10.

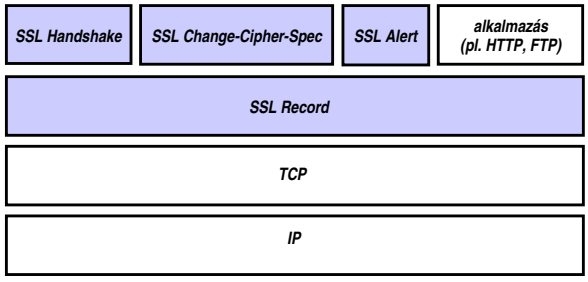
Internet biztonsági protokollok

Ebben a fejezetben három internet biztonsági protokollt mutatunk be. Ezek az SSL, az IPSec és a PGP protokollok. E három protokoll több szempontból is érdekes példának tekinthető. Egyrészt azért, mert e három protokoll a hálózati architektúra három különböző rétegében helyezkedik el: a PGP az alkalmazási, az SSL a szállítási, az IPSec pedig a hálózati réteghez tartozik. Ezért ezen protokollok azt példázzák, hogy a biztonsági szolgáltatásokat minden rétegben meg lehet valósítani, természetesen figyelembe véve az adott réteg sajátosságait.

Másrészt, ez a három protokoll három különböző típusú kommunikációs szolgáltatás biztonságát igyekszik megoldani. Az SSL feladata egy megbízható, kapcsolatorientált szolgáltatás (TCP) biztonságossá tétele, az IPSec feladata pedig egy megbízhatatlan, datagram szolgáltatás (IP) védelme. A PGP egyik fő alkalmazási területe az elektronikus levelezés biztonságossá tétele. Az e-mail sajátossága, hogy a kommunikáció aszinkron jellegű, azaz a küldő és a vevő nincs egy időben jelen. Látni fogjuk majd, hogy a kommunikációs szolgáltatás típusa nagy mértékben befolyásolja a védelemre szánt biztonsági protokoll tulajdonságait.

10.1. SSL (Secure Socket Layer)

A Netscape még a web korszak elején, a 90-es évek közepén felismerte, hogy a web technológia mindennapi életünk részévé fog válni, és olyan feladatokra is használni fogjuk majd, melyek biztonsági követelményeket is támasztanak. A Netscape célja tehát az volt, hogy a webböngésző és a web-



10.1. ábra. Az SSL illeszkedése az internet protokoll-architektúrájába

szerver közötti kapcsolatot biztonságossá tegye. Ezt a feladatot meg lehetett volna oldani a web protokollja, a HTTP szintjén is (voltak ilyen próbálkozások is, például Secure-HTTP). A Netscape azonban egy olyan általános célú protokollt fejlesztett ki, ami lehetővé teszi biztonságos TCP kapcsolatok kiépítését tetszőleges, amúgy TCP-t használó alkalmazások (például HTTP, FTP, SMTP stb.) között. Mivel a TCP programozói interfésze az ún. *socket* absztrakcióra épül, ezért az új protokollt Secure Socket Layer-nek, röviden SSL-nek nevezték el. Mára az SSL de facto szabvánnyá vált, szinte minden webböngésző és webszerver implementáció támogatja. Sőt, TLS néven (Transport Layer Security) és kisebb javításokkal, az SSL megindult az internet szabvánnyá válás útján is (lásd RFC 2246).

Az SSL protokoll tehát a TCP réteg felett és az alkalmazások alatt helyezkedik el az internet protokoll-hierarchiájában. Az SSL-lel kiegészített protokoll architektúrát a 10.1. ábra szemlélteti. Mint az az ábrán is látható, az SSL protokoll négy alprotokollból áll, melyek a következők:

- *Record protokoll*: A Record protokoll feladata a kliens és a szerver (például egy webböngésző és egy webszerver), valamint a felsőbb SSL protokoll entitások (Handshake, Change-Cipher-Spec és Alert) közötti kommunikáció védelme, mely titkosítást, integritásvédelmet és üzenetvisszajátszás elleni védelemet jelent.
- *Handshake protokoll*: A kliens és a szerver a Handshake protokoll segítségével egyezteti a Record protokollban használt kriptográfiai algoritmusokat és az algoritmusok paramétereit, beleértve a kapcsolatkulcsokat is. A Handshake protokoll további feladata a felek hitelesítése. Tipikusan a szerver mindig hitelesíti magát, a kliens hitelesítés azonban opcionális.

- *Change-Cipher-Spec protokoll*: A Change-Cipher-Spec protokoll egyetlen üzenetből áll, mely a Handshake protokoll kulcscsere részének végét jelzi. Ezen üzenet elküldése után az adott fél az új algoritmusokat és kulcsokat kezdi használni küldésre, a vétel azonban még mindig a Handshake előtti állapot szerint történik. Mikor az adott fél megkapja a másik fél Change-Cipher-Spec üzenetét, akkor a vételi állapotát megváltoztatja, azaz vételre is az új algoritmusokat és kulcsokat kezdi használni.
- *Alert protokoll*: Az Alert protokoll feladata a figyelmeztető- és hibaüzenetek továbbítása.

A továbbiakban a Record és a Handshake protokollokat tárgyaljuk részletesen. A Change-Cipher-Spec protokollra a Handshake protokoll leírásánál még utalni fogunk. Az Alert protokollal nem foglalkozunk.

10.1.1. Az SSL Record protokoll

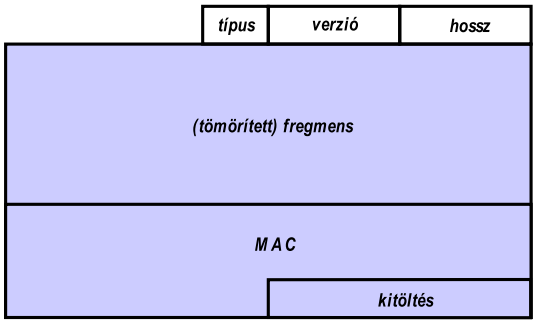
Az SSL Record protokoll a felsőbb protokoll rétegektől érkező üzeneteket a következő lépéseken keresztül dolgozza fel:

1. a hosszú üzeneteket fragmentálja,
2. a fragmenseket tömöríti,
3. minden tömörített fragmenst fejléccel lát el,
4. a fejléccel ellátott, tömörített fragmensre üzenethitelesítő kódot (MAC) számol, és azt a fragmenshez csatolja,
5. majd az üzenethitelesítő kóddal ellátott fragmenst rejtjelezi.

A vevő a feldolgozást értelemszerűen ellentétes sorrendben végzi. A feldolgozás során használt tömörítő, üzenethitelesítő és rejtjelezési algoritmusokban, paraméterekben, illetve kulcsokban a Handshake protokoll végrehajtása során egyeznek meg a felek.

A fenti lépések eredményeként előálló Record üzenet formátumát a 10.2. ábra mutatja. A fejléc három mezőt tartalmaz, melyek a következők:

- *típus (type)*: A típusmező értéke utal arra, hogy a Record üzenet belsejében melyik felsőbb protokoll üzenete (vagy üzenetfragmense) található. Ennek megfelelően a típus mező négyféle értéket tartalmazhat: SSL Handshake, SSL Change-Cipher-Spec, SSL Alert és alkalmazás.
- *verzió (version)*: A verziómező az éppen használt SSL verziót tartalmazza. A jelenlegi legmagasabb verzió a 3.0, mi is ezt tárgyaljuk. Néhány alkal-



10.2. ábra. Az SSL Record protokoll üzenet formátuma

mazás még mindig támogatja az SSL 2.0 verzióját is. Ez azonban komoly biztonsági hiányosságokkal rendelkezik, ezért használata kerülendő.

- *hossz (length)*: A hossz mező a Record üzenet belsejében található (tömörített) fragmens hosszát tartalmazza bájtban mérve.

Az üzenethitelesítő kód generálása a HMAC egy korai verziójával történik az alábbiak szerint:

$$MAC = H(K_{write}^{MAC} \mid pad_2 \mid H(K_{write}^{MAC} \mid pad_1 \mid seqnum \mid type \mid length \mid payload)),$$

ahol:

- H az MD5 vagy a SHA-1 hash függvény, attól függően, hogy a Handshake során miben egyeztek meg a felek.
- K_{write}^{MAC} a MAC érték generálásához használt titkos kulcs, melyet csak a kliens és a szerver ismer. Az SSL különböző irányokban különböző kulcsot használ, azaz a klienstől a szerver felé tartó üzenetek integritását egy $K_{C\to S}^{MAC}$ kulccsal, a szervtől a kliens felé tartó üzenetek integritását pedig egy $K_{S\to C}^{MAC}$ kulccsal védi. A kliens a $K_{C\to S}^{MAC}$ kulcsra K_{write}^{MAC} néven hivatkozik (küldésre használt MAC kulcs), a $K_{S\to C}^{MAC}$ kulcsra pedig a K_{read}^{MAC} néven (vételre használt MAC kulcs). A szerver oldalon az egyes kulcsok szerepe nyilván fordított.
- pad_1 és pad_2 két konstans bájtsorozat.
- *seqnum* az üzenet sorszáma. Az SSL implicit üzenetsorszámot használ, ami azt jelenti, hogy a sorszám nem jelenik meg explicit mezőként az üzenetben (lásd 10.2. ábra), de a MAC számításban felhasználják aktuális

értékét, melyet mindkét fél lokálisan számon tart. Ezt azért lehet így megtenni, mert az SSL a TCP protkoll felett helyezkedik el, és a TCP normális körülmények között biztosítja az üzenetek sorrendhelyes vételét. Ezért általában igaz az, hogy mindig a várt sorszámú üzenetet veszik a felek. Ha valamilyen támadás vagy hiba folytán nem a várt sorszámú üzenet érkezik meg (ami tehát abnormális jelenség), akkor a MAC ellenőrzés sikertelen lesz, és a vevő bontja a kapcsolatot.

