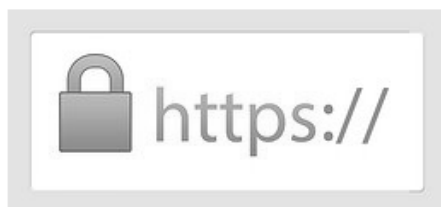


Ion BICA

Mihai TOGAN

**PROTOCOALE DE SECURITATE
PENTRU REȚELE DE CALCULATOARE**



Editura Univers Științific
București, 2015

[CASETA CIP]

Cuprins

Introducere.....	5
1. Rețele TCP/IP	7
1.1. Scurt istoric	7
1.2. Arhitectura TCP/IP	8
1.3. Nivelul Legătură de Date	9
1.4. Nivelul Internet	11
1.5. Nivelul Transport	17
1.6. Nivelul Aplicație.....	24
2. Tehnologii criptografice.....	25
2.1. Sisteme criptografice	25
2.2. Sisteme criptografice cu chei secrete	26
2.3. Sisteme criptografice cu chei publice.....	38
2.4. Funcții hash criptografice	42
2.5. Semnături digitale.....	46
2.6. Criptografia bazată pe curbe eliptice	50
2.7. Recomandări privind lungimile de chei.....	52
2.8. Infrastructuri de chei publice.....	52
3. Metode și protocoale de autentificare	66
3.1. Autentificarea folosind parole	67

3.2.	Autentificarea folosind generatoare de parole.....	77
3.3.	Autentificarea folosind certificate digitale	79
3.4.	Autentificarea folosind caracteristici biometrice.....	83
4.	Protocoale de securitate la nivel rețea	89
4.1.	Protocolul Authentication Header (AH).....	89
4.2.	Protocolul Encapsulating Security Payload (ESP)	91
4.3.	Moduri de operare IPSec	92
4.4.	Protocolul Internet Key Exchange (IKE)	95
4.5.	Rețele private virtuale (VPN)	99
5.	Protocoale de securitate la nivel transport	104
5.1.	Secure Sockets Layer (SSL).....	104
5.2.	Transport Layer Security (TLS)	116
5.3.	Aplicații ale SSL / TLS.....	119
6.	Protocoale de securitate la nivel aplicație.....	122
6.1.	Pretty Good Privacy (PGP)	122
6.2.	Secure/Multipurpose Internet Mail Extensions (S/MIME)	135
	Anexe.....	151
	Bibliografie.....	161

Introducere

Internetul a schimbat radical societatea actuală, deschizând noi canale de comunicație între oameni, cu impact major asupra modului în care trăim, muncim și interacționăm. Prin intermediul serviciilor Internet, o persoană poate să transmită mesaje altor persoane (e-mail), să urmeze un curs (e-learning), să își facă cumpărăturile de la calculator (e-commerce) sau să desfășoare tranzacții comerciale cu partenerii de afaceri (e-business).

Pe măsură ce Internetul devine o parte omniprezentă a vieții de zi cu zi, nevoia de securitate devine din ce în ce mai stringentă. Orice persoană sau organizație angrenată în activități online trebuie să evalueze și să gestioneze în mod corespunzător riscurile de securitate asociate cu această activitate. Rolul tehnologiilor criptografice este de a reduce aceste riscuri prin securizarea tranzacțiilor electronice. Această carte oferă detalii cu privire la tehnologiile criptografice și protocoalele de securitate disponibile în momentul de față pentru asigurarea confidențialității, integrității, autenticității și non-repudierii datelor transmise între entități prin intermediul rețelelor de calculatoare.

Lucrarea este structurată pe 6 capitole, abordarea fiind una progresivă. Capitolul 1 este dedicat rețelelor TCP/IP. Capitolul debutează cu un scurt istoric al Internetului urmat de prezentarea modelului arhitectural TCP/IP. În continuare, sunt prezentate principalele protocoale de comunicație de nivel legătură de date, de nivel rețea (IP și ICMP), de nivel transport (TCP și UDP) și de nivel aplicație, relevante pentru înțelegerea protocoalelor de securitate abordate în carte.

Capitolul 2 face o trecere în revistă a tehnologiilor criptografice moderne, cu accent pe algoritmi și mecanismele de securitate folosite în cadrul protocoalelor de securitate. Prima parte a acestui capitol este dedicată sistemelor criptografice cu chei secrete (simetrice), sistemelor criptografice cu chei publice (asimetrice), funcțiilor hash și semnăturilor digitale. În cea de-a doua parte sunt tratate aspectele legate de managementul cheilor criptografice și este prezentat rolul infrastructurilor de chei publice.

Capitolul 3 prezintă metodele și protocoalele de autentificare. Este explicat rolul autentificării în cadrul protocoalelor de securitate și metodele care pot fi folosite pentru autentificarea unei entități. În continuare, sunt descrise protocoalele de autentificare pe bază de parole (PAP, CHAP, NTLM și Kerberos), modul de funcționare al soluțiilor de autentificare bazate pe generatoare de parole de unică folosință, autentificarea pe bază de certificate

digitale împreună cu protocoalele aferente și sistemele biometrice folosite pentru recunoașterea persoanelor. În continuarea lucrării, sunt prezentate cele mai populare protocoale de securitate, grupate pe nivele, conform modelului arhitectural TCP/IP.

Capitolul 4 descrie în detaliu protocolul IPSec. IPSec este un protocol de securitate de nivel rețea care adaugă facilități de securitate la IPv4. În cadrul capitolului este prezentat protocolul Authentication Header (AH) care poate fi folosit pentru asigurarea integrității și autentificarea originii pachetelor IP. Urmează apoi protocolul Encapsulating Security Payload (ESP) care permite asigurarea confidențialității și integrității datelor din cadrul pachetelor IP. Ultimul protocol al IPSec este Internet Key Exchange (IKE) care asigură negocierea algoritmilor și parametrilor criptografici, autentificarea entităților și stabilirea de chei de sesiune între entitățile comunicante. În finalul capitolului este prezentat conceptul de rețea privată virtuală care poate fi implementat prin intermediul IPSec.

Capitolul 5 prezintă protocoalele de securitate de nivel transport: SSL și TLS. Secure Sockets Layer (SSL) și varianta îmbunătățită a acestuia, Transport Layer Security (TLS), permit stabilirea unui canal de comunicație sigur între două entități de nivel aplicație (procesele client și server). Acest capitol prezintă detalii cu privire la arhitectura și modul de funcționare al celor două protocoale precum și o serie de aplicații ale acestora.

Capitolul 6 este dedicat protocoalelor de securitate de nivel aplicație folosite pentru securizarea poștei electronice PGP și S/MIME. Pretty Good Privacy (PGP) este un pachet software, care furnizează servicii de confidențialitate și autenticitate pentru mesajele transmise prin e-mail sau fișierele stocate local. Secure/Multipurpose Internet Mail Extensions (S/MIME) este un protocol ce permite implementarea de servicii de securitate prin încapsularea MIME a mesajelor de e-mail semnate sau criptate.

Cartea se adresează în primul rând specialiștilor care pot găsi aici idei și soluții pentru implementarea de aplicații sau rețele de comunicații sigure. Pe lângă prezentarea teoretică a protocoalelor de securitate, cartea scoate în evidență avantajele și dezavantajele fiecărei tehnologii și prezintă o serie de considerente privind folosirea în practică a acestor protocoale. Nu în ultimul rând, cartea este utilă studenților pentru înțelegerea conceptelor cu privire la algoritmi criptografici, mecanismele, tehnologiile și serviciile care stau la baza protocoalelor de securitate moderne, folosite în momentul de față în cadrul rețelelor de calculatoare.

București, 2015

Autorii

1. Rețele TCP/IP

O *rețea de calculatoare* reprezintă un ansamblu de *calculatoare* interconectate prin intermediul unor *medii de comunicație* cu scopul de a permite utilizarea în comun, de către mai mulți utilizatori, a *resurselor hardware, software* și *informaționale* asociate calculatoarelor din rețea.

TCP/IP reprezintă în acest moment standardul de comunicație în Internet. Deși numele de TCP/IP este dat de principalele două protocoale: protocolul TCP și protocolul IP, modelul referă de fapt o suită întreagă de protocoale ce stau la baza funcționării Internetului.

Pentru o bună înțelegere a protocoalelor de securitate, este necesară cunoașterea modului de funcționare al rețelelor TCP/IP. Acest capitol prezintă în detaliu cele mai importante protocoale din suita TCP/IP, relevante pentru înțelegerea protocoalelor de securitate abordate în carte.

1.1. Scurt istoric

În contextul anilor 1960, când războiul rece era la apogeu, Departamentul Apărării al Statelor Unite ale Americii (DoD – Department of Defense) era interesat de dezvoltarea unei rețele de comunicații care să continue să funcționeze chiar și în cazul unui atac nuclear. Pentru asta, DoD a apelat la agenția sa de cercetare – ARPA (Advance Research Projects Agency) – pentru a inventa o tehnologie care să permită livrarea datelor la destinație chiar și atunci când o parte din rețeaua de comunicație a fost scoasă din funcțiune ca urmare a unui atac nuclear. Din această dorință a fost construită prima rețea de calculatoare cu comutare de pachete, ARPANET.

Fiabilitatea rețelei de calculatoare ARPANET este dată de tehnologia care stă la baza proiectării ei: *comutarea de pachete*. Aceasta presupune divizarea datelor ce trebuie transmise în rețea, într-o serie de blocuri numite pachete. Fiecare dintre aceste pachete este apoi rutat în mod independent, de la un calculator la altul prin rețea până ajunge la destinație.

ARPANET a devenit operațională în Decembrie 1969. Inițial, aceasta cuprindea 4 noduri: UCLA (University of California at Los Angeles), UCSB (University of California at Santa Barbara), SRI (Stanford Research Institute) și Universitatea din Utah. În următorii ani, rețeaua a început să se dezvolte: în

Iulie 1970, numărul nodurilor a crescut la 8; în Martie 1971, numărul a ajuns la 16, iar până în Septembrie 1972 acesta era de 34.

Această rețea experimentală a funcționat foarte bine în primele etape ale dezvoltării sale, când existau puține noduri interconectate. Însă, pe măsură ce numărul nodurilor creștea, funcționalitatea rețelei a fost afectată de o serie de blocări. Creșterea complexității, prin adăugarea rețelelor satelit și radio, a determinat căutarea unui nou protocol care să asigure serviciile de comunicație în rețea. Acesta trebuia să înlocuiască protocolul NCP (Network Control Protocol), utilizat până la momentul respectiv.

Ca urmare a acestui efort, în 1974 a apărut suita de protocoale TCP/IP. Faptul că TCP/IP nu necesită resurse de rețea importante și că este ușor de implementat, a făcut ca acesta să devină în scurt timp foarte popular. Astfel, în 1983 protocolul a fost integrat în versiunea 4.2 a BSD (Berkley Software Distribution) UNIX, sistemul de operare cel mai popular la acel moment. În același an, DoD a adoptat aceasta suită de protocoale ca și standard militar (MIL STD). De asemenea, TCP/IP a devenit standard și pentru ARPANET, precursorul Internetului.

Până la sfârșitul anilor 80, mare parte din lumea academică a fost conectată într-o rețea cu acoperire globală, cunoscută sub numele de Internet. În următorul deceniu a urmat o dezvoltare explozivă a acestei rețele, prin deschiderea ei către publicul larg. Pasul decisiv l-a constituit crearea unui sistem de organizare a informațiilor disponibile în rețea, WWW (World Wide Web) și a unui sistem de căutare a acestor informații.

În prezent, TCP/IP-ul continuă să fie standardul de comunicație pentru Internet. Faptul că acesta a fost proiectat să funcționeze indiferent de mediul de comunicație a permis integrarea sa pe scară largă în rețele și sisteme de operare de tipuri diferite.

1.2. Arhitectura TCP/IP

O arhitectură de rețea definește modul de interconectare a componentelor rețelei. Pentru a reduce complexitatea proiectării, majoritatea arhitecturilor folosesc o structură stratificată, pe nivele. Fiecare din aceste nivele oferă un anumit set de servicii de comunicație și definește protocoale aferente setului de servicii. Între două nivele adiacente există o interfață de acces la servicii. Aceasta stabilește operațiile și serviciile oferite de nivelul inferior pentru nivelul superior.

Un protocol de rețea este un set de reguli ce guvernează modul de comunicare între calculatoarele din rețea. În esență, un protocol definește sintaxa și semantica schimbului de date dintre calculatoare.

Modelul arhitectural stratificat oferă o serie de avantaje: fiecare nivel este independent de celelalte, astfel încât schimbările produse la un anumit nivel nu influențează funcționalitatea celorlalte nivele cât timp interfața rămâne nemodificată. De asemenea, ierarhizarea permite extinderea serviciilor pentru un anumit nivel.

Modelul TCP/IP referă o întreagă suită de protocoale, toate proiectate pentru a asigura transferul informațiilor într-o rețea de calculatoare și este ierarhizat sub forma unei stive de protocoale cu patru nivele. Fiecare nivel are anumite funcții și constă dintr-o combinație de protocoale diferite, după cum este ilustrat în Figura 1-1.

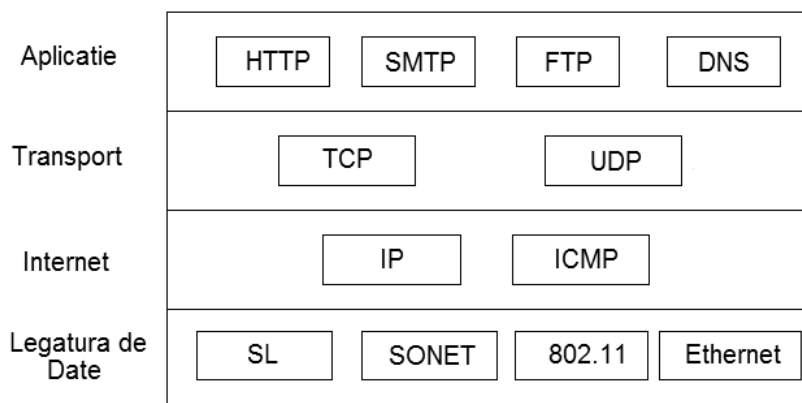


Figura 1-1. Modelul arhitectural TCP/IP

1.3. Nivelul Legătură de Date

Nivelul legătură de date reprezintă primul nivel din stiva TCP/IP. El mai poartă și denumirea de nivel de acces la rețea. Rolul acestuia este de a permite realizarea unei legături de comunicație stabilă între calculatoare prin intermediul mediului fizic de comunicație. Principalele funcții ale nivelului legătură de date sunt: încapsularea datelor, identificarea stațiilor între care se desfășoară comunicația și detecția erorilor de transmisie.

Cele mai populare tehnologii și protocoale de nivel legătură de date sunt Ethernet (IEEE 802.3), Token Ring (IEEE 802.5), Wireless (IEEE 802.11), Asynchronous Transfer Mode (ATM), Serial Link Interface Protocol (SLIP), Point-To-Point Protocol (PPP).

În momentul actual, la nivelul rețelelor locale de calculatoare se folosește tehnologia Ethernet. Ethernet-ul este standardizat de IEEE în seria de standarde IEEE 802.3. Aceste standarde permit transmisia datelor prin mai multe medii fizice, cum ar fi: cabluri coaxiale, cabluri torsadate sau fibra optică și asigură rate de transfer de 10,100 Mbps și 1,10 Gbps.

Transmisia datelor într-o rețea Ethernet se realizează prin intermediul *cadrelor (frame)*. Stațiile implicate în realizarea transmisiei datelor se numesc *sursă* și *destinație*. Un cadru conține atât datele de transmis, cât și doi identificatori unici: unul pentru stația sursă și unul pentru stația destinație. Într-o rețea Ethernet, identificatorul unei stații este adresa hardware a plăcii de rețea. Această adresă, cunoscută și ca adresă *MAC (Medium Access Control)*, constă dintr-o succesiune de 48 de biți (6 octeți). Dintre aceștia primii 24 de biți identifică în mod unic producătorul plăcii de rețea iar restul de 24 de biți reprezintă seria plăcii de rețea.

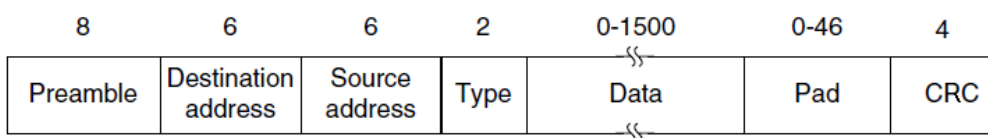


Figura 1-2. Formatul unui cadru Ethernet

Rețelele Ethernet folosesc transmisii de tip broadcast și au la bază o arhitectură de tip magistrală. În momentul transmisiei unui cadru în rețea, fiecare stație va compara adresa sa MAC cu adresa MAC destinație a cadrului care circulă prin rețea. Dacă cele două adrese sunt identice, atunci cadrul este pentru stația respectivă și trebuie procesat.

În cazul în care există mai multe stații care transmit în același timp, acestea se bruiază reciproc. Pentru a evita această situație, rețelele Ethernet folosesc protocolul *CSMA/CD (Carrier Sense Multiple Access with Collision Detection)* ca metodă de acces la mediul de comunicație.

Identificarea stațiilor dintr-o rețea TCP/IP se realizează nu prin intermediul adreselor MAC ci folosind adrese logice, adresele IP. Pentru a putea transmite un cadru Ethernet, stația sursă are nevoie de adresa MAC a stației destinație. Întrucât stația sursă știe doar adresa IP a stației destinație, trebuie să existe o metodă prin care aceasta să poată afla adresa MAC destinație.

Protocolul ARP (Address Resolution Protocol) este folosit pentru a determina adresa MAC a unei stații pornind de la adresa IP a acesteia. Pentru asta, fiecare stație menține o tabelă numită tabela ARP. Aceasta conține o corespondență între adresele IP și adresele MAC ale stațiilor din rețeaua locală. Atunci când o

stație are de transmis un pachet IP, compară adresa IP destinație cu intrările din tabela ARP. În cazul în care găsește o intrare cu această informație, va folosi adresa MAC din intrarea respectivă pentru transmisia cadrului Ethernet la destinație. În cazul în care nu există o astfel de informație, va face un broadcast ARP în rețea (adresa MAC pentru broadcast este FF:FF:FF:FF:FF:FF), pentru a afla care este adresa MAC corespunzătoare adresei IP destinație. Toate stațiile vor primi mesajul de broadcast iar stația care își recunoaște adresa IP va răspunde printr-un mesaj ce conține adresa sa MAC. Pe baza răspunsului, stația care a lansat cererea își va actualiza tabela ARP. Protocolul ARP este definit în RFC 826.

RARP (Reverse Address Resolution Protocol) este opusul protocolului ARP, rolul său fiind de a determina adresa IP pentru o adresă MAC dată. RARP este folosit în procesul de bootare din rețea al stațiilor.

1.4. Nivelul Internet

Nivelul Internet, denumit uneori și nivelul rețea, este responsabil de rutarea pachetelor de la sursă la destinație prin rețeaua de comunicație. Protocoale de nivel rețea discutate în acest capitol sunt: protocolul IP (Internet Protocol), protocolul ICMP (Internet Control Message Protocol) și protocolul IGMP (Internet Group Management Protocol).

1.4.1. Protocolul IP

Protocolul IP (Internet Protocol) reprezintă unul din cele mai importante protocoale ale stivei TCP/IP. IP este un protocol neorientat conexiune. Rutarea pachetelor de face în mod independent, în funcție de adresa IP destinație. Având în vedere aceste aspecte, IP nu garantează livrarea datelor și în plus, pachetele pot ajunge la destinație într-o ordine diferită de cea în care au fost transmise.

Versiunea curentă de protocol este IPv4, descrisă de către IETF (Internet Engineering Task Force) în publicația RFC 791. Treptat, treptat, se urmărește trecerea la IPv6, care elimină limitările specifice IPv4 și introduce o serie de funcționalități noi.

Formatul datagramelor IPv4

O datagramă IPv4 este formată din antet (header) și datele ce trebuie livrate la destinație (payload). Antetul are o parte de lungime fixă, 20 de octeți, și o parte de lungime variabilă (vezi Figura 1-3).

