4 PROTOCOALELE TLS și SSL

4.1 Arhitectura SSL/TLS

4.2 Serviciile de securitate ale SSL/TLS

4.3 Subprotocoalele SSL/TLS

4.4 Performanța SSL/TLS

4.5 Capcane de implementare

4.6 Arhitectura TLS fără fir

4.7 De la TLS la WTLS

4 PROTOCOALELE TLS și SSL

Secure Sockets Layer (SSL) și Transport Layer Security (TLS) sunt două protocoale utilizate pe scară largă pentru a asigura schimburi la nivelul Transport TCP/IP între un client și un server.

SSL versiunea 1.0 a fost folosită intern în cadrul Netscape (1994).

Versiunea 2.0 SSL a fost lansată publicului în 1994 și integrată în Netscape Navigator.

Versiunea 3.0 SSL a corectat deficiențele găsite în Versiunea 2.0 și a fost baza pentru RFC 2246 care a definit TLS 1.0 în 1999.

TLS 1.0 a fost permis pentru utilizare pentru a proteja datele federale din SUA; pe când, SSL v3 a fost tolerat în circumstanțe limitate, cu risc scăzut, cum ar fi accesarea siturilor Web ale furnizorilor care nu au sprijinit TLS.

TLS 1.0 a fost apoi adaptat la comunicațiile fără fir ca Wireless TLS (WTLS).

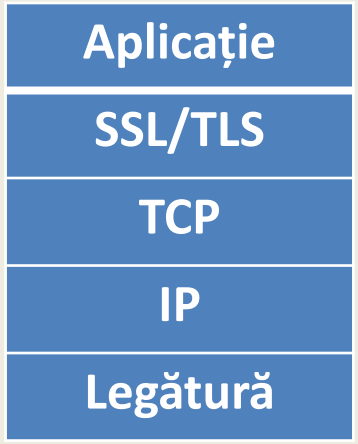
Două actualizări ale TLS în 2006 și 2008, respectiv TLS 1.1 și 1.2, sunt definite în RFC 4346 și 5246; aceste actualizări au inclus o varietate de contramăsuri și soluții la amenințările de securitate raportate.

În 2011, RFC 6151 a retras oficial Versiunea 2.0 SSL .

Datagram Transport Layer Security (DTLS) din RFC 4347 (actualizat în RFC 6347) a fost definit ca să ruleze pe lângă protocoalele nesigure de transport.

6.1 Arhitectura SSL/TLS

Locul SSL/TLS în stiva de protocoale TCP/IP.

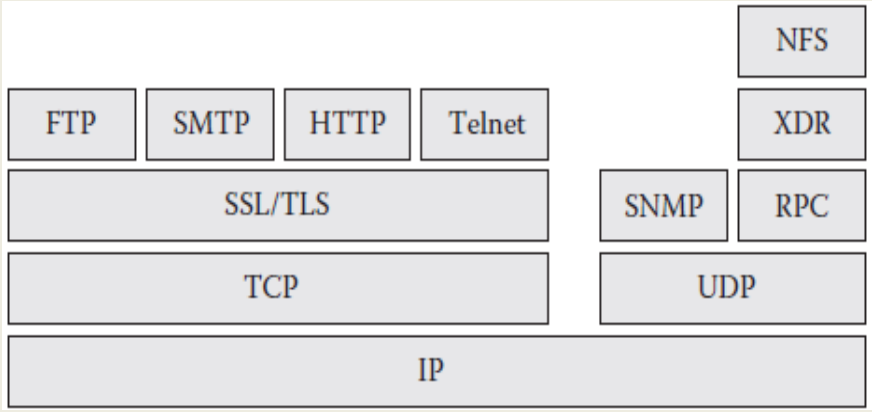


SSL/TLS funcționează deasupra stratului Transport, dar sub stratul Aplicație al modelului OSI. Strângerea de mână (*handshake*) pentru stabilirea sesiunii și a conexiunii are loc la nivelul Sesiune. În timpul acestei strângeri de mână, diferiți parametri cifrografici sunt stabiliți pentru a fi utilizați în stratul Prezentare pentru a suporta diferite protocoale de aplicație, cum ar fi S-HTTP (*Secure HyperText Transfer Protocol*) din RFC 2660 (1999).

SSL/TLS sunt integrate ușor în exploratoarele Web.

6.1 Arhitectura SSL/TLS

În figură este prezentată corespondența dintre SSL/TLS și alte protocoale Internet. Orice protocol de transport care, ca și TCP, oferă o transmisie fiabilă, poate profita de serviciile de securitate ale SSL/TLS.



Totodată, SSL/TLS nu protejează schimburile UDP (*User Datagram Protocol*), deoarece UDP nu oferă un serviciu de transport fiabil. În acest caz, întreruperile de flux din cauza pierderilor de pachete IP pot fi interpretate incorect ca pauze de securitate care ar forța deconectarea comunicației. În consecință, SSL/TLS nu protejează protocoalele care rulează pe UDP, cum ar fi SNMP, sistemul de fișiere de rețea (NFS), DNS (*Domain Name Service*) și VoIP (VoIP).

DTLS (RFC 4347) poate asigura traficul aplicațiilor ce rulează pe UDP cu mecanisme precum numere de secvențe explicite, cronometre de retransmisie și detectare a atacurilor de replicare (*replay*).

6.1 Arhitectura SSL/TLS

Numerele de porturi atribuite de IANA unor aplicații securizate pentru comunicarea cu SSL/TLS:

Secure Protocol	Port	Nonsecure Protocol	Application
S-HTTP	443	HTTP	Secure request–response transactions
SSMTP	465	SMTP	E-mail
SNNTP	563	NNTP	Network news
SSL-LDAP	636	LDAP	Light version of X.500
SPOP3	995	POP3	Remote access of mailbox with message download

6.1 Arhitectura SSL/TLS

Numerele de porturi folosite, prin convenție extinsă (fără o atribuire oficială de către IANA), de unele aplicații securizate pentru comunicarea cu SSL/TLS:

Secure Protocol	Port	Nonsecure Protocol	Application
ftp-DATA	889	ftp	File transfer
ftps	990	ftp	Control of file transfer
IMAPS	991	IMAP4	Remote access to mail box with or without downloading of messages
TELNETS	992	Telnet	Remote access to a computer
IRCS	993	IRC	Internet chat; text conferencing

6.2 Serviciile de securitate ale SSL/TLS

SSL/TLS definesc un cadru care să utilizeze algoritmi de cifrare și de hash-are negociați între cele două părți comunicante pentru a oferi trei servicii de securitate:

- **autentificare;**
- **Integritate;**
- **confidențialitate.**

Cu ajutorul unei i-semnături, este posibil să se furnizeze elementele necesare pentru un serviciu de nerepudiere.

Acest cadru este deschis pentru integrarea de algoritmi noi.

SSL/TLS combină operațiunile de stabilire a cheii, confidențialitate, semnătură și hash-are într-un pachet denumit suită de cifruri (*cipher suite*).

Există peste 300 de seturi de cifruri standardizate definite în diverse documente.

În TLS 1.2, suita de cifruri obligatorie este formată din RSA pentru semnătura certificatului, AES pentru cifrare cu o cheie de 128 biți și SHA-1 pentru verificarea integrității.

6.2 Serviciile de securitate ale SSL/TLS

6.2.1 Autentificarea

Autentificarea folosește un certificat conform Recomandării ITU-T X.509 Vers. 3. Aceasta are loc numai la stabilirea sesiunii și înainte ca primul set de date să fi fost transmis.

Acest serviciu a fost opțional în Vers. 2 a SSL și a devenit obligatoriu pentru server în Vers. 3.0 și pentru TLS. Serverul poate solicita clientului să se autentifice și poate refuza să stabilească sesiunea deoarece lipsește certificatul.

Stabilirea sau schimbul de chei este procesul de creare a unei chei secrete partajate care să fie utilizată pentru cifrare. În timpul stabilirii cheii, autentificarea poate fi statică sau dinamică.

În autentificarea statică, cheia publică de cifrare este extrasă din certificatul celeilalte părți, de obicei certificatul serverului.

În autentificarea dinamică, certificatul serverului conține o cheie de semnătură utilizată pentru a semna o cheie de cifrare temporară pentru a proteja integritatea acesteia. Câteva chei temporare sunt folosite pentru a proteja datele schimbate pentru a stabili cheia de cifrare simetrică a sesiunii.

6.2 Serviciile de securitate ale SSL/TLS

6.2.1 Autentificarea

Mai întâi a fost implementată autentificarea dinamică, pentru a evita limitarea cheilor de cifrare RSA la 512 de biți, impusă de restricțiile de export din SUA.

Cu autentificarea dinamică, o cheie mai mare (1024 biți) ar putea fi utilizată pentru a semna cheia temporară de 512 biți.

Un alt avantaj al autentificării dinamice este acela că evită utilizarea unei chei statice care, dacă este compromisă, ar expune riscului fiecare sesiune SSL/TLS care a fost stabilită vreodată cu cheia respectivă.

SSL/TLS utilizează RSA și algoritmul Diffie-Hellman pentru schimbul de chei.

În 1999, la protocolul TLS a fost adăugat (RFC 2712) protocolul de autentificare Kerberos (V5).

