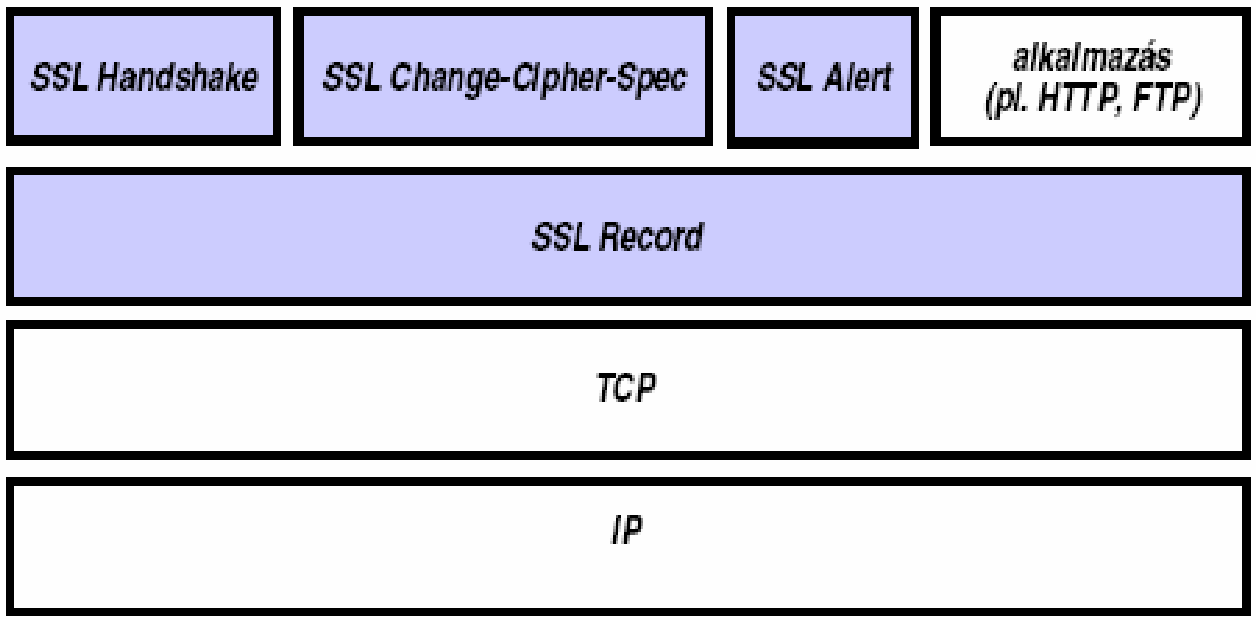
SSL

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 webszerver 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).

SSL elemei



Az SSL illeszkedése az internet protokoll-architektúrájába

SSL elemei

- *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.

SSL elemei

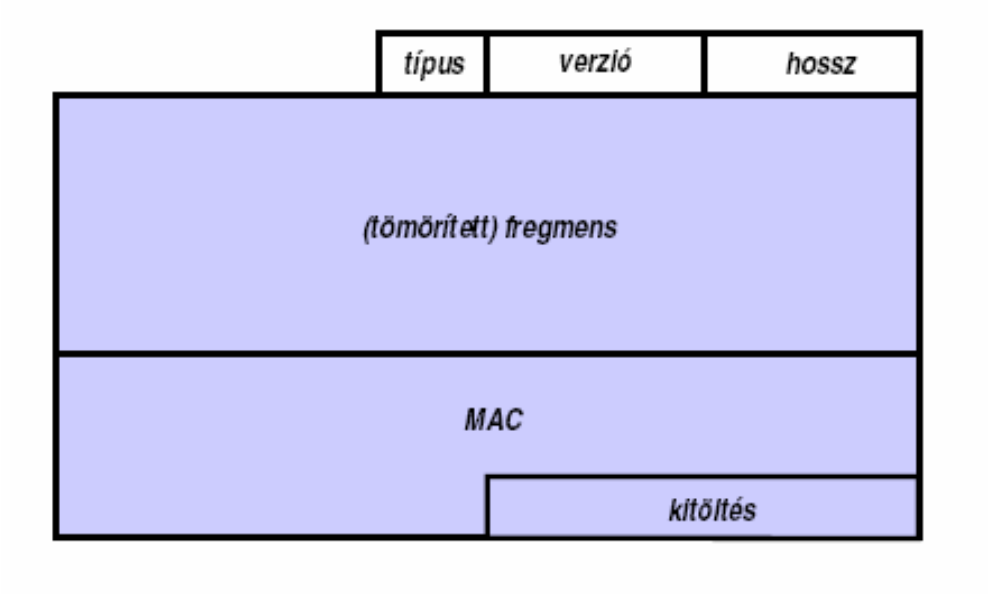
- *Change-Cipher-Spec protokoll*: A Change-Cipher-Spec protokoll egyetlen üzenetből áll, mely a Handshake protokoll kulcscsere részének végét jelzi. Ezen üzenet elküldése után az adott fél az új algoritmusokat és kulcsokat kezdi használni küldésre, a vétel azonban még mindig a Handshake előtti állapot 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.