- *type* és *length* az üzenet fejlécében található típus és hossz mezők értéke.
- *payload* az üzenetben található (tömörített) fragmens.

A 10.2. ábrán a Record üzenet rejtjelezett részét szürke színnel jelöltük. Mint látható, a fejléc kivételével az egész üzenet rejtjelezve van. Az SSL alapértlemezett (default) rejtjelező algoritmusa az RC4 kulcsfolyamatos rejtjelező, de a Handshake protokoll végrehajtása során a felek más algoritmusban is megegyezhetnek. Az SSL támogatja még az RC2, a DES, a három kulcsos 3DES, az IDEA és a Fortezza blokkrejtjelezők CBC módban történő használatát. Amennyiben a felek valamelyik blokkrejtjelező használatában egyeznek meg, úgy a rejtjelezés előtt minden Record üzenetet ki kell tölteni, hogy hossza a rejtjelező blokkméretének egész számú többszöröse legyen. Az SSL az 5.1. algoritmus szerinti kitöltési sémát használja. A kitöltés bájtjai a MAC után kerülnek az üzenetbe. Kulcsfolyamatos rejtjelező használata esetén nincs kitöltés.

Blokkrejtjelező használata esetén, a CBC mód miatt, szükség van *IV*-re is. Az első üzenet rejtjelezéséhez használt *IV*-t a Handshake során generálják a felek (a kulcsokkal együtt). Minden további üzenetnél az előző üzenet utolsó rejtjeles blokkját használják *IV*-nek. Ez gyakorlatilag azt jelenti, mintha a kapcsolat során elküldött összes üzenetet egyetlen nagy üzenetként CBC módban rejtjeleznék (az üzenetenkénti kitöltéstől eltekintve).

10.1.2. Az SSL viszony és az SSL kapcsolat

Mielőtt a Handshake protokoll részletes ismertetésébe kezdenénk, be kell vezetnünk az SSL viszony (session) és az SSL kapcsolat (connection) fogalmát és a közöttük fennálló kapcsolatot. Az SSL két fél között több párhuzamos viszony kialakítását teszi lehetővé, és minden viszonyon belül több kapcsolat lehetséges. Ezen lehetőségek közül azonban a legtöbb implementáció csak az egy viszonyon belüli több kapcsolatot támogatja, a több párhuzamos viszony lehetőségét nem. Tehát mi is azt fogjuk most feltételezni, hogy a felek között egy viszony van, és a viszonyon belül több kapcsolat lehetséges.

Mind a viszony, mind a kapcsolatok a felek közös állapotának egy-egy részletét írják le. A viszony tartalmazza az állapot azon részleteit, amit a viszonyon belüli kapcsolatok megosztanak, közösen használnak. A viszony része például a viszony azonosító, a felek nyilvános kulcs tanúsítványa (ha rendelkeznek ilyennel), a tömörítő algoritmus, a rejtjelező és MAC algoritmus, az ezen algoritmusok által használt méretek (például a MAC kulcs hossza) és egy mestertitok (master secret). A viszony része még az ún. „is resumable" jelzőbit, melynek 1 értéke azt jelenti, hogy az adott viszonyon belül még létrehozható új kapcsolat, 0 értéke pedig azt, hogy új kapcsolat létrehozása nem lehetséges. Ez a jelzőbit akkor válik 0 értékűvé, ha valamelyik, a viszonyhoz tartozó kapcsolatban egy fatális hiba keletkezik (például egy üzenet MAC értékének ellenőrzése sikertelen). Ekkor a még folyamatban levő kapcsolatok folytatódnak, de a jelzőbit nullázása azt eredményezi, hogy új kapcsolatot a viszonyon belül már nem lehet létrehozni.

A kapcsolat részét képezi például a kapcsolatban használt két rejtjelező kulcs, a két MAC kulcs (különböző irányokban különböző kulcsokat használ a protokoll mind a rejtjelezéshez, mind a MAC számításhoz), a különböző irányokban használt üzenetsorszámok, és blokkrejtjelező használata esetén az *IV*-k. Fontos tehát látni, hogy minden kapcsolatnak saját kulcsai vannak, ezért a kulcsok a kapcsolat részét képezik, ugyanakkor az egy viszonyhoz tartozó kapcsolatok ugyanazokat az algoritmusokat használják, és a mestertitok is közös, amiből a kapcsolatkulcsokat generálják. Ezért az algoritmusok és a mester titok a viszonyhoz tartoznak.

10.1.3. Az SSL Handshake protokoll

A Handshake protokoll feladata a két fél közötti viszony felépítése, vagy ha az már létezik, és annak „is resumable" jelzőbitje 1 értékű, akkor a viszonyon belül egy új kapcsolat létrehozása. Az SSL Handshake protokoll egy komplex protokoll. A komplexitás legfőbb oka az, hogy a Handshake protokoll többféle kulcscsere módszer használatát is támogatja, és a Handshake üzenetek értelmezése az egyes módszerek esetén más és más. A Handshake protokoll általános váza a 10.3. ábrán látható, ahol *C* jelöli a klienst, *S* pedig a szervert.

A Handshake protokoll négy fázisra tagolódik. Az első fázisban (1. és 2. üzenet) történik meg az algoritmusok egyeztetése, beleértve a Handshake hátralevő részében használt kulcscsere módszert is. Ezen kívül a felek az első fázisban kicserélnek két frissen generált véletlen számot is, melyet a további számításoknál használni fognak majd. A második fázisban (3–6. üzenetek) a

SSL Handshake protokoll		
(1)	$C \rightarrow S$:	**client-hello**
(2)	$S \rightarrow C$:	**server-hello**
(3)	$S \rightarrow C$:	certificate
(4)	$S \rightarrow C$:	server-key-exchange
(5)	$S \rightarrow C$:	certificate-request
(6)	$S \rightarrow C$:	**server-hello-done**
(7)	$C \rightarrow S$:	certificate
(8)	$C \rightarrow S$:	**client-key-exchange**
(9)	$C \rightarrow S$:	certificate-verify
	$C \rightarrow S$:	**change-cipher-spec**
(10)	$C \rightarrow S$:	**client-finished**
	$S \rightarrow C$:	**change-cipher-spec**
(11)	$S \rightarrow C$:	**server-finished**

10.3. ábra. Az SSL Handshake protokoll váza. A vastagon szedett üzenetek kötelezőek, a többi opcionális (a korábbi üzenetek tartalmától függ, hogy kell-e őket küldeni vagy sem).

szerver a kiválasztott módszernek megfelelően végrehajtja a kulcscsere ráeső részét, míg a kliens a harmadik fázisban (7–9. üzenetek) teszi meg ugyanezt. A harmadik fázis után, az addig kicserélt információkat felhasználva, mindkét fél előállítja az új kapcsolatkulcsokat. A negyedik fázisban (10. és 11. üzenet) a felek áttérnek az új algoritmusok és kulcsok használatára. Mindkét fél egy finished üzenet küldésével fejezi be a protokollt, mely az első olyan üzenet, ami már az új algoritmusokat használva, az új kulcsokkal van kódolva.

Vegyük észre, hogy a change-cipher-spec üzeneteket a fenti leírásban nem számoztuk meg. Ez azért van, mert ezek az üzenetek a Change-Cipher-Spec protokollhoz tartoznak és nem a Handshake protokoll részei. Erre az SSL analízisénél még vissza fogunk térni.

A client-hello üzenet a következő információkat tartalmazza:

- *kliens verzió*: A kliens tájékoztatja a szervert az általa támogatott legmagasabb SSL verzió számáról.
- *véletlenszám*: A kliens generál egy friss véletlenszámot, és elküldi azt a szervernek.

- *viszony azonosító*: Ha a kliens azt szeretné, hogy az új kapcsolat egy már létező viszonyon belül jöjjön létre, akkor elküldi ezen viszony azonosítóját a szervernek. Ha a kliens új viszonyt akar létrehozni, akkor ez a mező üres.
- *biztonsági algoritmusok*: A kliens tájékoztatja a szervert az általa támogatott biztonsági algoritmusokról. Ez a mező egy, a kliens preferenciái szerint rendezett listát tartalmaz, ahol a lista elemei a támogatott algoritmuskombinációk azonosítói. Egy ilyen listaelem lehet például az SSL-RSA-with-3DES-EDE-CBC-SHA azonosító, mely a következő algoritmusok kombinációját jelöli:
  - RSA alapú kulcscsere módszer,
  - 3DES rejtjelezés, EDE konfigurációban, CBC módban,
  - SHA hash függvény és arra épülő MAC.

