulokkal, így az azonosítás elvégezhető PAM-on vagy RADIUS szerveren keresztül is. Tartalmaz egy szkriptyűjteményt a gyors és egyszerű kulcspár és tanúsítvány generáláshoz. Ha kulcsos azonosítást választjuk, akkor minden kliensnek szüksége lesz a szerver tanúsítványhoz, valamint saját kulcspárra. A kulcspárban megjelölt név (CN, common name) mező segítségével megkülönböztethetjük a klienseket csatlakozáskor, és a közös, mindenire vonatkozó beállítások mellett kliensre szabott beállításokat is eljuttathatunk a távoli eszközökhöz, így például egyedi útválasztó szabályokat. Egy OpenVPN folyamat több kliens egyidejű csatlakozását is tudja kezelni.

9. Hálózatbiztonsági architektúrák (terheléselosztás, azonnali helyreállítás)

A tűzfal célja

Tűzfalak alkalmazása többes céllal történhet. Céljuk egyfelől, hogy forgalomszabályozási pontot képezzenek a szervezet belső hálózata és az internet között mindenki irányba. Az internet vagy a szervezet szemszögéből nézve bármely, külső hálózatnak minősülő irányból csak a szervezet házirendjének megfelelő, és csak a szükséges hálózati forgalmat szabad beengednie. Másfelől saját szervezetünk kifelé irányuló hálózati forgalmát is szabályozhatjuk tűzfalakkal. Ennek megvalósításához a tűzfalat a hálózat határán kell elhelyezni.

Ugyanakkor alkalmazhatunk tűzfalakat szerverekre és munkaállomásokra telepítve is. Ezek célja, hogy kikényszerítsék az adott szerverre vagy munkaállomásra vonatkozó házirendet, valamint védjék azokat mind a külső, mind pedig a belső hálózat felől érkező támadások ellen. Tipikusan elhárítandó jelenségek számít egy adott hálózat gépein futó szolgáltatások miatt futtatott portszkenelés (port scan) vagy a különböző elárasztásos támadások. Ne feledjük, hogy egy adott hálózat állomásai nem csupán kívülről, hanem belülről is támadhatók, ha a támadó már sikeresen hozzáférést szerzett valamely géphez vagy belső eszközözhöz.

Fontos szem előtt tartani, hogy a tűzfal nem tud védelmet nyújtani a felhasználók gondatlansága vagy rosszhiszemű viselkedése, valamint a megtévesztéses támadások (social engineering) ellen!

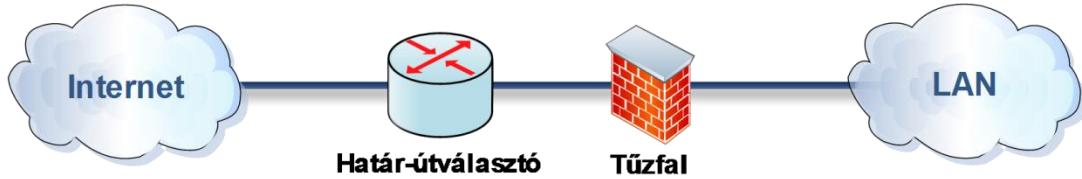
Megvalósítás

Tűzfalat szoftveres és hardveres eszközökkel egyaránt megvalósíthatunk, ahol a hardveres megvalósítás tartalmaz szoftverkomponenst is. Lássuk most ezeket egyenként!

Hardveres megvalósítás

Ebben az esetben a tűzfal szerepét egy célhardver látja el, mely rendelkezik a hálózat forgalmához mért áteresztőképességgel, akár több hálózati interfésszel, saját operációs rendszerrel, csomagszűrő vagy proxy szoftverrel, konfigurációs felülettel, melyen keresztül beállíthatók a hálózati paraméterek, valamint kialakíthatók a szabályrendszerek.

Ezen kívül lehetséges olyan hálózati kapcsolók telepítése is, melyek képesek egyszerű hozzáférési szabályok (access control list) alkalmazására switch-portonként, vagy belső, csak a switch szoftvere által létrehozott hálózati interfészenként. A hardveres túzfal struktúrája a 13. ábrán látható.



13. ábra: Tüzfal hardveres megvalósítással

Szoftveres megvalósítás

Olyan megoldások is szóba jöhetnek, melyeknél a túzfalszoftver telepítése a szervezet határ-útválasztójára, illetve a szerverekre és munkaállomásokra történik. Ekkor az operációs rendszer, a hálózati interfészek már adottak, a túzfalszoftver kiegészíti az útválasztó vagy a munkaállomás képességeit. A hardveres túzfal struktúrája a 14. ábrán látható.

Szoftveres túzfalra példa a Linux kernellel szállított Netfilter csomagszűrő, valamint a hozzá kapcsolódó, konfigurációs lehetőséget biztosító iptables szoftvercsomag.



14. ábra: Tüzfal szoftveres megvalósítással

A tüzfalak szabályrendszerei

A szabályrendszerek határozzák meg a tüzfal működését. A tüzfal kétféle döntést hozhat: adott forgalmat engedélyez vagy sem. A döntés meghozatala előre meghatározott szabályok szerint történik. A szabályoknak sorrendjét mi határozzuk meg. A szabályok fentről lefelé (top-down) értékelődnek ki. minden szabálynál megvizsgálja a tüzfal, hogy a szabályban rögzített feltételeknek megfelel-e a beérkező forgalom. Addig folytatódik a szabályok kiértékelése, amíg olyan szabályhoz nem ér, ami illeszkedik a beérkező csomagra. Ekkor a szabály eldöntheti, hogy a forgalom áthaladhat, visszautasításra kerül, de akár utasíthatja a tüzfalat további szabálycsoportok kiértékelésére. Ha egyik szabály esetén sem volt illeszkedés, akkor a beállított alapértelmezés szerinti akció hajtódik végre, azaz a tüzfal minden visszautasít vagy minden elfogad.

Külön szabálycsoportok is létrehozására is van lehetőség. A szabálycsoportok lehetővé teszik, hogy a vizsgált forgalomnak kevesebb szabályon kelljen végighaladnia, lerövidítve a kiértékeléshez szükséges időt és erőforrásigényt, valamint segíti a szabályrendszer olvashatóságát és áttekinthetőségét. Például a kiértékelési folyamat elején elágazást iktathatunk be, külön szabálycsoportra irányíthatjuk a

TCP/UDP/ICMP forgalmat, és ezekben folytathatjuk a szabályozást. De a csoportosítás történhet a forrás- vagy célcímek osztályai szerint is.

Azt sem szabad elfelejteni, hogy napjainkban történik az IPv6 protokoll széleskörű bevezetése, ami azt jelenti, hogy nem csak IPv4, hanem IPv6 címtartományainkra is ki kell alakítanunk szabályrendszerünket.