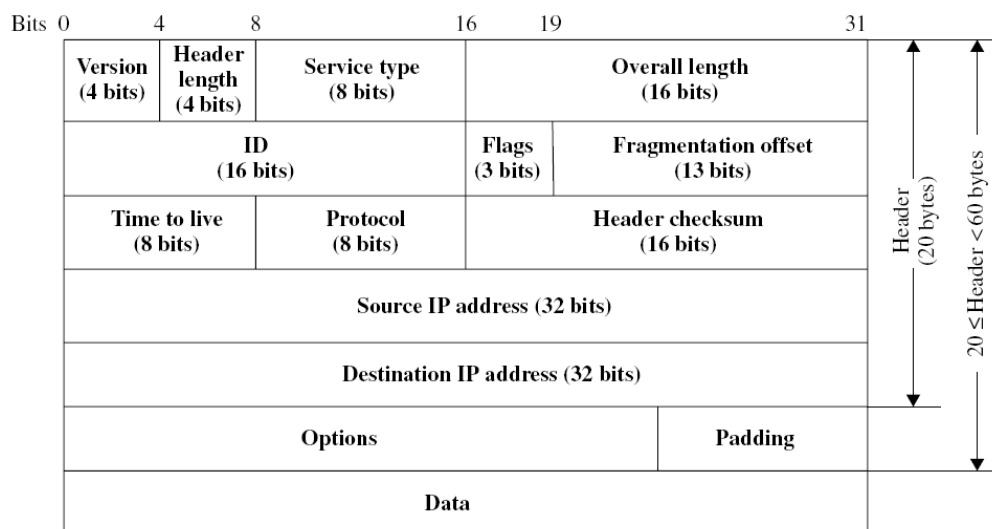


Figura 1-3. Formatul datagramelor IPv4

Câmpul *Versiune* (*Version*) specifică versiunea de format.

Câmpul *Lungimea antetului* (*Header length*) reprezintă lungimea antetului în cuvinte de 32 biți. Valoarea minimă este de 5 cuvinte (20 octeți), iar valoarea maximă este de 15 cuvinte (60 de octeți).

Câmpul *Tipul serviciului* (*Service type*) indică cerințele privind modul de transmisie al datagramelor în rețea (minimizare întârziere, maximizare rată de transfer, fiabilitatea ridicată, cost minim, etc). RFC 2474 redefinește semnificația acestui câmp în *Servicii diferențiate* (*Differentiated services*) pentru a putea trata în mod adecvat categorii diferite de trafic (voce, video, date).

Lungimea totală (*Overall length*) reprezintă lungimea datagramelor în octeți. Valoarea maximă este de 65.535 octeți. Acest câmp, împreună cu lungimea antetului, determină dimensiunea zonei de date dintr-o datagramă IP.

În funcție de *MTU-ul* (*Maximum Transmission Unit*) subrețelei în care este rutată o datagramă, aceasta poate fi sau nu fragmentată. Toate fragmentele care provin din aceeași datagramă vor avea câmpul *ID* identic.

Din cei trei biți a câmpului *Flags*, numai doi sunt utilizați: *DF* (*Don't Fragment*) și *MF* (*More Fragments*). Atunci când *DF* este setat, datagrama nu va putea fi fragmentată pe timpul transmisiei. Cel de-al doilea flag, *MF* este utilizat pentru a semnaliza ultimul fragment al unei datagramă. Pentru toate fragmentele, cu excepția ultimului, bitul *MF* este 1.

Offset-ul fragmentului (Fragmentation offset) indică poziția fragmentului în cadrul datagramelor, calculată în blocuri de 64 biți. Primul fragment dintr-o datagramă are offset-ul egal cu zero. Pe baza câmpurilor *ID*, *MF* și *Offset*, stația destinație va ști cum să reasambleze datagrama originală din fragmentele recepționate.

Câmpul *TTL (Time to Live)* este folosit pentru a limita durata de viață a datagramelor în rețea. TTL-ul previne rutarea la infinit a datagramelor în rețea datorită buclelor și are o valoare predefinită, care este decrementată de fiecare router de pe traseu, atunci când datagrama trece dintr-o subrețea în alta. Dacă TTL-ul ajunge la 0 atunci router-ul aruncă datagrama și trimite un mesaj ICMP de eroare stației sursă indicând faptul că TTL-ul a expirat.

Câmpul *Protocol (Protocol)* indică protocolul de nivel superior către care trebuie livrată datagrama IP pe stația destinație. Semnificația câmpului Protocol pentru diferite valori este:

Dec	Hex	Protocol
1	0x01	Internet Control Message Protocol
2	0x02	Internet Group Management Protocol
4	0x04	IP in IP
6	0x06	Transmission Control Protocol
8	0x08	Exterior Gateway Protocol
9	0x09	Interior Gateway Protocol
17	0x11	User Datagram Protocol
27	0x1B	Reliable Datagram Protocol
50	0x32	Encapsulating Security Payload
51	0x33	Authentication Header

Integritatea antetului este verificată prin intermediul *Sumei de control a antetului (Header checksum)*.

Adresa IP sursă (Source IP address) și *Adresa IP destinație (Destination IP address)* reprezintă adresele IPv4 ale stației sursă și respectiv destinație.

Câmpul *Opțiuni (Options)* conține informații suplimentare referitoare la datagramă. Acest câmp este utilizat destul de rar și este ignorat de majoritatea routerelor.

Dimensiunea antetului IP trebuie să fie multiplu de 32 biți. Pentru completare se folosește câmpul *Padding*.

Adresarea IPv4

Fiecare stație sau router conectat în rețea are asociată o adresă IP unică. O stație conectată la mai multe rețele, va avea asignată câte o adresă IP pentru fiecare interfață de rețea.

O adresă IPv4 reprezintă un număr pe 32 de biți și este formată dintr-o parte ce identifică rețeaua în care se află stația și o parte ce identifică stația în rețea. Pentru reprezentarea unei adrese IPv4 se utilizează notația zecimală cu punct. Fiecare octet din adresă este codificat printr-o valoare cuprinsă între 0 și 255. De exemplu adresa 11000001 11100111 00010101 00001010 va fi reprezentată prin 193.231.21.10.

Clase de adrese IP

Inițial, toate adresele IP alocate erau publice. Soluția aleasă pentru atribuirea IP-urilor a fost împărțirea spațiului de adrese IP în clase. Conform acestei alocări, toate adresele care aparțin unei anumite clase respectă formatul descris de clasa respectivă. În cazul adreselor IPv4 există cinci clase: A, B, C, D și E. Dintre acestea, D și E reprezintă adrese de tip special. Clasa D reprezintă clasa adreselor de multicast, folosite pentru a identifica grupuri de stații în Internet. Clasa E reprezintă o clasă specială de adrese, fiind rezervată pentru dezvoltări ulterioare. Figura 1-4 prezintă formatul pentru fiecare clasă de adrese IP.

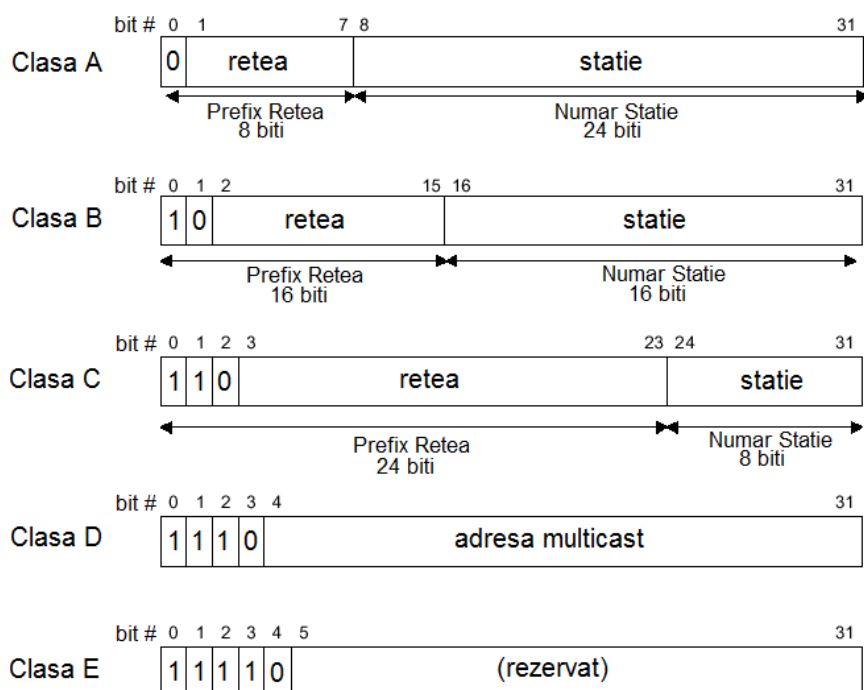


Figura 1-4. Clase de adrese IP

Adresele IP cu toți biții având valoarea 0 sau 1 în partea de rețea sau stație sunt adrese rezervate și au o semnificație aparte. Adresa 0.0.0.0 este folosită pentru a identifica rețeaua curentă și este validă doar dacă este folosită ca adresă sursă. O adresă în care toți biții de stație sunt 0 este folosită pentru a identifica o rețea (de exemplu, 193.231.21.0). O adresă de broadcast este o adresă în care toți biții de stație sunt 1 (de exemplu, 193.231.21.255 se referă la toate calculatoarele din rețeaua 193.231.21.0). Adresa de broadcast 255.255.255.255 este folosită pentru a transmite către toate calculatoarele din rețeaua locală. Adresele 127.0.0.0-127.255.255.255 sunt adrese de loopback folosite în scopul monitorizării și testării.

Împărțirea în clase de adrese s-a dovedit prea inflexibilă. Pe de o parte, împărțirea este inefficientă. Pentru o organizație cu 10 calculatoare se alocă în întregime o adresă de clasă C (254 adrese posibile) irosindu-se astfel 244 adrese. Pentru o organizație cu 500 de calculatoare trebuie alocată o adresă de clasă B (65.534 adrese posibile) ceea ce duce la o pierdere de 65.034 de adrese. Pe de altă parte, nu există nici o corelație între blocurile de adrese alocate unor organizații diferite dar din aceeași zonă geografică. În consecință, pentru majoritatea rutelor din Internet este nevoie de câte o regulă de dirijare pentru fiecare organizație căreia i s-a alocat un bloc de adrese.

Ca urmare, s-a decis folosirea unei noi scheme de alocare a blocurilor de adrese. Noua schemă se numește CIDR (Classless InterDomain Routing) și este descrisă în RFC 1519. În CIDR partea de rețea poate avea lungime variabilă. O organizație care dorește acces Internet poate solicita alocarea unui bloc de adrese, având un număr de adrese egal cu o putere a lui 2.

Reprezentarea unei adrese IP în CIDR se face folosind două componente: adresa IP propriu-zisă și masca de subrețea (subnet mask). Masca de subrețea reprezintă o valoare pe 32 de biți. Biții care se află în porțiunea de subrețea au valoarea 1, iar biții care se află în porțiunea de stație au valoarea egală cu 0. În urma operației de AND logic între adresa IP inițială și masca de subrețea se obține adresa subrețelei din care face parte stația. De exemplu, dacă adresa stației este 193.231.21.30 și masca de subrețea 255.255.255.248, atunci adresa subrețelei din care face parte stația este 193.231.21.24. Pentru identificarea celor două componente ale adresei stației se poate folosi și notația 193.231.21.30/29 în care /29 semnifică numărul de biți de 1 din masca de subrețea.

Adrese IP private

Adresele private sunt utilizate în rețelele interne ale organizațiilor și nu sunt rutate în Internet. Spre deosebire de adresele publice, aceste adrese nu trebuie să fie unice la nivel global. Definite în RFC 1918, blocurile de adrese IP private sunt:

10.0.0.0 – 10.255.255.255
172.16.0.0 – 172.31.255.255
192.168.0.0 – 192.168.255.255

Utilizarea acestor tipuri de adrese este motivată atât de numărul limitat de adrese publice disponibile, cât și de faptul că astfel de adrese pot fi gestionate local, de fiecare organizație în parte. Dezavantajul adreselor private constă în faptul că stațiile care folosesc aceste adrese nu pot stabili conexiuni în mod direct cu alte calculatoare din Internet. Pentru a stabili aceste conexiuni este necesar să se realizeze o „traducere” a adresei private într-o adresă publică. Mecanismul NAT (Network Address Translation), descris în RFC 1631, face posibilă traducerea unui bloc de adrese private într-o adresă publică, rutabilă în Internet.

1.4.2. Protocolul ICMP

ICMP (Internet Control Message Protocol) este folosit pentru semnalarea erorilor ce pot apărea în timpul transmiterii pachetelor IP. Protocolul presupune transmiterea în rețea a unor mesaje ICMP. Acestea sunt încapsulate în pachete IP, după cum se poate observa în Figura 1-5.

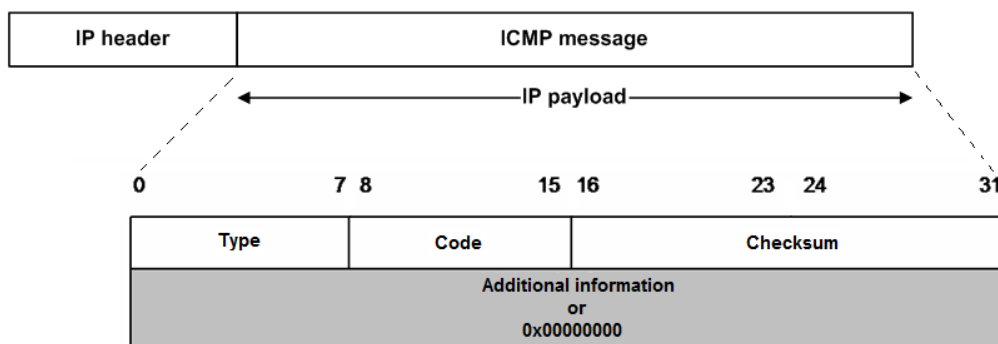


Figura 1-5. Formatul mesajelor ICMP

Mesajele ICMP, definite în RFC 792, sunt:

- *Destination unreachable*. Mesajul *destination unreachable* este generat de către un router atunci când destinația face parte din rețeaua în care se

află și router-ul, însă nu este accesibilă. Ca urmare, pachetul nu poate fi transmis.

- *Time exceeded.* Mesajul *time exceeded* este trimis de către un router atunci când valoarea TTL-ului a pachetului a ajuns la 0.
- *Parameter problem.* Mesajul *parameter problem* este generat atunci când un pachet IP primit de către o stație nu conține valori valide în câmpurile din antet.
- *Source quench.* Mesajul *source quench* indică supraîncărcarea cu pachete a unei stații. Acest lucru apare atunci când stația primește pachete mai repede decât le poate procesa.
- *Redirect message.* O stație trimite acest mesaj atunci când pachetul pe care îl primește a parcurs o rută ineficientă. Mesajul conține adresa IP a gateway-ului pe care sursa ar trebui să-l folosească astfel încât ruta parcursă să fie optimă.

ICMP permite, de asemenea, efectuarea de interogări simple în rețea. Pentru acest scop, se pot folosi următoarele mesaje:

- *Echo request/Echo reply.* Mesajele *echo request* și *echo reply* sunt utilizate pentru testarea conectivității între două stații. Utilitarul care verifică dacă o stație este activă este *ping*. Acesta utilizează mesajele de tip *echo request* și *echo reply*.
- *Timestamp request/Timestamp reply.* Mesajele *timestamp request* și *timestamp reply* permit unei stații să afle de la altă stație care este timpul curent.
- *Address mask request/ address mask reply.* Mesajele *address mask request* și *address mask reply* permit unei stații să afle de la router adresa de subrețea.

1.5. Nivelul Transport

În stiva TCP/IP, deasupra nivelului rețea se află *nivelul transport*. Aici găsim protocoale de tip end-to-end ce asigură servicii de comunicație pentru nivelul aplicație. Nivelul transport este cel care realizează adaptarea între serviciile oferite de nivelul rețea și cerințele proceselor de nivel aplicație.

La nivelul transport sunt implementate două protocoale: TCP (Transmission Control Protocol) și UDP (User Datagram Protocol). TCP este un protocol fiabil, orientat conexiune care presupune un overhead ridicat. UDP este neorientat conexiune, mai puțin fiabil dar cu un overhead redus. Funcție de cerințele specifice, aplicațiile pot opta pentru unul din cele două protocoale.

1.5.1. Protocolul TCP

TCP-ul, definit în RFC 793, este un protocol orientat conexiune, fiabil, care permite transmiterea fără erori a unui flux de octeți de la o aplicație la alta utilizând serviciile de comunicație puse la dispoziție de nivelul rețea (IP). TCP-ul folosește mecanismul cu fereastră glisantă pentru controlul fluxului și al erorilor.

Unitatea de transmisie și recepție a datelor este *segmentul*. Un segment TCP este format dintr-un antet de 20 de octeți și o zonă de date, de lungime variabilă.

Formatul segmentelor TCP

Segmentele TCP sunt încapsulate în datagrame IP. Valoarea atribuită de către IANA pentru TCP, în câmpul *Protocol* din antetul datagramelor IP, este 6. Într-o datagramă IP care conține un segment TCP, antetul TCP va urma imediat după antetul IP. Figura 1-6 ilustrează formatul segmentelor TCP.

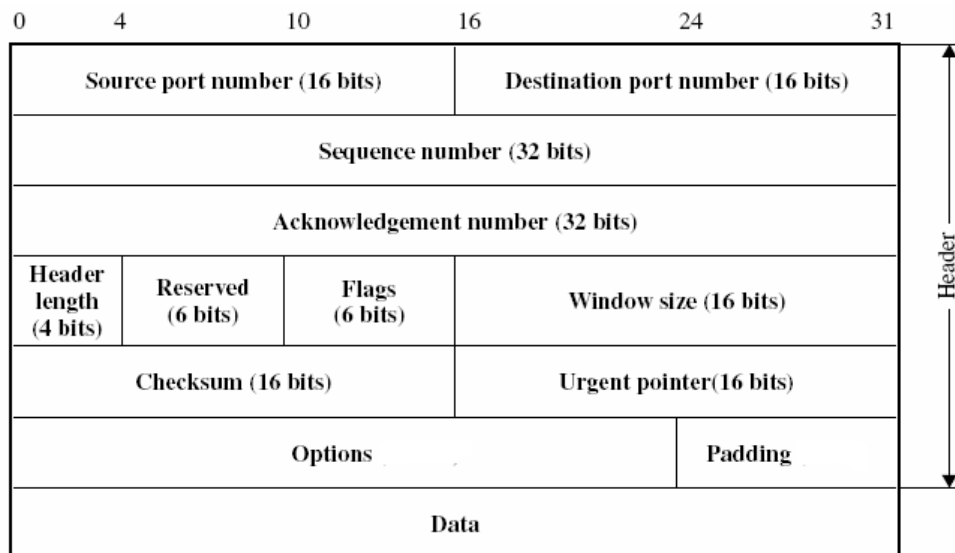


Figura 1-6. Formatul segmentelor TCP

Port sursă (Source port) și port destinație (Destination port) (16 biți). Stabilirea unei conexiuni TCP presupune ca fiecare dintre cele două puncte finale (procesul sursă și procesul destinație) să poată fi identificat printr-un număr. Acesta se numește număr de port. Porturile cu valori cuprinse între 1-1023 reprezintă porturi standard și sunt descrise în RFC 1700. Porturile standard sunt utilizate de aplicațiile server. Porturile cu valori cuprinse între 1024-65535 sunt porturi efemere și sunt folosite, de regulă, de aplicațiile client.

Număr de secvență (Sequence number) (32 biți). Fiecare octet din cadrul fluxului de date este numerotat. Numărul de secvență reprezintă numărul de ordine al primului octet de date din cadrul segmentului. Numerotarea octetilor începe de la o valoare aleatoare aleasă în momentul stabilirii conexiunii.