  A megjelölt algoritmusok közvetve definiálják a paramétereket is (például 3DES esetén az *IV* méret 64 bit, SHA esetén a MAC mérete 160 bit stb).
- *tömörítő algoritmusok*: A kliens tájékoztatja a szervert az általa támogatott tömörítő algoritmusokról. Az előző mezőhöz hasonlóan ez a mező is egy preferencia szerint rendezett listát tartalmaz, ahol azonban a lista elemeit tömörítő algoritmusok azonosítói alkotják.

A server-hello üzenet ugyanazokból a mezőkből áll, mint a client-hello üzenet, csak a mezők értelmezése kicsit más:

- *szerver verzió*: A szerver azt a legmagasabb verziót választja, melyet mind a kliens, mind a szerver támogat, és erről tájékoztatja a klienst.
- *véletlenszám*: A szerver is generál egy (a kliensétől független) friss véletlenszámot, és elküldi azt a kliensnek.
- *viszonyazonosító*: Ha a kliens javasolt egy viszonyazonosítót, akkor a szerver ellenőrzi, hogy az adott viszonyon belül lehet-e még új kapcsolatot létrehozni („is resumable” bit értéke 1). Ha igen, akkor a szerver az adott viszony azonosítójával válaszol. Ha a viszonyon belül már nem lehet új kapcsolatot létrehozni, vagy a kliens nem javasolt viszonyazonosítót, akkor a szerver generál egy új viszonyazonosítót, és azt küldi el a kliensnek.
- *biztonsági algoritmusok*: A szerver választ egy algoritmus-kombinációt a kliens listájáról, és csak a kiválasztott kombináció azonosítóját küldi vissza a kliensnek.

- *tömörítő algoritmusok*: A szerver választ egy algoritmust a kliens listájáról, és annak azonosítóját küldi vissza a kliensnek.

A további üzenetek értelmezése attól függ, hogy milyen kulcscseremódszerben egyeztek meg a felek. Az SSL öt kulcscseremódszert támogat. Ezek a következők:

- *RSA alapú*: RSA alapú kulcscsere esetén a kliens generál egy 48 bájt méretű véletlen blokkot, amit a szerver nyilvános RSA kulcsával kódolva elküld a szervernek. Ebből a véletlen blokkból aztán mind a kliens, mind a szerver előállítja a közös mestertitkot.
- *fix Diffie–Hellman*: Fix Diffie–Hellman-kulcscsere esetén a felek a Diffie–Hellman-algoritmust használják, és az ennek eredményeként kialakult közös értékből generálják a mestertitkot. A fix jelző arra utal, hogy a szerver Diffie–Hellman-paraméterei (p, g, $g^x \bmod p$) fixek, azokat egy hitelesítésszolgáltató aláírta, és az erről szóló tanúsítványt a kliens ellenőrizni tudja.
- *egyszer használatos Diffie–Hellman*: Az előző módszerhez hasonlóan Diffie–Hellman-algoritmust használnak a felek, de a szervernek nincsenek fix, aláírt Diffie–Hellman-paraméterei. Helyette a szerver egyszer használatos paramétereket generál, és azokat RSA vagy DSS aláíró kulcsával aláírva juttatja el a kliensnek. A kliens mind a fix, mind az egyszer használatos Diffie–Hellman-kulcscsere esetén egyszer használatos Diffie–Hellman nyilvános értéket generál a szerver p és g paramétereit használva.
- *anonim Diffie–Hellman*: Ezen módszer használata esetén a felek az eredeti, hitelesítés (aláírás) nélküli Diffie–Hellman-algoritmust hajtják végre. Ezen módszer használata nem tanácsos, ha az aktív támadások veszélyét nem lehet kizárni.
- *Fortezza*: A Fortezza kulcscsere protokoll a Netscape saját fejlesztésű módszere, amivel itt most nem foglalkozunk.

A Handshake protokoll második fázisának első üzenete a certificate üzenet. Ez egy hitelesítésszolgáltató által aláírt tanúsítvány (vagy ilyen tanúsítványok lánca), amit az anonim Diffie–Hellman kivételével, minden kulcscsere módszer esetén elküld a szerver. A tanúsítvány tartalma azonban változó. Tartalmazhat egy nyilvános RSA rejtjelező kulcsot (RSA alapú kulcscsere), vagy egy RSA, vagy DSS aláírás ellenőrző kulcsot (RSA alapú kulcscsere, vagy egyszer használatos Diffie–Hellman), vagy a szerver Diffie–Hellman paramétereit (fix Diffie–Hellman).

A következő üzenet a server-key-exchange üzenet. Ezt csak abban az esetben küldi a szerver, ha az előző lépésben küldött tanúsítvány nem tartalmaz elegendő információt a kulcscsere befejezéséhez. Tipikusan, ha a tanúsítvány csak egy aláírásellenőrző kulcsot tartalmazott, akkor a szerver a server-key-exchange üzenetben küldi el a kliensnek a rejtjelezésre alkalmas RSA kulcsot (RSA alapú kulcscsere), vagy az egyszer használatos Diffie–Hellman paramétereit. A server-key-exchange üzenetet a szerver digitálisan aláírja. Ehhez először a server-key-exchange üzenetben elküldött kulcscsere paramétereket és az első fázisban kicserélt véletlenszámokat összefűzi, majd az eredmény hash értékén képezi az aláírást.

Ha a szerver szeretné hitelesíteni a klienst (ez opcionális), akkor küld egy certifi cate-request üzenetet, melyben specifikálja, hogy milyen tanúsítványokat vár a klienstől. Végül, a második fázist a server-hello-done üzenet zárja le, melyben a szerver jelzi a kliensnek, hogy befejezte a kulcscsere ráeső részét.

Ha a szerver küldött certifi cate-request üzenetet, azaz szeretné hitelesíteni a klienst, akkor a protokoll harmadik fázisa egy certifi cate üzenettel kezdődik, melyben a kliens elküldi a szerver által kért tanúsítványokat.

A következő üzenet a client-key-exchange, melyet mindig küld a kliens. A megegyezett kulcscsere módszertől függően, a client-key-exchange üzenet tartalmazhatja a kliens által generált és a szerver nyilvános RSA kulcsával rejtjelezett 48 bájtos véletlen blokkot (RSA alapú kulcscsere), vagy a kliens egyszer használatos Diffie–Hellman nyilvános értékét.

Végül, ha a kliens küldött certifi cate üzenetet, akkor a harmadik fázist a certifi cate-verify üzenettel fejezi be. Ez az üzenet tartalmazza az összes eddigi (a kliens által vett és küldött) Handshake üzenet hash értékét digitálisan aláírva, ahol az aláírás a kliens által korábban küldött certifi cate üzenetben található aláírásellenőrző kulccsal ellenőrizhető.

A Handshake protokoll harmadik fázisa után a kliens és a szerver az következő módon előállít egy közös K mestertitkot: Jöljük K'-vel a kliens által generált 48 bájtos véletlen blokkot, melyet RSA alapú kulcscsere esetén a kliens a szerver nyilvános RSA kulcsával rejtjelezve juttat el a szervernek. Fix, egyszer használatos vagy anonim Diffie–Hellman-kulcscsere esetén pedig K' jelölje a Diffie–Hellman-algoritmus $g^{xy} \bmod p$ végeredményét. Ekkor

$$\begin{aligned} K = {} & \mathrm{MD5}(K' \mid \mathrm{SHA}(\text{"A"} \mid K' \mid N_C \mid N_S)) \mid \\ & \mathrm{MD5}(K' \mid \mathrm{SHA}(\text{"BB"} \mid K' \mid N_C \mid N_S)) \mid \\ & \mathrm{MD5}(K' \mid \mathrm{SHA}(\text{"CCC"} \mid K' \mid N_C \mid N_S)), \end{aligned}$$

ahol N_C és N_S az első fázisban kicserélt véletlen számok, "A", "BB", és "CCC" pedig az "A", a "B" illetve a "C" ASCII karakterekből alkotott egy, kettő, illetve három hosszú karakterláncok. Mivel az MD5 hash függvény kimenetének mérete 16 bájt, ezért a három hash érték összefűzéséből nyert K mestertitok 48 bájt hosszú.

A Handshake protokoll negyedik fázisában a felek egy-egy finished üzenetet küldenek egymásnak. A finished üzenetek célja az egész Handshake hitelesítése, azaz az esetleges aktív támadások detektálása. Ezen cél elérése érdekében mindkét fél kiszámítja az összes vett és küldött Handshake üzenet, valamint az imént kiszámolt mestertitok összefűzésével nyert „szuper üzenet” hash ertékét. Ez a következő formula szerint történik:

$$\begin{aligned}&\mathrm{MD5}(K \mid pad_2 \mid \mathrm{MD5}(msgs \mid sender \mid K \mid pad_1)) \mid \\ &\mathrm{SHA}(K \mid pad_2 \mid \mathrm{SHA}(msgs \mid sender \mid K \mid pad_1)),\end{aligned}$$

ahol *msgs* jelöli a Handshake üzeneteket, *sender* pedig egy, a küldő féltől függő konstans.