Tűzfalak csoportosítása

A tűzfalakat csoportosíthatjuk az alapján, hogy az OSI rétegmodell mely rétegében működnek. Ez alapján a következő típusokat különböztethetjük meg:

- Csomagszűrő tűzfalak
- Nem állapottartó (stateless) működés
- Állapottartó (stateful) működés
- Proxy tűzfalak

Csomagszűrő tűzfalak

A csomagszűrő tűzfalak az OSI rétegmodell adatkapcsolati (L2), hálózati (L3) és szállítási (L4) rétegeiben működnek. A tűzfal megkap minden csomagot az operációs rendszer kernelétől, és azokat egyesével vizsgálja, legyen az kintről befelé, vagy bentről kifelé irányított csomag. A tűzfal dönti el, hogy adott csomag áthaladhat vagy eldobásra kerül, esetleg a megfelelő ICMP válaszüzenet küldésével egy kapcsolat visszautasítása vagy – már felépült kapcsolatok esetén – bontása történjen meg.

A szabályok a csomag egy vagy több paraméterét is vizsgálhatják. Ilyen lehet például a forrás- vagy célcím, a forrás- vagy célport, a szállítási protokoll (pl.: TCP/UDP/ICMP), IP-verzió (IPv4, IPv6), fizikai cím (MAC address), esetleg valamely TCP/UDP jelzőbit (flag).

A csomagszűrő tűzfalak általában nem vizsgálják a csomagok adattartalmát, kizárolag a csomagok fejléceiben található információk alapján hoznak döntést.

Nem állapottartó működés

Ebben a működési módban a csomagszűrő tűzfal nem tart fenn saját adatbázist az új, már felépült vagy lezárásra váró kapcsolatokról. Nincs úgynevezett kapcsolatkövetés (connection tracking), így nincs információ egy adott adatfolyam állapotáról. A döntés mindenkorán kizárolag a kapott csomag fejléceinek vizsgálata alapján történik. Sok hálózati protokoll esetén ez a mechanizmus elegendőnek bizonyulhat, főleg olyanoknál, ahol egy szolgáltatás csak egy portot használ. A webszervereknél elterjedt HTTP protokoll például tipikusan a 80-as TCP portot használja. Amennyiben egy publikus IP-címen hallgató szerver webes szolgáltatásának elérését engedélyezni kell az internet felől, akkor két szabály megadása szükséges. Egyfelől engedni kell a kapcsolat létrejöttét külső címek felől, másfelől engedélyezni kell, hogy a szerver válaszolhasson kifelé, a kezdeményező állomás irányába. Így a szerverenként minden egyes szolgáltatáshoz tartozó két szabály a szabályrendszerünk gyors hizását eredményezi.

Állapottartó működés

Több olyan, napi szinten használatos protokoll létezik, amely nem csak egy portot használ. Ilyen például az FTP protokoll, amely tipikusan a 21-es TCP porton figyel. Ezen a porton működik a protokoll működéséhez szükséges kontroll csatorna, viszont az adatátvitel a 20-as TCP porton megy végbe. Ebben

az esetben viszonylag egyszerű a helyzet, viszont léteznek olyan protokollok, amelyeknél a különböző segédfolyamok portszáma nem előre definiált. Ilyen például az NFS protokoll, vagy a VoIP hívásoknál a SIP/RTP protokollpáros. Látható, hogy ezekben az esetekben megoldás lehet a tűzfalon áthaladó kapcsolatok nyomon követése. A tűzfal feljegyzi az engedélyezett kapcsolatokat, az abban résztvevő állomások IP-címeit, a portszámokat, a kapcsolat állapotát (új kapcsolat, felépült kapcsolat, bontásra váró kapcsolat). Így megadhatók olyan szabályok is, amelyek megengedik, hogy a feljegyzett kapcsolatok további ellenőrzés nélkül átjussanak a tűzfalon. Az előző példában szereplő webszerver esetén elegendő lenne egy szabályt definiálni, ami átengedi a feljegyzett kapcsolatot, és egyet, ami engedi a webszerver elérését. minden további szolgáltatás engedélyezéséhez elegendő egy új szabály felvétele. Hasonlóan, az FTP esetében is elegendő a 21-es TCP port engedélyezése, a nyilvántartás miatt az adatátvitel is menni fog.

Proxy tűzfalak

A proxy tűzfal az OSI rétegmodell alkalmazási rétegében (L7) működik. Nem egyszerűen csak a csomagok fejléceiben szereplő paramétereket vizsgálja, hanem az adatrészt is. Ismer több hálózati protokollt, sokszor transzparens módon működik a felhasználók szemszögéből nézve. Az ismert protokollok szabályos működése is vizsgálható proxy tűzfalakkal, amellett, hogy eszközök biztosít a hozzáférés-szabályozáshoz is. Tipikus példa proxy tűzfalra a webes forgalom szabályozására használt webproxy. A HTTP kérést a kliens gépek valójában a proxy irányába továbbítják. A proxy feldolgozza a kérést, a meghatározott szabályok alapján engedélyezi vagy visszautasítja azt. Engedélyezés esetén a webtártalmat a proxy tölti le a megadott célcímről, majd továbbítja a kérést indító kliens felé. Ez a technika használható egyfelől a szervezet gépeinek kifelé irányuló kéréseinek szabályozására, de lehetőséget ad a webszerver külső támadások elleni védelmére is az úgynevezett fordított proxy (reverse proxy) mód használatával, amely kikényszeríti a protokollsabályok pontos betartását, valamint védi az elárasztásos próbálkozások ellen is.

Hátránya, hogy nagyobb az erőforrásigénye és késleltetése a csomagszűrő tűzfalakkal szemben, mivel nem csomagonként történik a vizsgálat, hanem több csomag bevárása, az adatmezők összefűzése után.

Előnye, hogy naplózhatóvá teszi a tevékenységeket, szűrhető, szabályozható lesz az adatfolyam az alkalmazási rétegen. A csomagszűrő tűzfal csak azt tudja szabályozni, hogy egy adott szervezet munkatársai elérhessék az interneten levő FTP szerverek 21-es TCP portját. Azt viszont csak a proxy tűzfal képes kiszűrni, hogy egy tetszőleges irányba indított, 21-es TCP portra irányuló kapcsolat valóban egy FTP szervert ér-e el, és nem pedig a cég munkatársa próbál olyan protokoltt használni, amelynek az alapértelmezett portja a házirend szerint tiltott, és kizárolag a tűzfal megkerülése a cél.

Manapság szinte minden szerveren beállítható, hogy adott szolgáltatás mely porton figyeljen, ezért a rendszergazdának megfelelő technikákat kell alkalmaznia, hogy az elképzelt szabályrendszer valóban a kívánt szabályozást biztosítsa.