Numărul de confirmare (Acknowledgement number) (32 biți). Numărul de confirmare reprezintă numărul următorului octet pe care receptorul se așteaptă să-l primească. Este folosit pentru a confirma recepția datelor. Confirmările sunt cumulative. Câmpul are semnificație numai dacă flag-ul ACK este setat.

Lungimea antetului (Header length) (4 biți). Câmpul conține lungimea antetului TCP în cuvinte de 32 de biți. Este folosit pentru a specifica unde începe zona de date în cadrul segmentului TCP.

Câmpul *Rezervat (Reserved)* este alocat pentru extensii ulterioare ale protocolului.

Flag-urile de control (Flags) sunt: URG, ACK, PSH, RST, SYN, FIN.

Atunci când *URG* este setat, segmentul conține un mesaj urgent. Valoarea câmpului *Urgent pointer* specifică offset-ul din cadrul segmentului până la care se află datele urgente.

Dacă *ACK* este setat atunci numărul de confirmare din cadrul segmentului este valid.

Flagul *PSH* notifică receptorul să transfere toate datele din bufferul său de recepție către nivelul aplicație.

Setarea flag-ului *RST* are drept efect resetarea forțată a conexiunii TCP.

Flagul *SYN* este folosit în cadrul fazei de stabilire a conexiuni pentru sincronizarea numerelor de secvență între emițător și receptor.

Flag-ul *FIN* este folosit pentru a solicita închiderea unei conexiuni TCP.

Câmpul *Dimensiunea ferestrei (Window size)* specifică numărul maxim de octeți pe care receptorul îi poate primi.

Valoarea *Sumei de control (Checksum)* (16 biți) se calculează peste un pseudo-header IP și segmentul TCP propriu-zis. Ea are rolul de a detecta erorile de transmisie.

Câmpul *Opțiuni (Options)* permite adăugarea unor opțiuni suplimentare. Lungimea acestuia este variabilă.

Lungimea antetului TCP trebuie să fie multiplu de 32 de biți. Câmpul *Padding* se folosește pentru a completa antetul, dacă este cazul.

Zona de date conține datele primite de la nivel aplicație și care trebuie transportate la destinație.

TCP asigură următoarele servicii:

1. *Fiabilitate*. TCP-ul realizează transmisia fără erori a datelor primite de la nivel aplicație. TCP dispune de mecanisme de recuperare a datelor pierdute, duplicate sau livrate în afara rândului. În timpul transmisiei, fiecare octet are asociat un număr de secvență iar emițătorul este notificat de faptul că datele au ajuns la destinație, prin intermediul ACK-urilor. Dacă nu se recepționează niciun ACK, datele sunt retransmise. La destinație, numărul de secvență este utilizat pentru ordonarea datelor și pentru a elimina duplicate. Destinația verifică, de asemenea și suma de control. Dacă aceasta este eronată, segmentul trebuie retransmis.
2. *Controlul fluxului*. Protocolul TCP asigură faptul că niciodată un emițător mai rapid nu va sufoca un receptor mai lent. Acest lucru este realizat prin intermediul mecanismului cu fereastră glisantă. Câmpul dimensiunea ferestrei specifică volumul maxim de octeți pe care receptorul îl poate primi la un moment dat.
3. *Multiplexare*. Multiplexarea permite ca mai multe procese de nivel aplicație să folosească simultan mecanismele de comunicație oferite de TCP. Acest lucru se realizează prin intermediul porturilor de comunicație.
4. *Conexiunea*. TCP-ul menține informații de stare pentru fiecare flux de date transmis sau recepționat. Informațiile memorate constituie o conexiune. Aceasta este formată din: numerele de porturi, numerele de secvență, dimensiunile ferestrelor de recepție, etc. O conexiune între două calculatoare este stabilită atunci când informația de stare este inițializată. La închiderea conexiunii se eliberează toate resursele alocate pentru conexiunea respectivă.

Transmisia de date între două procese de nivel aplicație (client și server) folosind protocolul TCP poate fi privită ca un proces în trei etape: stabilirea conexiunii, transferul efectiv de date și închiderea conexiunii.

Prima etapă, stabilirea conexiunii TCP, se realizează prin intermediul unui mecanism denumit *3-way handshake*. În timpul acestei etape se stabilesc numerele de secvență inițiale, dimensiunile ferestrelor de recepție și se alocă resursele necesare conexiunii. Figura 1-7 prezintă modul de stabilirea al unei conexiuni TCP.

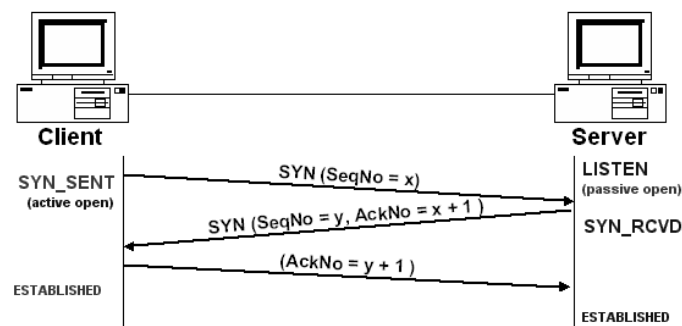


Figura 1-7. Stabilirea unei conexiuni TCP

După stabilirea conexiunii, între procesul client și procesul server se pot transfera date. TCP permite comunicație full duplex – schimb de date în ambele direcții simultan.

Ultima etapă, închiderea conexiunii TCP, începe atunci când unul dintre cele două procese nu mai are date de transferat și trimite un segment cu flagul FIN setat. Procesul pereche răspunde cu ACK. Pentru închiderea completă a conexiunii, procesul trebuie repetat și în celălalt sens. Figura 1-8 prezintă modul de închidere al unei conexiuni TCP.

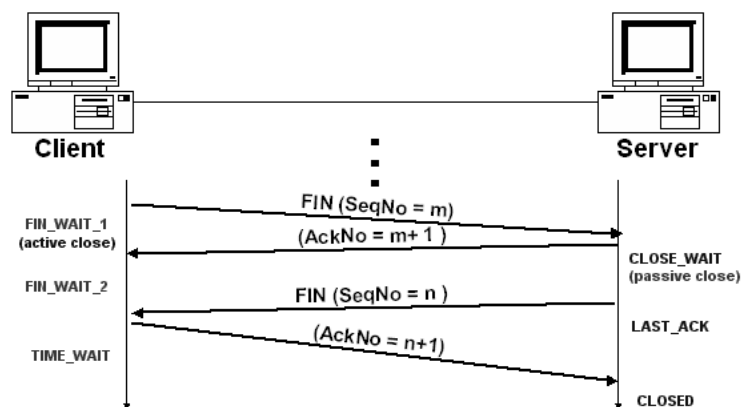


Figura 1-8. Închiderea unei conexiuni TCP

În timpul acestor etape, cele doua procese implicate se pot afla în una din următoarele *stări*:

- *LISTEN*: reprezintă starea în care se află un proces care așteaptă să primească o cerere de stabilire a unei conexiuni TCP.
- *SYN-SENT*: este starea în care se află un proces care a trimis o cerere de inițiere a unei conexiuni TCP; procesul așteaptă primirea unui ACK.
- *SYN-RECEIVED*: este starea în care se află un proces care așteaptă primirea unui ACK, după ce a primit cererea de inițiere a unei conexiuni TCP și a transmis, la rândul său, o cerere de începere a conexiunii.
- *ESTABLISHED*: reprezintă starea în care se află cele două procese imediat după ce s-a încheiat etapa de stabilire a conexiunii.
- *FIN-WAIT-1*: este starea în care se află un proces care a trimis un pachet cu flag-ul FIN setat; procesul așteaptă primirea unui ACK.
- *FIN-WAIT-2*: reprezintă starea în care se află un proces care a trimis un FIN și a primit un ACK.
- *CLOSE-WAIT*: este starea în care TCP așteaptă o cerere de închidere a sesiunii de la un proces local.
- *CLOSING*: este starea în care TCP așteaptă o cerere de închidere a sesiunii de la stația remote.
- *LAST-ACK*: este starea în care TCP a primit o confirmare pentru cererea trimisă de terminare a sesiunii și a transmis la rândul său un ACK pentru confirmarea primită.
- *TIME-WAIT*: este starea în care TCP așteaptă să treacă un interval de timp suficient de lung pentru a fi sigur că procesul remote a primit ACK-ul pentru cererea sa de închidere a conexiunii.
- *CLOSED*: starea în care nu există conexiune.

Principalele protocoale de nivel aplicație care folosesc TCP pentru transmitia datelor prin rețea și porturile standard asociate, sunt enumerate mai jos:

- File Transfer Protocol (20,tcp)
- Telnet (23/tcp)
- Simple Mail Transfer Protocol (25/tcp)

- Hypertext Transfer Protocol (80/tcp)
- Post Office Protocol (110/tcp)
- Internet Message Access Protocol (143/tcp)
- Lightweight Directory Access Protocol (389/tcp)

1.5.2. Protocolul UDP

UDP (User Datagram Protocol) este alt protocol utilizat la nivel transport. Spre deosebire de TCP, acesta este un protocol neorientat conexiune. UDP nu garantează livrarea datelor. El asigură doar o interfață simplă între nivelul aplicație și nivelul rețea.

Spre deosebire de TCP, UDP are un overhead redus. Din această cauză, UDP este folosit în aplicații pentru care timpul de transmisie al datelor este mai important decât transmitia integrală a acestora (de exemplu, transmitia de voce sau video). UDP este definit în RFC 768.

Formatul datagramelor UDP

Valoarea atribuită de către IANA pentru UDP în câmpul *Protocol* din antetul datagramelor IP este 17.

Antetul unei datagrame UDP are patru câmpuri de dimensiune fixă, fiecare având câte 16 biți, după care urmează zona de date de lungime variabilă. Formatul datagramelor UDP este prezentat în Figura 1-9.

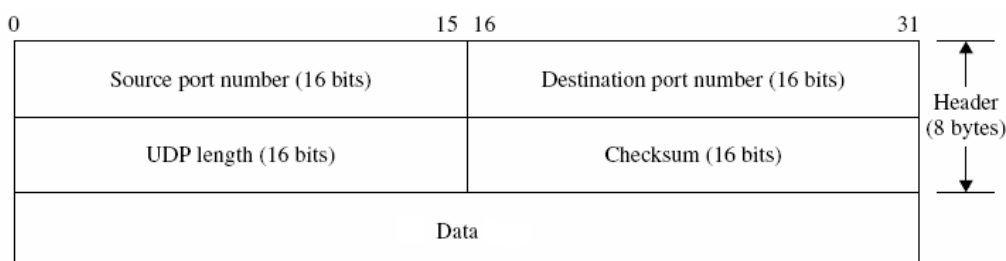


Figura 1-9. Formatul datagramelor UDP

Port sursă și *port destinație* (16 biți) au același rol ca și în cazul protocolului TCP. Acestea identifică procesele de nivel aplicație implicate în transmitia datelor.

Lungimea (UDP length) (16 biți) reprezintă dimensiunea în octeți a datagramei UDP.

Valoarea *Sumei de control (Checksum)* (16 biți) se calculează peste un pseudo-header IP și datagrama UDP propriu-zisă. Ea are rolul de a detecta erorile de transmisie.

UDP este folosit de următoarele protocoale de nivel aplicație:

- Domain Name System (53/udp)
- Dynamic Host Configuration Protocol (67,68/udp)
- Trivial File Transfer Protocol (69/udp)
- Simple Network Management Protocol (161/udp)

1.6. Nivelul Aplicație

Nivelul superior al stivei TCP/IP este nivelul aplicație. Rolul acestui nivel este de a furniza servicii utilizatorilor rețelei. La nivel aplicație funcționează protocoale diverse dintre care cele mai cunoscute sunt:

- *DNS* (Domain Name System) este utilizat pentru conversia numelor de calculatoare în adrese IP și invers;
- *Telnet* oferă serviciul de conectare la distanță a utilizatorilor;
- *FTP* (File Transfer Protocol) este utilizat pentru transfer de fișiere între calculatoare;
- *SMTP* (Simple Mail Transfer Protocol) este folosit pentru transmiterea de mesaje electronice (e-mail) între utilizatori;
- *HTTP* (Hypertext Transfer Protocol) folosit pentru accesul la paginile WWW (World Wide Web).

2. Tehnologii criptografice

Acest capitol face o trecere în revistă a celor mai moderne tehnologii criptografice folosite în cadrul protocoalelor de securitate pentru asigurarea confidențialității, integrității, autenticității și non-repudierii datelor transmise între entități prin intermediul rețelelor de calculatoare.

2.1. Sisteme criptografice

Sistemele criptografice, denumite uneori și *cifruri*, se folosesc pentru asigurarea confidențialității mesajelor transmise între entități încă de pe vremea Romei Antice. *Cifrul lui Cezar* este primul sistem de acest gen iar principiul substituției care stă la baza sa, s-a menținut nealterat aproape două milenii. Între timp, sistemele criptografice au fost perfecționate continuu iar apariția calculatoarelor electronice a permis implementarea unor algoritmi de o complexitate și securitate ridicată.

Un *sistem criptografic* constă dintr-o transformare de *criptare* și una de *decriptare* după cum se poate observa în Figura 2-1. Prin criptare, textul clar este convertit într-un format neinteligibil numit text criptat sau *criptogramă*. Transformarea inversă, de decriptare, se aplică pe criptogramă pentru a reface textul inițial. Atât transformarea de criptare cât și cea de decriptare sunt controlate de către una sau mai multe *chei criptografice*.

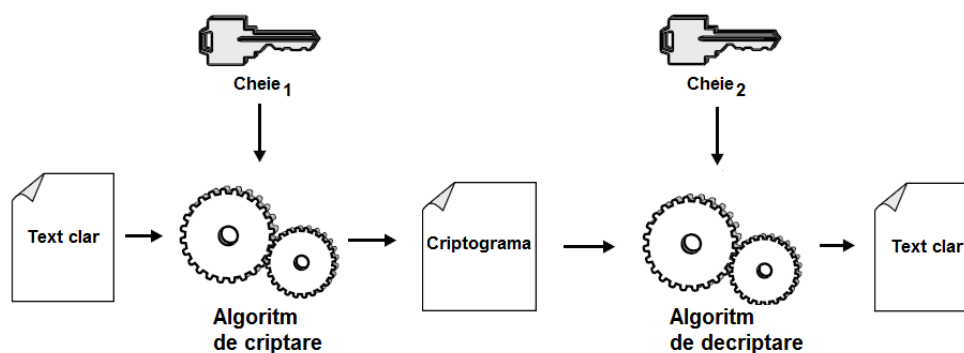


Figura 2-1. Sistem criptografic

În criptografia modernă, transformările de criptare și de decriptare au la bază operații matematice complexe iar pentru implementările practice se folosesc sisteme de calcul.

Experiența de până acum în domeniul criptografiei a arătat că securitatea unui sistem criptografic trebuie să depindă numai de chei și nu de ținerea secretă a transformărilor de criptare și de decriptare folosite. În majoritatea cazurilor, algoritmi de criptare și de decriptare sunt publici pentru a putea fi analizați din punct de vedere al securității sau pentru a permite implementarea eficientă a acestora pe diferite platforme de calcul. Prin urmare, securitatea sistemului depinde de modul în care se realizează gestiunea și protecția cheilor criptografice.

Sistemele criptografice sunt de două tipuri:

1. *Sisteme criptografice cu chei secrete (simetrice)* ce folosesc aceeași cheie atât pentru criptarea cât și pentru decriptarea datelor.
2. *Sisteme criptografice cu chei publice (asimetrice)* în care cele două chei sunt diferite.

2.2. Sisteme criptografice cu chei secrete

Un sistem criptografic cu chei secrete folosește aceeași cheie atât pentru criptarea mesajelor cât și pentru decriptarea acestora. Din această cauză, sistemele criptografice cu chei secrete se mai numesc și simetrice. Figura 2-2 prezintă modul de asigurare a confidențialității mesajelor transmise între două entități. Înaintea oricărui schimb de mesaje, entitățile trebuie să stabilească în comun o cheie de criptare secretă numită și cheie de sesiune.

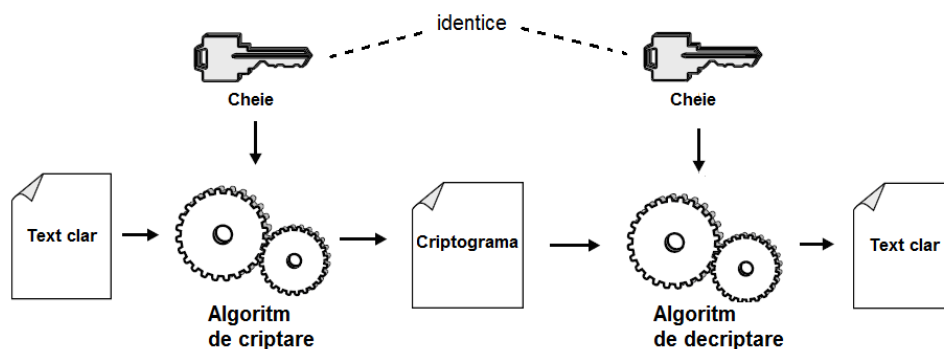


Figura 2-2. Sistem criptografic cu chei secrete

Tăria unui sistem criptografic cu chei secrete este dată de o serie de factori. Unul dintre aceștia îl reprezintă gradul de aleatorism al criptogramelor rezultate. Este foarte important ca porțiuni de text similare din cadrul mesajului inițial să nu producă rezultate identice la ieșire. Acest parametru poartă denumirea de

entropie. Cu cât gradul de entropie este mai mare cu atât rezistența algoritmului la *atacuri criptanalitice* este mai ridicată.

Lungimea cheilor este de asemenea foarte importantă pentru a putea face față la atacuri prin încercări repetate de determinare a cheii de criptare (*brute force attack*). Algoritmii criptografici simetrici actuali folosesc chei având o lungime cuprinsă între 40 și 256 biți. În cazul algoritmilor pe 40 de biți, numărul de încercări necesare este de aproximativ 2^{40} (1.099.511.627.776). Fiecare bit adăugat suplimentar duce la dublarea spațiului de chei posibile.

În funcție de modul în care se realizează procesarea datelor de intrare, sistemele criptografice simetrice se împart în două mari categorii: *cifruri bloc* (*block ciphers*) și *cifruri șir* (*stream ciphers*).

Cifruri bloc

Cifrurile bloc operează pe blocuri de date. Ei primesc la intrare câte un bloc de n biți din textul în clar și generează la ieșire criptograme de aceeași dimensiune (vezi Figura 2-3). În cazul în care textul de la intrare nu este multiplu de n biți, atunci acesta trebuie completat în mod corespunzător. Operația de completare se numește *padding*.

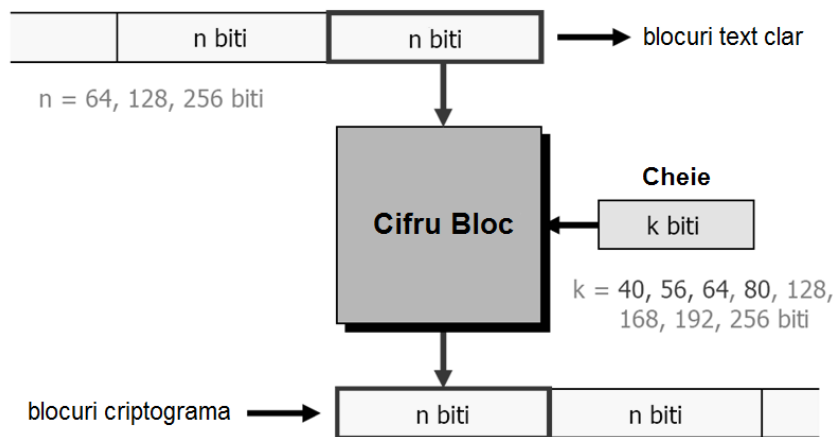


Figura 2-3. Cifru bloc

După cum se poate observa în figura Figura 2-3, un cifru bloc criptează blocurile independente și prin urmare, dacă un bloc de biți din textul în clar se repetă, criptogramele rezultate vor fi identice. Acest lucru poate fi exploatat de un criptanalist pentru a desfășura atacuri prin reluare sau pentru a decifra mesajele dacă cheia secretă este folosită pentru o perioadă mare de timp.