10.1. Példa. Tegyük fel, hogy a felek az RSA alapú kulcscsere módszerben egyeztek meg, a szerver tanúsítványa tartalmazza a szerver RSA rejtjelező kulcsát, és a szerver nem szeretné hitelesíteni a klienst. Ekkor a Handshake protokoll üzenetei a következőképpen alakulnak:

(1)	$C \to S$:	client-hello
(2)	$S \to C$:	server-hello
(3)	$S \to C$:	certificate
(6)	$S \to C$:	server-hello-done
(8)	$C \to S$:	client-key-exchange
	$C \to S$:	change-cipher-spec
(10)	$C \to S$:	client-finished
	$S \to C$:	change-cipher-spec
(11)	$S \to C$:	server-finished

A hello üzenetek cseréje után a szerver a certificate üzenetben elküldi az RSA rejtjelező kulcsát tartalmazó tanúsítványát. A kliens generál egy 48 bájtos véletlen blokkot, majd a kapott rejtjelező kulcsot használva rejtjelezi azt, és az eredményt a client-key-exchange üzenetben elküldi a szervernek. Ezután mindkét fél kiszámolja a mestertitkot, és a change-cipher-spec, valamint a finished üzenetek elküldésével befejezik a protokollt. ♣

10.2. Példa. Tegyük fel, hogy a felek az egyszer használatos Diffie–Hellman-kulcscsere módszerben egyeztek meg, a szerver tanúsítványa egy DSS aláírásellenőrző kulcsot tartalmaz, és a szerver DSS aláírással szeretné hitelesíteni a klienst. Ekkor a Handshake protokoll minden üzenetét elküldik a felek.

A hello üzenetek cseréje után a szerver a certifi cate üzenetben elküldi a DSS aláírás ellenőrző kulcsát tartalmazó tanúsítványát. Utána generálja az egyszer használatos Diffie–Hellman paramétereket (p és g) és publikus értékét ($g^x \bmod p$), majd ezeket DSS aláírással ellátva a server-key-exchange üzenetben elküldi a kliensnek. Végül a certifi cate-request üzenetben DSS aláírás ellenőrző kulcsot tartalmazó tanúsítványt kér a klienstől.

A kliens ennek megfelelően a certifi cate üzenetben elküldi a DSS aláírás ellenőrző kulcsát tartalmazó tanúsítványát a szervernek. Ezután a szerver p és g Diffie–Hellman paramétereit használva, ő is generál egy egyszer használatos Diffie–Hellman publikus értéket ($g^y \bmod p$), majd elküldi azt a szervernek a client-key-exchange üzenetben. Végül az összes eddig vett és küldött Handshake üzenet hash értékének DSS aláírását küldi el a szervernek a certifi cate-verify üzenetben.

Ezután mindkét fél kiszámolja a közös mestertitkot, és a change-cipher-spec valamint a fi nished üzenetek elküldésével befejezik a protokollt. ♣

A fi nished üzeneteket a Record protokoll már az új algoritmusokat és kulcsokat használva kódolja. Az ehhez szükséges MAC kulcsokat, rejtjelező kulcsokat, *IV*-ket stb. a mestertitokból generálják a felek a következő módon: először a mester titokból létrehoznak egy megfelelő hosszúságú random bájtsorozatot, majd ezt a kulcsok és *IV*-k méretének megfelelő szeletekre vágják. A random bájtsorozat generálása a következő kifejezés szerint történik:

$$\begin{aligned} &\mathrm{MD5}(K \mid \mathrm{SHA}(\text{"A"} \mid K \mid N_C \mid N_S)) \mid \\ &\quad \mathrm{MD5}(K \mid \mathrm{SHA}(\text{"BB"} \mid K \mid N_C \mid N_S)) \mid \\ &\qquad \mathrm{MD5}(K \mid \mathrm{SHA}(\text{"CCC"} \mid K \mid N_C \mid N_S)) \mid \ldots \end{aligned} \tag{10.1}$$

A fenti kifejezésben minden sor 16 bájtot állít elő, és ezeket fűzzük össze, hogy megfelelő mennyiségű véletlen bájtot kapjunk. Ha például 80 bájtra van szükségünk, akkor azt öt 16 bájtos blokkból tudjuk összerakni, azaz ötször kell meghívni az MD5 és SHA függvényeket a fenti módon.

A teljes Handshake protokoll végrahajtására csak akkor van szükség, ha egy új viszonyt akarnak létrehozni a felek. Ha a kliens egy már meglévő

viszonyon belül szeretne egy új kapcsolatot létrehozni, és a szerver ezt engedélyezi („is resumable" bit 1 értékű), akkor a Handshake első fázisa után rögtön a harmadik fázis végére ugranak a felek a végrehajtásban, vagyis (10.1) alapján új kapcsolatkulcsokat generálnak a meglévő K mestertitok és az első fázisban generált friss N_C és N_S véletlenszámokat használva, majd elküldik change-cipher-spec és fi nished üzeneteiket. Ekkor tehát a protokoll futása sokkal rövidebb.

10.1.4. Az SSL protokoll analízise

Az SSL az évek során intenzív analízisnek volt kitéve. A 2.0 verzióban számos gyengeséget fedeztek fel, amit a 3.0 verzióban a Netscape kijavított. Az SSL 3.0 verziója tehát alapvetően stabil és biztonságos protokoll. Apróbb hiányosságok azonban még akadnak. Ezeket igyekszünk összefoglalni ebben a szakaszban. Úgy érezzük, hogy az itt leírt gondolatok megértése elmélyíti a fejezet első részének elolvasása során megszerzett ismereteket.

10.1.4.1. A Record protokoll analízise

A Record protokollt a következő négy szempont szerint vizsgáljuk:

- bizalmasság megsértésére irányuló támadások elleni védelem,
- forgalomanalízis elleni védelem,
- üzenet-visszajátszás elleni védelem,
- üzenetintegritás és hitelesség biztosítása.

Bizalmasság. Általában megállapítható, hogy a bizalmasság megsértésére irányuló támadások ellen jól védekezik az SSL Record protokollja. A védelem alapja a Record üzenetek rejtjelezése. A rejtjelezéshez az SSL rövid élettartamú kapcsolatkulcsokat használ, amiket a hosszabb élettartamú mestertitokból állít elő a kapcsolatban generált friss véletlen értékeket (a kliens és a szerver által generált friss véletlenszámokat) is felhasználva. A kapcsolatkulcsok előállítása egyirányú függvénnyel történik, ezért egy kapcsolatkulcs kompromittálódása nem veszélyezteti a mestertitkot. Külön pozitívumként mondható el, hogy az SSL különböző kapcsolatokban, és a kapcsolatokon belül különböző irányokban különböző kulcsokat használ. Ez nehezebbé teszi a támadó dolgát, és általában jó tervezési hozzáállásnak minősül. Ezen kívül az SSL által támogatott rejtjelező algoritmusok mind erősek. Meg kell azonban jegyezni, hogy különböző amerikai export szabályok miatt, az al-

goritmusok csonkított (40 bites) kulcsokkal történő használatát is támogatja a protokoll, ami a biztonság szempontjából nem előnyös tuljadonság.

Forgalomanalízis. A Record protokoll egyáltalán nem védekezik a forgalomanalízis ellen. Az IP címeket és TCP port számokat nyilván nem is rejtheti, hiszen a TCP és az IP protokollok felett helyezkedik el. Az üzenetek méretét viszont rejthetné, de nem teszi. Ha a felek az RC4 kulcsfolyamatos rejtjelező használatában egyeznek meg, akkor nincs kitöltés, és így a rejtett üzenetek hossza megegyezik a nyílt üzenetek hosszával. Ha a felek blokkrejtjelezőt használnak, akkor csak a minimálisan szükséges kitöltést alkalmazzák, így egy forgalmat figyelő támadó jó eséllyel tud tippelni a nyílt üzenetek hosszára.

Az átküldött nyílt üzenetek hosszának ismerete további információk megszerzését teheti lehetővé. Tegyük fel például, hogy egy webböngésző és egy webszerver közötti HTTP forgalmat védjük az SSL protokollal. Ha a támadó egy legális felhasználó, aki szintén hozzáfér a szerveren tárolt weboldalakhoz, akkor könnyen megfigyelheti az oldalak és a hozzájuk tartozó URL-ek hosszát. Mivel az SSL nem rejti a nyílt üzenetek hosszát, ezért a támadó egy másik felhasználó rejtjelezett forgalmát lehallgatva, az üzenetek méretéből következtetni tud arra, hogy a felhasználó mely weboldalakat tölti le a szerverről.

Visszajátszás. A Record protokoll az üzenetek sorszámozásával védekezik a visszajátszás ellen. Mivel az üzeneteken számolt MAC tartalmazza a sorszám aktuális értékét is, ezért a támadó nem tudja az üzenetek sorrendjét észrevétlenül megváltoztatni, illetve nem tud üzeneteket törölni és beszúrni. A sorszám 64 bites, azaz praktikusan sosem fordul körbe.