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.

SSL Record protokoll



Az SSL Record protokoll üzenet formátuma

SSL Record protokoll

- *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 alkalmazás 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.

SSL Record protokoll

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

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

- 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.

SSL viszony (session) és az SSL kapcsolat (connection)

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.

(Mi is azt fogjuk most feltételezni, hogy a felek között egy viszony van, és a viszonyon belül több kapcsolat lehetséges.)

Viszony (session)

A viszony tartalmazza az állapot azon részleteit, amit a viszonyon belüli kapcsolatok megosztanak, közösen használnak.

A viszony részei:

- viszony azonosító,
- felek nyilvános kulcs tanúsítványa (ha rendelkeznek ilyennel),
- tömörítő algoritmus,
- rejtjelező és MAC algoritmus,
- ezen algoritmusok által használt méretek (pl. MAC kulcs hossza)
- mestertitok (master secret)
- “is resumable” jelzőbit (0/1: nem hozható létre/létrehozható új kapcsolat a viszonyon belül)

Kapcsolat (connection)

A kapcsolat részei:

- 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,
- blokkrejtjelező használata esetén az *IV*-k.

Tehát 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 mestertitok a viszonyhoz tartoznak.

SSL Handshake protokoll

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**

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.

Handshake protokoll négy fázisa

A Handshake protokoll négy fázisra tagolódik.

1. fázis (1. és 2. üzenet): 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.

2. fázis (3.6. üzenetek): a szerver a kiválasztott módszernek megfelelően végrehajtja a kulcscsere ráeső részét, míg a kliens a

3. fázis (7.9. üzenetek): a kliens teszi meg ugyanezt.

A harmadik fázis után, az addig kicserélt információkat felhasználva, mindkét fél előállítja az új kapcsolatkulcsokat.

4. fázis (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.

client-hello üzenet

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 (kliens preferenciái szerint rendezett lista, egy eleme pl.

SSL-RSAwith-3DES-EDE-CBC-SHA

RSA alapú kulcscsere módszer,
3DES rejtjelezés, EDE kongurációban, CBC módban,
SHA hash függvény és arra épülő MAC.

tömörítő algoritmusok: kliens preferenciái szerint rendezett lista

server-hello üzenet

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 véletlensz ámot, 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 és a tömörítő algoritmusok: A szerver választ egy algoritmus-kombinációt a kliens listájáról, és csak a kiválasztott kombináció azonosítóját küldi vissza a kliensnek.

Kulcscsere

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 méretu véletlen blokkot, amit a szerver nyilvános RSA kulcsával kódolva elküld a szervernek. Ebbol a véletlen blokkból aztán mind a kliens, mind a szerver eloállítja a közös mestertitkot.

Fix Diffie-Hellman: Fix Dife.Hellman-kulcscsere esetén a felek a Dife. Hellman-algoritmust használják, és az ennek eredményeként kialakult közös értékbol generálják a mestertitkot. A x jelző arra utal, hogy a szerver Diffie-Hellman-paraméterei (p, g, g^x mod p) xek, azokat egy hitelesítésszolgáltató aláírta, és az erről szóló tanúsítványt a kliens ellenőrizni tudja.

Kulcscsere

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.

Certificate üzenet

A Handshake protokoll második fázisának első üzenete a certicate 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

- nyilvános RSA rejtjelező kulcsot (RSA alapú kulcscsere), vagy
- RSA, vagy DSS aláírás ellenőrző kulcsot (RSA alapú kulcscsere, vagy egyszer használatos Diffie-Hellman), vagy
- szerver Diffie-Hellman paramétereit (fix Diffie-Hellman).

server-key-exchange

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.

client-key-exchange

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.

Ha a kliens küldött certificate üzenetet, akkor a harmadik fázist a **certificate-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 certificate üzenetben található aláírásellenőrzo kulccsal ellenőrizhető.

Közös mestertitok

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

Finished

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.

Record protocol kulcsai

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

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

Példa 1: RSA alapú kulcscsere

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

A hello üzenetek cseréje után a szerver a **certicate** 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.

Példa 2: Egyszer használatos Diffie-Hellman

A **hello** üzenetek cseréje után a szerver a **certicate** 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 mod *p*), majd ezeket DSS aláírással ellátva a **server-key-exchange** üzenetben elküldi a kliensnek. Végül a **certicate-request** üzenetben DSS aláírás ellenőrző kulcsot tartalmazó tanúsítványt kér a klienstől.

A kliens ennek megfelelően a **certicate** 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 mod *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 **certicate-verify** üzenetben.