Modul independent de criptare a blocurilor de biți din textul în clar poartă denumirea de ECB (Electronic Code Book). Având la dispoziție suficientă memorie (2^{64} cuvinte de 64 biți), pentru o cheie dată se poate construi o tabelă de conversie (look-up table) prin care să se pună în corespondență orice bloc de criptogramă cu blocul de text în clar corespunzător fără să mai fie necesar să se aplice transformarea de decriptare. De aici și denumirea acestui mod de lucru. ECB este prezentat în Figura 2-4.

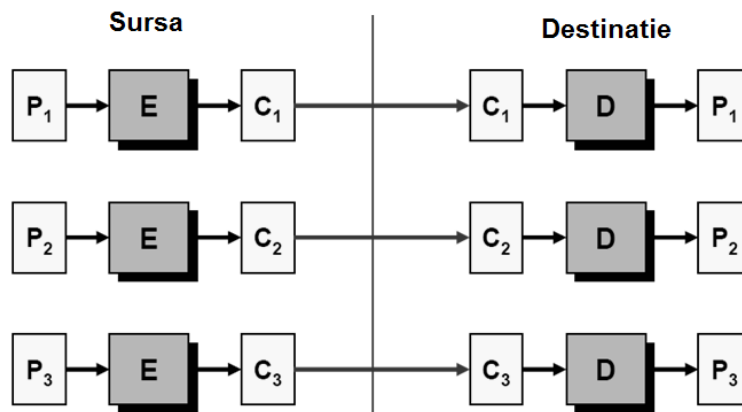


Figura 2-4. Modul ECB de criptare a datelor

Pentru a elimina neajunsurile ECB, au fost dezvoltate modurile de criptare cu feedback în care criptograma curentă depinde de textul în clar și de criptograma anterioară. Cel mai cunoscut mod de criptare cu feedback este CBC (Cipher Block Chaining) prezentat în Figura 2-5.

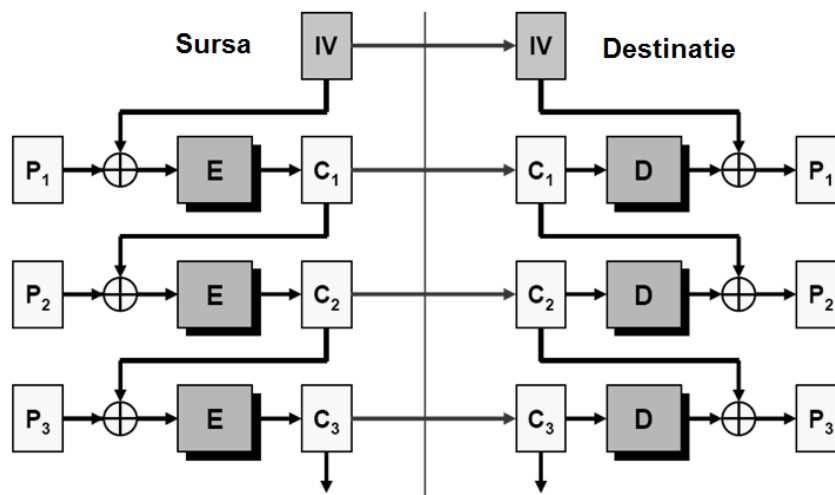


Figura 2-5. Modul CBC de criptare a datelor

Când se criptează date în modul CBC, fiecare bloc din textul în clar este făcut XOR cu blocul anterior din criptogramă și apoi se aplică transformarea de criptare. Pentru primul bloc din textul în clar neexistând o criptogramă anterioară, se folosește un vector de inițializare (IV). Astfel, blocuri identice din textul în clar vor produce la ieșire criptograme diferite.

Cei mai cunoscuți algoritmi de cifrare bloc sunt:

- *DES (Data Encryption Standard)*. DES a reprezentat timp de 20 de ani cel mai popular standard pentru criptarea datelor. El a fost elaborat de IBM în jurul anilor 1970 și adoptat apoi ca standard federal în 1977 (FIPS 46). Ultima actualizare a acestui standard a avut loc în 1999 când a fost publicat FIPS 46-3. DES operează pe blocuri de date de 64 de biți, lungimea efectivă a cheii de criptare fiind de 56 biți (64 biți din care se scad 8 biți de paritate).
- *3DES (Triple-DES)*. 3DES reprezintă transformarea obținută prin aplicarea de trei ori succesiv a algoritmului DES. Exista două variante de 3DES: DES-EDE și DES-EEE. În cazul DES-EDE ordinea transformărilor este criptare-decriptare-criptare, fiecare tip de transformare folosind chei diferite. DES-EEE semnifică trei criptări consecutive. Principalul avantaj al 3DES față de DES îl reprezintă creșterea lungimii cheilor de criptare.
- *AES (Advanced Encryption Standard)*. AES este succesorul algoritmului DES. El reprezintă noul standard federal (FIPS 197) pentru criptarea datelor și a fost selectat pe baza unui concurs inițiat de NIST în 1998 în urma căruia a fost desemnat câștigător algoritmul Rijndael inventat de Joan Daemen și Vincent Rijmen. AES operează pe blocuri de date de 128 de biți și suportă lungimi de chei de 128, 192 și 256 de biți.
- *RC2*. RC2 a fost inventat de celebrul cercetător Ron Rivest de la MIT (ceea ce explică și semnificația acronimului RC – “Rivest Cipher”). Algoritmul RC2 s-a dorit a fi un înlocuitor pentru DES, permițând folosirea unor lungimi de chei de dimensiune variabilă. RC2 operează pe blocuri de date de 64 de biți și este de două până la trei ori mai rapid decât DES în implementările software. RC2 este vulnerabil la o serie de atacuri criptanalitice și nu mai este considerat ca fiind sigur.
- *Blowfish*. Acest algoritm a fost propus de Bruce Schneier de la firma Counterpane Systems, autorul cărții “Applied Cryptography”. Blowfish operează pe blocuri de date de 64 de biți și suportă lungimi variabile de chei de până la 448 de biți. Blowfish a fost proiectat special pentru procesoare pe 32 de biți fiind semnificativ mai rapid decât DES.

- *Twofish*. Twofish reprezintă succesorul algoritmului Blowfish, fiind propus de același autor, Bruce Schneier. Twofish a candidat pentru AES ajungând până în faza finală a procesului de selecție.
- *IDEA (International Data Encryption Algorithm)*. IDEA a fost propus de către Xuejia Lai și James Massey de la ETH Zurich. Celebritatea algoritmului se datorează PGP-ului care folosește acest algoritm simetric pentru criptarea datelor. IDEA operează pe blocuri de date de 64 de biți, lungimea cheii de criptare fiind de 128 de biți. Viteza de procesare este comparabilă cu a DES-ului.
- *CAST*. CAST reprezintă prescurtarea de la Carlisle Adams și Stafford Tavares, autorii algoritmului. CAST este un cifru bloc pe 64 de biți ce suportă chei de criptare de până la 128 de biți. CAST-128 este proprietatea firmei Entrust Technologies însă poate fi folosit atât în scopuri comerciale cât și non-comerciale. CAST-256 este o extensie a CAST-128, disponibilă gratuit, ce operează pe blocuri de date de 128 de biți, lungimea cheilor de criptare fiind de până la 256 de biți. CAST-256 a candidat, de asemenea, pentru AES.

Cifruri șir

Cifrurile șir operează la nivel de bit. Cifrurile șir au la bază un generator pseudo aleator de biți inițializat de o cheie secretă. Între biții astfel generați și șirul de biți corespunzător textului în clar se aplică operația *XOR* pentru a obține criptograma. La recepție, folosind aceeași cheie secretă, se generează aceeași secvență de biți pseudo aleatori și se face XOR șirul de biți din criptogramă pentru a reface textul în clar (vezi Figura 2-6).

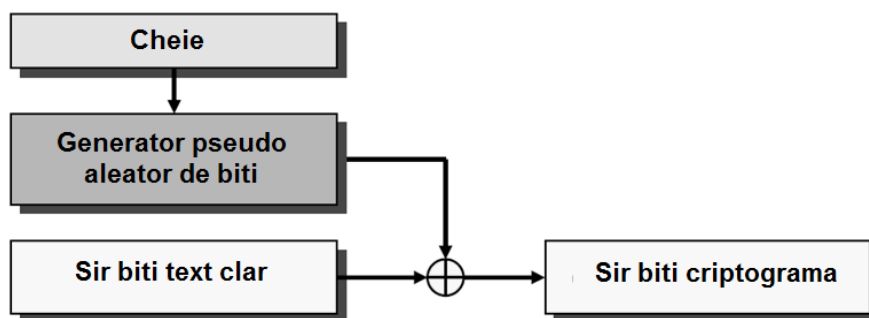


Figura 2-6. Cifru șir

Cifrurile șir sunt, de regulă, mai rapizi și necesită mai puține linii de cod decât cifrurile bloc. Un alt avantaj al cifrurilor șir îl reprezintă faptul că o eroare de un bit în criptogramă afectează un singur bit din textul în clar și nu se propagă mai departe cum se întâmplă în cazul cifrurilor bloc în modul CBC, de exemplu.

Dintre cei mai utilizați algoritmi de cifrare șir, amintim:

- RC4. RC4 a fost creat de Ron Rivest și permite o lungime variabilă a cheii de criptare de până la 2048 de biți. RC4 a fost unul din algoritmi standard ai SSL folosit pentru criptarea traficului Web. În ultima perioadă au fost descoperite o serie de atacuri asupra RC4 fapt pentru care algoritmul nu mai este considerat ca fiind sigur și nu se mai recomandă folosirea sa.
- SEAL (Software Efficient Algorithm). Autorii acestui algoritm sunt Phil Rogaway și Don Coppersmith de la IBM. SEAL folosește o cheie de 160 biți și este considerat unul din cei mai siguri algoritmi de cifrare șir.
- WAKE (World Auto Key Encryption). WAKE a fost propus de David J Wheeler și reprezintă un algoritm pentru criptare șir, de viteză medie, care oferă însă o securitate ridicată.

Un cifru șir poate fi obținut și dintr-un cifru bloc prin intermediul unor moduri speciale de criptare cum ar fi Cipher Feedback (CFB), Output Feedback (OFB) sau Counter (CTR).

Sistemele criptografice simetrice au performanțe ridicate din punct de vedere al timpului de procesare permițând criptarea rapidă a mesajelor de diferite dimensiuni. De asemenea, resursele de calcul necesare sunt relativ reduse, algoritmi simetrici actuali fiind special proiectați a se executa eficient pe sisteme de calcul uzuale.

Principalul dezavantaj al sistemelor criptografice simetrice îl reprezintă managementul cheilor de criptare. Managementul cheilor este un factor vital în asigurarea securității sistemelor criptografice cu chei secrete și cuprinde aspecte precum: generarea, distribuția sau stocarea cheilor. În cazul în care n entități doresc să comunice confidențial, numărul de schimburi de chei este de ordinul $n(n-1)/2$ iar managementul cheilor devine o problemă critică.

2.2.1. Algoritmul AES (Advanced Encryption Standard)

În prezent, AES (Advanced Encryption Standard) este noul Standard Federal pentru Procesarea Informației (FIPS – Federal Information Processing Standard), care specifică algoritmul criptografic recomandat a fi utilizat de către organizațiile guvernamentale ale SUA pentru protejarea informațiilor sensitive neclasificate. AES fost selectat pe baza unui concurs inițial de NIST în 1998 în urma căruia a fost desemnat câștigător algoritmul Rijndael inventat de Joan Daemen și Vincent Rijmen.

Intrarea și ieșirea pentru algoritmul AES constă fiecare în secvențe (blocuri) de 128 de biți. Cheia de criptare pentru AES este o secvență de 128, 192 sau 256 de biți. În algoritmul AES, unitatea de bază pentru procesare este octetul. Intrarea, ieșirea și secvența de biți a cheii de criptare sunt procesate ca matrici de octeți, formate prin împărțirea acestor secvențe de biți în grupuri de câte 8 biți contigui. Pentru o intrare, ieșire sau cheie de cifrare indicată prin a , octeții care alcătuiesc matricea rezultată vor fi referiți prin forma a_n , unde n are valori în intervalele: lungimea cheii = 128 biți, $0 \leq n < 16$; lungimea cheii = 192 biți, $0 \leq n < 24$; lungimea cheii = 256 biți, $0 \leq n < 32$, pentru o lungime fixă a blocului = 128 biți, $0 \leq n < 16$.

Toate valorile biților vor fi prezentate ca o concatenare a valorilor biților (0 sau 1) între paranteze, în ordinea $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$. Acești biți sunt interpretați ca elemente în câmpul finit $GF(2^8)$, folosind următoarea reprezentare polinomială:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^7 b_i x^i$$

În interiorul algoritmului se execută operații matematice asupra unor matrici de octeți, care definesc o *stare*. Matricile de octeți sunt reprezentate sub forma $a_n = \{input_{8n}, input_{8n+1}, \dots, input_{8n+7}\}$, unde o secvență este definită ca $(input_{8n}, \dots, input_{8n+7})$. O stare constă în patru rânduri de octeți, fiecare conținând Nb octeți, unde Nb este lungimea blocului împărțită la 32. În matricea de stare, notată cu simbolul s , fiecare octet are doi indici, r - numărul rândului cu $0 \leq r < 4$ și c - numărul coloanei cu $0 \leq c < Nb$. Aceste notații permit ca un octet să fie definit unic ca $s[r, c]$ în matricea de stare. În AES, $Nb=4$, ceea ce înseamnă că $0 \leq c < 4$.

La începutul algoritmului de criptare, intrarea - matricea de octeți: $in_0, in_1, \dots, in_{15}$ - este copiată în matricea de stare (Figura 2-7). Operațiile de criptare sau decriptare se efectuează în matricea de stare, iar valoarea finală este copiată la ieșire - matricea de octeți: $out_0, out_1, \dots, out_{15}$. Prin generalizarea operației de criptare, matricea de intrare in este copiată în matricea de stare conform schemei:

$$s[r, c] = in[r + 4c], \text{ pentru } 0 \leq r < 4 \text{ și } 0 \leq c < 4,$$

iar la sfârșitul cifrării sau descifrării, matricea de stare este copiată în matricea de ieșire conform schemei:

$$out[r + 4c] = s[r, c], \text{ pentru } 0 \leq r < 4 \text{ și } 0 \leq c < 4.$$

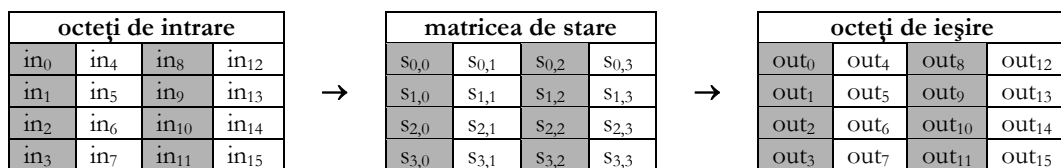


Figura 2-7. Matricele de intrare, stare și ieșire ale algoritmului AES

Se observă maparea din matricea de stare a patru octeți pe coloane, care formează cuvinte pe 32 de biți și în care rândul furnizează un index, al fiecărui cuvânt. De exemplu, dacă blocul de intrare constă în următoarea secvență de octeți $a_0, a_1, a_2, \dots, a_{15}$, atunci blocul este mapat în matricea de stare ca în Figura 2-8, ceea ce înseamnă că matricea de stare se poate interpreta ca o matrice coloană cu 4 cuvinte de 32 de biți:

$$w_0 = a_0 a_1 a_2 a_3, w_1 = a_4 a_5 a_6 a_7, w_2 = a_8 a_9 a_{10} a_{11}, w_3 = a_{12} a_{13} a_{14} a_{15}$$

a_0	a_4	a_8	a_{12}
a_1	a_5	a_9	a_{13}
a_2	a_6	a_{10}	a_{14}
a_3	a_7	a_{11}	a_{15}

Figura 2-8. Maparea unui bloc de octeți

În algoritmul AES, toți octeții sunt interpretați ca elemente în câmpul finit $GF(2^8)$, păstrând notațiile anterioare. Elementele câmpului finit pot fi adunate și înmulțite după reguli specifice.

Lungimea *blocului de intrare*, *blocului de ieșire* și a *stării* din AES este de 128 biți ($Nb=4$), iar lungimea cheii de criptare, K , este 128, 192 sau 256 de biți. Lungimea cheii este reprezentată de $Nk=4, 6$ sau 8 , care arată numărul de cuvinte de 32 de biți (numărul de coloane) din cheia de cifrare (Figura 2-9).

	Lungimea cheii (Nk)	Mărime bloc (Nb)	Număr de runde(Nr)
AES – 128	4	4	10
AES – 192	6	4	12
AES – 256	8	4	14

Figura 2-9. Schema de organizare AES

Atât pentru criptare, cât și pentru decriptare, algoritmul AES folosește o funcție circulară care este compusă din 4 transformări asupra unui octet (prezentată în Figura 2-10): substituția octetului folosind o tabelă de substituție (*S-box*), deplasarea rândurilor din matricea de stare cu diferite offset-uri, mixarea datelor în interiorul fiecărei coloane a matricii de stare și adunarea unei chei de rotire ciclică la matricea de stare.

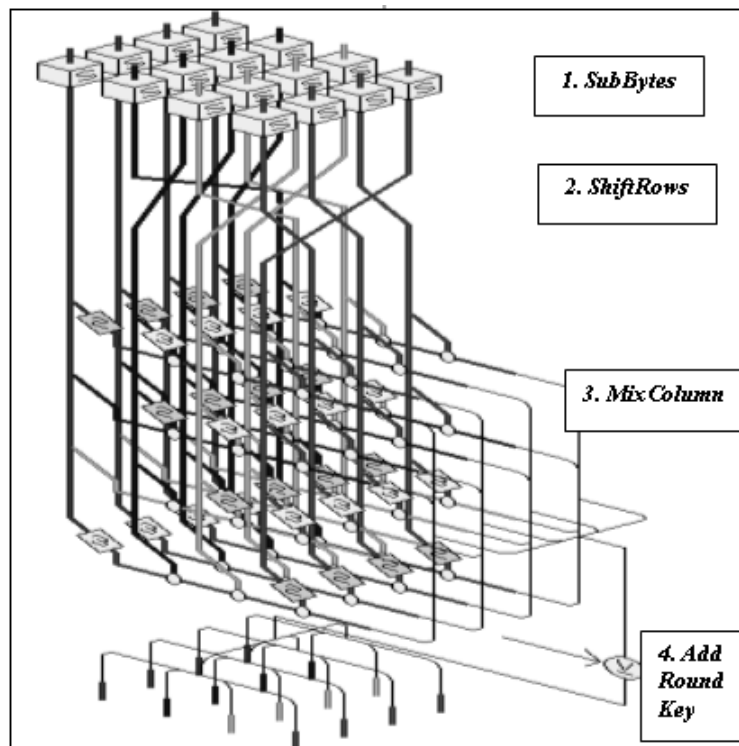


Figura 2-10. Diagrama tridimensională a algoritmului AES

La începutul algoritmului de criptare, intrarea este copiată în matricea de stare folosind convențiile descrise anterior. După efectuarea unei adunări inițiale a cheii de rotire, matricea de stare este transformată prin implementarea funcției circulare de 10, 12 sau 14 ori (în funcție de lungimea cheii: 128, 192 sau 256), cu runda finală diferită de runda ($Nr-1$). Funcția circulară este parametrizată folosind o cheie ordonată, care constă dintr-o matrice coloană cu patru octeți derivați, folosind rutina de expansiune a cheii.