Integritásvédelem. A Record protokoll minden üzenethez üzenethitelesítő kódot (MAC) csatol, mely védi az üzenet integritását és hitelesíti annak küldőjét. Az alkalmazott MAC algoritmus a HMAC egy korábbi verziója, a MAC kulcsok mérete pedig 128 bit, ami megfelelő biztonságot nyújt. A rejtjelezéshez hasonlóan, a MAC kulcsok is különbözőek minden kapcsolatban, és azon belül mindkét irányban.

A MAC számítással kapcsolatos egyetlen gyengeség, hogy a MAC függvény bemenete nem tartalmazza a Record üzenet verziómezőjét. Ez nem feltétlenül ad lehetőséget támadásra, de mindenenképpen felveti a következő kérdést: Ha a verziómező értékét a vevő felhasználja a Record üzenet feldol-

gozásánál, akkor azt a MAC számítás bemenetének tartalmaznia kellene; ha viszont a vevő nem használja a verziómezőt, akkor miért van egyáltalán erre a mezőre szükség?

10.1.4.2. A Handshake protokoll analízise

Cipher suite rollback. Az SSL 2.0 verziójában egy támadó el tudta érni azt, hogy a felek 40 bitesre csonkított kulcsokkal használják a rejtjelező algoritmust. Ehhez a támadó a client-hello üzenetet módosította, mégpedig úgy, hogy a kliens által javasolt biztonsági algoritmusok listájáról törölte a 40 bitnél nagyobb kulcsokat használó algoritmus-kombinációkat. Így a szerver már csak olyan algoritmus-kombinációt választhatott, ami 40 bites kulcsot használt rejtjelezésre. A kliens és a szerver a támadást nem detektálta, mert a Handshake protokoll 2.0 verziója nem használt fi nished üzeneteket, melyek az aktív támadások detektálását lehetővé tették volna.

A Handhskake protokoll 3.0 verziója tehát a fi nished üzenetek segítségével detektálja a Cipher suite rollback támadást. Meg kell azonban jegyezni, hogy a fi nished üzenetek sikeres ellenőrzése csak azon feltétel mellett biztosítja a teljes Handshake hitelességét és sértetlenségét, hogy a mestertitok valóban titkos, sértetlen és hiteles.

A mestertitok hitelességét az alkalmazott kulcscsere módszertől függően többféleképpen biztosíthatja a protokoll. Ha a Handshake során a szerver küld server-key-exchange üzenetet, akkor az tartalmazza a szerver aláírását a szerver kulcscsere paraméterein, és ez explicit módon hitelesíti a kliens számára a később létrehozott mestertitkot. Ha nincs server-key-exchange üzenet, akkor a mestertitok hitelesítése a kliens számára implicit. Például RSA alapú kulcscsere esetén a kliens tudja, hogy csak a szerver képes dekódolni a kliens által rejtjelezett 48 bájtos véletlen blokkot, a szerver nyilvános kulcsát pedig egy megbízható hitelesítés-szolgáltató hitelesíti.

A szerver számára a legtöbb esetben nem biztosított a mestertitok hitelessége, hiszen a kliens hitelesítés opcionális, és ritkán használják. Más szavakkal, a szerver legtöbbször nem tudja biztosan, hogy ki a kliens, és így azt sem, hogy ki ismeri a mestertitkot. Ugyanakkor, ha a szerver kéri a kliens hitelesítést, akkor a kliens digitális aláírása a certifi cate-verify üzenetben explicit módon hitelesíti a szerver számára a később létrehozott mestertitkot.

Version rollback. Mivel az SSL 2.0 verziója számos biztonsági rést tartalmaz, ezért a támadó megpróbálhatja elérni, hogy a felek a 2.0 verziót futtassák a 3.0 verzióc helyett. Ez nyilván csak akkor lehetséges, ha a felek még

támogatják a 2.0 verziót. A támadónak ehhez csak annyit kell tennie, hogy a kliens client-hello üzenetében a verziómezőt 2.0-ra változtatja. A szerver így azt hiszi, hogy ez a kliens által támogatott legmagasabb verzió, és így nem tehet mást, mint maga is 2.0-ra állítja a verzió mező értékét a server-hello üzenetben. A kliens ezt úgy értelmezi, hogy a szerver nem támogatja a 3.0 verziót, ezért helyette a 2.0 verziót javasolja. Mindketten azt hiszik tehát, hogy a másik fél által támogatott legmagasabb verzió a 2.0, ezért a továbbiakban azt használják. A dolog pikantériája, hogy a csalást észre sem veszik, mert a Handshake protokoll 2.0 verziójában nincsenek fi nished üzenetek, melyek ezt lehetővé tennék.

Szerencsére az SSL 3.0 verziója képes detektálni a version rollback támadást. Ezt a következő módon teszi. Ha a kliens támogatja a 3.0 verziót, akkor úgy generálja a mester titok alapját képző 48 bájtos véletlen blokkot, hogy abból igazából csak 46 bájt véletlen, az első két bájt pedig a kliens által támogatott legmagasabb verziót, azaz a 3.0 értéket tartalmazza. Ezt még akkor is így teszi, ha amúgy a protokoll 2.0 verzióját futtatja. Az így előállított 48 bájtot küldi el aztán a szerver nyilvános RSA kulcsával rejtjelezve a szervernek[1]. Ha a szerver támogatja a 3.0 verziót, akkor mindig ellenőrzi a kapott blokk első két bájtját, még akkor is, ha amúgy a 2.0 verziót futtatja. Vegyük észre, hogy a fenti version rollback támadás csak akkor minősül támadásnak, ha mindkét fél támogatja a 3.0 verziót. Ekkor azonban a kliens jelzi, a szerver pedig detektálja, hogy a kliens által támogatott legmagasabb verzió a 3.0. Ha ennek ellenére a 2.0 verziót használják, akkor a szerver megszakítja a kapcsolatot.

A change-cipher-spec üzenet elnyelése. Említettük, hogy a change-cipher-spec üzenet nem a Handshake része, és ezért a fi nished üzenetek számításánál és ellenőrzésénél a change-cipher-spec üzeneteket nem veszik figyelembe a felek. Elképzelhető továbbá, hogy valamely SSL implementáció megengedi a fi nished üzenet feldolgozását annak ellenére, hogy nem kapott change-cipher-spec üzenetet. Bár ez nem tűnik logikusnak, az SSL specifikációja nem zárja ki ezt a lehetőséget, és a Netscape egyik saját korábbi referencia implementációja (SSLRef 3.0b1) is pont ezt teszi. Ez súlyos hiba, amit egy támadó bizonyos esetekben ki tud használni

Tegyük fel, hogy a felek olyan algoritmus-kombinációt szeretnének használni, amiben csak üzenethitelesítés van, de nincs rejtjelezés (például SSL-

[1] Megjegyezzük, hogy a Handshake 2.0 verziója csak az RSA alapú kulcscserét támogatja.

RSA-with-NULL-SHA). A támadó hagyja, hogy a Handshake első három fázisa lefusson, majd a kliens change-cipher-spec üzenetét elfogja és eldobja, így a szerver nem kapja azt meg. Ekkor a kliens és a szerver felemás állapotba kerülnek: a kliens már az új algoritmusokat használva, azaz MAC értékkel kiegészítve küldi üzeneteit, de a szerver még mindig MAC nélkül várja azokat. Az első üzenet, amit a kliens MAC értékkel ellátva küld a fi nished üzenet. A támadó ezt az üzenetet úgy módosítja, hogy a MAC értéket törli belőle. A szerver elfogadja az így módosított fi nished üzenetet, mert

- MAC nélkül várja a kliens üzeneteit, és
- a fi nished üzenet ellenőrzése sikeres lesz, hiszen annak számítása nem függ az elküldött vagy vett change-cipher-spec üzenetektől.

Hasonlóképpen, a támadó elnyeli a szerver change-cipher-spec üzenetét is, majd a szerver fi nished üzenetéből törli a MAC értéket. A kliens az előbbiekhez hasonlóan elfogadja a módosított fi nished üzenetet. A Handshake tehát sikeresen lefut, és innentől kezdve a támadó minden Record üzenetből törli a MAC értéket. Ez persze azt is jelenti, hogy a MAC törlése után minden Record üzenetet módosítani tud, és ezt a felek nem veszik észre, mert még mindig felemás állapotban vannak.

A támadás ellen úgy védekezhetünk, hogy a fi nished üzenet feldolgozását addig nem kezdjük el, amíg a change-cipher-spec üzenet meg nem érkezett. Az újabb implementációk erre már ügyelnek. Vegyük észre továbbá, hogy a támadás nem működik, ha rejtjelezést is használnak a felek, mert a MAC értékkel ellentétben a rejtjelezést nem tudja eltüntetni a támadó.

Key exchange algorithm rollback. Egy másik lehetséges aktív támadás, mikor a támadó eléri, hogy a kliens RSA alapú kulcscserét, míg a szerver Diffie–Hellman-algoritmust futtasson egyazon Handshake során. Ekkor egy esetleges implementációs figyelmetlenség a következő végzetes támadáshoz vezethet.