6.2.1 Autentificarea

Există trei variante ale comunicațiilor Diffie-Hellman – statice (sau fixe), efemere și anonime:

1. În cazul *Diffie-Hellman statice*, parametrii publici sunt conținuți în certificatul serverului. Clientul oferă parametrii publici corespunzători, fie într-un certificat, fie prin schimb de mesaje, dacă nu posedă. În acest caz, serverul oferă o cheie statică Diffie-Hellman în timp ce cheia clientului este efemeră sau temporară.

2. *Diffie-Hellman efemer* este o tehnică pentru a crea chei secrete de o singură dată. Fiecare parte trimite cheile publice Diffie-Hellman semnate cu cheia privată RSA. Destinatarul utilizează cheia publică corespunzătoare pentru a verifica certificatul. Cheile efemere ating secretul înainte perfect (*perfect forward secrecy* – PFS), deoarece cheile temporare de pe ambele părți nu pot fi recuperate odată ce acestea sunt distruse. Astfel, compromisul unei chei de sesiune sau al uneia private nu expune riscului nici o cheie de sesiune anterioară.

3. În cazul *Diffie-Hellman anonim*, parametrii publici Diffie-Hellman sunt schimbați fără autentificare. Această metodă, totuși, este susceptibilă la atacurile omul-în-mijloc.

6.2.1 Autentificarea

SSL și TLS 1.0 au folosit algoritmi clasificați anterior, aprobați pentru aplicații cifrografice pe cartela Fortezza PCMCIA (NIST, 1994).

Cartela Fortezza folosește o variantă a algoritmului Diffie-Hellman numit algoritmul de schimb al cheilor Fortezza (*Fortezza Key Exchange Algorithm* – KEA) pentru acordul privind cheile și un algoritm de cifrare bloc numit SKIPJACK pentru cifrare.

Semnăturile sunt calculate cu DSA, iar rezumatele mesajelor sunt calculate cu ajutorul funcției hash SHA-1.

Fortezza a fost integrat și în TLS 1.1.

În TLS 1.2, pentru schimbul de chei și semnătură, a fost introdusă cifrografia curbelor eliptice. ♦

6.2.2 Confidențialitatea

Confidențialitatea mesajelor se bazează pe utilizarea algoritmilor de cifrare simetrică, fie flux (continuă), fie bloc. Același algoritm este folosit de ambele părți, dar fiecare parte utilizează cheia proprie, partajând-o cu cealaltă parte. Pe partea clientului, cheia este numită *client_write_key*, iar pe partea server aceasta este numită *server_write_key*.

Inițial, algoritmii care puteau fi utilizați cu SSL au fost DES, triple DES (3DES), DES40, RC2, RC4 cu o cheie de 128 biți (RC4-128) sau 40 de biți (RC4-40), IDEA și algoritmul SKIPJACK de la Fortezza.

DES40 și RC4-40 au folosit chei de 40 de biți pentru a respecta restricțiile pe care guvernul federal al SUA le-a impus exportului de programe cifrografice; aceste restricții au fost ridicate în 1999.

Multe exploratoare utilizează suite de cifruri cu RC4, deoarece sunt rapide. Începând cu 2010, în cererile aprobate de autoritățile federale americane sunt necesare dimensiuni ale cheilor mai mari de 1024 de biți.

RFC 4132 a adăugat suitele de cifruri Camellia la TLS 1.0.

În 2006, TLS 1.1 a adăugat AES în suitele de cifruri, renunțând la Fortezza.

6.2.2 Confidențialitatea

TLS 1.2 suportă cifrografia curbelor eliptice (ECC) definită în ANSI X9.62 (2005) și Cifrarea autentificată cu date asociate (*Authenticated Encryption with Associated Data* – AEAD), eliminând DES și IDEA.

În AEAD, textul este cifrat simultan și protejat de integritate. Intrarea poate fi de orice lungime; ieșirea cu cifru este în general mai mare decât intrarea din cauza valorii de verificare a integrității.

Compatibilitatea înapoi cu SSL Vers. 2.0 a fost eliminată ca cerință.

Astfel, TLS 1.2 oferă trei metode de cifrare:

1. Cifrarea în modul CBC folosind cifrurile bloc 3DES sau AES.

2. Cifrarea în flux folosind cifrul fluxului RC4 cu o cheie de 128 biți.

3. AEAD într-unul din cele două moduri de operare: Galois/Counter Mode (GCM) sau contorul cu blocarea blocului de cifru și codul de autentificare a mesajelor (*cipher block chaining and message authentication code* – CCM) așa cum este descris în RFC 5116. CCM și GCM se bazează pe algoritmul AES cu chei de 128 sau 256 de biți. AES-CGM, cu toate acestea, reduce performanța.

S-a constatat că modul CBC de cifrare în TLS și cifrarea flux cu RC4 sunt vulnerabile la atacuri. ♦

6.2.3 Integritatea

Integritatea datelor este asigurată cu funcții hash care utilizează procedura HMAC. Funcțiile hash folosite pot fi Secure Hash (SHA) sau MD5. Rezumatul este tratat printr-o serie de operații care depind de o cheie secretă, iar rezultatul este numit cod de autentificare a mesajelor (MAC). Această operație servește și ca autentificare, deoarece cunoașterea secretului utilizat în cifrarea rezumatului este restricționată la cele două părți.

Până la TLS 1.2, au fost utilizate HMAC-MD5 si HMAC-SHA1.

Cu TLS 1.2, clientul poate specifica funcția hash care va fi utilizată în i-semnatură independent de algoritmul de cifrare. În plus, utilizarea unei chei de 256 de biți pentru hash-are a devenit implicită (HMAC-SHA256), rezultând un MAC de 32 de octeți. Astfel, cu TLS1.2, SHA poate fi folosit cu chei de 224, 256, 384 și 512 biți.

În 2011, RFC 6151 a indicat că MD5 nu mai este acceptabil în cazul în care este necesară rezistența la coliziune, cum ar fi i-semnăturile, dar aplicațiile moștenite pot încă să utilizeze HMAC-MD5.

În TLS 1.2, un certificat care conține cheia pentru un algoritm de semnătură poate fi semnat utilizând un algoritm diferit. În specificațiile anterioare, algoritmul trebuia să fie același. De asemenea, verificarea integrității poate utiliza SHA-256 în suitele de cifruri relevante. ♦

6.2.4 Sumarul algoritmilor de securitate

Diferitele securizări susținute de SSL/TLS [3]:

	Algorithms		
Function	**SSL/TLS 1.0**	**TLS 1.1**	**TLS 1.2**
Key exchange	RSA, Diffie–Hellman, Fortezza, Kerberos	RSA, Diffie–Hellman	RSA, Diffie–Hellman, Elliptic Curve Diffie–Hellman
Signature	Digital Signature Algorithm (DSA), RSA	Diffie–Hellman	RSA, Digital Signature Algorithm (DSA), Elliptic Curve RSA. Elliptic Curve Digital Signature Algorithm (ECDSA)
Stream symmetric encryption	RC4 with keys of 40 or 128 bits	RC4 with 128 bits, Camellia	RC4 with 128 bits
Block symmetric encryption	DES, DES40, 3DES RC2 with 128 bits, IDEA, SKIPJACK	DES, 3DES, RC2, IDEA, AES with 128 and 256 bits, Camellia	3DES, AES with 128 and 256 bits
Authenticate Encryption with Associated Data (AEAD)	—	—	AES CGM, AES CCM
Hashing	MD5, SHA-1	MD5, SHA-1	MD5, SHA-1 with keys of 64, and 256

În urma dezvăluirii în 2013 a supravegherii masive a traficului pe Internet de către diferite guverne, multe companii și organizații au răspuns prin creșterea mărimii cheilor lor de cifrare TLS/SSL, de obicei de la 1024 la 2048 de biți. ♦

6.2.5 Vulnerabilități cifrografice ale TLS

Chiar dacă TLS 1.2 specifică două clase generale de metode de cifrare, au fost deja constatate unele vulnerabilități ce afectează securitatea tranzacțiilor, inclusiv:

- **BEAST (*browser exploit against* SSL/TLS) ce exploatează vectorul de inițializare utilizat în modul CBC de cifrare bloc;**
- **folosirea părtinirilor (tendențiozități - *biases*) statistice la cifrare.**

Atacul vectorial de inițiere (Atacul BEAST)

În modul CBC de cifrare bloc, SSL 3.0 și TLS 1.0 generează vectorul de inițializare a primului bloc. Pentru mesaje mai lungi decât un bloc, vectorii de inițializare ulteriori sunt înlănțuiți, adică fiecare vector nou este ultimul text cifrat din mesajul precedent.

Lanționarea vectorilor de inițializare între mesaje este considerată o variantă slabă a modului CBC de cifrare, care necesită în mod obișnuit un vector proaspăt de inițializare pentru fiecare bloc.

6.2.5 Vulnerabilități cifrografice ale TLS

Vectorii de inițiere a lanțului în modul CBC permit recuperarea completă a textului, în următoarele condiții:

1. Atacatorul știe ce bloc de text conține așa informații ca parola sau un modul cookie de sesiune HTTP.

2. Blocurile cifrate de text sunt cunoscute.

3. Atacantul cunoaște valorile vectorului de inițializare pentru blocul următor.

4. Atacatorul este capabil să aleagă următorul text care urmează să fie cifrat într-un mesaj lung. Acest lucru este posibil prin intermediul unui plug-in rău intenționat sau al unui JavaScript în exploratorul utilizatorului.