Helyreállítás áramszünet után

Évente pár alkalommal szinte mindenhol elfordulnak rövidebb-hosszabb ideig tartó áramszünetek. Fontos, hogy a hálózat központi tűzfala minél magasabb rendelkezésre állást biztosítson! Ezért fontos,

hogy a tűzfalként szolgáló eszköz szünetmentes tápellátással legyen felszerelve, ami át tudja hidálni az átlagos hosszúságú áramszüneteket. Hogy milyen az átlagos hosszúságú áramszünet, azt tapasztalat útján tudjuk megállapítani, egy adott környezetre nézve. Ugyanakkor az áthidalandó idő hosszúságát meghatározhatták a szervezettel szemben támasztott elvárások is. Kis iroda esetén elfogadható lehet, ha az áram visszakapcsolása után a tűzfal üzemkész állapotáig el kell tennie egy-két percnek, és nem okoz gondot, ha az áramszünet alatt a tűzfal legfeljebb 30 percig üzemet, hisz az irodai asztali gépekhez csatlakoztatott szokványos szünetmentes tápok 15-30 perces intervallumot tudnak áthidalni. Egy internetszolgáltató vagy irodaház gerinchálózati eszközei, illetve tűzfala esetén azonban jogosan elvárható a hosszabb tartásidő, hiszen a kliensek nagy területen való elhelyezkedése miatt előfordulhat olyan szituáció, amikor csak a szerverszoba környékén – illetve internetszolgáltató esetén csak a központi eszközöknél – van áramszünet, de az épület más részein vagy akár a teljes felhővel hálózatban nincs áramkimaradás.

Általánosságban elmondható, hogy minél nagyobb tudással, feldolgozási és áteresztőképességgel rendelkezik egy tűzfaleszköz, annál több időbe telik kikapcsolt állapotból az üzemi állapot elérése. Azt is figyelembe kell venni, hogy újraindítás után a tűzfal elveszíti állapotterét! A kapcsolatokról nyilvántartott minden információ elveszik az állapottartó és proxy tűzfalak esetén egyaránt, így a klienseknek újra fel kell építeniük a hálózati kapcsolataikat, ami löketszerű terhelést jelenthet a tűzfal számára.

Tűzfal-architektúrák

A tűfalépítési feladat komplexitása, az elvárt rendelkezésre állás és a megkövetelt biztonsági szint határozza meg, hogy milyen architektúrát alkalmazunk. Az alábbiakban néhány építkezési mód leírása olvasható.

Egyedülálló tűzfal

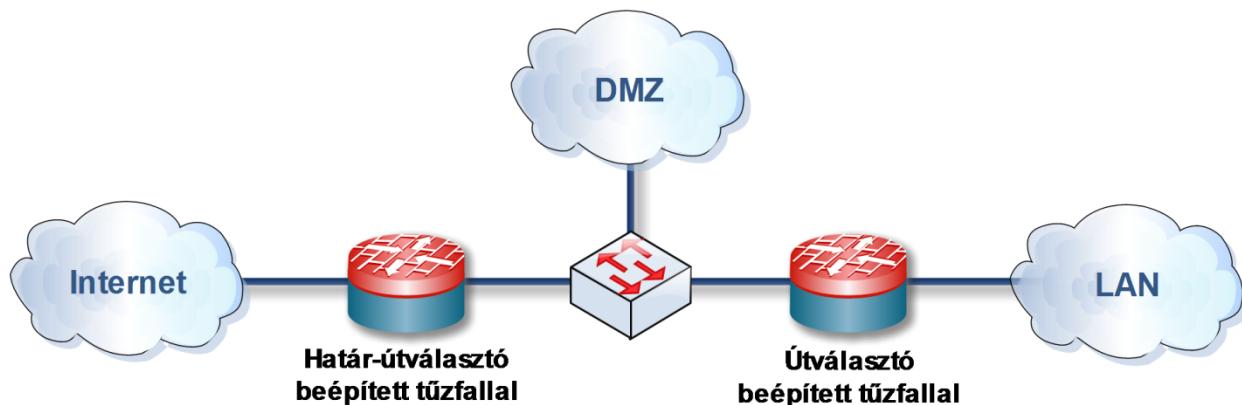
Egyedülálló tűzfal esetén a belső hálózat és a külső hálózat között csak egy tűzfal helyezkedik el, amely minden irányból szabályoz. Előnye, hogy alacsony a kialakítás költsége, egy eszközt kell konfigurálni és üzemeltetni. Hátránya, hogy csak egy védelmi vonalat teremt, azaz kritikus meghibásodási pontnak (single point of failure) tekinthető. A tűzfal sikeres feltörése vagy szabályainak kijátszása a külső támadó számára direkt hozzáférést biztosít a belső hálózathoz. Az egyedülálló tűzfal struktúrája a 15. ábrán látható.



15. ábra: Egyedülálló tűzfal

Kettős (szendvics) tűzfal

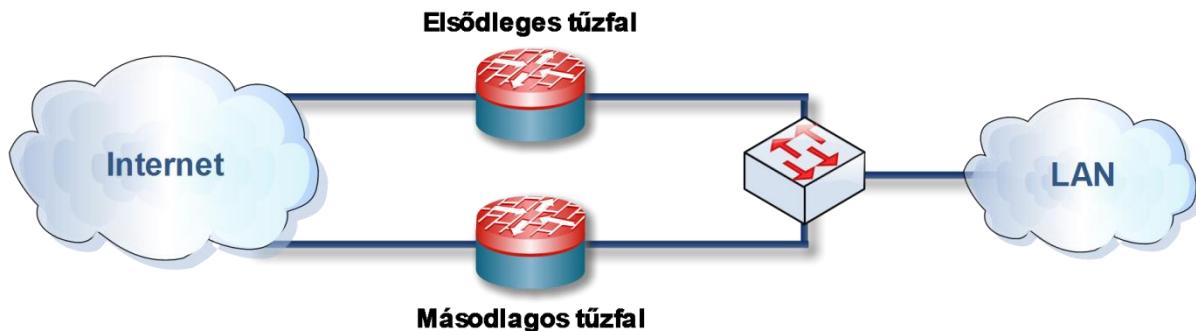
Kettős tűzfalról akkor beszélünk, ha külön tűzfal szűri a külső hálózat forgalmát, amelynek a belső hálózat felé néző hálózati interfésze egy demilitarizált zónához (DMZ) csatlakozik. A DMZ-ben található szerverek kommunikálhatnak külső hálózatokkal, megbíznak egymásban, a belső hálózat felé külön kapcsolattal rendelkeznek, valamint forgalmuk – a teljes befelé irányuló forgalommal együtt – áthalad egy belső tűzfalon, mely mögött húzódik a belső hálózat. Hátránya, hogy implementációja költségesebb, több tűzfal- és hálózati eszközt igényel. Fontos, hogy kettős tűzfal esetében két helyen kell a szabályrendszerünket felépíteni és karbantartani! Előnye, hogy a külső tűzfal kompromittálása csak az első védelmi vonal elestét jelenti, a rendszer-üzemeltetőnek detektálási lehetőséget és időt ad a támadás megállításához, a szükséges lépések megtételéhez. A támadó nem jut direkt hozzáféréshez a belső hálózat irányába, mivel a DMZ-ből csak erősen korlátozott kommunikáció indítható a belső hálózat irányába. A kettős tűzfal struktúrája a 16. ábrán látható.