Operația de criptare este descrisă în continuare în pseudocod.

```

Cipher(byte in[4 * Nb], byte out[4 * Nb], word w[Nb * (Nr + 1)])
begin
    byte state[4, Nb]
    state = in

    AddRoundKey(state, w)

    for round = 1 step 1 to Nr - 1
        SubBytes(state)    // punctul a)
        ShiftRows(state)   // punctul b)
        MixColumns(state)  // punctul c)
        AddRoundKey(state, w + round * Nb) //punctul d)
    end for

```

```

SubBytes(state)
ShiftRows(state)
AddRoundKey(state, w + Nr * Nb)

out = state
end

```

Transformările individuale – *SubBytes()*, *ShiftRows()*, *MixColumns()* și *AddRoundKey()* procesează starea și constau din următoarele operații:

a) Transformarea *SubBytes(state)* - reprezintă o substituție neliniară, care constă în înlocuirea fiecărui octet cu inversul lui în câmpul finit $GF(2^8)$, mai puțin elementul $\{00\}$ care este mapat cu el însuși. Apoi, se aplică o transformare afină peste $GF(2^8)$ definită de:

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

pentru $0 \leq i < 8$, unde b_i este bitul i din octet, iar c_i este bitul i din octetul c adunat cu valoarea $\{63\}$ sau $\{01100011\}$.

Tabela *S-box* folosită în transformarea *SubBytes()* este prezentată în hexazecimal în Tabelul 2-1.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Tabelul 2-1. Tabela S-box a algoritmului AES prezintă valorile substituite pentru octetul xy (în hexazecimal)

b) Transformarea *ShiftRows(state)* – octeții din ultimele 3 rânduri aflați în matricea de stare sunt roțiți ciclic cu diferite offset-uri. Primul rând, r_0 rămâne același, celelalte rânduri fiind rotite astfel:

$$s'_{r,c} = s_{r,(c+shift(r,Nb)) \bmod Nb}, \text{ pentru } 0 \leq r < 4 \text{ și } 0 \leq c < Nb.$$

Pentru $Nb=4$ și valorile specifice ale r , rezultă că $shift(1,4)=1$; $shift(2,4)=2$; $shift(3,4)=3$ (Figura 2-11):

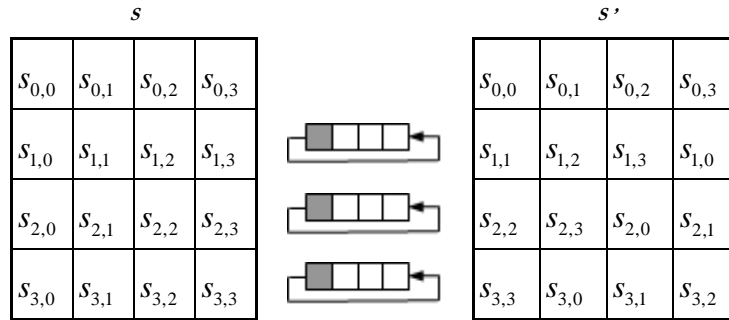


Figura 2-11. Schema de transformare *ShiftRows()*

c) Transformarea *MixColumns(state)* - are loc asupra matricei de stare, coloană cu coloană, considerând fiecare coloană ca un polinom de 4 termeni. Coloanele sunt considerate ca polinoame peste câmpul finit $GF(2^8)$ și multiplicare cu polinomul fix

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \bmod (x^4 + 1)$$

d) Transformarea *AddRoundKey(state, RoundKey)* – se adăugă o cheie de ciclare printr-o operație *XOR* la matricea de stare. Fiecare cheie de ciclare constă din cele Nb cuvinte ale schemei cheii de criptare.

Algoritmul preia cheia de criptare K și execută o rutină, numită *KeyExpansion()*, pentru a genera cheile de ciclare necesare în transformarea *AddRoundKey()*. *KeyExpansion()* generează un total de $Nb(Nr+1)$ cuvinte: inițial, algoritmul are nevoie de un set de Nb cuvinte și fiecare rundă Nr necesită Nb cuvinte ale cheii. Rezultatul acțiunii de planificare a cheii constă în formarea unei matrice liniare de cuvinte de 4 octeți, notată $[w_i]$, cu $0 \leq i < Nb(Nr+1)$.

Pseudocodul *KeyExpansion()* conține funcțiile:

- *SubWord()* – la intrare primește un cuvânt care este introdus în *S-box* și aplicat fiecărui octet pentru a produce la ieșire un cuvânt;
- *RotWord()* – primește la intrare un cuvânt cu care realizează o permutare ciclică, iar la ieșire produce un cuvânt;
- *Rcon[i]* – reprezintă un vector de cuvinte constant.

```

KeyExpansion(byte key[4 * Nk], word w[Nb * (Nr + 1)], Nk)
begin
    i=0
    while (i < Nk)
        w[i] = word[key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]]
        i = i + 1
    end while
    i = Nk
    while (i < Nb * (Nr + 1))
        word temp = w[i - 1]
        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i / Nk]
        else if (Nk = 8 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i - Nk] xor temp
        i = i + 1
    end while
end

```

Tehnicile standard ale criptanalizei lineare și diferențiale pot fi adaptate pentru a fi utilizate împotriva algoritmului AES. Datorită modului de efectuare a înmulțirii matricelor, dar și pentru că în câmpul $GF(2^8)$ toți coeficienții matricei *MixColumns()* au inverși, poate fi utilizat un atac specific, care mai întâi a fost destinat predecesorului său, algoritmul *Square*, și care se numește “*atacul square*”.

Dacă se utilizează 256 de blocuri de text clar ales și se mențin constante, mai puțin unul, iar acestuia i se dau toate cele 256 de valori posibile, atunci după prima rundă a algoritmului AES, patru octeți vor lua toate cele 256 de valori posibile, iar restul octeților vor rămâne constanți. După a doua rundă, 16 octeți vor trece fiecare prin cele 256 de valori posibile, fără nici o duplicare, în criptarea celor 256 de blocuri de text clar ales. Pentru blocuri de 128 de octeți, acest lucru se întâmplă pentru fiecare octet, iar pentru blocuri mai mari, restul de octeți vor rămâne constanți. Această proprietate importantă, chiar dacă nu este ușor de exploatat, poate fi utilizată pentru a impune anumite condiții cheii de criptare pentru a realiza o rundă adițională, înainte sau după cele două runde efectuate mai sus.

Avantajul de a folosi blocuri de mărimi diferite în algoritmul AES permite o variantă de “*a construi mai multe straturi cu cărămizi de diferite mărimi*”, unde cele trei nivele folosesc blocuri de 128, 192 și 256 de biți ca în Figura 2-12, asemănător cu algoritmul *Triple-DES*.

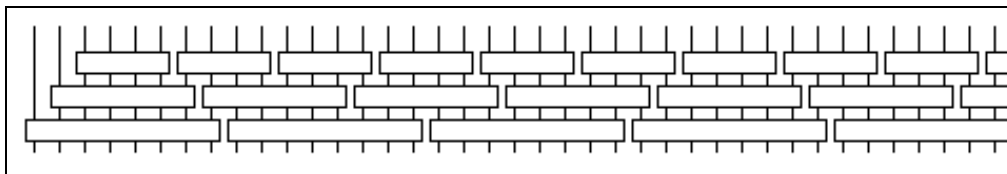


Figura 2-12. Folosirea blocurilor de diferite mărimi: 128, 192 și 256 de biți

2.3. Sisteme criptografice cu chei publice

În anul 1976, W. Diffie și M. Hellman, cercetători la Universitatea Stanford din California, au pus bazele criptografiei cu chei publice. În locul unei singure chei secrete, criptografia asimetrică folosește două chei diferite: una pentru criptare și alta pentru decriptare. Cheia de criptare a unei entități este făcută publică iar cea pentru decriptare este privată, fiind cunoscută numai de către entitatea respectivă. Pentru ca schema să fie sigură trebuie ca obținerea cheii de decriptare din cheia de criptare să fie imposibilă din punct de vedere computațional.

R. Rivest, A. Shamir și L. Adleman au arătat cum, probleme matematice grele ca factorizarea numerelor întregi mari sau calculul logaritmulor discreți într-un câmp finit, pot fi folosite pentru a realiza sisteme criptografice cu chei publice, propunând totodată și primul algoritm de acest gen, algoritmul RSA.

Pentru asigurarea confidențialității unui mesaj, acesta este cifrat cu cheia publică a destinatarului. Odată criptat, mesajul nu va mai putea fi decriptat decât de către destinatar, singurul care posedă cheia privată, pereche a cheii publice (Figura 2-13). Se observă că în acest caz distribuția cheilor se simplifică.

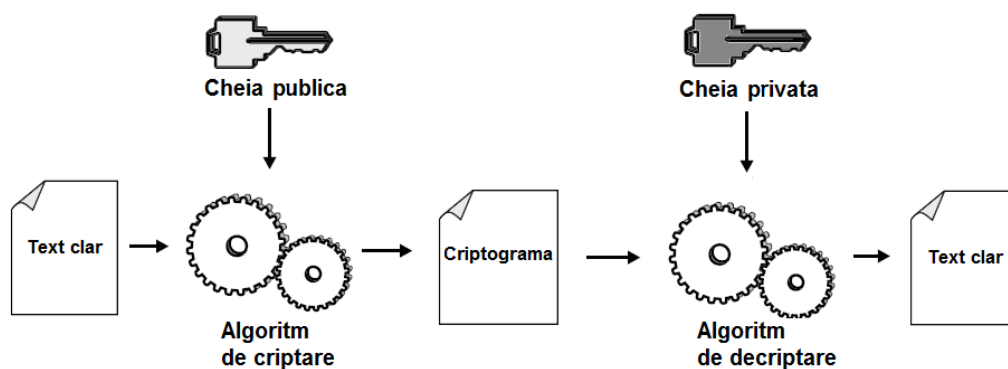


Figura 2-13. Sistem criptografic cu chei publice

Criptografia cu chei publice funcționează pe baza principiului conform căruia fiecare entitate are acces la cheile publice ale entităților cu care corespundează. Pentru aceasta, trebuie să existe un mecanism care să garanteze că o cheie publică aparține unei anumite entități. În caz contrar, sistemul devine vulnerabil la atacuri prin impersonare. Infrastructurile de chei publice (PKI – Public Key Infrastructure) permit utilizatorilor să obțină chei publice autentice sub forma de certificate digitale. Astfel, procesul de management al cheilor se simplifică în cazul sistemelor criptografice cu chei publice comparativ sistemele criptografice simetrice.

Cei mai populari algoritmi cu chei publice sunt: RSA, Merkle-Hellman, McEliece, ElGamal sau cei bazați pe curbe eliptice.

Datorită complexității operațiilor matematice efectuate, sistemele criptografice cu chei publice au viteze de procesare cu câteva ordine de mărime mai mici decât cele cu chei secrete și din această cauză se folosesc numai pentru criptarea unor mesaje reduse ca dimensiune. Transportul (distribuția) cheilor de simetrice de criptare reprezintă un astfel de exemplu.

Pentru asigurarea confidențialității datelor, criptarea acestora se realizează folosind un algoritm simetric (din motive de performanță) și o cheie de sesiune generată aleator. Cheia de sesiune este apoi criptată cu cheia publică a destinatarului folosind un algoritm asimetric și transmisă la destinație împreună cu criptograma rezultată. Cheia de sesiune poate fi decriptată numai de către destinatar, singurul care deține cheia privată corespunzătoare și este folosită apoi la refacerea mesajului inițial din criptograma recepționată. Întregul proces este descris în Figura 2-14 și poartă denumirea de anvelopare digitală.

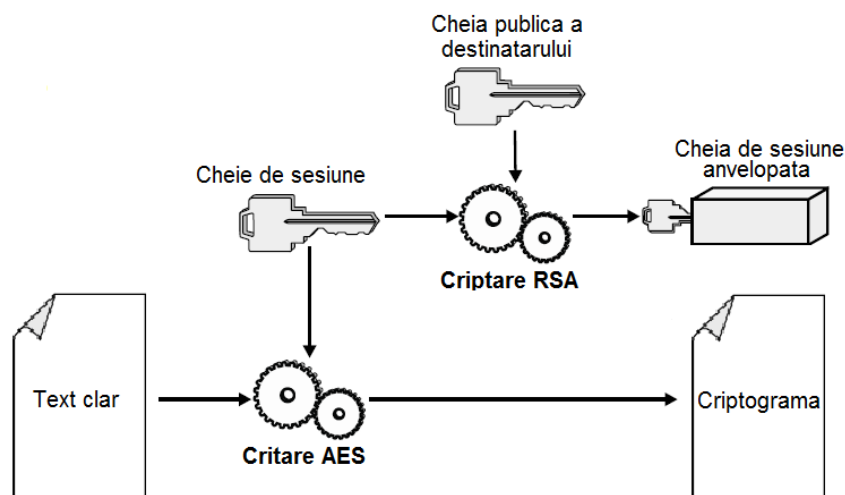


Figura 2-14. Anveloparea cheii de sesiune cu cheia publică a destinatarului

2.3.1. Algoritmul RSA

Algoritmul RSA, denumit după inventatorii acestuia R. Rivest, A. Shamir, și L. Adleman, este în momentul de față cel mai utilizat algoritm criptografic cu chei publice. RSA este un sistem criptografic cu chei publice reversibil în care atât spațiul mesajelor cât și spațiul criptogramelor reprezintă mulțimea $Z_n = \{0, 1, 2, \dots, n-1\}$ unde $n = pq$ este produsul a două numere prime distincte, alese aleator. Fiind reversibil, sistemul se poate utiliza atât pentru criptarea mesajelor cât și pentru semnarea digitală a acestora.

Algoritmul de generare a cheilor

Algoritmul de generare a cheilor RSA de către entitatea A este următorul:

1. Se generează două numere prime mari distincte, p și q , de dimensiuni diferite.
2. Se calculează $n = pq$ și $\phi = (p-1)(q-1)$.
3. Se alege în mod aleator un număr întreg e , $1 < e < \phi$ astfel încât $\text{cmmdc}(e, \phi) = 1$.
4. Folosind algoritmul extins al lui Euclid se determină d , $1 < d < \phi$ astfel încât $ed = 1 \pmod{\phi}$.
5. Cheia publică a lui A este (n, e) iar cheia privată este d .

Valoarea întreagă n poartă denumirea de *modul*, iar e și d reprezintă *exponenți de criptare* respectiv *decriptare*.

Algoritmul de criptare

Pentru a transmite un mesaj criptat lui A, entitatea B trebuie să execute următorii pași:

1. Se obține cheia publică autentică (n, e) a lui A.
2. Se reprezintă mesajul sub forma unei valori întregi m în intervalul $[0, n-1]$.
3. Se calculează criptograma $c = m^e \pmod{n}$.
4. Se transmite lui A criptograma c .

Algoritmul de decriptare

Refacerea mesajului m din criptograma c de către entitatea A , se face astfel:

1. Folosind cheia privată d se calculează $m = c^d \bmod n$.

Se demonstrează relativ ușor că:

$$c^d = (m^e)^d = m^{ed} = m \pmod{n}$$

Securitatea schemei RSA constă în intractabilitatea următoarei probleme: “Fie n un număr întreg, astfel încât $n = pq$, unde p și q sunt două numere prime mari. Să se determine p și q atunci când se cunoaște n ”. Această problemă matematică dificil calculabilă este cunoscută sub denumirea de *problema factorizării numerelor întregi*.

Implementările practice actuale folosesc chei RSA cu lungimi de peste 2048 de biți. De asemenea, există implementări atât software cât și hardware ale algoritmului RSA.

2.3.2. Algoritmul Diffie-Hellman

Algoritmul Diffie-Hellman (D-H) nu poate fi folosit pentru criptarea datelor. În schimb, el permite stabilirea în comun a unei valori secrete între două entități, peste un canal public de comunicație. Valoarea secretă poate fi utilizată apoi de către cele două entități pentru a construi o cheie de sesiune pentru criptarea datelor folosind un algoritm simetric. Algoritmul Diffie-Hellman permite deci negocierea unei chei de sesiune sau schimbul de chei de sesiune între două entități.

Etapele stabilirii în comun a unei chei de sesiune între două entități A și B , folosind algoritmul Diffie-Hellman sunt următoarele:

1. Se alege un număr prim mare p și un generator g al Z_p^* ($2 \leq g \leq p-2$). Numerele p și g sunt publice.
2. A alege în mod aleator o cheie privată x_A , ($1 \leq x_A \leq p-2$) și calculează cheia sa publică ca fiind $y_A = g^{x_A} \bmod p$ pe care o transmite către B .

$$A \rightarrow B: y_A = g^{x_A} \bmod p$$

3. B alege în mod aleator o cheie privată $x_B, (1 \leq x_B \leq p-2)$ și calculează cheia sa publică ca fiind $y_B = g^{x_B} \bmod p$ pe care o transmite către A .

$$B \rightarrow A: y_B = g^{x_B} \bmod p$$

4. B recepționează cheia publică a lui A și calculează cheia de sesiune ca fiind:

$$K_{AB} = y_A^{x_B} \bmod p = (g^{x_A})^{x_B} \bmod p$$

5. A recepționează cheia publică a lui B și calculează cheia de sesiune ca fiind:

$$K_{BA} = y_B^{x_A} \bmod p = (g^{x_B})^{x_A} \bmod p = K_{AB}$$

Securitatea schemei constă în *imposibilitatea calculului logaritmilor discreți într-un câmp finit*. Un atacator care interceptează o cheie publică transmisă între A și B nu va putea niciodată să deducă cheia privată corespunzătoare.

2.4. Funcții hash criptografice

Funcțiile hash criptografice reprezintă primitive de bază în cadrul protocoalelor criptografice actuale fiind folosite pentru asigurarea integrității sau autentificarea originii datelor. Rolul acestora este de a crea un rezumat (*digest*) a unui mesaj sau document electronic.

O *funcție hash* transformă un mesaj de lungime arbitrară într-o valoare de lungime fixă (de exemplu, 160 biți în cazul SHA-1), după cum se poate observa în Figura 2-15.

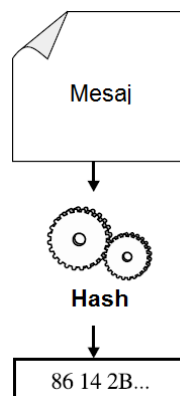


Figura 2-15. Calcularea rezumatului unui mesaj

Pentru a putea fi folosite în aplicații criptografice, funcțiile hash trebuie să respecte o serie de proprietăți:

- să fie ușor de calculat și să poată fi aplicate pe mesaje de orice lungime;
- să nu fie inversabile;
- să fie imposibil din punct de vedere computațional să se poată determina două mesaje diferite astfel încât hash-urile acestora să fie identice.

Funcțiile hash sunt transformări publice și nu necesită folosirea vreunei chei secrete. Atunci când acestea sunt folosite doar pentru asigurarea integrității datelor, cum este cazul schemelor de semnături digitale, ele se mai numesc și *Coduri pentru Detectarea Modificărilor* (MDC – Modification Detection Codes).

Pentru a asigura atât integritatea cât și autenticitatea datelor se folosesc funcții hash în conjuncție cu o cheie secretă. Această clasă de funcții hash poartă denumirea de *Coduri pentru Autentificarea Mesajelor* (MAC – Message Authentication Codes).

MAC-urile permit asigurarea integrității și autenticitatea datelor din moment ce nimeni nu poate genera un MAC valid dacă nu cunoaște cheia secretă. Procesul de generare a unui MAC este prezentat în Figura 2-16. Pentru verificare se recalculează MAC-ul folosind aceeași funcție hash și aceeași cheie secretă și apoi se compară cu MAC-ul original. Dacă cele două MAC-uri sunt identice, mesajul este autentic.