În condițiile menționate mai sus, atacatorul poate folosi textul ales pentru a valida ghiciri repetate cu privire la valoarea unui anumit bloc de text până când este identificat cel corect.

O metodă de rezolvare este de a folosi un mesaj cu lungimea zero (adică, numai cu HMAC și padding) doar pentru a genera un vector de inițializare pe care un atacator ar putea să nu îl cunoască în avans. Multe servere, cu toate acestea, consideră un mesaj gol ca un semnal pentru un sfârșit de fișier, deci acest lucru nu este dovada completă.

TLS 1.1 și TLS 1.2 nu sunt vulnerabile la acest atac, deoarece un vector de inițializare este generat din nou pentru fiecare bloc.

6.2.5 Vulnerabilități cifrografice ale TLS

Atacul de părtinire statistică a RC4

Utilizarea RC4 a crescut după dezvăluirea atacului BEAST în 2011, ca protecție față de suitele CBC vulnerabile în SSL/TLS 1.0. În 2012 aproximativ 50% din traficul TLS utiliza RC4.

Din păcate, în 2013, s-a demonstrat că, în anumite condiții, părtinirile (*biases*) statistice cunoscute din textul cifrat RC4 ar putea fi utilizate pentru recuperarea primilor 220 de octeți după mesajul terminat al protocolului Handshake, chiar și fără cunoașterea prealabilă a textului scris.

Mai întâi, același text trebuie să se afle întotdeauna în aceeași poziție cu textul original. În special, segmentul cifrat poate include parola pentru a accesa un server de i-poștă sau antete cookie HTTP codificate, care încep întotdeauna cu șirul "Cookie" și se termină cu un caracter de linie nouă. De asemenea, același text trebuie sa fie cifrat cu diferite chei.

Pentru a efectua analiza statistică, trebuie sa fie disponibile între 2^{28} si 2^{32} cifrări independente.

6.2.5 Vulnerabilități cifrografice ale TLS

Pentru a genera un număr așa de mare (2^{28}) de cifrări independente, un explorator ar putea fi infectat cu un program rău intenționat JavaScript care ar trimite în mod repetat solicitări HTTPS către un server distant. Deoarece cookie-urile sunt incluse automat în fiecare dintre aceste cereri într-o locație predefinită, aceste solicitări pot fi direcționate spre analiză.

Altă posibilitate constă în încheierea sesiunii TLS după transmiterea cookie-ului cifrat, forțând astfel crearea unei noi sesiuni TLS pentru următoarea solicitare HTTPS. În cazul unui server de i-poștă, resetarea conexiunii TCP între clientul de i-poștă și server după autentificare poate declanșa anumite configurații ale clienților pentru a relua sesiunea TLS și a retransmite parola cifrată.

Atacul nu poate fi încă practic în toate situațiile, dar pot exista și alte vulnerabilități necunoscute. De fapt autoritățile federale ale SUA nu au aprobat RC4 pentru aplicațiile cifrografice și au permis utilizarea lui în circumstanțe foarte limitate, cum ar fi comandarea bunurilor furnizorilor care nu susțin nimic mai puternic decât RC4.

Există un consens general pentru a evita RC4 complet în TLS.♦

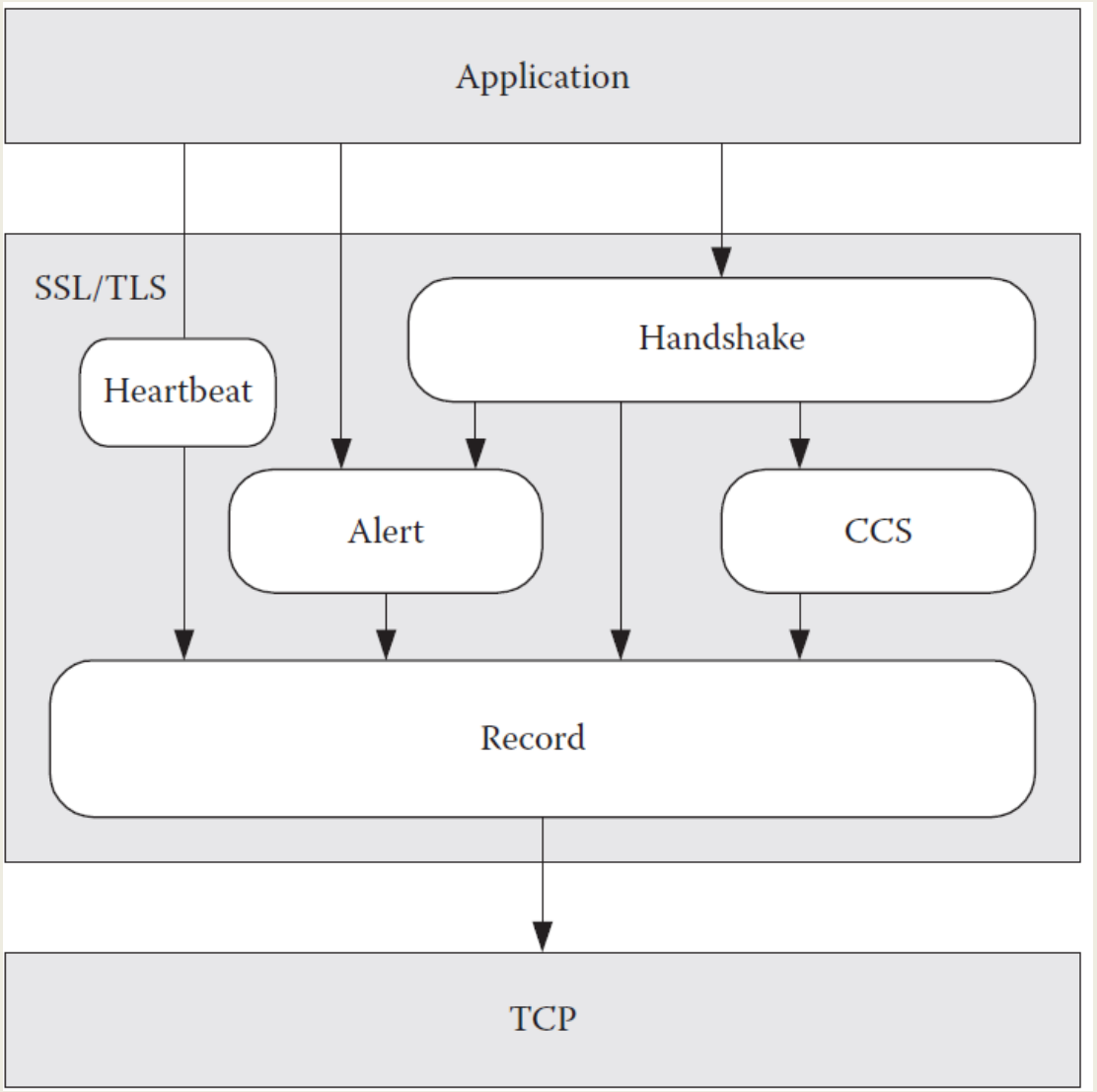
6.3 Subprotocoalele SSL/TLS

6.3.1 Aspecte generale

Protocoalele inițiale SSL/TLS includ patru subprotocoale:

- **Handshake;**
- **Record;**
- **ChangeCipherSpec (CCS);**
- **Alert;**

în 2012, a fost adăugat și subprotocolul Heartbeat.



6.3 Subprotocoalele SSL/TLS

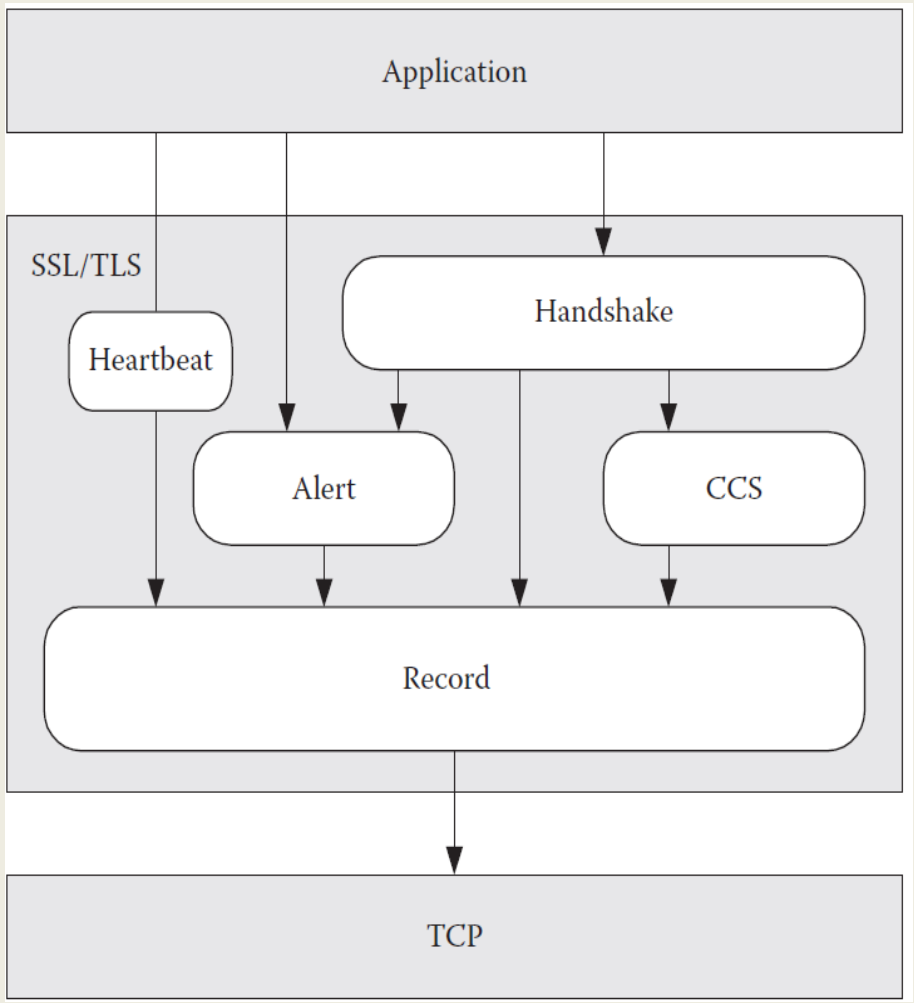
Protocolul Handshake folosește cifrografia asimetrică pentru autentificarea părților comunicante, pentru negocierea algoritmilor de cifrare și hash-are și pentru schimbul unui secret, PreMasterSecret.