16. ábra: Kettős tűzfal topológia

Tűzfalak tartalékolása, hibatűrő elosztott tűzfalak

A rendelkezésre állás növelhető, a szolgáltatás-kiesés pedig csökkenthető, amennyiben tartalék tűzfal(ak) áll(nak) folyamatos készenlétben. Olyan architektúra kialakítása szükséges, amelyben alapértelmezés szerint az elsődleges tűzfal üzemel, de ki van jelölve egy tartalék tűzfal. Ha az elsődleges eszköz meghibásodik vagy tervszerű karbantartáson megy keresztül, akkor a tartalék átveszi a szerepét. Nem állapottartó tűzfalaknál a váltás szinte észrevehetetlen lehet a kliensek szemszögéből. Viszont az állapottartó és proxy tűzfalak esetében ahhoz, hogy az átállás észrevehetetlen legyen, az állapottér folyamatos replikációja, az elsődleges eszközzel való szinkronban tartása lenne szükséges. A nagy hálózati átviteli sebesség, a kliensek nagy száma és a folyamatos kapcsolatnyitási és -lezárási kérések, a magas – akár több tízezer – másodpercenkénti áthaladó csomagszám miatt ez nem kis feladat, implementációja nem triviális, és nagy körültekintést igényel, ezért a legtöbb megvalósítás ezt nem foglalja magába.



17. ábra: Tartalékolt tűzfal topológia

Léteznek megoldások klaszterezett tűzfalakra is, ahol nem egy, hanem több, erre a célra dedikált tűzfal végzi a munkát. A klaszter egy tagjának kiesése esetén a klaszter többi tagja megosztva veszi át a szerepét. Implementációja költséges, csak nagy szervezeteknél használatosak. A tartalékolt tűzfal topológia a 17. ábrán látható.

A tűzfalak ellenőrzése

Tűzfalépítés során kerül meghatározásra a házirend, amelyben megadott paraméterek alapján körültekintően kialakítjuk a szabályrendszerünket. Fontos, hogy ne fogadjuk el ellenőrzés nélkül, hogy a kigondolt logika pontosan fedi a megállmodott házirendet! Munkánkat minden egészítük ki ellenőrzésekkel! Vizsgáljuk meg, hogy adott kliens eléri-e a számára szükséges hálózati erőforrásokat, és csak kizárolag azokat! Ugyanígy ellenőrizzük a szervereknél is, hogy szolgáltatásaik csak a meghatározott kliensek számára elérhetők! A vizsgálatokhoz az egyszerű kapcsolódási tesztek mellett számtalan ingyenes és fizetős eszköz érhető el. A kapcsolódási teszt egyik legegyszerűbb módjához minden össze egy telnet kliens szükséges. Egy webszerver elérése például szinte bármely operációs rendszer alól tesztelhető a telnet www.example.com 80 parancs kiadásával. Ha a szolgáltatás elérhető, akkor sikeres TCP kapcsolat épül fel a kliens és a szerver között. Ha a szolgáltatást blokkolja a tűzfal, akkor a kapcsolódási kísérlet sikertelen lesz (pl.: időtúllépés miatt).

Ha kíváncsiak vagyunk, hogy egy szerver milyen szolgáltatásai érhetők el adott gépről, akkor a kliens gépen használhatjuk az ingyenes Nmap programot, amely többek között portszennelést tesz lehetővé. Képes nem csak a nyitott, de a szűrt – csak bizonyos irányból elérhető – portokat is felderíteni, valamint a kapott válaszcsomagok mintázata alapján megállapítani a vizsgált cél operációs rendszerét. A szoftverrel feltérképezhetők a hálózatban található gépek is. Fontos a körültekintő használat, tanácsos csak saját hálózatunk ellenőrzésére használni, hiszen a portszennelés sok helyen tiltott tevékenységnek és automatikusan támadásnak minősül!

Hasznos és ingyenes eszköz a Wireshark is. Létezik konzolos és grafikus felülete is, így alkalmazható szervereken és munkaállomásokon egyaránt. A Wireshark a kijelölt hálózati interfész teljes forgalmához hozzáfér, ennek eléréséhez rendszergazdai jogosultsággal kell futtatni. Képes a csomagokat elfogni, azokat elmenteni, fejlécüket és adatrészüket egyaránt elemzni, ismert protokollok esetén a szabályos működést ellenőrizni, teljes analizálást végezni. Az így kapott információk birtokában a rendszergazda képes lehet hálózati kommunikációs hibák felderítésére és megoldási terv kidolgozására.

A hálózati forgalmi adatok és a hálózati hozzáférés auditálása

Előfordulhat, hogy egy támadást már csak akkor veszünk észre, ha már sikeresen lezajlott. Megtörtéhet az is, hogy egy rosszhiszemű alkalmazott a biztonsági házirendet meg nem sértve adatokat szivárogtat ki harmadik fél számára. De adatszivárgás előfordulhat akaratlanul is, például kémprogrammal fertőzött gépekről. Az sem ritka, hogy egy oktatási intézmény a hallgatói számára publikus Wi-Fi hozzáférést biztosít, amely esetben a hálózat használata a nap bármely szakában történhet, időtartama eltérő lehet. Ugyanakkor az intézmény elvárja a hallgatóktól, hogy betartsák a házirendet és a hatályos jogszabályokat, törvényeket.

Ezekből az életszerű szituációkból is látszik, hogy a legrészletesebb házirend kidolgozása esetén is történhetnek biztonsági incidensek, vagy kaphat értesítést az intézmény, hogy hálózatából jog- vagy törvénysértő tartalmat tettek közzé (pl.: webes feltöltő űrlap vagy FTP használatával). A hálózat üzemeltetőjének ilyenkor szüksége lenne arra, hogy vissza tudja keresni, mi történt egy adott időszakban a hálózaton. Ezért fontos, hogy minden szerverszolgáltatás naplózva legyen az alkalmazási rétegen. Ha egy szervezet tagjai számára lehetséges a világhálón történő böngészés, akkor webforgalmukat csak a szervezet webproxy eszközén keresztül engedélyezzük, közvetlenül ne érhessenek el külső webszervereket! Az SMTP szerver is jegyezze fel a pontos levélmozgásokat minden két irányban, levelet küldeni és fogadni csak a kijelölt SMTP szerveren keresztül lehessen! A DNS szerver is rögzítse a feloldási kéréseket! A belső hálózat állandó gépei előre kialakított címzési logika alapján kapjanak IP-címet a szervezet DHCP szerverétől! Az időszakosan megjelenő (pl.: az előbb említett hallgatói) gépek kerüljönek külön hálózatba, mivel nem számítanak megbízhatónak, nincs közvetlen kontroll felettük! Legalább a határ-útválasztó jegyezze fel az összes hálózati kapcsolatot! Egy incideks okainak feltárása és a lezajlás pontos menetének kielemzése akkor lehetséges, ha rendelkezünk megfelelő adathalmazzal a vizsgált időszakról. Ehhez nyújt kiváló segédeszközt a Netflow.