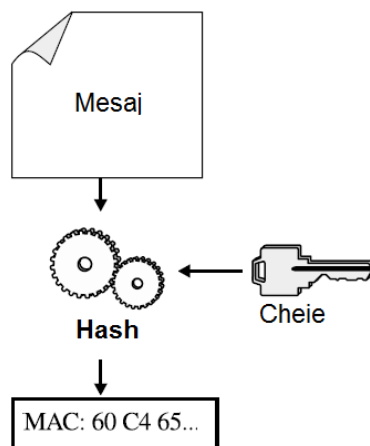


Figura 2-16. Coduri pentru autentificarea mesajelor (MAC)

Principalele funcții hash folosite în practică sunt:

- *MD5*. MD5 a fost proiectat de Ron Rivest în 1991. MD5 generează o valoare hash de 128 biți, fiind folosit în multe protocoale criptografice. În timp, au fost identificate o serie de atacuri fapt pentru care algoritmul nu se recomandă a se mai folosi.
- *SHA-1*. SHA-1 a fost proiectată de către NSA (National Security Agency) din SUA și adoptată ca standard federal în 1995 (FIPS 180-1). SHA-1 generează o valoare hash de 160 biți, fiind mai sigură decât MD5. În timp, au fost propuse o serie de variante îmbunătățite denumite generic SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512), descrise în FIPS 180-4.
- *SHA-3*. SHA-3 reprezintă un nou standard federal (FIPS 202), complementar SHA-1 și SHA-2, și a fost selectat pe baza unui concurs inițial de NIST în 2007 în urma căruia a fost desemnat câștigător algoritmul Keccak proiectat de Guido Bertoni, Joan Daemen, Michael Peeters și Gilles Van Assche.
- *RIPEMD-160*. RIPEMD este o familie de funcții hash criptografice dezvoltate în Leuven, Belgia, de către Hans Dobbertin, Antoon Bosselaers și Bart Preneel. RIPEMD-160 generează o valoare hash de 160 biți.

2.4.1. Algoritmul SHA-1

Algoritmul SHA (Secure Hash Algorithm) a fost dezvoltat de către Institutul Național de Standarde și Tehnologii al SUA (NIST – National Institute of Standards and Technologies) și publicat sub forma unui standard federal (FIPS 180) în 1993. O versiune revizuită a acestui standard (FIPS 180-1) a fost publicată în 1995, versiune cunoscută sub denumirea de SHA-1.

SHA-1 primește la intrare un mesaj de lungime arbitrară și produce la ieșire un rezumat de 160 de biți. Mesajul de intrare M este completat la multiplu de 512 biți, astfel: se adaugă un bit de 1 urmat de mai mulți biți de 0, până se ajunge la 448 biți, iar în ultimii 64 biți se memorează lungimea mesajului inițial. Rezumatul MD , de 160 de biți, văzut ca 5 regiștri A , B , C , D , E de 32 biți, se inițializează cu următoarea constantă MD_0 :

$A = 0x67452301$, $B = 0xEFCDAB89$, $C = 0x98BADCFE$,

$D = 0x10325476$, $E = 0xC3D2E1F0$.

Fiecare bloc M_j de 512 biți al mesajului este supus unei prelucrări în 4 runde a câte 20 de operații fiecare (Figura 2-17).

Funcția neliniară F , care se modifică la fiecare rundă, este definită astfel:

- runda 1 : $F_t(B, C, D) = (B \wedge C) \vee (B' \wedge D)$, pentru $t = [0, 19]$
- runda 2 : $F_t(B, C, D) = B \oplus C \oplus D$, pentru $t = [20, 39]$
- runda 3 : $F_t(B, C, D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$, pentru $t = [40, 59]$
- runda 4 : $F_t(B, C, D) = B \oplus C \oplus D$, pentru $t = [60, 79]$,

unde t reprezintă numărul operației, $t = [0, 79]$.

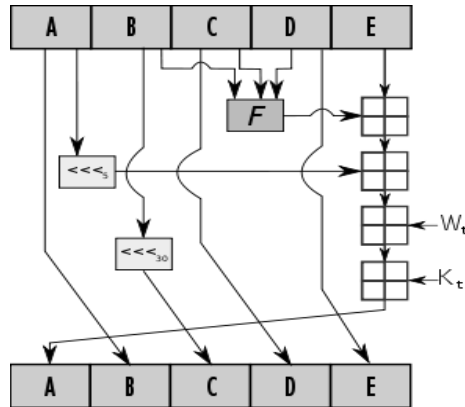


Figura 2-17. Operațiile unei runde SHA

Fiecare bloc M_j , $j = [0, 15]$, de 16 cuvinte de 32 biți este transformat în 80 de sub-blocuri W_j , $j = [0, 79]$, folosind următorul algoritm:

$$W_t = M_t, \text{ pentru } t = [0, 15]$$

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1, \text{ pentru } t = [16, 79].$$

Pseudocodul corespunzător unei runde SHA-1 este următorul:

```

for t = 1 step 1 to 79
    TEMP = (A<<<5) + Ft(B, C, D) + E + Wt + Kt
    E = D
    C = B<<<30
    B = A
    A = TEMP
end for

```

La sfârșitul celor 4 runde de prelucrare a unui bloc de 512 biți, valorile noi ale lui MD_j (A , B , C , D și E) se adună la valorile anterioare, obținute în urma prelucrării blocului MD_{j-1} .

$$MD_j = MD_j + MD_{j-1}$$

Având în vedere limitările specifice SHA-1, se recomandă înlocuirea treptată din implementările practice a acestuia cu versiunea îmbunătățită, SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512).

2.5. Semnături digitale

Conceptul de semnătură digitală a fost introdus de W. Diffie și M. Hellman odată cu prezentarea sistemelor criptografice cu chei publice. Prima schemă de semnături digitale a fost RSA. Cercetările ulterioare au condus la apariția unor scheme noi cu arii de aplicabilitate diferite.

Semnăturile digitale sunt larg folosite în protocoalele de securitate actuale pentru asigurarea integrității, autenticității și non-repudierii datelor.

Schemele de semnături digitale folosesc algoritmi criptografici cu chei publice și funcții hash. Procesul de semnare digitală a unui document folosind algoritmul RSA este prezentat în Figura 2-18. Folosind o funcție hash, se calculează rezumatul documentului. Rezumatul este apoi criptat folosind cheia privată a semnatarului obținându-se astfel semnătura digitală care este atașată documentului original.

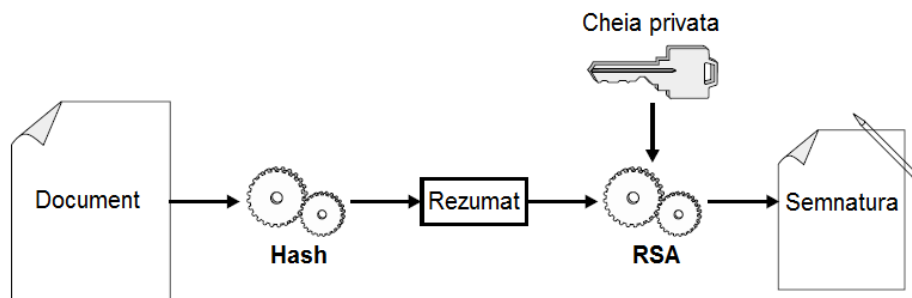


Figura 2-18. Procesul de semnare digitală a unui document

Pentru verificarea semnăturii se folosește aceeași funcție hash și se calculează rezumatul documentului recepționat. Se decriptează apoi semnătura cu cheia publică a semnatarului și se obține rezumatul din momentul semnării documentului. Se compară cele două rezumate. Dacă sunt identice, atunci semnătura este validă ceea ce înseamnă că documentul nu a fost modificat pe

parcurs și că el aparține semnatarului respectiv. Procesul de validare a unei semnături digitale este prezentat în Figura 2-19.

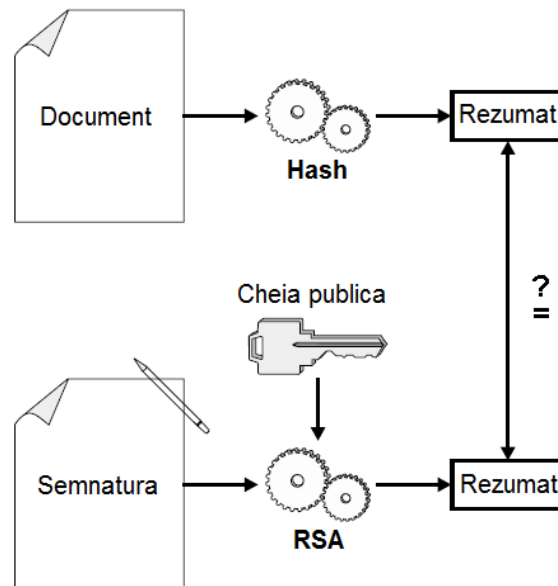


Figura 2-19. Procesul de validare a unei semnături digitale

Cei mai utilizați algoritmi pentru semnarea digitală a documentelor sunt: RSA, ElGamal, DSA (Digital Signature Algorithm) și ECDSA (algoritmul DSA pe curbe eliptice).

2.5.1. Schema de semnătură RSA

RSA a fost prima schemă de semnătură digitală implementată în practică care a rămas și în zilele noastre cea mai populară și versatilă metodă criptografică.

Algoritmul de generare a cheilor

Pentru a putea semna mesaje, fiecare entitate A va trebui să-și genereze o pereche de chei. Algoritmul de generare a cheilor RSA este următorul:

1. Se generează două numere prime mari distincte, p și q , de dimensiuni diferite.
2. Se calculează $n = pq$ și $\phi = (p-1)(q-1)$.
3. Se alege în mod aleator un număr întreg e , $1 < e < \phi$ astfel încât $\text{cmmdc}(e, \phi) = 1$.

4. Folosind algoritmul extins al lui Euclid se determină d , $1 < d < \phi$ astfel încât $ed = 1(\text{mod } \phi)$.
5. Cheia publică a lui A este (n, e) iar cheia privată este d .

Algoritmul de generare a semnăturilor

Pentru a semna un mesaj $m \in M$, entitatea A va efectua următoarele procesări:

1. Se calculează rezumatul mesajului folosind o funcție hash rezistentă la coliziuni $\tilde{m} = h(m)$.
2. Se calculează $s = \tilde{m}^d (\text{mod } n)$.
3. Semnătura lui A asupra mesajului m este s . Pentru verificarea semnăturii, A va trebui să transmită celor interesați m și s .

Algoritmul de verificare a semnăturilor

Orice entitate B poate verifica semnătura lui A astfel:

1. Se obține cheia publică autentică (n, e) a lui A.
2. Se calculează rezumatul mesajului recepționat folosind aceeași funcție hash $\tilde{m}' = h(m)$.
3. Se extrage hash-ul calculat de semnatar $\tilde{m} = s^e (\text{mod } n)$.
4. Dacă $\tilde{m}' \stackrel{?}{=} \tilde{m}$ atunci semnătura este validă, altfel nu.

2.5.2. Schema de semnătură DSA

În 1991, Institutul Național pentru Standarde și Tehnologie al SUA (NIST – National Institute of Standards and Technology) a propus un nou algoritm de semnătură digitală purtând denumirea de DSA (Digital Signature Algorithm).

DSA a devenit standard federal (FIPS 186 – Digital Signature Standard (DSS)) și a reprezentat prima schemă de semnături digitale recunoscută de guvernul SUA. Standardul DSS recomandă folosirea explicită a funcției hash SHA-1 descrisă în FIPS 180-1.

Algoritmul de generare a cheilor

Algoritmul de generare a cheilor DSA este următorul:

1. Se selectează un număr prim mare, p , astfel încât $2^{L-1} < p < 2^L$, unde $512 \leq L \leq 1024$, L fiind multiplu de 64.
2. Se selectează un număr prim q , divizor al lui $(p-1)$, astfel încât $2^{159} < q < 2^{160}$.
3. Se generează $g = \alpha^{(p-1)/q} \bmod p$ unde α este o valoare întreagă $1 < \alpha < p-1$, astfel încât $\alpha^{(p-1)/q} \bmod p > 1$.
4. Se alege în mod aleator un număr întreg x cu $1 \leq x \leq q-1$.
5. Se calculează $y = g^x \bmod p$.
6. Cheia publică a lui A este y iar cheia privată este x . Parametrii (p, q, g) reprezintă parametrii de domeniu și pot fi folosiți în comun de mai multe entități sau pot fi specifici unui singur utilizator.

Algoritmul de generare a semnăturilor

Pentru a semna un mesaj m , entitatea A va efectua următoarele procesări:

1. Se selectează în mod aleator un parametru k , $1 \leq k \leq q-1$.
2. Se calculează $r = (g^k \bmod p) \bmod q$. Dacă $r = 0$, se reia procesul de la pasul 1.
3. Se calculează $s = (k^{-1}(h(m) + xr)) \bmod q$. Dacă $s = 0$, se reia procesul de la pasul 1. $h(m)$ este o funcție hash rezistentă la coliziuni.
4. Semnătura lui A asupra mesajului m este (r, s) . Pentru verificarea semnăturii, A va trebui să transmită celor interesați și mesajul m .

Dimensiunea semnăturilor DSA nu este dependentă de lungimea cheii sau a mesajului, ea fiind formată din două părți de 160 biți fiecare.

Algoritmul de verificare a semnăturilor

Orice entitate B poate verifica semnătura lui A astfel:

1. Se obține cheia publică autentică y și parametrii de domeniu (p, q, g) ai lui A .
2. Se verifică dacă valorile r și s recepționate sunt în intervalul $[1, q-1]$.
3. Se calculează $w = s^{-1} \bmod q$ și $h(m)$.
4. Se calculează $u_1 = (h(m)w) \bmod q$ și $u_2 = (rw) \bmod q$.
5. Se calculează $v = (((g)^{u_1} (y)^{u_2}) \bmod p) \bmod q$.
6. Dacă $v \stackrel{?}{=} r$ atunci semnătura este validă, altfel nu.

Deoarece r și s sunt valori întregi mai mici decât q , dimensiunea unei semnături DSA este de cel mult 320 biți.

Securitatea schemei DSA contă în intractabilitatea următoarelor probleme: “Fie p un număr prim mare, g un întreg $1 \leq g \leq p-1$ și $y = g^x \bmod p$. Să se determine x atunci când se cunosc p , g și y ”. Această problemă matematică dificil calculabilă este cunoscută sub denumirea de *problema logaritmilor discreți* (DLP – Discrete Logarithm Problem).

2.6. Criptografia bazată pe curbe eliptice

Criptografia bazată pe curbe eliptice (ECC – Elliptic Curve Cryptography) a fost introdusă de V. Miller și N. Koblitz în 1985. Aceștia au propus, în mod independent, un sistem criptografic cu chei publice analog schemei ElGamal în care mulțimea numerelor întregi Z_p^* (unde p este un număr prim) a fost înlocuită cu mulțimea punctelor de pe o curbă eliptică definită peste un câmp finit.

Figura 2-20 prezintă operația de adunare a două puncte de pe o curbă eliptică. Aceasta este echivalentă cu înmulțirea a două numere în Z_p^* . Multiplicarea unui punct cu o constantă, care înseamnă adunare succesivă, este echivalentă cu exponențierea din Z_p^* .

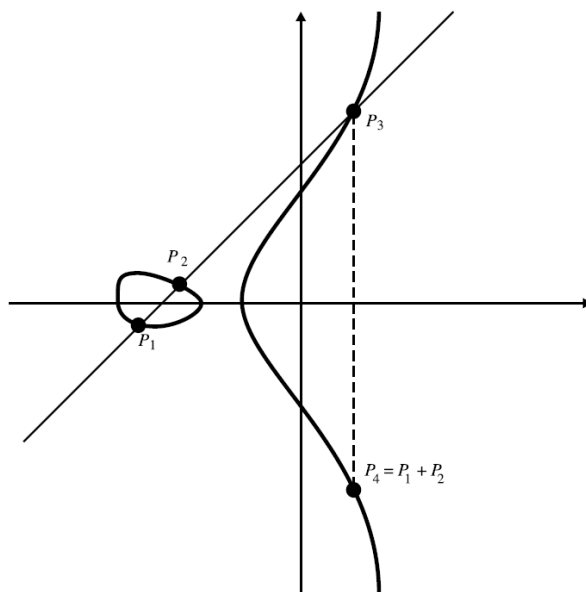


Figura 2-20. Operația de adunare a două puncte de pe o curbă eliptică

Cel mai important avantaj al criptografiei bazată pe curbe eliptice față de sistemele criptografice clasice este dimensiunea mai mică a parametrilor și cheilor în comparație cu RSA sau DSA, pentru același nivel de securitate. Tabelul 2-2 face o comparație între dimensiunea parametrilor și a cheilor specifice sistemelor criptografice RSA, DSA și ECC.

	Parametrii sistem (biți)	Cheie publică (biți)	Cheie privată (biți)
RSA	-	1088	2048
DSA	2208	1024	160
ECC	481	161	160

Tabelul 2-2. Dimensiunea parametrilor și a cheilor

Datorită dimensiunii reduse a parametrilor și cheilor, timpul de execuție, spațiul de memorare, banda de rețea și puterea consumată scad semnificativ în cazul sistemelor criptografice bazate pe curbe eliptice. Cele mai performante implementări ale acestor sisteme au demonstrat faptul că generarea de semnături electronice este cu un ordin de mărime mai rapidă în cazul ECC față de RSA sau DSA. Timpii de verificare a semnăturilor sunt însă comparativi ca valoare. Aceste avantaje fac ca sistemele criptografice bazate pe curbe eliptice să poată fi folosite cu succes în implementarea de soluții de securitate pentru dispozitive cu puteri de procesare scăzute și spațiu de memorare limitat, cum ar fi smart card-urile, PDA-urile sau telefoanele mobile.

2.7. Recomandări privind lungimile de chei

Pentru asigurarea confidențialității datelor se recomandă folosirea algoritmilor criptografici simetrici. Aceștia au performanțe ridicate din punct de vedere al timpului de procesare permițând criptarea rapidă a mesajelor de diferite dimensiuni. Securitatea sistemelor criptografice simetrice depinde în mare măsură de modul de gestiune al cheilor folosite pentru criptarea datelor.

Sistemele criptografice cu chei publice sunt semnificativ mai lente decât cele cu chei secrete. Criptografia cu chei publice este folosită în practică pentru transportul (distribuția) cheilor de criptare simetrice sau pentru criptarea unor mesaje reduse ca dimensiune.

Un alt aspect foarte important în domeniul criptografiei îl reprezintă modul de selectare al lungimii cheilor de criptare. În general, lungimea cheilor trebuie să țină cont de o serie de factori precum: valoarea datelor protejate, puterea de calcul a adversarului sau intervalul de timp pentru care se dorește asigurarea confidențialității datelor.

Tabelul 2-3 prezintă recomandările RSA Laboratories privind lungimea minimă a cheilor criptografice necesare pentru asigurarea confidențialității datelor pe parcursul unui orizont de timp dat, atât în cazul algoritmilor simetrici cât și pentru algoritmul RSA. Aceste recomandări sunt similare celor făcute de NIST în Ianuarie 2003 (NIST SP 800-57).

Perioada de protecție a datelor	Până în 2010	Până în 2030	După 2030
Lungime minimă chei – algoritmi simetrici	80 biți	112 biți	128 biți
Lungime minimă chei – algoritmul RSA	1024 biți	2048 biți	3072 biți

Tabelul 2-3. Recomandări privind lungimea minimă a cheilor

2.8. Infrastructuri de chei publice

Criptografia cu chei publice constituie fundamentul serviciilor de securitate pentru documentele electronice. Ea funcționează pe baza principiului conform căruia fiecare entitate are acces la cheile publice ale celorlalte entități. De exemplu, validarea unei semnături digitale se face folosind cheia publică a semnatarului. Pentru a cripta date, este necesară, de asemenea, cheia publică a destinatarului.

Distribuirea pe scară largă a cheilor publice necesită însă existența unui mecanism care să garanteze integritatea acestora și faptul că o cheie publică aparține unei anumite entități. În caz contrar, sistemul devine vulnerabil la atacuri prin impersonare.

Infrastructurile de Chei Publice (PKI – Public Key Infrastructure) permit entităților să obțină chei publice autentice sub formă de certificate digitale. Un certificat digital reprezintă o legătură imposibil de falsificat între o cheie publică și un anumit atribut al posesorului acesteia.

În realitate, un certificat digital este o structură de date ce conține în principal valoarea cheii publice a unei entități și o serie de informații ce identifică în mod unic entitatea respectivă. Această structură de date este semnată digital de către un terț de încredere denumit Autoritate de Certificare (CA – Certification Authority) care confirmă astfel validitatea datelor înscrise în certificat.

O Infrastructură de Chei Publice reprezintă totalitatea echipamentelor hardware, aplicațiilor software, personalului, politicilor și procedurilor necesare pentru a genera, stoca, distribui și revoca certificatele digitale asociate cheilor publice.