Funcția protocolului CCS este de a semnala la nivelul Record toate modificările parametrilor de securitate.

Protocolul Alert indică erorile întâmpinate în timpul verificării mesajelor, precum și orice incompatibilitate care poate apărea în timpul operației de Handshake.

Protocolul Record aplică cifrografia simetrică cu toți parametrii de securitate negociați pentru a proteja datele aplicației, precum și mesajele generate de protocoalele Handshake, CCS sau Alert.

Protocolul Heartbeat este folosit pentru a menține o conexiune chiar și în absența transferului de date.



6.3.2 Schimburi SSL/TLS

Schimburile SSL/TLS au loc în două etape:

1. **Identificarea părților, negocierea atributelor cifrografice și generarea și partajarea cheilor.**
2. **Schimbul de date. Semnalarea intruziunii sau erorilor (dacă apar).**

Sesiunea SSL/TLS reprezintă o asociere între două entități ce au un set comun de parametri și atribute cifrografice. Ori de câte ori un client se conectează la un server, este lansată o sesiune SSL sau TLS. Dacă clientul se conectează la un alt proces de pe server, o nouă sesiune este pornită fără a întrerupe sesiunea inițială. Dacă clientul se întoarce mai târziu la primul server și dorește să conserve alegerile cifrografice deja efectuate, clientul va cere primului proces să reia vechea sesiune în loc să pornească una nouă.

Pentru a limita riscurile atacurilor prin interceptarea mesajelor, SSL/TLS recomandă limitarea duratei sesiunii la cel mult 24 de ore, durata fiind la discreția serverului. În plus, sesiunea întreruptă poate fi reluată numai dacă au fost utilizate procedurile adecvate de suspendare.

Conceptul de conexiune în SSL/TLS a fost introdus pentru a permite unei aplicații să actualizeze anumite atribute de securitate (de exemplu, cheia de cifrare) fără a afecta toate celelalte atribute negociate la începutul sesiunii. O sesiune poate conține mai multe conexiuni aflate sub controlul aplicațiilor. Sesiunile și conexiunile SSL/TSL pot fi ilustrate cu ajutorul variabilelor lor de stare și a parametrilor de securitate asociați.♦

6.3.3 Variabilele de stare ale unei sesiuni SSL/TLS

O sesiune SSL/TLS este identificată în mod unic de șase variabile de stare:

1. *ID sesiune* - o secvență arbitrară de 32 octeți pe care serverul o selectează pentru a identifica o sesiune activă sau una ce poate fi reactivată.

2. *Certificatul egal-la-egal* - al corespondentului, este conform cu Vers. 3 a X.509.

3. *Metoda de comprimare* - negocierea unei metode de comprimare a datelor.

4. *Specificația cifrului* - definește algoritmii de cifrare și hash-are de folosit dintr-o listă prestabilită.

5. *MasterSecret*, de 48 octeți, este partajat între client și server. Este folosit pentru a genera toate celelalte secrete; rămâne valabil pentru întreaga sesiune.

6. Un steag marcat *este reluabil* pentru a descrie dacă sesiunea poate fi utilizată pentru a deschide noi conexiuni.

Parametrii suitei de cifruri sunt negociați în mod clar în timpul stabilirii sesiunii prin definirea a cinci elemente (în SSL și TLS 1.0):

1. Tipul cifrării, indiferent dacă este vorba de un cifru flux sau unul bloc.

2. Algoritmul de cifrare.

3. Algoritmul de hash-are.

4. Dimensiunea rezumatului.

5. O valoare binară care indică permisiunea de a exporta algoritmul de cifrare conform legii SUA privind exportul de cifrografie. Acest element a fost eliminat în TLS 1.1 și 1.2.♦

6.3.4 Variabilele de stare ale unei conexiuni SSL/TLS

Parametrii care definesc starea unei conexiuni în timpul unei sesiuni SSL/TLS sunt cei care vor fi "actualizați" când se va stabili o nouă conexiune.

Fiecare conexiune are proprii parametri cifrografici (chei și vectori de inițializare), dar toate conexiunile din aceeași sesiune partajează MasterSecret.

În plus, cheile de confidențialitate pentru fiecare direcție rămân independente unele de altele.

Generarea de numere aleatorii de calitate este de interes atât pentru client, cât și pentru server.

Programul de Validare a Modulelor Cifrografice (*Cryptographic Module Validation Program* – CMVP) este o colaborare a NIST și a Instituției de Securitate a Comunicațiilor din Canada (*Communications Security Establishment of Canada*) pentru validarea produselor comerciale care presupun generarea de numere aleatorii.

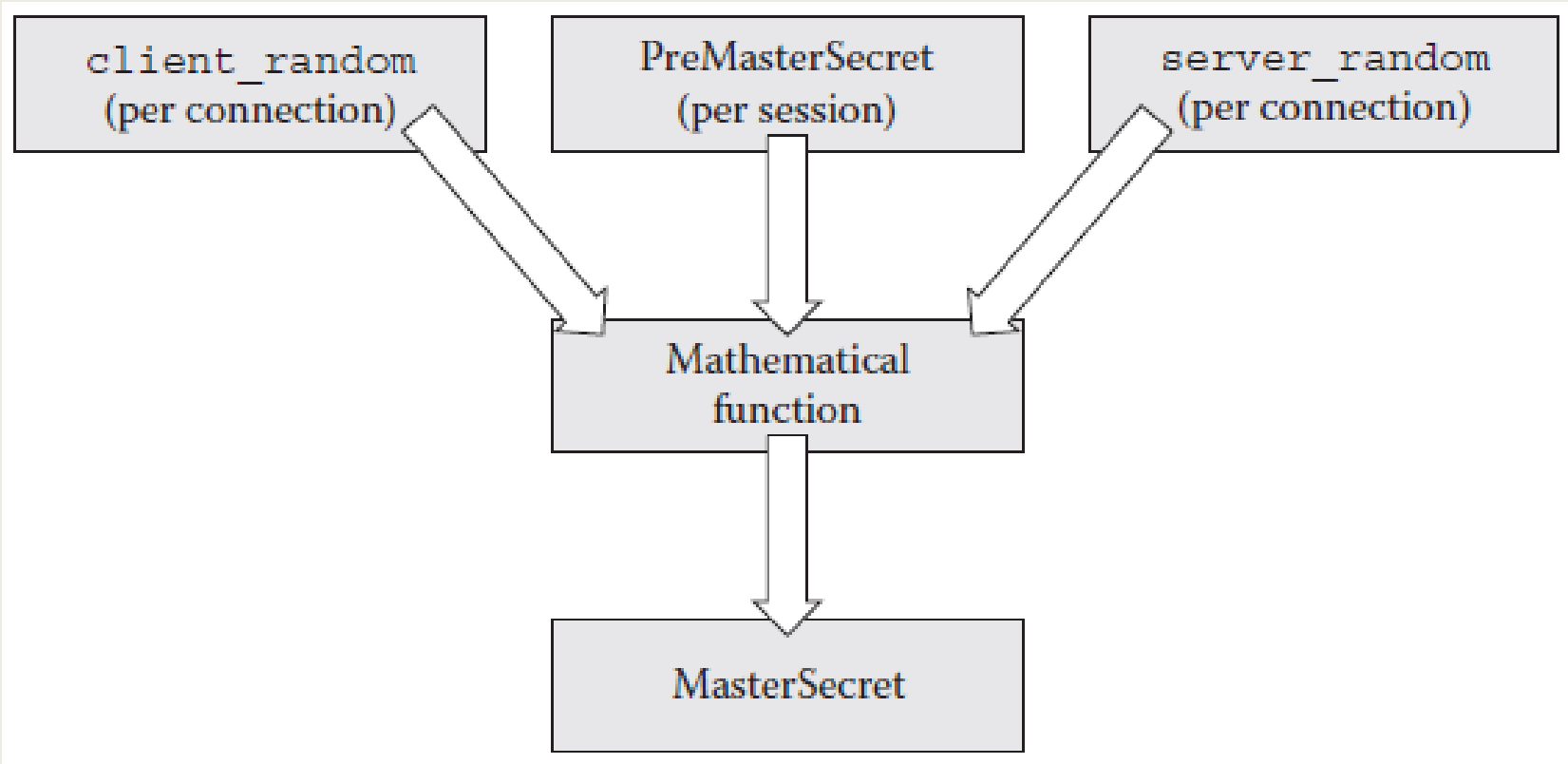
6.3.4 Variabilele de stare ale unei conexiuni SSL/TLS