Tegyük fel, hogy a kliens egyszer használatos RSA kulcscserét szeretne futtatni, a támadó azonban oly módon módosítja a kliens client-hello üzenetét, hogy a szerver számára úgy tűnjön, hogy a kliens egyszer használatos Diffie–Hellman-kulcscserét javasol (azaz a kliens által javasolt SSL-RSA-with-... azonosítókat SSL-DHE-with-... azonosítókra cseréli a client-hello üzenetben). A szerver így biztosan olyan algoritmus-kombinációt választ, melyben a kulcscsere módszer az egyszer használatos Diffie–Hellman. A támadó a server-hello üzenetet úgy módosítja, hogy a szerver által választott

SSL-DHE-with-... azonosítót SSL-RSA-with-... azonosítóra cseréli, így a kliens azt hiszi, a szerver RSA alapú kulcscserét választott.

A Handshake protokoll második fázisában a szerver elküldi egyszer használatos Diffie–Hellman paramétereit (p, g, $g^x \bmod p$) a kliensnek. A kliens azonban a szerver egyszer használatos nyilvános RSA kulcsára vár, ezért a kapott üzenetet úgy értelmezheti, hogy p az RSA modulus, g pedig az exponens. A Handshake harmadik fázisában a kliens generál egy 48 bájtos véletlen blokkot, amit K_1'-gyel jelölünk, és elküldi $(K_1')^g \bmod p$-t a szervernek. A támadó azonban elfogja ezt az üzenetet, és helyette a szerver által várt $g^y \bmod p$-t küldi el a szervernek, ahol y-t a támadó választja.

Mivel p prím, ezért a támadó az elfogott $(K_1')^g \bmod p$-ből hatékonyan ki tudja számolni K_1'-et. Továbbá, mivel a támadó generálta y-t, ezért könnyen ki tudja számolni $K_2' = g^{xy} \bmod p$-t is. K_1'-ről a kliens azt hiszi, hogy az egy, a szerverrel megosztott közös titok, K_2' pedig a szerver által kiszámolt érték, melyről a szerver azt hiszi, hogy a klienssel osztja meg. Valójában azonban mindketten a támadóval osztják meg a titkukat. A támadó minden olyan információnak birtokában van, ami lehetővé teszi számára a K_1 és a K_2 mestertitkok kiszámítását K_1'-ből, illetve K_2'-ből. Ezek ismeretében azonban minden kulcsot ki tud számítani, és az ezen kulcsokkal védett Record üzeneteket fel tudja törni, illetve meg tudja változtatni. Mivel a fi nished üzenetek elküldésére csak a negyedik fázisban kerül sor, ezért azokat is tetszőlegesen manipulálni tudja a támadó, és ezzel képes eltüntetni az aktív támadás nyomait is. Más szavakkal, a fi nished üzenetek ellenére a kliens és a szerver nem szerez tudomást arról, hogy becsapták őket.

A key exchange algorithm rollback támadást az teszi lehetővé, hogy a server-key-exchange üzenetet a kliens nem hitelesített információkra támaszkodva dolgozza fel. Egészen pontosan, a kliens a szerver fi nished üzenetének sikeres ellenőrzéséig nem lehet biztos abban, hogy a szerver valóban RSA alapú kulcscserét választott. Ennek ellenére, a kliens az RSA alapú kulcscsere módszer szerint dolgozza fel a szerver üzenetét. Figyeljük meg, hogy a szerver aláírása a server-key-exchange üzenetben nem oldja meg a problémát, mert az csak az üzenet tartalmát hitelesíti. Itt azonban a feldolgozás kontextusa az, amit hitelesíteni kellene az üzenet értelmezése előtt. Sajnos a mestertitok kompromittálódása után a fi nished üzenetek már nem nyújtanak védelmet, hiszen korábban is kihangsúlyoztuk, hogy a fi nished üzenetek sikeres ellenőrzése csak azon feltétel mellett biztosítja a teljes Handshake hitelességét és sértetlenségét, hogy a mestertitok valóban titkos, sértetlen és hiteles.

A problémát úgy lehetne megoldani, hogy a szerver nemcsak a kulcscsere paramétereit írná alá a server-key-exchange üzenetben, hanem például az összes addigi vett és küldött Handshake üzenet hash értékét is. Ez a megoldás összhangban volna az SSL filozófiájával abban az értelemben, hogy hasonló hash értéket ír alá a kliens a certifi cate-verify üzenetben, és a fi nished üzenetek számítása is az összes addigi Handshake üzenet hash értékéből történik.

10.1.5. A TLS protokoll

A TLS protokollt gyakorlatilag az SSL protokoll 3.1 verziójának is tekinthetjük, melyben a fenti elemzésben azonosított hiányosságokat kijavították. A TLS és az SSL közötti főbb különbségek a következők:

- a TLS Record protokollja véletlen hosszúságú kitöltést alkalmaz a forgalomanalízis megnehezítésére;
- a TLS Record protokollja a legújabb HMAC verziót használja és a MAC számítás magában foglalja a Record üzenet verziómezőjét is;
- a TLS Handshake protokollja a fi nished üzenetet addig nem dolgozza fel, amíg nem érkezett change-cipher-spec üzenet;
- az SSL Handshake során használt különböző MAC jellegű függvényeket (certifi cate-verify és fi nished üzenetek számítása, valamint a mestertitok és a kapcsolatkulcsok generálása) egységesítették a TLS-ben.

10.2. IPSec

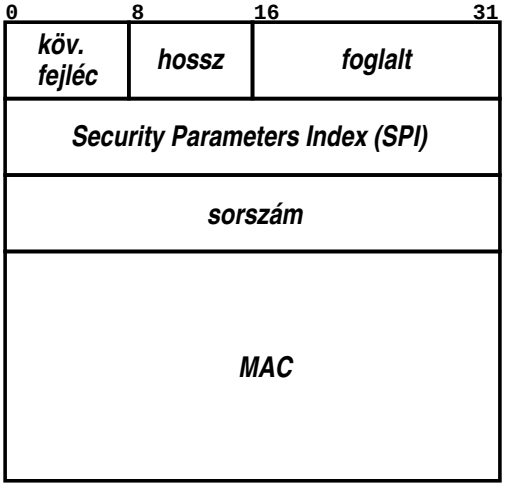
Az IPSec protokoll a TCP/IP architektúra hálózati rétegének szabványosított (RFC 2401, 2402, 2406, 2408, 2409) biztonsági protokollja. Ez azt jelenti, hogy az IP és minden fölötte található protokoll (TCP, UDP, ICMP, stb.) számára védelmet biztosít. Két alprotokollja van, az AH (Authentication Header) és az ESP (Encapsulated Security Payload). Az AH protokoll funkciói közé tartozik az IP csomagok integritásának védelme, eredetének hitelesítése és visszajátszásuk detektálása. Az ESP protokoll fő feladata az IP csomagok tartalmának rejtése, opcionálisan azonban az ESP is nyújthat integritásvédelem szolgáltatást. Természetesen az AH és az ESP protokollok kombinálhatók az IP csomagok teljeskörű védelme érdekében.

Az IPSec protokollhoz tartoznak még az ISAKMP (Internet Security Association and Key Management Protocol) és az IKE (Internet Key Exchange) protokollok. Mindketten kulcscserével kapcsolatos feladatokat látnak el. Az

ISAKMP egy általános célú keretprotokoll, mely bármely konkrét kulcscsere protokoll üzeneteit képes szállítani. Egy ilyen konkrét kulcscsere protokoll az IKE, mely jelenleg az IPSec hivatalos kulcscsere protokolljának szerepét tölti be. Az ISAKMP és az IKE protokollokat a teljesség kedvéért említettük meg, a továbbiakban azonban kizárólag az AH és az ESP protokollok ismertetésével foglalkozunk.

10.2.1. Az AH protokoll

Az AH protokoll tehát integritásvédelmet, eredethitelesítést és visszajátszás elleni védelmet biztosít az IP csomagok számára. Az integritásvédelmet és az eredethitelesítést úgy éri el, hogy az IP fejléc és az azt követő felsőbb szintű protokoll fejléce közé beszúr egy AH fejlécet, mely egy, a teljes IP csomagra számolt üzenethitelesítő kódot (MAC) tartalmaz. A visszajátszások detektálásának érdekében, az IP csomagokat sorszámozza. Az AH fejlécben található MAC érték a sorszámot is védi.

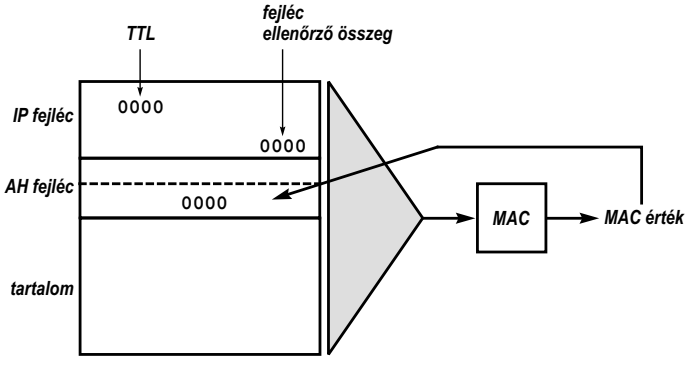


10.4. ábra. Az AH fejléc felépítése

Az AH fejléc felépítését a 10.4. ábra szemlélteti. A fejlécben található mezők és azok jelentése a következő:

- *következő fejléc (next header):* Ez a mező az AH fejlécet követő fejléc típusát adja meg, azaz az IP csomag tartalmára utal.