Autoritățile de Certificare reprezintă elementele de bază ale Infrastructurilor de Chei Publice, având rolul de a emite și revoca certificatele digitale. O Infrastructură de Chei Publice este alcătuită din una sau mai multe Autorități de Certificare conectate între ele prin căi de încredere. Autoritățile de Certificare pot fi interconectate folosind topologii diferite, asigurând astfel un grad ridicat de flexibilitate și scalabilitate.

O Autoritate de Certificare este cunoscută după nume și cheia sa publică. Numele autorității este inclus în toate certificatele și CRL-urile pe care aceasta le generează iar cheia sa publică este folosită pentru validarea semnăturilor. Odată ce un utilizator acceptă o Autoritate de Certificare ca fiind de încredere (direct sau prin intermediul unei căi de certificare) el poate determina nivelul de încredere al tuturor certificatelor emise de către autoritatea respectivă.

Acest mecanism de obținere a cheilor publice este simplu și economic de implementat, certificatele digitale putând fi distribuite fără a necesita protecție prin serviciile de securitate suplimentare deoarece:

1. Certificatele digitale conțin informații publice (numele entității, cheia publică a acesteia, etc) și prin urmare nu se pune problema asigurării confidențialității acestora. Ele pot fi distribuite prin legături de comunicație nesigure precum: servere de fișiere, servere de directoare sau protocoale de comunicație fără mecanisme de securitate.

2. Certificatele digitale sunt semnate digital de Autoritatea de Certificate asigurându-se astfel atât integritatea cât și autenticitatea acestora.

Trebuie subliniat faptul că la baza acestui mecanism se află încrederea pe care entitățile o acordă Autorităților de Certificare în a emite certificate digitale valide.

2.8.1. Certificate digitale

Noțiunea de certificat de cheie publică a fost introdusă pentru prima oară de către Kohnfelder în 1978. Acesta a propus folosirea unor structuri de date semnate digital (certificate digitale) pentru distribuirea cheilor publice. Ulterior, certificatele digitale au fost folosite pentru implementarea unor mecanisme de control al accesului la servere de directoare X.500, mecanisme bazate pe semnături digitale.

Formatul certificatelor digitale a fost definit în cadrul Recomandărilor X.509 de către ITU-T și a evoluat de-a lungul a trei versiuni. Versiunea 1, adoptată în 1988, se caracterizează printr-o flexibilitate scăzută datorită faptului că formatul certificatelor nu putea fi extins pentru a suporta atribute suplimentare. Versiunea 2, adoptată în 1993, a adus doar îmbunătățiri minore primei versiunii prin introducerea a două câmpuri opționale. Deoarece utilitatea acestor câmpuri a fost și continuă să fie neglijabilă, versiunea 2 a certificatelor de chei publice nu a fost acceptată pe scară largă. Pentru a corecta deficiențele versiunilor 1 și 2, ITU-T a elaborat versiunea 3 a certificatelor de chei publice în cadrul Recomandării X.509 din 1997. Versiunea 3 introduce noțiunea de extensii opționale, mecanism prin intermediul căruia formatul certificatelor digitale poate fi extins în mod standard și generic pentru a include atribute suplimentare. De remarcat faptul că odată cu trecerea la o nouă versiune s-a încercat păstrarea compatibilității cu versiunea anterioară. Altfel spus, versiunea 1 este un subset al versiunii 2 iar versiunea 2 este un subset al versiunii 3.

Ultima actualizare a formatului certificatelor de chei publice a fost făcută odată cu elaborarea Recomandării X.509 din 2012. Versiunea din 2012 aduce o serie de modificări la versiunea anterioară, în special prin adăugarea unor extensii noi.

În continuare este prezentată structura și semantica câmpurilor unui certificat digital conform Recomandării X.509 din 2000. De remarcat faptul că deși X.509 definește o serie de cerințe privind modul de folosire al câmpurilor și extensiilor standard dintr-un certificat, acestea nu sunt suficiente pentru a rezolva problemele de interoperabilitate. Pentru aceasta este necesar a se defini

profile specifice de certificate care să specifice în mod clar modul de folosire al câmpurilor și extensiilor dintr-un certificat pentru un domeniu PKI.

Grupul de Lucru pentru Infrastructuri de Chei Publice X.509 (PKIX) din cadrul Internet Engineering Task Force (IETF) a elaborat un astfel de profil în Ianuarie 1999 prin publicarea RFC 2459. În Aprilie 2002, RFC 2459 a fost înlocuit cu RFC 3280. Profilul definit în RFC 3280 se adresează comunității Internet însă o serie de recomandări ale acestuia pot fi folosite în egală măsură și în cadrul PKI-urilor organizaționale. Figura 2-21 ilustrează structura certificatelor digitale X.509 v3.

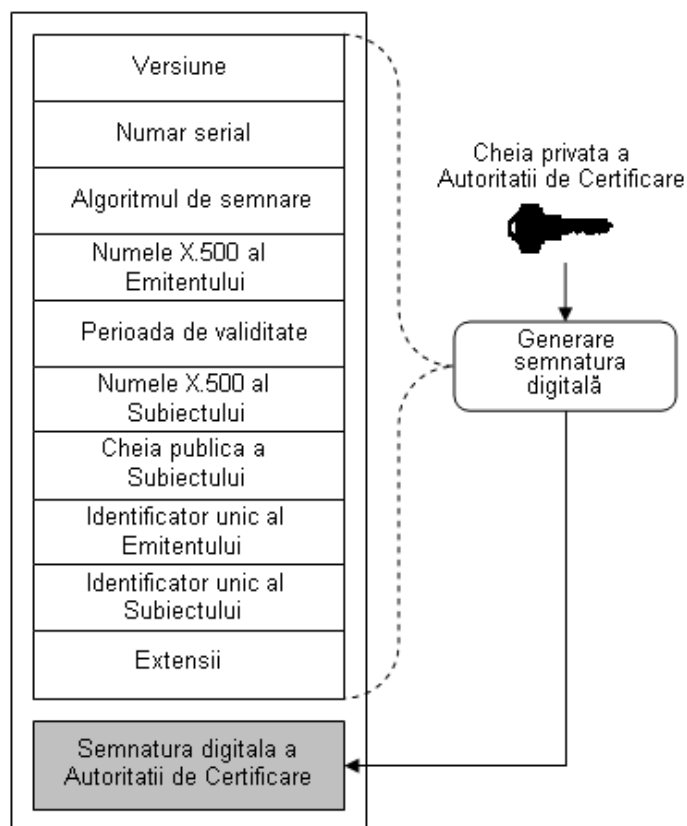


Figura 2-21. Formatul certificatelor digitale X.509 v3

Semnificația câmpurilor din figură este următoarea:

- *Versiunea (Version)* indică versiunea formatului de certificat (1, 2 sau 3)
- *Numărul serial (Serial Number)* reprezintă un identificator numeric unic al certificatului relativ la mulțimea tuturor certificatelor emise de o Autoritate de Certificare. Atunci când certificatul este revocat, numărul

său serial este trecut în lista de certificate revocate (CRL – Certificate Revocation List) emisă de Autoritatea de Certificare respectivă. Din acest motiv numărul serial al fiecărui certificat emis de o Autoritate de Certificare trebuie să fie unic.

- *Algoritmul de semnare (Signature Algorithm)* identifică algoritmul folosit de Autoritatea de Certificare pentru a semna digital certificatul. Identificarea algoritmului se face folosind identificatori de obiect (OID – Object Identifier). Un OID este o reprezentare unică a unui obiect sub forma unei secvențe de numere întregi separate prin punct. OID-urile sunt organizate ierarhic și trebuie înregistrare la autoritățile de înregistrare internaționale, naționale sau organizaționale. De exemplu, OID-ul pentru cazul în care semnătura digitală este realizată folosind funcția hash SHA-1 și algoritmul RSA este 1.2.840.113549.1.1.5.
- *Numele X.500 al Emitentului (Issuer X.500 Name)* reprezintă numele distinct (DN – Distinguished Name) al Autorității de Certificare care a emis certificatul. Un DN este o convenție de denumire ierarhică a entităților, definită în Recomandările X.500, cu scopul de a asigura unicitatea numelor. DN-urile sunt exprimate ca o concatenare de nume distincte relative (RDN – Relative Distinguished Name) pornind de la nivelul cel mai înalt până la cel mai de jos. De exemplu, “C=RO, O=Military Technical Academy, OU=Computer Science Department, CN=Ion Bica” reprezintă un DN.
- *Perioada de validitate (Validity Period)* specifică intervalul de timp pe parcursul căruia certificatul poate fi considerat valid, dacă nu cumva a fost revocat între timp.
- *Numele X.500 al Subiectului (Subject X.500 Name)* reprezintă numele distinct (DN – Distinguished Name) al proprietarului certificatului.
- *Cheia publică a Subiectului (Subject Public Key Info)* conține valoarea cheii publice a subiectului precum și identificatorul algoritmului cu care aceasta poate fi folosită.
- *Identificatorul unic al Emitentului (Issuer Unique Identifier)* este un identificator unic al Autorității de Certificare care a emis certificatul. Acest câmp este opțional și a fost introdus odată cu versiunea 2 a certificatelor X.509 pentru a putea face distincție între cazurile în care același nume X.500 a fost atribuit mai multor Autorități de Certificare de-a lungul timpului. El este însă rar întâlnit în implementările practice iar RFC 3280 nu recomandă folosirea lui.

- *Identificatorul unic al Subiectului (Subject Unique Identifier)* este un identificator unic al proprietarului certificatului. Acest câmp este opțional și a fost introdus odată cu versiunea 2 a certificatelor X.509 pentru a putea face distincție între cazurile în care același nume X.500 a fost atribuit mai multor entități de-a lungul timpului. El este însă rar întâlnit în implementările practice iar RFC 3280 nu recomandă folosirea lui.

Câmpul de extensii este opțional și a fost introdus odată cu versiunea 3 a certificatelor X.509. El permite extinderea certificatelor cu atribute suplimentare, într-un mod standard și generic. Fiecare extensie constă din trei câmpuri: tipul extensiei, un flag care indică dacă extensia este critică sau nu și valoarea extensiei.

Câmpul *tip extensie* definește tipul de date din câmpul ce conține *valoarea extensiei*. Tipul poate reprezenta, de exemplu, un șir de caractere, o valoare numerică, o dată calendaristică sau o structură complexă de date. Pentru a asigura interoperabilitatea între domenii PKI diferite, toate tipurile de extensii trebuie înregistrate la o organizație de standardizare recunoscută internațional.

Dacă o extensie a fost marcată ca fiind critică aceasta indică faptul că ea conține o informație importantă pe care o aplicație nu trebuie să o ignore. Dacă aplicația nu poate interpreta extensia, atunci ea trebuie să nu accepte certificatul. O extensie marcată ca fiind necritică se recomandă a fi procesată dacă este posibil însă poate fi ignorată dacă aplicația nu o poate interpreta.

Trebuie subliniat faptul că există o diferență majoră între o extensie critică și o informație dintr-un certificat necesară unei anumite aplicații. O aplicație specifică poate cere ca o extensie să fie disponibilă în orice certificat pe care îl procesează. Aceasta nu înseamnă însă că extensia trebuie să fie marcată ca fiind critică. Extensiile critice se folosesc doar pentru informații importante ce trebuie interpretate de către toate aplicațiile (de exemplu, informație critică necesară pentru a preveni utilizarea greșită sau nesigură a unui certificat). Din această cauză, majoritatea extensiilor sunt necritice. Extensiile critice trebuie incluse într-un certificat numai după o analiză temeinică datorită problemelor de interoperabilitate ce pot apărea la nivelul aplicațiilor PKI.

Extensiile standard, definite în cadrul Recomandării X.509 din 2000, sunt următoarele:

- *Identificatorul cheii autorității (Authority Key Identifier)*
- *Identificatorul cheii subiectului (Subject Key Identifier)*
- *Modul de utilizare al cheii (Key Usage)*

- *Modul extins de utilizare a cheii (Extended Key Usage)*
- *Punctul de distribuție a CRL (CRL Distribution Point)*
- *Perioada de utilizare a cheii private (Private Key Usage Period)*
- *Politicile de certificare (Certificate Policies)*
- *Mapări de politici (Policy Mappings)*
- *Nume alternativ al Subiectului (Subject Alternative Name)*
- *Nume alternativ al Emitentului (Issuer Alternative Name)*
- *Atribute de director ale Subiectului (Subject Directory Attributes)*
- *Constrângeri de bază (Basic Constraints)*
- *Constrângeri de nume (Name Constraints)*
- *Constrângeri de politică (Policy Constraints)*
- *Anularea oricărei politici (Inhibit Any Policy)*
- *Pointer către cel mai recent CRL (Freshest CRL Pointer)*

Pe lângă extensiile standard, se pot defini o serie de extensii private pentru a fi folosite în cadrul unui domeniu PKI. De exemplu, RFC 3280 definește două extensii private pentru a fi folosite în cadrul comunității Internet:

- *Modul de acces la informațiile despre Autoritate (Authority Information Access)*
- *Modul de acces la informațiile despre Subiect (Subject Information Access)*

Orice modificare a informațiilor din câmpurile unui certificat digital, survenită înainte de data expirării acestuia, atrage după sine revocarea certificatului și emiterea unui nou certificat. De aceea, atributele ce urmează a fi incluse într-un certificat trebuie analizate cu atenție pentru a evita revocarea înainte de termen a certificatelor. Atributele asociate unei entități care se modifică des (de exemplu, drepturile de acces) nu se recomandă a fi incluse în cadru unui certificat de chei publice.

2.8.2. Componente și funcțiile PKI

Principalele componente ale unei Infrastructurii de Chei Publice sunt:

- *Autoritățile de Certificare* (CA – Certification Authority) reprezintă componentele de bază ale unei Infrastructurii de Chei Publice și au rolul de a emite și revoca certificate digitale;
- *Autoritățile de Înregistrare* (RA – Registration Authority) au rolul de a valida cererile de emitere certificate și identitatea entităților finale;
- *Depozite de certificate* (Repository) pentru stocarea și distribuirea certificatelor și a listelor de certificate revocate (CRL – Certificate Revocation List);
- *Entitățile Finale* (End Entity) reprezintă un termen generic folosit pentru a desemna utilizatori, dispozitive sau aplicații software care folosesc certificatele digitale pentru implementarea de servicii de securitate.

Componentele PKI și relațiile dintre acestea sunt prezentate în Figura 2-22, conform modelului arhitectural propus de *IETF PKIX Working Group*.

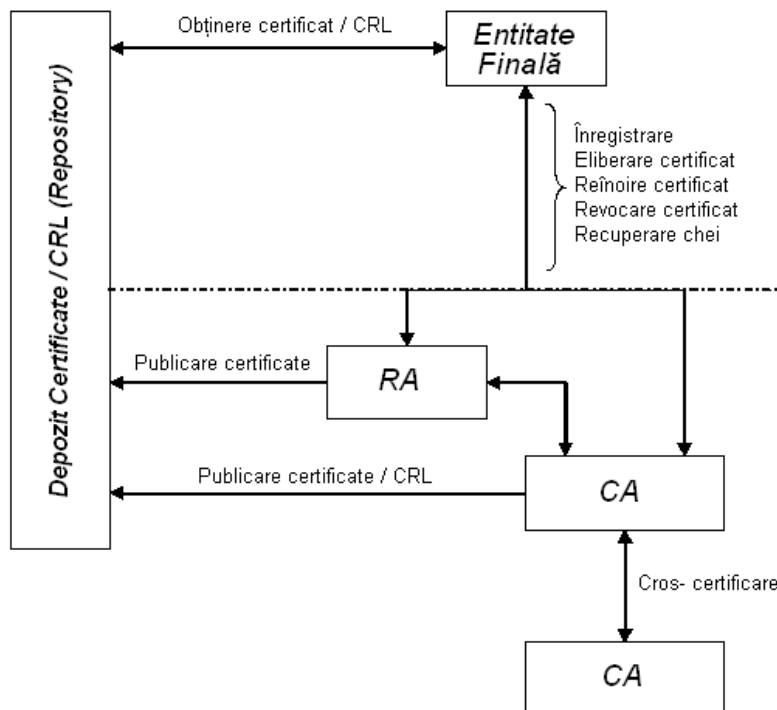


Figura 2-22. Modelul arhitectural PKIX

Autoritatea de Certificare emite certificate digitale garantând prin intermediul acestora că o cheie publică aparține unei anumite entități. După emiterea certificatelor digitale, Autoritatea de Certificare este responsabilă pentru actualizarea stării acestora, emiterea de CRL-uri, publicarea certificatelor și a CRL-urilor și păstrarea unei arhive cu informații despre certificatele emise și operațiile efectuate asupra acestora.

O Autoritate de Certificare poate emite certificate digitale atât pentru entitățile finale cât și pentru alte Autorități de Certificare. Pentru a garanta autenticitatea informațiilor înscrise într-un certificat digital, Autoritatea de Certificare trebuie mai întâi să verifice că acestea corespund/apartin entității solicitante. Procesul de verificare poate fi realizat direct de către Autoritatea de Certificare sau aceasta poate delega una sau mai multe Autorități de Înregistrare să se ocupe de acest lucru.

Atunci când o Autoritate de Certificare emite un certificat digital unei alte Autorități de Certificare, se creează o relație de cross-certificare între cele două autorități prin care prima autoritate garantează faptul că certificatele emise de cea de-a doua autoritate pot fi, de asemenea, considerate de încredere.

Autoritatea de Certificare trebuie să mențină și să publice periodic o listă a certificatelor digitale revocate (CRL). Certificatele pot fi revocate înainte de termenul de expirare dacă, de exemplu, cheia privată asociată cheii publice din certificat a fost compromisă sau pierdută, utilizatorul părăsește organizația sau informațiile înscrise în certificat (numele utilizatorului, drepturile de acces, etc) nu mai sunt valide. CRL-urile sunt de obicei semnate de aceeași autoritate care a emis certificatele însă există posibilitatea creării unei autorități separate care să se ocupe numai de acest lucru. Pe lângă CRL-uri există posibilitatea folosirii și altor mecanisme pentru determinarea stării certificatelor, cum ar fi OCSP.

Pentru rezolvarea unor dispute ce pot apărea ulterior, Autoritatea de Certificare trebuie să păstreze o arhivă cu toate certificatele și CRL-urile emise pentru determinarea cu exactitate a validității unui certificat digital la un moment dat de timp. Aceste informații sunt foarte utile, ele permițând, de exemplu, determinarea autenticității unei semnături digitale de pe un document mai vechi. Într-adevăr, o semnătură datată poate fi considerată autentică dacă data la care a fost creată se încadrează în perioada de validitate a certificatului și CRL-ul din acel moment nu conținea respectivul certificat.

Certificatele digitale și CRL-urile sunt semnate cu cheia privată a Autorității de Certificare. Prin urmare, este foarte important ca această cheie să fie protejată corespunzător prin folosirea de module criptografice hardware (HSM – Hardware Security Module).

Autoritatea de Înregistrare este o componentă opțională a unei Infrastructuri de Chei Publice care are rolul de a degreva Autoritatea de Certificare de task-uri administrative precum înregistrarea entităților finale în vederea emiterii de certificate digitale.