Parametrii de stare ale unei conexiuni sunt:

- **două numere aleatoare (*server _random* și *client_random*) a câte 32 octeți fiecare. Aceste numere sunt generate de server și de client, respectiv, la stabilirea unei sesiuni și pentru fiecare conexiune nouă. Cheia secretă va fi derivată folosind aceste numere. La deschiderea unei sesiuni, aceste numere sunt în clar. Ulterior, în timpul stabilirii unei conexiuni suplimentare, cele două numerele sunt transmise cifrate. Utilizarea acestor numere protejează împotriva atacurilor de replicare (replay);**
- **două chei secrete, *server_MAC_write_secret* și *client_MAC_write_secret*. Aceste chei vor fi utilizate în cadrul funcțiilor hash pentru a calcula codul de autentificare a mesajelor (MAC). Dimensiunea MAC depinde de algoritmul utilizat, de exemplu, 16 octeți pentru SHA-1 sau 20 octeți pentru MD5;**
- **două chei pentru cifrarea simetrică a datelor, una pentru partea serverului și cealaltă pentru partea clientului. În timp ce același algoritm este utilizat de ambele părți, fiecare poate să utilizeze cheia proprie, server_write_key sau client_write_key, respectiv, cu condiția să le partajeze cu cealaltă parte;**
- **doi vectori de inițializare (IV) pentru cifrarea simetrică în modul CBC - un vector pe partea server și celalalt pe partea client. Fiecare este un șir aleatoriu care este "exclusiv SAU-izat" cu mesajul text simplu înainte de cifrare. Dimensiunea acestora depinde de algoritmul selectat. IV-urile adaugă stocasticitate la mesaj și sunt trimise de-a lungul mesajului în clar.**
- **două numere de secvențe, una pentru server și cealaltă pentru client, fiecare codificată pe 8 octeți. Aceste numere de secvență sunt menținute separat pentru fiecare conexiune și sunt incrementate ori de câte ori un mesaj este trimis pe această conexiune. Acest mecanism oferă o anumită protecție împotriva atacurilor de replicare, prevenind refolosirea mesajelor vechi.♦**

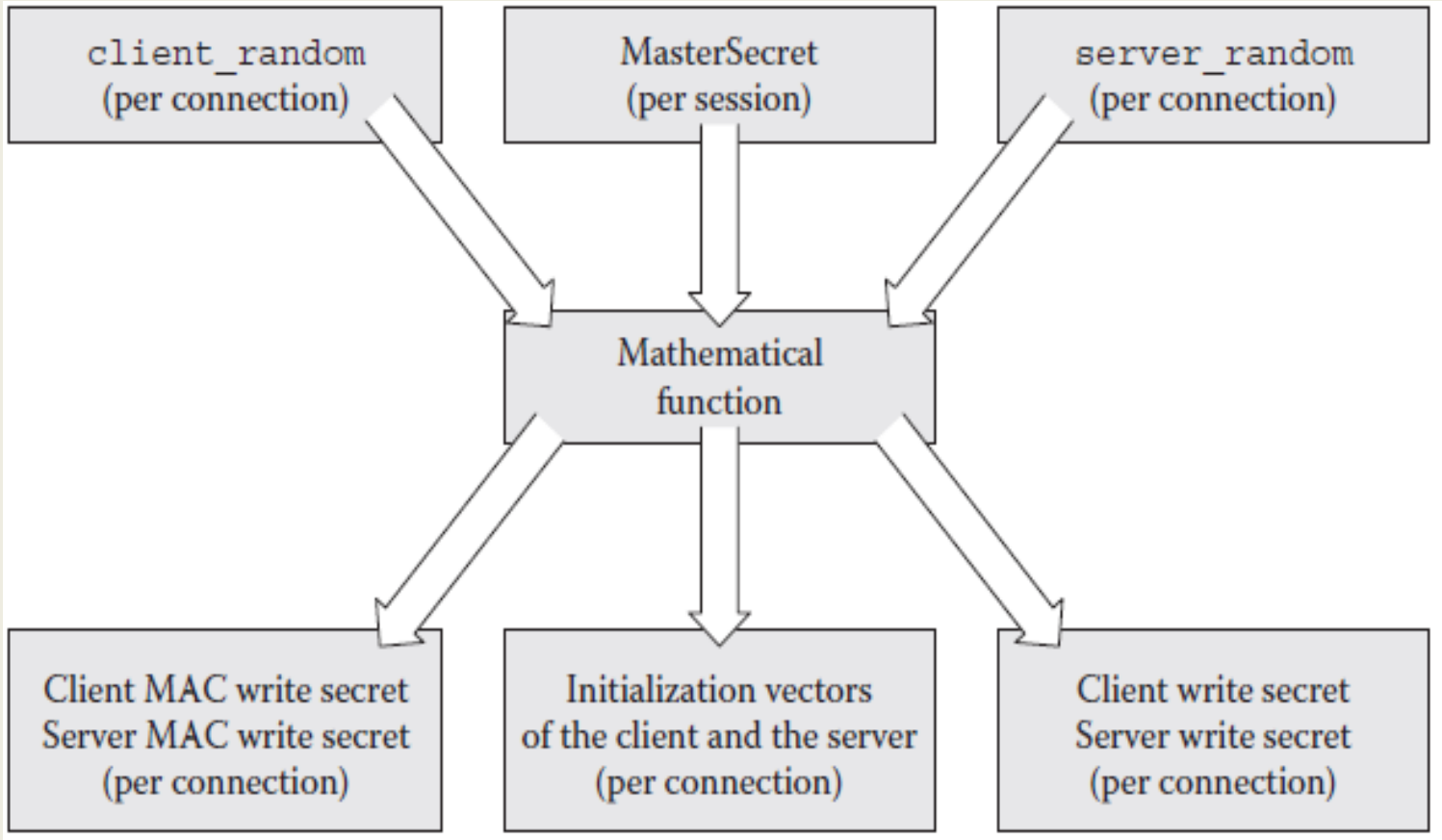
6.3.5 Rezumatul calculului parametrilor

Figura ilustrează calcularea MasterSecret pornind de la PreMasterSecret și parametrii *client_random* și *server_random*. Valoarea MasterSecret va rămâne constantă pe durata sesiunii. Parametrii *client_random* și *server_random* sunt schimbați în modul clar în timp ce PreMasterSecret este schimbat confidențial cu ajutorul algoritmului de schimb al cheilor.



6.3.5 Rezumatul calculului parametrilor

Calculul cheilor de cifrare, de hash-are și a vectorilor de inițializare începe de la variabilele MasterSecret, *client_random* și *server_random* în maniera descrisă în figură. Deschiderea unei noi conexiuni va conduce la recalcularea variabilelor *client_random* și *server_random*, deși valoarea MasterSecret rămâne neschimbată. În consecință, la deschiderea unei conexiuni, variabilele *client_write_key* și, respectiv, *server_write_key* vor fi recalculate. ♦



6.3.6 Protocolul Handshake

Operarea generală

Protocolul Handshake descrie o serie de schimburi de mesaje între client și server pentru a stabili canalul SSL/TLS, utilizând un set de algoritmi și parametri de securitate negociați.

Handshake începe cu autentificarea obligatorie a serverului, autentificarea clientului fiind opțională.

Odată ce autentificarea este stabilită, ambele părți se îndreaptă către faza de negociere pentru a selecta suita de cifruri care va fi utilizată pe durata sesiunii.

Astfel, protocolul Handshake condiționează întregul proces de transfer de date securizat, ceea ce îl face o țintă primordială pentru potențialii agresori.

6.3.6 Protocolul Handshake

Deschiderea unei sesiuni noi

O sesiune, o inițiază clientul, începe cu transmiterea mesajului ClientHello către server. Prin mesajul ServerHello, serverul avertizează clientul că este gata. Schimburile ulterioare au loc în același mod.

În SSL Vers. 3.0 și TLS, schimburile care au loc în timpul deschiderii unei sesiuni noi cuprind următoarele patru etape:

1. **Identificarea suitelor de cifruri disponibile la fiecare sit Web.**
2. **Autentificarea serverului.**
3. **Schimbul de secrete.**
4. **Verificarea și confirmarea mesajelor schimbate.**

În 2011, RFC 6176 a interzis suportul SSL Vers. 2.0 în noile implementări. Serverele TLS pot continua să accepte mesajele ClientHello Vers. 2.0, dar ar întrerupe conexiunea în cazul în care clientul nu acceptă un protocol de versiune mai recentă.

Primele mesaje schimbate între client și server, ClientHello și ServerHello, permit negocierea parametrilor sesiunii, algoritmilor de cifrare și secretele sesiunii. La finalul negocierii, clientul și serverul aleg versiunea protocolului comun între cele două părți.