- *hossz (length):* Ez a mező az AH fejléc 32 bites szavakban mért hosszára utal.
- *Security Parameter Index (SPI):* Ez egy azonosító, mellyel a küldő azt jelzi a vevő számára, hogy milyen módon és mely kulcsokat használva kell az AH fejlécet feldolgozni. Az AH protokoll feltételezi, hogy a küldő és a vevő korábban már megegyezett az alkalmazható algoritmusokban és kulcsokban, tipikusan az ISAKMP/IKE protokollokat használva.
- *sorszám (sequence number):* Ez a mező az aktuális IP csomag sorszámát tartalmazza.
- *üzenethitelesítő kód (MAC)*: Ez a teljes IP csomagra számolt üzenethitelesítő kód. A MAC számításának módját a 10.5. ábra szemlélteti. A számításhoz az IP fejléc változó mezőit (például TTL) és az AH fejléc MAC mezőjét nullával töltjük ki, majd az így kapott teljes csomagra kiszámoljuk a MAC értéket, és az eredményt visszaírjuk az AH fejléc MAC mezőjébe.



10.5. ábra. Az AH fejléc MAC mezőjének számítása.

A régi IP csomagok visszajátszásának detektálását az AH protokoll az AH mezőben található sorszám alapján végzi. A vevőnek van egy konstans W méretű vételi ablaka. A vételi ablak e_r jobb széle megegyezik az eddigi legnagyobb sorszámú sikeresen vett csomag sorszámával. Az ablak bal széle pedig mindig $e_\ell = e_r - W + 1$. A vevő számon tartja, hogy az $[e_\ell, e_r]$ intervallumban mely csomagokat vette már egyszer. Tegyük fel, hogy egy új csomag érkezik, melynek sorszámát jelöljük s-sel. Ekkor a vevő a következőt teszi:

- Ha $s < e_\ell$, akkor a vevő eldobja a csomagot.

- Ha $e_\ell \leq s \leq e_r$ és s sorszámmal a vevő már vett egy csomagot, akkor eldobja a csomagot.
- Ha $e_\ell \leq s \leq e_r$ és s sorszámmal a vevő még nem vett csomagot, akkor a vevő ellenőrzi a csomag AH fejlécében található MAC értéket. Ha az ellenőrzés sikeres, akkor a vevő megjegyzi az s sorszámot, és elfogadja a csomagot.
- Ha $s > e_r$, akkor a vevő ellenőrzi a csomag AH fejlécében található MAC értéket. Ha az ellenőrzés sikeres, akkor a vevő megjegyzi az s sorszámot, a vételi ablak jobb szélét s-re állítja, és elfogadja a csomagot.

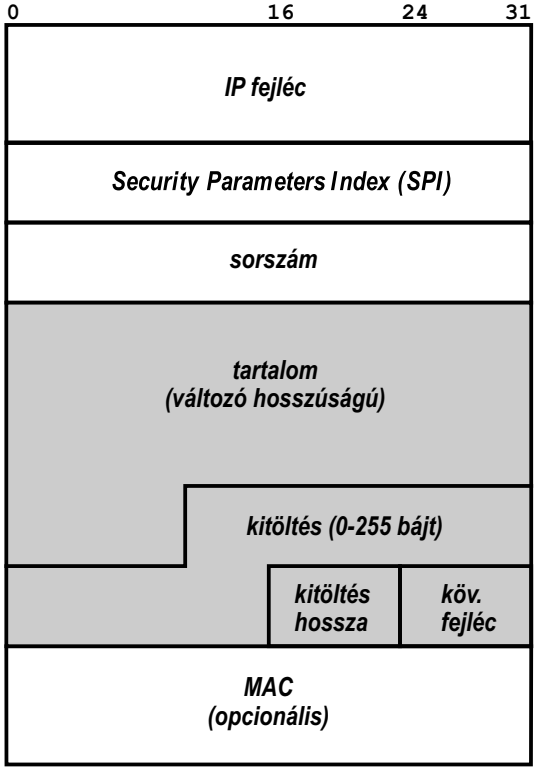
A vételi ablakra azért van szükség, mert az IP protokoll nem garantálja a csomagok sorrendhelyes kézbesítését. Ez azt jelenti, hogy normális körülmények között is érkezhet olyan csomag, melynek sorszáma kisebb, mint az eddigi legnagyobb vett sorszám. Ha ez a csomag még benne van az ablakban, akkor a vevő elfogadja. Az ablak véges mérete biztosítja, hogy „túl régi” csomagokat nem fogad el a vevő. A véges méret miatt a vevőnek nem kell az összes eddig vett sorszámot megjegyeznie, elegendő csak azokat számon tartani, melyek beleesnek az aktuális ablakba.

10.2.2. Az ESP protokoll

Az ESP protokoll feladata az IP csomag tartalmának rejtése, és opcionálisan a tartalom integritásának védelme. Az előbbit az IP csomag tartalmának rejtjelezésvél oldja meg a protokoll, az utóbbit pedig úgy, hogy az ESP fejlécre és a csomag tartalmára számít MAC kódot és azt a csomaghoz csatolja. Az AH-val ellentétben tehát az ESP MAC nem védi az IP fejléc mezőit.

A 10.6. ábra az ESP-vel védett IP csomag felépítését szemlélteti. Az ESP fejléc tartalmaz egy azonosítót (SPI), mely az AH-hoz hasonlóan itt is azt jelzi, hogy a vevőnek milyen módon és mely kulcsokat használva kell a csomagot feldolgoznia. Az ESP protokoll is feltételezi, hogy a küldő és a vevő korábban már megegyezett az alkalmazható algoritmusokban és kulcsokban, tipikusan az ISAKMP/IKE protokollokat használva. Az ESP fejléchez tartozik még a csomag sorszáma is.

A 10.6. ábrán szürkével jelöltük a csomag rejtjelezett részét. Blokk-kódoló használata esetén a csomagot természetesen ki kell tölteni. A kitöltés hossza és az ESP fejlécet követő fejléc típusa (azaz a felsőbb protokoll fajtája) a kitöltés után kerül, és szintén rejtjelezve van. Végül az ESP MAC zárja le a csomagot, mely opcionális, és nincs rejtjelezve. A blokkrejtjele-



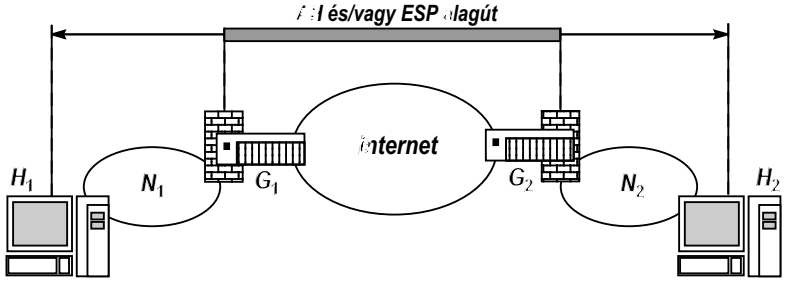
10.6. ábra. Az ESP-vel védett IP csomag felépítése

zőt CBC módban használja a protokoll. Az ehhez szükséges *IV* nyíltan kerül átvitelre a csomag tartalomrészének elején (az ESP sorszám után).

10.2.3. IPSec a gyakorlatban

Mind az AH, mind az ESP protokollt két üzemmódban lehet használni. Ezeket szállítási (transport) és alagút (tunnel) módoknak nevezzük. Szállítási módban az AH vagy az ESP fejléc a csomag eredeti IP fejléce és a felsőbb szintű protokoll (például TCP, UDP) fejléce közé kerül. Alagút módban azonban az eredeti IP csomagot teljes egészében beágyazzuk egy másik IP csomagba (IP tunneling), és az AH vagy az ESP fejléc az új, és az eredeti IP fejléc közé kerül. Ekkor tehát az AH fejléc vagy az ESP trailer *következő fejléc* mezője IP-re utal.

Az alagút módot tipikusan biztonsági átjárók (security gateway), például tűzfalak között használjuk. Segítségével könnyen létrehozhatunk virtuális magánhálózatokat (Virtual Private Network –VPN), ahol két tűzfallal védett belső hálózatot az interneten keresztül, IPSec-et használva biztonságosan összekötünk. Ezt szemlélteti a 10.7. ábra. Tegyük fel például, hogy az N_1 belső hálózaton található H_1 gép egy IP csomagot küld az N_2 belső hálózaton található H_2 gépnek. Mikor az IP csomag eléri a G_1 tűzfalat, a tűzfal az egészet beágyazza egy G_2-nek szóló IP csomagba, és azt IPSec védelemmel ellátva küldi tovább. Így a csomag biztonságban jut át az interneten a G_2 tűzfalhoz. G_2 elvégzi az IPSec feldolgozást (dekódolja a csomagot, ellenőrzi a MAC kódot stb.), majd a csomagban található eredeti IP csomagot (most már nyíltan) továbbküldi az eredeti címzettnek.