Netflow a hálózati audit segítésére

A Netflow eredetileg egy olyan szoftveres eszköz, amelyet a Cisco fejlesztett ki, de ma már a legtöbb neves gyártó útválasztói és hálózati kapcsolóeszközei ismerik, emellett elérhető Linux, BSD, VMware vSphere 5 rendszerekre is. A Netflow jelenleg használatos verziói már ipari szabvánnyá váltak, működésük többek közt RFC dokumentumokban rögzített irányelvek alapján valósul meg. A Netflow rendszer három részre tagolódik:

- A Netflow Exporter az útválasztóban vagy hálózati kapcsolóban megvalósított adatszolgáltató szoftverkomponens, ami a hálózati folyamokról gyűjt adatokat. Jegyzi a forrás- és célcímeket, forrás- és célportot, protokollt, átvitt adatmennyiséget a kijelölt hálózati interfészen. A Netflow Exporter a beállított időérték leteltével vagy a hálózati folyam befejezével küldi tovább az adatokat.
- A Netflow Collector veszi át az adatokat a Netflow Exporter-től, majd letárolja megfelelő rekordszerkezetet használva a háttértáron. Általában pár percenként új állományt nyit, mivel kisebb állományokban gyorsabban lehet keresni a későbbiekbén.
- A Netflow Analyzer – kérésünkre – kiolvassa a letárolt rekordokat, és megjeleníti a vizsgált hálózati folyamokról kapott adatokat. Így az elemzést végző rendszergazda lekérdezésekkel intézhet bármely tárolt paraméterre vonatkozóan, de visszakövetheti egy adott időszak

hálózati történései is. Az elemzés időben bármikor történhet, amíg a gyűjtött adatokat megőrzük.

Fontos látni a hármas tagolás előnyeit, hiszen ennek köszönhetően:

- Az adatgyűjtés kis erőforrásigényteljes megvalósítható nagy forgalmú és nagy sávszélességű kapcsolatokon.
- Az adatok tárolásáról külső tároló gondoskodik, például egy Linuxot futtató PC, így nagy mennyiséggel adat olcsón tárolható, az adatok több hónapra visszamenőleg is megőrizhetők, nem a hálózati forgalmat bonyolító eszköz erősen véges memóriaterülete határozza meg a megőrzött adatok mennyiségét.
- Az elemzés a tárolt adatokon végezhetők, nem kell előre meghatározni, hogy mire leszünk kíváncsiak a későbbiekben. Szabadon elemezhetjük a rendelkezésre álló adatokat, ugyanakkor lehetőség van időzített jelentések generálására is.

A Netflow Analyzer kimenete nfdump használatával (példa):

Date	flow start	Duration	Proto	Src IP	Addr:Port	Dst IP	Addr:Port	Packets	Bytes	Flows
2012-07-01	16:40:18.941	1.121	TCP	195.39.12.117	:80 ->	172.21.3.8	:49156	3	132	1
2012-07-01	16:40:18.676	0.000	UDP	172.21.3.8	:58418 ->	193.6.33.2	:53	2	122	1
2012-07-01	16:40:18.915	0.000	UDP	193.6.33.2	:53 ->	172.21.3.8	:58418	2	352	1
2012-07-01	16:40:18.941	1.122	TCP	172.21.3.8	:49156 ->	195.39.12.117	:80	5	467	1
2012-07-01	16:40:18.340	0.000	ICMP	172.21.0.1	:0 ->	172.21.3.8	:8.0	1	48	1
2012-07-01	16:40:22.874	1.442	TCP	149.7.241.118	:80 ->	172.21.3.8	:49159	3	132	1
2012-07-01	16:40:22.873	1.444	TCP	172.21.3.8	:49159 ->	149.7.241.118	:80	5	471	1
2012-07-01	16:40:23.713	1.486	TCP	172.21.3.8	:49160 ->	149.7.241.118	:80	5	446	1
2012-07-01	16:40:23.713	1.482	TCP	149.7.241.118	:80 ->	172.21.3.8	:49160	3	132	1

Támadás bármikor érheti a hálózatot. A biztonsági felkészültség szinten tartásához fontos alkalmazni a már rendelkezésre álló biztonsági, naplázási és auditsegítő technológiákat. De ne feledjük el, hogy gyors és robbanásszerű fejlődésen megy keresztül az informatika, ezért nem elég, ha a hálózat biztonsági struktúrája egy adott technológiai fejlettséghez és felhasználói szokásokhoz kerül kidolgozásra. A házirendeket, szabályrendszereket, az alkalmazott IDS és IPS rendszereket folyamatosan hozzá kell igazítani a változó körülményekhez!

10. Vezeték nélküli hálózatok biztonsága

A vezeték nélküli hálózati kommunikációt a vezetékes kommunikáció kiegészítésére kezdték el kidolgozni. Tervezői nem számítottak akkora térhódításra, mint amit elért napjainkra. Kisegítő alternatívának szánták olyan szituációkra, amikor a vezetékes hálózat kiépítése nem volt lehetséges, vagy csak ad-hoc, ideiglenes hálózati hozzáférésre volt szükség. A kezdeti elgondolások nyújtotta szolgáltatási szint hamar szűkösnek bizonyult. Nem csak otthoni környezetben, de nagyvállalati szinten is egyre nagyobb piaci részesedésre tett szert. Kényelmes, gyorsan implementálható hálózati megoldássá, széles körben elterjedt kommunikációs eszközévé vált. A népszerűség és a vállalatok részéről érkező elvárások arra késztették a technológiai újítókat, hogy a vezeték nélküli kommunikációt több lépcsőben fejlesszék. Nézzük végig ezeket az állomásokat a biztonság oldaláról, kezdve a technológia alapjainak rövid áttekintésével!

Betekintés a vezeték nélküli technológia alapjaiba

Ugyan az első próbálkozások 1979-ben indultak infravörös fény alkalmazásával, rövidesen kézenfekvővé vált, hogy az átviteli közeget érdemes levegőben terjedő rádióhullámokra cserélni. Hosszabb előkészítő munka után 1997-ben szabványosította az IEEE (Institute of Electrical and Electronics Engineers) szervezet a 802.11 nevű szabványban azt a fajta vezeték nélküli, rádióhullámokkal működő kommunikációs technológiát, ami napjainkra meghódította a világot, később több utódszabvány követte. A szabványokban közös, hogy mindegyik 802.11 elnevezéssel kezdődik, de kiegészül különböző betűjelölésekkel, így például a kezdeti szabványt követte az IEEE 802.11a, majd az IEEE 802.11b, IEEE 802.11g, és jelenleg az IEEE 802.11n a manapság kapható legfejlettebb megoldás.

Kik kommunikálnak?