6.3.6 Protocolul Handshake

Toate mesajele din protocolul Handshake au un antet de 5 octeți:

- **tipul mesajului este pe 1 octet;**
- **versiunea protocolului este pe 2 octeți;**
- **lungimea mesajului este pe 2 octeți.**

Dimensiunile câmpurilor variabile din mesajele ClientHello și ServerHello sunt:

Field	Maximum Length in Octets	Number of Octets to Code the Length
Session identifier	32	1
List of cipher suites	65,534	2
List of compression methods	255	1
List of extensions	65,534	2

TLS 1.2 permite clientului și serverului să negocieze următorii parametri:

- **versiunea de protocol;**
- **suita de cifruri;**
- **algoritmul de compresie;**
- **lista de extensii.**

6.3.6 Protocolul Handshake

Datele mesajului ClientHello includ următoarele elemente:

- **numerele aleatoare *client_random* și *server_random*, fiecare de 32 octeți în două câmpuri: primul de 4 octeți conține timpul universal al ceasului intern al clientului; cel de-al doilea pe 28 octeți include un număr aleator;**
- **identificatorul de sesiune de 0-32 octeți, precedat de un câmp ce indică lungimea acestuia. Când o sesiune este deschisă pentru prima dată, lungimea identificat. este de 1 octet, iar după deschiderea unei conexiuni - de 32 octeți;**
- **Lista suitelor de cifruri acceptate de client pe 2 octeți. Odată ce este selectată o suită de cifruri, ea va fi aplicată tuturor conexiunilor din aceeași sesiune;**
- **Lista metodelor de comprimare pe care clientul le poate suporta. Metoda este codificată pe 1 octet. SSL/TLS suportă până la 255 de metode distincte, dar trei au fost standardizate: fără compresie, identificată prin codul "0"; algoritmul DEFLATE (RFC 1951), identificat prin "1" (RFC 3749); algoritmul Lempel-Ziv-Stac (LZS) (RFC 1967) și identificat prin "64" (RFC 3943);**
- **începând cu TLS 1.2, ClientHello include extensii TLS, prin care clientul poate solicita serverului să furnizeze funcționalități suplimentare. Serverele indică în mesajul ServerHello extensiile pe care le suportă; cu toate acestea, nu li se permite să inițieze o solicitare de extensie.**

6.3.6 Protocolul Handshake

După trimiterea mesajului ClientHello, clientul așteaptă sosirea mesajului ServerHello. Acest mesaj trebuie să indice versiunea protocolului și identificatorul de sesiune unic pe care serverul l-a selectat. Mesajul conține, de asemenea, numărul aleator *server_random*. Prin schimbarea numerelor aleatoare, *client_random* și *server_random*, fiecare parte va putea să reproducă secretele corespondentului său, partajând astfel secretele.

Mesajul include suita de cifruri pe care serverul a selectat-o din lista prezentată de client. Dacă nu este disponibilă o suită comună, serverul (sau clientul, după caz) va genera mesajul de eroare *close_notify* așa cum este specificat de protocolul Alert și sesiunea va fi abandonată. Unele servere nu trimit mesajul de *close_notify*, ci închid sesiunea TCP fără avertisment.

Această negociere are loc în mod clar. Un intrus poate încerca să intercepteze mesajul ClientHello și să îl înlocuiască cu unul fals pentru a solicita un algoritm mai puțin robust. Protecția împotriva acestui atac poate fi realizată prin mesajul *Finished* care este schimbat la sfârșitul Handshake-ului. Acest mesaj este primul care trebuie protejat doar cu algoritmi, chei și secrete negociate. Destinatarul mesajului *Finished* trebuie să verifice dacă conținutul este corect. Odată ce o parte și-a trimis mesajul *Finished* și a primit și a validat mesajul *Finished* de la partenerul său, acesta este gata să transmite datele aplicației prin conexiune.

6.3.6 Protocolul Handshake

Autentificarea serverului

Serverul se autentifică prin trimiterea mesajului *Certificate* care conține:

1. Certificatul X.509 Vers. 3.0 al serverului, care include cheia publică a suitei de cifruri selectate anterior. Mesajul Alert al protocolului va indica o eroare dacă certificatul nu este inclus.

2. Calea de certificare și certificatul autorității de certificare.

Dacă serverul nu are un certificat, în loc de mesajul *Certificate*, acesta transmite mesajul ServerKeyExchange. Evident, în aplicațiile de i-comerț, serverele trebuie să fie autentificate.

Chiar dacă serverul prezintă un certificat, acesta poate fi obligat să transmită mesajul ServerKeyExchange în următoarele condiții:

- **cu Fortezza Key Exchange Algorithm, mesajul ServerKeyExchange este trimis fără cheie semnată, deoarece este furnizat în certificat. Mesajul conține un număr aleator care este folosit ca parte a procesului acordului KEA;**
- **dacă cheia RSA schimbată este una efemeră, pentru semnătură este certificatul X.509, iar mesajul ServerKeyExchange conține cheia publică efemeră a serverului și semnătura sa pentru verificarea integrității cheii;**
- **dacă pentru schimbul de chei este folosită metoda Diffie-Hellman, în mesajul ServerKeyExchange este inclusă o cheie semnată pentru a verifica integritatea acesteia înainte de a stabili cheia de comunicație.**

6.3.6 Protocolul Handshake

Astfel, pentru autentificarea dinamică, serverul trebuie să aibă un certificat de semnătură și transmite două mesaje consecutive: mesajul *Certificate,* iar apoi și ServerKeyExchange. Mesajul *Certificate* include cheia publică pentru semnătura serverului. Clientul verifică certificatul cu cheia publică a autorității de certificare. Mesajul ServerKeyExchange conține parametrii publici ai algoritmului utilizat pentru a schimba cheia secretă pentru cifrarea simetrică. Serverul semnează acești parametri utilizând cheia privată care corespunde cheii publice deja recuperate în primul pas. După primirea celui de-al doilea mesaj, clientul se asigură că parametrii publici ai KEA sunt cei ai serverului, verificând semnătura utilizând cheia publică deja recepționată. Acest mesaj conține rezumatul concatenării variabilelor *client_random* și *server_random*, precum și alți parametri ai serverului.

După autentificare, serverul poate cere clientului să facă același lucru. Ulterior, serverul trimite mesajul CertificateRequest care conține o listă a tipurilor de certificate solicitate, aranjate în ordinea preferințelor serverului pentru autoritățile de certificare. În urma acestor mesaje, serverul trimite mesajul ServerHelloDone care semnifică pentru client că este terminat și că așteaptă un răspuns.

6.3.6 Protocolul Handshake

Schimbul de secrete

Dacă serverul cere clientului să se autentifice utilizând mesajul CertificateRequest, clientul trebuie să răspundă prin includerea certificatului său, dacă are unul, în mesajul Certificate. Dacă clientul nu poate da un certificat, răspunsul va fi *no_certificate*. Aceasta este doar o avertizare și nu o eroare fatală, cu excepția cazului în care serverul cere autentificarea clientului, caz în care conexiunea va fi întreruptă. În caz contrar, autentificarea utilizatorilor va avea loc la nivel de aplicație cu login și parolă. Clientul trimite în continuare mesajul ClientKeyExchange cu conținutul în funcție de algoritmul utilizat pentru schimbul de chei și tipul de certificat în conformitate cu următoarele:

- **dacă RSA este utilizat pentru schimbul de chei, PreMasterSecret este cifrat, fie cu cheia publică a serverului, fie cu o cheie temporară conținută în mesajul ServerKeyExchange;**
- **dacă schimbul de chei este în conformitate cu algoritmul Diffie-Hellman, mesajul ClientKeyExchange conține cheia publică pe care clientul o trimite către server. Fiecare sit Web poate efectua separat calculele necesare care vor genera un secret partajat, a cărui valoare va fi PreMasterSecret;**
- **dacă schimbul de chei se bazează pe algoritmul Fortezza, se calculează un TEK din parametrii pe care serverul le-a trimis în mesajul ServerKeyExhange. Această cheie va fi apoi utilizată pentru a cifra cheia clientului, cheia *client_write_key*, secretul PreMasterSecret și vectorii de inițializare. ♦**

6.3.7 Protocolul ChangeCipherSpec

Protocolul ChangeCipherSpec (CCS) constă dintr-un singur mesaj de 1 octet cu același nume ca și protocolul.

Mesajul ChangeCipherSpec semnalează protocolului Record că cifrarea poate începe cu algoritmii cifroografici deja negociați.

Înainte de acest mesaj, cifrarea a fost sarcina protocolului Handshake.

După primirea acestui mesaj, stratul protocolului Record de pe partea de transmitere va avea metoda sa de cifrare a mesajelor care trebuie trimise.

Pe partea de recepție a entității distante, stratul protocolului Record va trebui să modifice atributele sale de citire pentru a putea să descifreze mesajele recepționate. ♦

6.3.8 Protocolul Record

Protocolul Record intervine numai după transmiterea mesajului ChangeCipherSpec. În timpul stabilirii sesiunii, rolul protocolului Record este de a încapsula datele Handshake și de a le transmite fără modificări spre stratul TCP.