10.7. ábra. VPN létrehozása IPSec segítségével

A fenti példában a G_1 és G_2 tűzfalak alagút módban használják az AH és ESP protokollokat. Ekkor az IP csomagok a $H_1 - G_1$ és a $G_2 - H_2$ szakaszon védelem nélkül kerülnek továbbításra. Egy adott alkalmazásban ez esetleg nem jelent problémát, hiszen a belső hálózatok megbízhatónak tekinthetők. Ha ez nem így van, akkor biztonságos vég-vég kommunikációra van szükség H_1 és H_2 között. Erre a célra az AH és az ESP szállítási módja alkalmazható.

Az AH és ESP protokollok szállítási módú alapkombinációja (transport adjacency) az, amikor az IP csomagon először az ESP feldolgozást végezzük el hitelesítés szolgáltatás (azaz MAC) nélkül, majd az így kapott csomagra elvégezzük az AH feldolgozást is. Ekkor tehát a fejlécek sorrendje: IP, AH, ESP, TCP/UDP, ...

Alagút módban többféle kombinációt hozhatunk létre. Elékpzelhető például, hogy egy AH és/vagy ESP szállítási móddal védett IP csomagot még egy AH és/vagy ESP alagút móddal védett IP csomagba ágyazunk be. Sőt,

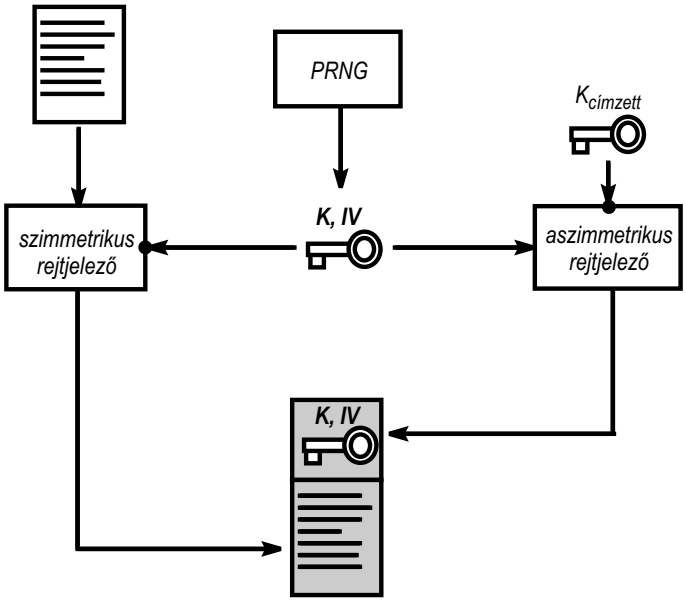
alagutak egymásba ágyazása is lehetséges, ennek alapkombinációi a következők: az alagutak két végpontja azonos (tipikusan hosztok között), az alagutak egyik végpontja azonos (tipikusan hoszt és tűzfal között) és az alagutak egyik végpontja sem közös (tipikusan tűzfalak között).

10.3. PGP (Pretty Good Privacy)

A PGP egy általános célú szoftver, melyet fájlok digitális aláírására és rejtjelezésére lehet használni. Azért tárgyaljuk mégis az internet biztonságról szóló részben, mert a PGP megalkotásának fő motivációja az elektronikus levelezés védelme volt (alapjában véve az e-mail nem más, mint egy fájl). Ebben a vonatkozásában a PGP azért is érdekes, mert az SSL-től és az IPSec-től eltérően, egy aszinkron jellegű kommunikáció védelmét oldja meg. Az aszinkron jelzőt itt abban az értelemben használjuk, hogy a kommunikáló felek nincsenek egy időben jelen a kommunikáció során. Ez a tulajdonság alapvetően meghatározza a protokollt, hiszen az interakció lehetősége ki van zárva, és olyan üzenetekre van szükség, melyek önmagukban hordoznak minden információt, ami a feldolgozásukhoz szükséges.

A PGP alapvetően nyilvános kulcsú kriptográfiára épül. Fő szolgáltatásai közé tartozik az üzenetek digitális aláírása, az aláírt üzenetek tömörítése, és a tömörített üzenetek rejtjelezése. Ezen szolgáltatások közül a rejtjelezést tárgyaljuk részletesebben. A digitális aláírás a hash-and-sign módszerrel történik, amiről már korábban esett szó (lásd 7.2. fejezet).

Gondoljuk meg, hogy az aszinkron kommunikáció miatt a PGP-nek (és más hasonló rendszereknek) nincs lehetősége arra, hogy egy hagyományos kulcscsereprotokollt használva először létrehozzon egy kapcsolatkulcsot, majd azzal rejtjelezze az üzenetet (elektronikus levelet). Ezért kézenfekvőnek látszik a nyilvános kulcsú rejtjelezés használata, hiszen a címzett nyilvános kulcsa rendelkezésre állhat, vagy letölthető valamilyen kulcstárból. Tudjuk azonban, hogy hosszú üzenetek nyilvános kulcsú algoritmussal történő rejtjelezése nem praktikus. Ezért a PGP az ún. digitális boríték (digital envelop) technikát használja. Ennek az a lényege, hogy az üzenetet először egy frissen generált szimmetrikus kulccsal rejtjelezzük (például AES-sel), majd a szimmetrikus kulcsot kódoljuk a címzett nyilvános kulcsával, és a rejtjeles üzenetet a rejtjelezett szimmetrikus kulccsal együtt elküldjük a címzettnek. A címzett először a szimmetrikus kulcsot állítja vissza saját privát kulcsa segítségével, majd a visszaállított szimmetrikus kulccsal dekódolja az üzenetet. A digitélis boríték technikát a 10.8. ábra szemlélteti.



10.8. ábra. A PGP által is használt digitális boríték technika szemléltetése

A PGP rendszer egy másik érdekessége a kulcsgondozás, illetve azon belül is a nyilvános kulcsok hitelesítésének módja. A PGP rendszerben ugyanis nem hitelesítés szolgáltatók (CA) bocsátják ki a felhasználók nyilvános kulcs tanúsítványait, hanem maguk a felhasználók. Ebben az értelemben tehát a PGP egy önszerveződő rendszer, nincs szüksége kiépített PKI-re.

A PGP rendszerben minden felhasználó kiadhat egy tanúsítványt más felhasználók nyilvános kulcsára. A tanúsítvány formája hasonlít a hagyományos PKI-ben használt tanúsítványhoz, azaz minimálisan tartalmazza a felhasználó nevét, nyilvános kulcsát és a tanúsítványt kibocsátó felhasználó aláírását. Ezeket a tanúsítványokat a felhasználók egymás között cserélgethetik, terjeszthetik, vagy valamilyen nyilvános adatbázisban elhelyezhetik.

Minden U felhasználónak van egy lokális publikus kulcsadatbázisa is. Ezen adatbázis minden bejegyzése tartalmaz egy V felhasználó-azonosítót, V vélt K_V nyilvános kulcsát, és más felhasználók által V számára kiadott tanúsítványokat. Ezek után U PGP szoftvere kiszámol egy hitelességmértéket, ami azt fejezi ki, hogy a rendelkezésre álló információk alapján U mennyire bízhat meg abban, hogy K_V valóban V kulcsa.

A hitelesség mértékének kiszámítása a következőképpen történik. U egy szubjektív bizalom értéket rendel minden tanúsítány kibocsátóhoz. A szubjektív bizalom értékek a következők lehetnek:

- ismeretlen felhasználó,
- általában nem megbízható felhasználó,
- általában megbízható felhasználó és
- mindig megbízható felhasználó,

ahol a megbízhatóság azt jelenti, hogy U mennyire bízik meg az adott felhasználó által kibocsátott tanúsítványokban. Ezután U szoftvere kiválasztja azokat a tanúsítókat, amelyek kulcsának hitelességéről már korábban meggyőződött (az U által aláírt kulcsokat hitelesnek fogadja, minden további ellenőrzés nélkül), majd kiszámítja az ezen tanúsítókhoz rendelt bizalom értékek súlyozott összegét, ahol a súlyok U által konfigurálható paraméterek. Az eredmény adja a $V - K_V$ összerendelés hitelességének mértékét.

10.3. Példa. Tegyük fel, hogy a $V - K_V$ összerendelést A, B és C tanúsítja, ahol U a következő szubjektív bizalom értékeket rendelte a tanúsítókhoz: A – általában megbízható, B – ismeretlen, C – általában megbízható. Tegyük fel továbbá, hogy A, B és C kulcsának hitelességéről már meggyőződött U szoftvere. Legyenek a súlyok olyanok, hogy a kulcs hitelességének megállapításához legalább két, általában megbízható felhasználó, vagy legalább egy, mindig megbízható felhasználó aláírása szükséges. Ekkor U szoftvere hitelesnek minősíti a $V - K_V$ összerendelést, mert két, általában megbízható felhasználó, A és C aláírta, és mindkettőjük kulcsát hitelesnek tartja U rendszere. Ellenben, ha például C kulcsa nem lenne hiteles, vagy C-t általában nem megbízható felhasználónak tartaná U, akkor K_V-t nem fogadná el hitelesnek. ♣