Több felállást tesz lehetővé a technológia. Az architektúra építőkövei az állomások. Azokat az eszközöket nevezzük állomásnak, melyek részt vesznek a kommunikációban, rendelkeznek megfelelő rádiós adóvevő komponenssel. Állomás lehet egy vezeték nélküli hálózati kártyával rendelkező munkaállomás vagy laptop, illetve a vezetékes hálózat oldaláról nézve tipikus összekötő elem: a vezeték nélküli hozzáférési pont (AP, Access Point). Manapság elterjedtek már az okostelefonok, tabletok, melyek ugyancsak rendelkeznek a megfelelő képességekkel, hogy egy vezeték nélküli hálózatban állomások lehessenek.

Azon eszközök kommunikálhatnak egymással, akik azonos csatornán tartózkodnak. Ezen eszközök gyűjtőneve a Basic Service Set (BSS). A kommunikáció rögzített protokollt követve zajlik.

Rádióhullámok és az interferencia

A kijelölt frekvenciatartomány a 2.4GHz volt kezdetekben. Az IEEE 802.11n hozott változást ebben, hiszen ez a szabvány megengedi már a 2.4GHz mellett az 5GHz-es tartomány használatát is 20, illetve 40MHz széles csatornákkal.

Fizikai tanulmányainkból ismerős lehet, hogy azonos frekvenciájú rádióhullámok találkozásakor a hullámok csillapíthatják, erősíthetik vagy teljesen kiolthatják egymást. Ezt nevezzük interferenciának. Ez nyilván nem kívánatos jelenség esetünkben. Az interferencia minimalizálása érdekében 20MHz-es csatornákat alakítottak ki. Az Európában érvényes szabályozások szerint 11 darab csatorna áll

rendelkezésre 2.4GHz-en és lényegesen több 5GHz-en, de ez utóbbi tartományban országokonként változik a használható csatornák száma.

Ha egy időben több állomás ad, akkor fellép az interferencia jelensége. A küldött csomag sérülhet, nem jut célba, újraküldése szükséges, ami erősen degradálja a hálózat teljesítményét. Ennek csökkentésére a közös osztott közegben időosztásos átviteli technikákat alkalmaznak. De a jelenséget így sem lehet teljesen elkerülni. Mivel azonos földrajzi helyen több, egymástól független hálózat is üzemelhet. Előfordulhat, hogy egymás hatósugarán belül találhatók olyan hálózatok, amelyek azonos csatornán üzemelnek, viszont egymás forgalmát szabályozni nem tudják. Törvényi előírások szerint a vezeték nélküli rádióadók legfeljebb 100mW teljesítménnyel sugározhatnak, hogy a hálózati frekvenciaátłapolódás kisebb területen léphessen fel. Tanácsos telepítés előtt feltérképezni, hogy mely csatornák foglaltak, és a saját hozzáférési pontokat olyan frekvenciára hangolni, amelyen a legkevesebb hálózat „érzékelhető”.

Azt azonban fontos megjegyezni, hogy egy esetlegesen szabad csatorna sem garantálja a zavartalan működést. Egyrészt az elterjedtség miatt bármikor megjelenhetnek új állomások az addig szabad csatornán, másrészt a 2.4GHz-es frekvenciatartományt használja a Bluetooth is, valamint számos háztartási gép (pl.: mikrohullámú sütő) is itt kelt hullámokat. Ezért ha az eszközök támogatják és lehetőség van rá, javasolt az 5GHz-es tartomány használata.

Látható, hogy már az átviteli közeg is számtalan lehetőséget ad a vezeték nélküli hálózat zavarására. A támadó érkezhet olyan eszközzel, ami folyamatosan zavaró jelet sugároz a megfelelő frekvencián, ezzel használhatatlanná téve a hálózatot. Továbbá érkezhet adatgyűjtési szándékkal, hiszen a csomagok a levegőben terjednek, a rádióadók pedig körsugárzó antennával rendelkeznek, így a tér minden irányába indulnak rádióhullámok. A támadónak nincs más dolga, mint elhelyezni eszközét a hatósugáron belül, és lehallgatni a forgalmat.

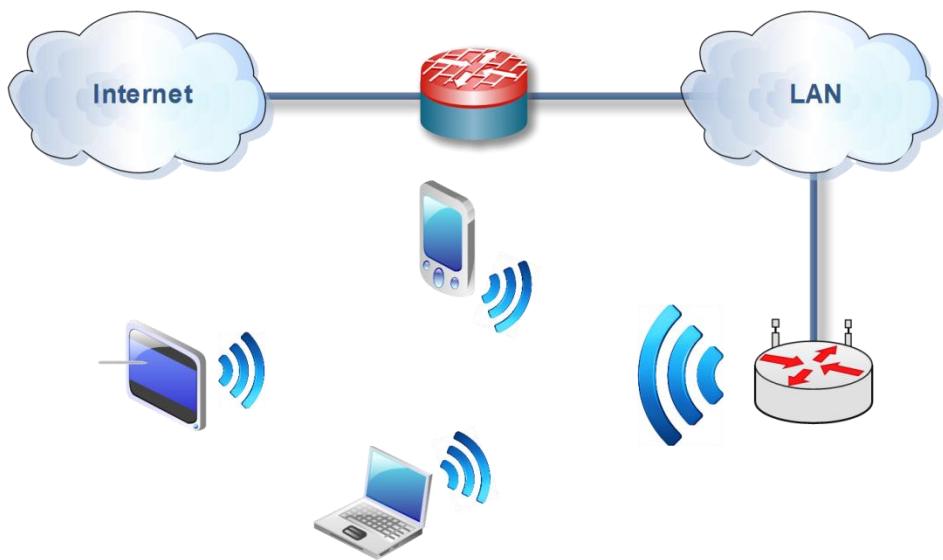
Topológiák

Alapvetően két topológia létezik: ad-hoc és infrastruktúra kialakítás. Ad-hoc hálózatról akkor beszélünk, amikor az állomások halmazában nem található AP. A kommunikációs partnerek pont-pont kapcsolatokat építenek fel szomszédjaikkal. Ez a topológia tartalmazhat mindössze két állomást, de megengedett több állomás csatlakoztatása is, a felek saját pont-pont kapcsolatot alakítanak ki a közelükben levő résztvevőkkel, a hálózat további tagjait a szomszédokon keresztül éri el. Az ad-hoc vezeték nélküli hálózatok működését a 18. ábra szemlélteti.



18. ábra: Ad-hoc vezeték nélküli hálózati topológia

Az infrastruktúra mód használatához szükséges legalább egy AP, ami egy szöveges hálózati azonosítót hirdet saját interfészének MAC-címével együtt. Az állomások az AP-hez csatlakoznak egy társítási folyamat keretén belül, melynek során egyeztetik a hálózat paramétereit, felépítik munkamenetüket. A hálózat bármely két állomása közti kommunikáció az AP eszközön keresztül történik. Jellemzően az AP kapcsolódik a vezetékes hálózathoz, így kapcsolja össze a vezeték nélküli eszközöket a vezetékes belső hálózattal, illetve az internettel. De nem csupán ez az előnye az AP-k alkalmazásának. Fontos szempont lehet az is, hogy egy AP-hez nem csak azonos szabványt támogató állomások csatlakozhatnak, hanem egyszerre asszociálhatnak a 802.11b, illetve a 802.11g szabványt támogató eszközök is. Az infrastruktúra mód működését a 19. ábra illusztrálja.