În timpul cifrării datelor, protocolul Record primește date din straturile superioare (Handshake, Alert, CCS, HTTP, FTP, etc.) și le transmite la TCP după efectuarea următoarelor funcții:

1. **Fragmentarea datelor în blocuri de dimensiune maximă de 214 octeți.**
2. **Compresia datelor, o funcție considerată dar neacceptată în versiunea actuală a specificațiilor.**
3. **Generarea rezumatului pentru a asigura serviciul de integritate.**
4. **Cifrarea datelor pentru a asigura serviciul de confidențialitate.**

Secvența de operații constă în calculul MAC și apoi cifrare. Protocolul Record fragmentează mesajul primit de la straturile superioare în sarcini utile și precompune 13 octeți la fiecare încărcătură utilă după cum urmează: un număr de secvență de 8 octeți și un antet de 5 octeți. Acest antet indică tipul mesajului în funcție de subprotocolul său de origine (Handshake, Alert, CCS sau date de aplicație). Identifică versiunea protocolului SSL/TLS utilizat și lungimea blocurilor de date încapsulate. Acest lucru este diferit de lungimea reală a înregistrării, deoarece aceasta include MAC și eventuala umplutură (*padding*). Acest număr de secvență nu este inclus în mesajul Record.

6.3.8 Protocolul Record

Formarea unei înregistrări SSL/TLS.

Sarcinile inverse sunt efectuate pe partea de recepție: descifrarea, verificarea integrității, decompresia și reasamblarea.

Dacă rezumatul calculat nu este identic cu cel primit, protocolul Record invocă protocolul Alert pentru a transmite mesajul de eroare unității de transmisie.

6.3.9 Stabilirea conexiunii

Stabilirea primei conexiuni SSL/TLS este aceeași cu stabilirea unei sesiuni.

Dacă o conexiune SSL/TLS a fost deja stabilită, fluxurile TCP pot tranzita în ambele direcții. Astfel, stabilirea unei noi conexiuni constă în actualizarea parametrilor *client_random* și *server_random* cu mesajele ClientHello și ServerHello, păstrând în același timp algoritmii de cifrare și hash-are deja selectați.

O nouă autentificare este evitată și, spre deosebire de ceea ce se întâmplă pentru stabilirea sesiunii, mesajele ClientHello și ServerHello sunt cifrate.

6.3.9 Stabilirea conexiunii

Mesajul ClientHello conține identificatorul sesiunii în cadrul căreia se stabilește conexiunea. Dacă acest identificator lipsește în tabelele serverului, fie din cauza unei erori, fie pentru că sesiunea la care se referă a expirat, clientul nu este respins, iar serverul pornește un nou Handshake complet pentru a stabili o nouă sesiune SSL/TLS.

Dacă serverul recunoaște identificatorul sesiunii, clientul și serverul confirmă acordul lor trimițând mesajul ChangeCipherSpec din fiecare parte și terminând Handshake abreviat cu mesajul Finished.

Deschiderea unei conexiuni poate fi astfel peste o sesiune deja stabilită sau pe o sesiune suspendată. Atunci când o sesiune este reluată, algoritmul de compresie este reținut, dar trebuie reinitializat, adică istoricul său șters și variabilele de stare ale acestuia resetate.

Specificațiile nu sunt clare cu privire la acțiunea potrivită dacă schimburile introduc un nou set de cifruri. În mod similar, nu este clar dacă într-o sesiune stabilită anterior cu Versiunea 3.0 noua conexiune poate fi de Versiunea 2.0. De fapt, degradarea versiunii poate degrada securitatea din cauza slăbiciunilor Versiunii 2.0, iar unele atacuri pot fi inițiate prin forțarea unei astfel de degradări. ♦

6.3.10 Renegocierea

Renegocierea este o caracteristică opțională prin care un server sau un client poate solicita noi acreditări de client pentru a înlocui cele deja utilizate.

Aceste acreditări noi pot, de exemplu, să creeze o cifrografie mai puternică pentru o anumită tranzacție, să acorde un acces mai restrâns sau să prevină atacurile asupra conexiunilor de lungă durată prin reîmprospătarea parametrilor cifrografici, cum ar fi o cheie de sesiune nouă sau noi parametri cifrografici.

Spre deosebire de strângerea de mână (*handshake*) inițială, schimburile în timpul renegocierii sunt cifrate utilizând parametrii cifrografici existenți. ♦

6.3.11 Protocolul Alert

Protocolul Alert semnalează evenimentele canalului pentru a genera mesaje de alarmă ca răspuns la erori și pentru a indica schimbările în starea unei conexiuni, cum ar fi închiderea unei conexiuni.

Stratul Record cifrează mesajele de alertă utilizând atributele de cifrare existente pentru toate mesajele care provin de la straturile superioare.

În funcție de gravitatea amenințării, alarma poate fi un avertisment simplu sau poate conduce la deconectarea sesiunii.

Un mesaj de avertizare nu justifică o acțiune specifică.

Un mesaj fatal forțează partea de emisie să închidă imediat conexiunea fără a aștepta o confirmare din partea celeilalte părți. Din partea sa, receptorul va închide conexiunea imediat ce va sosi mesajul de alarmă fatal.

6.3.11 Protocolul Alert

Această folosire a mesajelor fatale face, totuși, ca SSL/TLS să fie vulnerabile la atacurile de refuz al serviciului, în cazul în care un intrus reușește să înlocuiască mesajele reglementare cu mesaje neconforme care ar provoca deconectarea sesiunilor.

Mesajele de alertă SSL/TLS nu sunt autentificate.

În consecință, un atacator poate trimite mesaje false de alertă folosind aceleași numere de secvență ca cele utilizate de pachetele de date cifrate, cauzând eliminarea lor pentru fluxul de date.

De asemenea, unele alerte pot scurge informații laterale care ar putea fi utilizate pentru atacuri.

Protocolul Alert poate fi invocat în unul din următoarele moduri:

- prin aplicație, de exemplu, pentru a indica sfârșitul unei conexiuni;
- prin protocolul Handshake dacă întâmpină o problemă;
- prin protocolul Record direct, de exemplu, dacă se pune problema integrității unui mesaj.

6.3.12 Atacuri de refuzare a serviciului

Încărcarea de procesare a unui server în timpul stabilirii unei sesiuni TLS/SSL poate fi substanțială, în special după primirea mesajului ClientKeyExchange. Prin supraîncărcarea serverului cu cereri, un atacator poate supraîncărca resursele de procesare ale unui server SSL Version 3.0. Implementările TLS 1.0 sau 1.1 pot ignora blocurile Record primite dacă tipul lor este necunoscut (RFC 2246, RFC 4346). Folosind această proprietate, este posibil de înlăturat un atac de refuz al serviciului în timpul înființării unei sesiuni TLS. Protecția constă în trimiterea unui puzzle către client atunci când încărcarea de procesare pe server depășește un anumit prag. În consecință, clienții sunt obligați să reducă transmisiile, permițând serverului un anumit interval de timp pentru recuperare.

Puzzle-ul: serverul selectează un număr aleatoriu x, calculează hash-ul $h(x)$ și apoi formează un nou număr x' prin zeroarea a n biți de ordin inferior ai lui x. Tripletul $\{n, x', h(x)\}$ formează puzzle-ul trimis după mesajul Certificate. Clientul trebuie să descopere valoarea lui x încercând toate valorile pentru biții n absenți și apoi comparând hash-ul pentru fiecare combinație cu hash-ul primit $h(x)$. Numărul mediu de hash-uri pe care clientul le calculează este 2^{n-1}, în timp ce serverul calculează numai două: primul este parte din puzzle și cel de-al doilea verifică dacă hash-ul valorii pe care clientul o descoperă corespunde hash-ului trimis.

6.4 Performanța SSL/TLS

Principala sarcină de procesare în SSL/TLS provine din cifrografie, în special, în timpul stabilirii sesiunii. Algoritmul temporal Diffie-Hellman pentru stabilirea cheilor este laborios și este folosit numai dacă este necesar secretul perfect înainte; în acest caz, pentru fiecare strângere de mână se folosește o cheie nouă.

Performanța unui server de i-comerț este măsurată prin trei criterii:

- numărul sesiunilor cifrate simultan (adică numărul de tranzacții pe secundă);
- timpul de răspuns la solicitări;
- capacitatea de transfer disponibilă pentru fiecare dintre aceste sesiuni.

Protocoalele SSL/TLS utilizează resurse semnificative de calcul din cauza folosirii algoritmilor cu chei publice. Încărcarea totală pe server este substanțială, deoarece va trebui să se ocupe simultan cu multiple solicitări de stabilire a sesiunii. Sarcina crește odată cu numărul de sesiuni simultane ceea ce conduce la saturarea serverului și, respectiv, scăderea substanțială a performanței.

În general, pentru un anumit client, încărcarea pe server poate fi de două ori sau de trei ori cea a clientului (dar dacă clientul are un certificat de autentificare, atunci sarcina clientului va fi de patru ori mai mare, iar cea a serverului va crește cu aproximativ 20 %). Mai exact, utilizarea algoritmului de semnătură DSA și a schimbului de chei cu algoritmul Diffie-Hellman efemer măresc sarcina pe server de 5-7 ori.

6.4 Performanța SSL/TLS

Printre măsurile utilizate pentru reducerea sarcinii se numără:

- reluarea sesiunii pentru a evita un nou Handshake;
- utilizarea acceleratoarelor pentru a descărca procesările cifrografice.