19. ábra: Infrastruktúra módban működő vezeték nélküli hálózat

Az infrastruktúra mód egy kiterjesztése a központilag felügyelt vezeték nélküli hálózat, amely kétféle komponensből áll: pehelysúlyú hozzáférési pontok (LWAP, Lightweight Access Point) csoportjából és egy központi irányítóeszközből, a kontrollerből.

Pehelysúlyú hozzáférési pont

A pehelysúlyú hozzáférési pontra jellemző, hogy nem képes önállóan működni, beüzemelése két módon történhet. Az egyik esetben a hálózat üzemeltetője az alapkonfiguráció részeként beállítja vezetékes interfészét, valamint hogy milyen címen és paraméterekkel fér hozzá a kontrollerhez. A másik esetben az eszköz a rendszergazda csatlakoztatja a hálózathoz, az DHCP protokollon keresztül megkapja a hálózati beállításokat és opcionálisan a kontroller fellelhetőségét, esetleg azonos szórási tartományon belül felderíti azt.

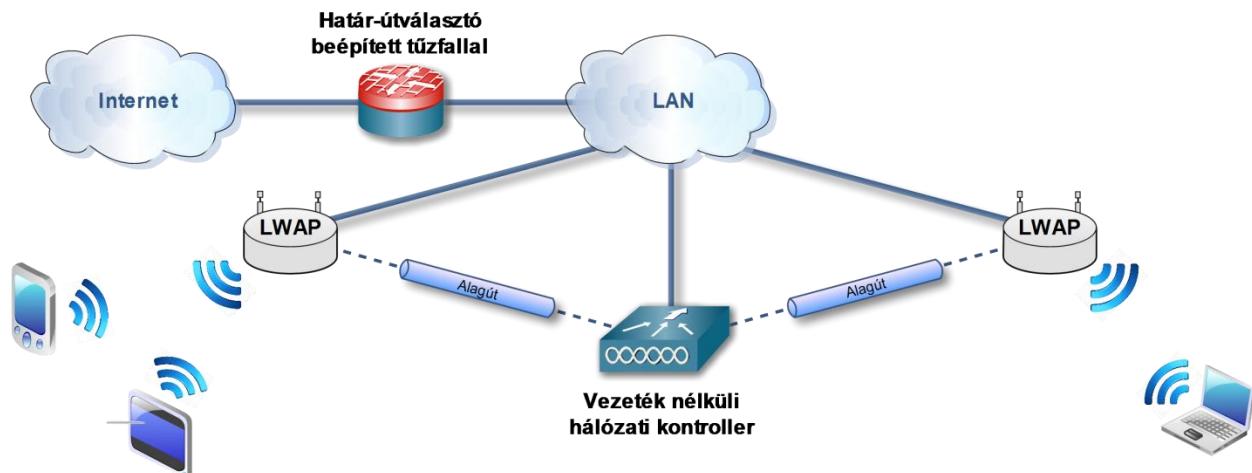
Mindkét esetben a hozzáférési pont kapcsolatot kezdeményez a kontroller irányába, amely egy titkosított alagutat hoz létre a két eszköz között. Az AP és a kontroller közti teljes hálózati kommunikáció ezután az alagúton keresztül történik.

A kontroller

A kontroller a központi vezérlőeszköz. A rendszergazda ezen konfigurálja fel a kialakítandó vezeték nélküli hálózatokat. A pehelysúlyú hozzáférési pontok a kontrollertől kapják meg a konfigurációs állományt, amely az összes, eszközre szabott beállítást tartalmazza. Átvárat valósít meg a vezetékes és a vezeték nélküli hálózatok közt. A forgalom nem a hozzáférési pontokon lép ki a belső hálózatba, hanem az alagúton keresztül továbbítódik a kontrollerig. Az architektúra lehetővé teszi, hogy hozzáférési szabályozást alakítsunk ki ezen a ponton.

Azon túlmenően, hogy egy helyen, egy felületen látható mindegyik eszköz, lekérdezhetővé válik az állapotuk és a hozzájuk csatlakoztatott kliensek, firmware frissítés indítható, naplózásra kerülhetnek központilag a kliensek forgalmi adatai vagy a kliensek vándorlása (roaming). Algoritmizálható az AP-k

adóteljesítményének automatikus állítása és csatornaválasztása. A központilag felügyelt vezeték nélküli hálózat működése a 20. ábrán látható.



20. ábra: Központilag felügyelt vezeték nélküli hálózat

Roaming, az állomások vándorlása

Roamingnak azt nevezik, amikor egy állomás az AP hatósugarának széléhez érve lecsatlakozik a hálózatról, majd a társítási folyamat keretében csatlakozik egy másik, erősebb jelerőséggel rendelkező AP-hoz, azaz egyik AP-ról átvándorol egy másikra. Központosított rendszer esetén ez a váltás milliszekundumok alatt lezajlódik. A kliens megtartja hálózati konfigurációját, nem szakadnak meg a hálózati kapcsolatai. A váltás gyorsasága különösen a hang- és video-átviteli alkalmazásoknál rendkívül lényeges.

A vezeték nélküli hálózat forgalmának titkosítása

Kezdetekben a vezeték nélküli forgalom titkosítás nélkül haladt az állomások között, melynek következtében a forgalom viszonylag egyszerűen lehallgathatóvá vált. Az adatátvitel nem volt védett a visszajátszásos vagy beékelődéses támadásokkal szemben. Nem állt rendelkezésre olyan mechanizmus, amivel a fogadó fél ellenőrizhette volna, hogy egy csomag adatrésze sértetlen és hiteles. Ez komoly gátat szabott annak, hogy a technológiát olyan környezetben alkalmazzák, ahol fontos az adatok biztonságos továbbítása. A mérnökök első válasza a felmerülő problémára a WEP titkosítási eljárás volt.

WEP

A WEP (Wired Equivalent Privacy) egy titkosítási eljárás, amely a hálózati forgalom biztonságos átvitelét két függvényel biztosítja:

- Az RC4 folyamtitkosító (stream cipher) függvény felel a csomagok bizalmas kezeléséért.
- A CRC-32 (Cyclic Redundancy Check) ellenőrzőösszeg-generáló függvény pedig a csomagok sértetlenségét ellenőrizi.

A WEP leírását tartalmazza az első 802.11 szabvány, ami 64 bit hosszú kulcs alkalmazását teszi lehetővé. A 64 bit hosszú kulcs két részből tevődik össze. Egy 40 bit hosszúságú kulcsból, ami tetszőleges karakterlánc lehet, és mi határozzuk meg. Ezt hagyományosan „jelszónak” nevezzük. A másik komponens egy 24 bit hosszúságú inicializáló vektorból áll. Az így kapott kulcsot használja fel az RC4 a kulcsfolyam (key stream) előállításához. A kulcsfolyam és a titkosítani kívánt adatfolyam között bitenként elvégzi a XOR logikai műveletet, így áll elő a WEP által titkosított folyam.