Acești acceleratori provin fie sub formă de mașini separate, fie ca plăci care se introduc în server. Mașinile separate funcționează ca proxy-uri intre server si clienți si pot fi organizate pentru a creste puterea de calcul prin distribuirea sarcinii pe mai multe mașini. Un aranjament de cluster are un avantaj suplimentar: în caz de defecțiune, calculul poate fi schimbat automat de la mașina căzută la una în așteptare.

Comunicările între mașini adaugă o încărcare suplimentară de procesare care poate neutraliza câștigul în viteză dacă numărul de mașini pe cluster depășește trei-patru. În plus, deoarece SSL/TLS sunt protocoale punct-la-punct, proxy-ul trebuie să înceteze conexiunea, deci să descifreze mesajele de la client pentru a extrage informațiile care urmează să fie distribuite altor mașini (sau unei mașini oglindă) astfel încât să se asigure recuperarea în caz de eșec, înainte de a le trimite la server. În mod evident, devine imperativ să se protejeze toate aceste mașini împotriva potențialelor intruziuni fizice sau logice; în caz contrar, întregul sistem de securitate va fi afectat.

6.4 Performanța SSL/TLS

Alegerea plăcii de accelerație este dificilă: câștigurile de performanță estimate variază în funcție de placa de bord, configurația serverului și profilul de trafic. Îmbunătățirea depinde de utilizarea procesoarelor centrale, ceea ce afectează răspunsul lor la trafic de vârf neașteptat. Alți factori includ rata la care performanța lor se saturează și toleranța lor la defecțiuni. Ca urmare, chiar și atunci când timpul mediu de răspuns se îmbunătățește, timpul de răspuns poate fi, în unele cazuri, semnificativ. De asemenea, prezența unor mașini suplimentare între client și server pentru a efectua operațiile cifrografice adaugă noi riscuri de securitate. Prin urmare, utilizarea plăcilor de accelerație necesită o bună înțelegere a contextului serviciilor; altfel, ele pot fi inutile dacă nu sunt dăunătoare.

Alte tehnici de îmbunătățire a performanței au rolul de a accelera descifrarea. De exemplu, cu RSA reechilibrată, parametrii algoritmului *e*, *d* sunt aleși astfel încât, în timp ce *d* este mare, *d* mod (*p* - 1) și *d* mod (*q* - 1) sunt numere mici. O altă abordare a vitezei de descifrare RSA este utilizarea tehnicilor de dozare.♦

6.5 Capcane de implementare

Implementările SSL/TLS cuprind două module: primul pentru funcțiile cifrografice și cel de-al doilea pentru biblioteca protocolului. Această arhitectură permite modificarea proprietăților serverului compozit SSL/TLS în conformitate cu restricțiile legale sau opțiunile tehnice.

Multe slăbiciuni au fost descoperite în SSL vers. 2.0 (RFC 6176):

- autentificarea mesajelor utilizează MD5;
- mesajele de tip "Handshake" nu sunt protejate. Acest lucru permite un atac de tip omul-în-mijloc pentru a păcăli clientul în a alege o suită de cifruri mai slabă decât ar alege în mod normal;
- integritatea mesajelor și cifrarea mesajelor utilizează aceeași cheie – este o problemă dacă clientul și serverul negociază un algoritm de cifrare slab;
- sesiunile pot fi terminate cu ușurință. Un om-în--mijloc poate introduce cu ușurință un TCP FIN pentru a închide sesiunea, iar partenerul nu poate determina dacă a fost un sfârșit legitim al sesiunii.

De aceea, RFC 6176 (2011) interzice utilizarea SSL vers. 2.0 în noile implementări. În timp ce SSL 3.0 este mai sigur, acesta nu a fost aprobat la nivel federal din SUA, deoarece se bazează parțial pe algoritmi cifrografici neaprobați (RSA pentru stabilirea cheilor sau RC4 pentru confidențialitate).

TLS este aprobat pentru protecția informațiilor federale din S.U.A.

6.5 Capcane de implementare

RFC 5246 enumeră câteva capcane, printre care:

- **fragmentarea mesajelor Handshake (ClientHello, Certificate și CertificateRequest) de peste 214 octeți în mai multe înregistrări TLS;**
- **la cifrarea PreMasterSecret cu RSA, versiunea mesajului ClientKeyExchange trebuie să fie aceeași ca în mesajul ClientHello și nu cea negociată. Acest lucru este pentru a preveni atacurile *rollback*;**
- **de a nu trimite un mesaj de certificat vid, dacă clientul nu are un certificat adecvat ca răspuns la mesajul CertificateRequest al serverului.**
- **când Handshake întâlnește o eroare, strângerea de mână trebuie continuată;**
- **TLS 1.2 permite clientului să solicite funcții suplimentare de la server. Regulile de manipulare a acestor extensii sunt complicate și ar putea conduce la probleme de interoperabilitate. De exemplu, unele servere vor refuza conexiunea dacă sunt incluse extensii TLS în ClientHello;**
- **verificarea corectă prin implementarea de către client a parametrilor pe care serverul i-a trimis pentru schimbul de chei Diffie-Hellman;**
- **utilizarea vectorilor de inițializare imprevizibili în cifrurile mod CBC sau a unui generator de numere aleatoare puternic;**
- **unele implementări de server închid incorect conexiunea atunci când clientul oferă o versiune mai nouă decât TLS 1.0. Versiunile anterioare ale TLS nu au fost pe deplin clare cu privire la ce să se întreprindă în asemenea situații.**

6.7 Generalități și arhitectura TLS fără fir

Securitatea stratului Transport fără fir (WTLS) este o specificație a forumului WAP (Wireless Application Protocol) pentru securizarea tranzacțiilor în rețelele mobile [*Wireless Application Protocol Forum*, 2001].

Prima versiune a WTLS a fost o revizuire a TLS pentru a răspunde constrângerilor comunicațiilor de date prin rețele GSM (*Global System for Mobile Communication*) folosind serviciul de mesaje scurte (SMS) pentru rate de biți de 9,6 Kbps și serviciul de pachete radio generale (GPRS) pentru rate de biți între 28 și 56 Kbps. Au fost luate în considerare și tehnologiile celulare de generația a treia (3G) sau UMTS (*Universal Mobile Telecommunication System*).

Protocolul a fost conceput ținând cont de limitarea telefoanelor în ceea ce privește memoria, puterea de calcul, durata de viață a bateriei, ecranele și tastatura.

Au existat mai multe revizuiri, WTLS Vers. 6 fiind aprobată în aprilie 2001. Cu toate acestea, având în vedere complexitatea operațională a WAP 1.x, în special cea a securității capăt-la-capăt WTLS în tandem cu TLS, WAP 2.0 a fost publicat în august 2001 bazat pe stiva TCP/IP. Acest lucru a necesitat unele extensii la TLS pentru a permite comunicarea mobilă.

6.7 Generalități și arhitectura TLS fără fir

Prima versiune a WTLS face parte din stiva WAP (*Application Application Protocol*) din aprilie 1998. Această stivă a fost proiectată pentru a satisface următoarele cerințe:

- **menținerea operării peste UDP chiar și cu pierderi de pachete;**
- **operare chiar și atunci când întârzierile transmisiei sunt importante;**
- **luarea în considerare a puterii de calcul reduse și a capacității de stocare relativ mici a terminalelor mobile.**

6.7 Generalități și arhitectura TLS fără fir

În stratul superior, limbajul Wireless Markup Language (WML), care este mai adaptat decât cel HTML la constrângerile limitelor de bandă și capabilităților terminale, este folosit pentru a descrie paginile de acces. În plus, a fost nevoie de un microexplorator (explorator WAP), iar comunicarea cu poarta WAP a fost codificată binar pentru eficiență.

Utilizarea WTLS în tranzacțiile bancare necesită o infrastructură cu chei publice sub controlul instituției financiare a utilizatorului.

Dispozitivele mobile tipice au un Modul de identificare a abonatului (*Subscriber Identity Module* – SIM), o cartelă de circuite integrate care stochează identitatea internațională a abonatului mobil (*International Mobile Subscriber Identity* – IMSI) și cheile folosite pentru identificarea și autentificarea abonaților în rețea. Este necesar un PIN pentru a accesa conținutul stocat pe cartela SIM.

Terminalele utilizate pentru serviciile WAP au un modul de identificare fără fir (*Wireless Identification Module* – WIM) pentru a ține cheile și acreditările utilizatorului necesare pentru tranzacții financiare.

Relația dintre SIM și WIM poate fi în una din următoarele configurații:

- SIM și WIM se găsesc pe aceeași placă de circuite integrate;
- receptorul este un telefon cu slot dual cu două cititoare de cartele separate, una pentru cartela SIM și cealaltă pentru cartela WIM;
- cititorul WIM este extern și comunică cu receptorul utilizând un protocol fără fir, cum ar fi Bluetooth.♦

6.8 De la TLS la WTLS

Principalele modificări introduse de WTLS în SSL/TLS se referă la următoarele elemente:

- **formatul identificatorilor și al certificatelor;**
- **algoritmii cifrografici;**
- **conținutul anumitor mesaje Handshake;**
- **protocolul de schimb în timpul Handshake;**
- **calculul secretelor;**
- **mărimea parametrilor;**
- **mesajele de avertizare;**
- **rolul protocolului Record.**

Toate aceste modificări fac WTLS total incompatibil cu TLS/SSL, chiar dacă WTLS este compus tot din patru protocoale: Handshake, Record, ChangeCipherSpec (CCS) și Alert.