Későbbi törvényi változások lehetővé tették 128 bit hosszúságú kulcsok alkalmazását, ahol a 104 bit hosszúságú „jelszó” mellett 24 bit hosszúságú maradt az inicializáló vektor.

Az alkalmazott RC4 folyamtitkosító függvény rendkívül gyors. Viszont a 64, illetve 128 bit hosszúságú kulcsok alkalmazása, valamint a megkötés, hogy a 40, illetve 104 bit hosszúságú kulcsrész csak az [A-Za-z], valamint [0-9] karakterosztályokból állhatott elő, beszűkíti a generálható kulcsok terét.

Nem kellett sokat vájni, és megjelentek az első eljárások a WEP titkosítás feltörésére. Nagyságrendileg elegendő 40-60000 csomag elfogása a kulcs visszafejtéséhez. Ekkora csomagmennyiség percek alatt összegyűjthető, főleg, hogy a vezeték nélküli hálózat protokollja kijátszható, könnyen generálható hálózati forgalom. Ennek következtében a WEP bármely kereskedelmi forgalomban kapható vezeték nélküli hálózati kártya és megfelelő, szabadon elérhető szoftver felhasználásával pár perc alatt feltörhető, az általa védett hálózathoz illetéktelen hozzáférés szerezhető. Épp ezért a WEP használata erősen nem javasolt!

WPA és WPA2

A WPA (Wi-Fi Protected Access) és a WPA2 (Wi-Fi Protected Access II) kívánt megoldást nyújtani a WEP gyengeségeire, ezek kidolgozását a „The Wi-Fi Alliance” vállalta fel.

A WPA áthidaló megoldást nyújtott, hiszen az eszközök felkészíthetők voltak az új eljárás használatára egy firmware frissítés elvégzésével. A WPA2 alkalmazásához már a rádióeszközök hardverfelépítésén is változtatni kellett.

A WPA az IEEE 802.11i szabvány egy részét valósítja meg, különösképp a TKIP (Temporal Key Integrity Protocol) eljárást. A munkamenetenként statikus WEP kulccsal szemben a TKIP csomagonként generál egy 128 bit hosszúságú titkosítási kulcsot a kulcsfolyam legenerálásához.

A csomagok integritás-ellenőrzésében is történt változtatás, a CRC szerepét a Michael algoritmus vette át, amely sokkal nehezebben kijátszható ellenőrzőösszeget adott, viszont erőforrásigényét kielégítettek a piacra akkoriban forgalomban levő vezeték nélküli hálózati kártyák.

A WPA2 újabb algoritmusokat alkalmaz. A csomagok titkosításához és hitelesítéséhez az AES (Advanced Encryption Standard) eljárásra épülő CCMP (Counter Cipher Mode with Block Chaining Message Authentication Code Protocol) algoritmust alkalmazza. A kereskedelmi termékek mégsem a CCMP megnevezést használják a konfigurációs felületen, hanem az AES megnevezést.

Előre kiosztott kulcsok, WPA-PSK mód

A WPA-PSK (WPA Pre-Shared Key) a WPA, illetve a WPA2 azon működési módja, amikor a kulcs előállításához tetszőleges ASCII karaktereket tartalmazó saját „jelszót” használunk. Ezt a módot WPA-Personal néven is ismerhetjük. Otthoni, kisvállalati telepítések során ez a legszélesebb körben alkalmazott működési mód.

WPA-802.1x mód

A WPA-802.1x mód lehetővé teszi, hogy központi RADIUS hitelesítési szerveren keresztül történjen a kliensek azonosítása. A 802.1x egy olyan folyamat, amelynek során a sikeres azonosítás előtt kizárolag hitelesítési keretek utazhatnak az Ethernet hálózaton. A keretek csak az azonosítandó eszköz és az azonosítást elvégző hálózati eszköz között közzélehetnek. A központi adatbázisból történő lekérdezést az azonosítást végző eszköz hajtja végre, RADIUS protokollon keresztül.

Mit tanácsos, és mi nem tanácsos?

- Soha ne építsünk olyan vezeték nélküli hálózatot, ami nem védett legalább WPA2-PSK titkosítási móddal.
- Nem tanácsos olyan hálózathoz csatlakozni, ami nem használ titkosítást. Ha mégis elkerülhetetlen, akkor olyan alkalmazások használata javasolt, amik alkalmazási (L7) rétegen gondoskodnak a biztonságos kommunikációról (pl. HTTPS, SSH, IMAPS).
- WEP titkosítást nem tanácsos alkalmazni, mert csak látszólagos biztonságérzetet kelt.
- A vezeték nélküli hálózat olyan csatornán üzemeljen, amelyen a legkevesebb a zavaró jel.
- Törekedni kell az 5GHz frekvenciasáv használatára, amennyiben a csatlakoztatni kívánt eszközök erre felkészítettek.

Bár számos helyen keletkezhet támadási felület a vezeték nélküli hálózatokban, gondos tervezés és körültekintő implementálás esetén a hálózat biztonsággal használható és üzemeltethető. Mégis a rendszer üzemeltetőjének fel kell készülnie arra is, hogy a hálózatot támadás éri, ezért itt is fontos szerepet kap a hálózat naplózása és az audit. Biztonsági kérdésekben a [11] könyv adhat további ismeretanyagot.

11. Irodalomjegyzék

- [1] Andrew S. Tanenbaum: Számítógép-hálózatok, Panem Kiadó, 2004.
- [2] Ryan Trost: Practical Intrusion Analysis (Prevention and Detection for the Twenty-First Century), Addison-Wesley, 2009.
- [3] Stephen Northcutt , Judy Novak: Network Intrusion Detection (3rd Edition), New Riders, 2002.
- [4] Matt Fearnlow , Stephen Northcutt , Karen Frederick , Mark Cooper: Intrusion Signatures and Analysis, New Riders, 2001.
- [5] Niels Provos, Thorsten Holtz: Virtual Honeypots (From Botnet Tracking to Intrusion Detection), Addison-Wesley, 2007.

- [6] Jon C. Snader: VPNs Illustrated (Tunnels, VPNs, and IPsec), Addison-Wesley, 2005
- [7] J. Michael Stewart: Network Security, Firewalls, and VPNs, Jones & Bartlett Learning, 2010.
- [8] Greg Holden: Guide to Firewalls and Network Security (Intrusion Detection and VPNs), Course Technology, 2003.
- [9] Michael E. Whitman, Herbert J. Mattord, Andrew Green: Guide to Firewalls and VPNs, Delmar Cengage Learning, 2011.
- [10]Oleg Kolesnikov, Brian Hatch: Building Linux Virtual Private Networks (VPNs), Sams, 2002.
- [11]John R. Vacca: Guide to Wireless Network Security, Springer, 2006.