Rolul principal al unei Autorități de Înregistrare este de a valida cererile de emitere certificate și identitatea entităților finale. Pe lângă aceasta, o Autoritate de Înregistrare mai poate îndeplini și următoarele funcții:

- Autentificarea entităților care solicită certificate digitale
- Validarea informațiilor furnizate de entitățile finale
- Validarea drepturilor entităților finale de a obține certificate de un anumit tip
- Verificarea faptului că entitatea finală deține într-adevăr cheia privată asociată cheii publice pentru care a solicitat un certificat digital. Acest proces este referit în literatura de specialitate ca “dovada deținerii cheii private” (POP – Proof of Possession)
- Anunțarea Autorității de Certificare în cazurile în care este necesară revocarea certificatului digital al unei entități finale
- Generarea perechii cheie privată/cheie publică pentru entitățile finale
- Generarea de parole de acces pentru entitățile finale, necesare pentru fazele de înregistrare și eliberare certificate digitale de la Autoritatea de Certificare
- Inițierea procesului de înregistrare la Autoritatea de Certificare pentru eliberarea de certificate digitale
- Arhivarea cheilor private aparținând entităților finale
- Inițierea procesului de recuperare a cheilor private
- Stocarea cheilor private pe token-uri criptografice (smart card-uri) și distribuirea acestora către entitățile finale

Fiecare Autoritate de Certificare trebuie să mențină o listă cu Autoritățile de Înregistrare acreditate. Mesajele transmise de Autoritățile de Înregistrare către Autoritatea de Certificare trebuie semnate digital pentru a garanta integritatea și autenticitatea acestora. Protecția cheii private de semnătură a Autorității de Înregistrare este foarte importantă și se recomandă pentru aceasta, folosirea de module criptografice hardware (HSM – Hardware Security Module).

Instalarea de Autorități de Înregistrare este recomandată în situațiile în care există un număr mare de entități finale, distribuite geografic sau din motive de securitate prin limitarea accesului direct al entităților finale la Autoritatea de Certificare.

Depozitul de Certificate este mecanismul folosit pentru publicarea certificatelor și a CRL-urilor. Inițial, standardul X.509 elaborat de ITU prevedea folosirea în acest sens a serverelor de directoare X.500 și protocolului DAP (Directory Access Protocol) pentru interogarea acestora de către clienți. Protocolul LDAP (Lightweight Directory Access Protocol) dezvoltat ulterior reprezintă o versiune simplificată a DAP. Standardele PKIX permit însă publicarea certificatelor și a CRL-urilor folosind, pe lângă LDAP, mecanisme precum HTTP sau FTP.

Entitățile Finale reprezintă persoane, aplicații sau echipamente care folosesc certificatele digitale pentru implementarea de servicii de securitate. De remarcat faptul că o cheie publică și prin urmare și certificatul digital asociat, are semnificație numai în conjuncție cu cheia privată asociată. Securitatea sistemelor bazate pe algoritmi criptografici cu chei publice este în mare măsură dependentă de modul în care sunt protejate cheile private. Pentru aceasta, posesorii de certificate digitale trebuie să asigure o protecție corespunzătoare cheilor private. Se pot folosi în acest sens cartele inteligente (smart card) ce permit generarea perechilor de chei, stocarea în siguranță a cheilor private și efectuarea de operații criptografice intern astfel încât cheile private să nu părăsească niciodată cardul.

2.8.3. Validarea certificatelor digitale

Revocarea certificatelor digitale reprezintă acțiunea prin care un certificat este declarat invalid înainte de expirarea termenului de valabilitate asociat. Un certificat digital trebuie revocat în situații precum: compromiterea cheii private, schimbarea numelui subiectului sau schimbarea relației dintre subiect și Autoritatea de Certificare. Prin urmare, este necesar ca în cadrul oricărei Infrastructuri de Chei Publice să se implementeze metode care să permită revocarea certificatelor în caz de necesitate și notificarea entităților interesate despre această schimbare a stării certificatului.

Există mai multe abordări posibile pentru revocarea certificatelor digitale dintre care cele mai întâlnite în practică sunt: listele de certificate revocate (CRL – Certificate Revocation List) și protocolul OCSP (OCSP – Online Certificate Status Protocol).

Liste de certificate revocate

Cea mai folosită metodă de revocare a certificatelor digitale se bazează pe publicarea periodică a unei liste de certificate revocate (CRL – Certificate Revocation List), semnată digital de Autoritatea de Certificare. Formatul CRL-urilor este definit în cadrul standardului X.509 iar versiunea curentă este 2. Acest format este prezentat în Figura 2-23.

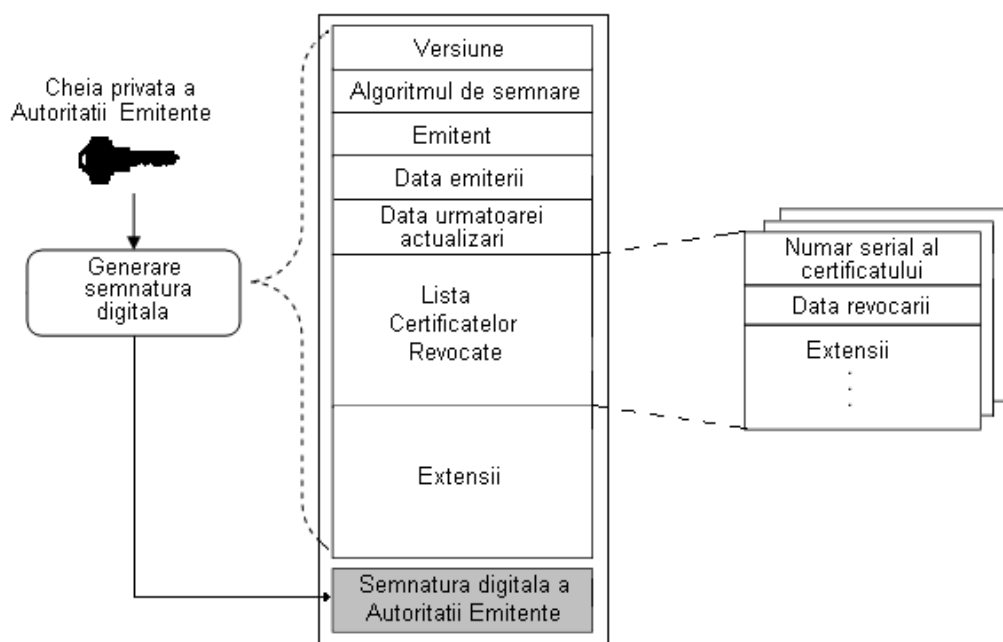


Figura 2-23. Structura unui CRL

CRL-urile oferă informații detaliate despre certificatele revocate, cum ar fi momentul revocării sau motivul revocării. În plus CRL-urile pot fi ușor extinse pentru a include, pe lângă extensiile standard, extensii specifice unui domeniu sau produs software. Extensiile pot fi folosite atât la nivelul fiecărui certificat revocat cât și la nivelul întregului CRL.

Autoritatea de Certificare publică periodic câte un CRL în repository. CRL-ul conține toate certificatele neexpirate, emise de Autoritatea de Certificare, care au fost revocate. Conform standardului X.509, fiecare CRL conține un câmp *nextUpdate* care specifică momentul în care va fi emis următorul CRL. Pentru a valida un certificat digital, entitățile finale obțin din repository și stochează în cache-ul local cel mai recent CRL. Odată obținut cel mai recent CRL, entitățile finale nu vor mai face alte cereri la repository ci vor folosi CRL-ul stocat în cache-ul local până când acesta va expira. Prin urmare, putem presupune că fiecare entitate finală va face o singură cerere la repository pentru un CRL.

Exactitatea schemei de revocare bazată pe CRL-uri este dependentă de lungimea perioadei de actualizare a CRL-urilor. Dacă un certificat digital este revocat în intervalul de timp dintre două actualizări, atunci entitățile finale vor afla acest lucru abia în momentul publicării următorului CRL. În cel mai rău caz, intervalul de timp maxim dintre revocarea efectivă a unui certificat și notificarea tuturor entităților din sistem poate fi egal cu perioada de actualizare a CRL-urilor. Scăderea perioadei de actualizare a CRL-urilor poate duce la creșterea exactității schemei dar dă naștere la alte probleme. Cu cât perioada de actualizare a CRL-urilor este mai scurtă, cu atât rata cererilor va fi mai mare generând o creștere a încărcării rețelei. Prin urmare va trebui găsit un optim între perioada de actualizare a CRL-urilor și rata de revocare a certificatelor pentru o dimensiune dată a domeniului PKI. Din acest motiv, schemele de revocare bazate pe CRL-uri nu pot fi folosite în aplicații care necesită determinarea în timp real a stării de revocare a certificatelor digitale.

Schemele bazate pe CRL-uri au fost criticate datorită costurilor mari asociate și a faptului că nu permit determinarea cu exactitate a stării de revocare a unui certificat digital. Cu toate acestea, schemele bazate pe CRL-uri sunt standardizate și folosite pe scară largă în implementările actuale de PKI. Pentru eficientizarea acestor scheme au fost propuse o serie de îmbunătățiri cum ar fi segmentarea CRL-urilor sau delta CRL-urile ce permit reducerea dimensiunii listelor și implicit a încărcării rețelei precum și creșterea exactității schemei.

Protocolul OCSP

Pentru a rezolva problema determinării cu exactitate a stării de revocare a certificatelor digitale, specifică schemelor bazate pe CRL-uri, au fost proiectate o serie de scheme bazate pe servere on-line și protocoale de interogare a acestora care să permită determinarea în timp real a stării certificatelor.

Protocolul de validare on-line a stării certificatelor (OCSP – Online Certificate Status Protocol) reprezintă un protocol simplu de tip cerere/răspuns destinat exclusiv determinării stării de revocare a certificatelor. Clienții OCSP trimit o cerere conținând identifiantii certificatelor de validat către serverul OCSP iar acesta furnizează un răspuns privind starea fiecărui certificat.

Răspunsurile furnizate de serverele OCSP sunt întotdeauna semnate digital pentru a se asigura integritatea și autenticitatea informațiilor conținute. Semnarea cererilor clienților este opțională și se folosește pentru autentificarea acestora la serverul OCSP.

Serverele OCSP pot fi parte integrantă a unei Autorității de Certificare sau independente (terț de încredere care oferă servicii de validare a certificatelor).

Mecanismul folosit de serverul OCSP pentru obținerea informației de revocare poate fi: prin interogarea directă a bazei de date a Autorității de Certificare, prin procesarea CRL-urilor emise de Autoritatea de Certificare sau prin apelarea serviciilor unui alt server OCSP, autoritar pentru domeniul PKI în care a fost emis certificatul respectiv.

OCSP suportă un număr arbitrar de extensii în cadrul mesajelor de cerere și răspuns. Toate extensiile specifice CRL-urilor sunt permise în OCSP. Acest lucru conferă protocolului flexibilitate ridicată, permițând extinderea acestuia în diverse scopuri.

Principalul dezavantaj al OCSP îl constituie faptul că fiecare răspuns trebuie semnat digital ceea ce necesită timp de procesare și bandă de rețea suplimentară. Pentru a optimiza acest proces, serverele OCSP pot calcula în avans răspunsuri și atașa o perioadă de valabilitate fiecărui răspuns. În acest mod nu mai este necesar să se repete procesul de construire și semnare a răspunsului pentru fiecare cerere de validare. Această optimizare poate duce la creșterea performanțelor sistemului dar schema își pierde proprietatea de determinare în timp real a stării certificatelor.

3. Metode și protocoale de autentificare

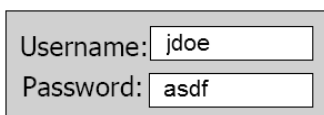
Autentificarea entităților joacă un rol foarte important pentru controlul accesului la un sistem informatic sau pentru implementarea unui protocol de securitate. În cazul unui sistem informatic, accesul la resurse sau date trebuie permis numai utilizatorilor legitimi.

Majoritatea protocoalelor de securitate, după cum o să vedem, presupun negociere între entități, ori această negociere nu are sens decât dacă entitățile se autentifică în prealabil.

Autentificarea are ca scop determinarea univocă a identității unei entități. Pentru asta, entitatea respectivă trebuie să demonstreze că este într-adevăr cine pretinde a fi. Acest lucru se poate face numai pe baza unor factori aflați sub controlul exclusiv al entității respective.

Principalii factori folosiți în procesul de autentificare sunt:

- **Ceea ce utilizatorul cunoaște.** Pentru autentificare, utilizatorul trebuie să prezinte ceva ce numai el cunoaște, cum ar fi, de exemplu, o parolă sau un PIN (Personal Identification Number).



A screenshot of a web-based login form. It contains two input fields: 'Username:' with the text 'jdoe' and 'Password:' with the text 'asdf'.

Figura 3-1. Parole folosite în procesul de autentificare

- **Ceea ce utilizatorul deține.** Pentru autentificare, utilizatorul trebuie să prezinte ceva ce se află numai în posesia sa, cum ar fi, de exemplu, un token, un certificat, un card sau un telefon mobil.



Figura 3-2. Dispozitive folosite în procesul de autentificare

- **Ceea ce utilizatorul este.** În acest caz, autentificarea se face pe baza caracteristicilor biometrice ale utilizatorului. Amprenta, irisul, geometria feței, vocea, ADN-ul, etc, sunt trăsături unice, specifice fiecărui individ și prin urmare pot fi folosite în procesul de autentificare.

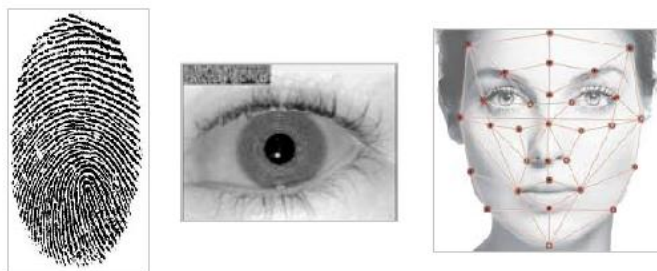


Figura 3-3. Caracteristici biometrice folosite în procesul de autentificare

Luată individual, niciuna din aceste metode de autentificare nu este complet sigură. Parolele pot fi ghicite sau aflate, dispozitivele pot fi pierdute sau furate iar sistemele biometrice pot fi înșelate. Pentru a asigura autentificarea sigură a utilizatorilor, se recomandă combinarea metodelor. De exemplu, un smart card este întotdeauna folosit în conjuncție cu un PIN.

În continuare sunt prezentate principalele protocoale de autentificare dezvoltate pe baza metodelor enumerate mai sus.

3.1. Autentificarea folosind parole

Parolele reprezintă în momentul de față cea mai folosită metodă de autentificare. Accesul în sistemele de operare sau în aplicații este, de regulă, controlat prin intermediul unei parole. Deși parolele sunt vulnerabile la diverse tipuri de atacuri, prin alegerea lor în mod corespunzător și prin folosirea unor protocoale de autentificare adecvate, acestea vor putea fi folosite și în viitor pentru autentificarea utilizatorilor. În secțiunile ce urmează sunt prezentate principalele protocoale de autentificare pe bază de parole, plecând de la cele simple, cu transmiterea în clar a parolei până la cele complexe cum este Kerberos.

3.1.1. Protocoale cu transmiterea în clar a parolei

Cel mai simplu mod de autentificare este cel în care numele / identificatorul (ID) utilizatorului și parola sunt transmise în clar către serverul de autentificare. Serverul de autentificare verifică aceste credențiale în baza de date și dacă sunt corecte, permite accesul utilizatorului.

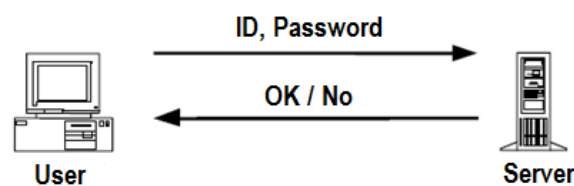


Figura 3-4. Autentificare cu transmitere în clar a parolei

Acest mod de autentificare este nesigur, parola putând fi interceptată cu ușurință atunci când este transmisă prin rețea. Password Authentication Protocol (PAP) folosit în cadrul Point-to-Point Protocol (PPP) este un exemplu de astfel de protocol. De asemenea, protocoale de nivel aplicație precum Telnet, FTP, POP3, IMAP, HTTP Basic Auth, transmit în clar parola utilizatorului în momentul autentificării. Figura 3-5 prezintă parola captată prin intermediul unui analizor de pachete (Wireshark) în momentul în care utilizatorul își accesează căsuța poștală folosind protocolul POP3. Pentru a evita interceptarea parolilor, se recomandă criptarea comunicației prin SSL/TLS.

Source	Destination	Protocol	Info
78.47.178.172	192.168.0.4	POP	S: +OK Dovecot ready.
192.168.0.4	78.47.178.172	POP	C: USER alice
78.47.178.172	192.168.0.4	TCP	pop3 > 57870 [ACK] Seq=21 Ack=11 win=14624 Len=0
78.47.178.172	192.168.0.4	POP	S: +OK
192.168.0.4	78.47.178.172	POP	C: PASS 123456
78.47.178.172	192.168.0.4	TCP	pop3 > 57870 [ACK] Seq=26 Ack=24 win=14624 Len=0
78.47.178.172	192.168.0.4	POP	S: +OK Logged in.
192.168.0.4	78.47.178.172	POP	C: STAT
78.47.178.172	192.168.0.4	TCP	pop3 > 57870 [ACK] Seq=42 Ack=30 win=14624 Len=0
78.47.178.172	192.168.0.4	POP	S: +OK 0 0
192.168.0.4	78.47.178.172	POP	C: QUIT
78.47.178.172	192.168.0.4	POP	S: +OK Logging out.

Figura 3-5. Interceptarea parolilor transmise prin rețea

3.1.2. Protocoale de tip întrebare - răspuns

În protocoalele de tip întrebare-răspuns (challenge-response), o parte prezintă o întrebare (challenge) iar cealaltă parte trebuie să furnizeze un răspuns (response) valid pentru a se autentifica. Altfel spus, pentru a autentifica o entitate, se verifică dacă aceasta este în posesia unei informații secrete, fără a transmite această informație prin rețea.

Challenge-Handshake Authentication Protocol (CHAP), folosit în cadrul Point-to-Point Protocol (PPP), este un exemplu de astfel de protocol. CHAP este descris în RFC 1994 și constă din trei pași:

1. serverul trimite o valoare aleatoare către client (challenge);
2. clientul răspunde cu o valoare calculată folosind o funcție hash aplicată peste parola secretă și valoarea aleatoare trimisă de server (response);

3. serverul verifică răspunsul făcând același calcul și comparând hash-urile. Dacă hash-urile sunt identice înseamnă că clientul cunoaște parola secretă.

Întregul proces este prezentat generic în Figura 3-6. Se observă că parola nu circulă în clar prin rețea ci numai hash-ul acesteia. Cum funcțiile hash nu sunt inversabile, este imposibil de aflat parola prin interceptarea traficului de rețea.

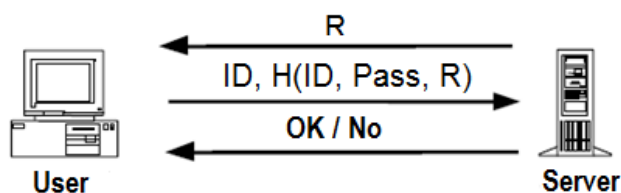


Figura 3-6. Autentificarea de tip întrebare-răspuns (varianta 1)

Pentru a preveni atacurile prin reluare, valorile aleatoare trimise de server (challenge) nu trebuie să se repete niciodată. Asta înseamnă că pe server trebuie folosit un generator de numere pseudo aleatoare foarte bun.

În scopul de a contracara orice încercare a unui eventual atacator de a obține informații cu privire la parola secretă, clientul poate adăuga și el o valoare aleatoare în momentul calculării hash-ului parolei. Schema îmbunătățită este prezentată în Figura 3-7. Valoarea generată de server este notată cu R_s iar cea generată de client cu R_c .

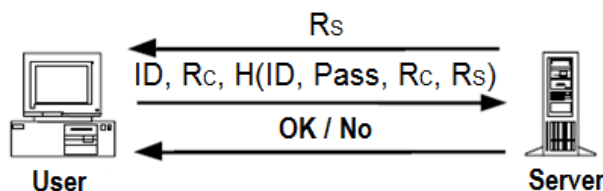


Figura 3-7. Autentificarea de tip întrebare-răspuns (varianta 2)

Această tehnică de autentificare este folosită în multe aplicații datorită simplității și securității oferite. Pe lângă CHAP, au fost dezvoltate și alte protocoale de tip întrebare-răspuns, cum ar fi: CRAM-MD5, DIGEST-MD5, SCRAM, etc.

3.1.3. Windows NT LAN Manager (NTLM)

Windows NT LAN Manager (NTLM) este protocolul folosit pentru autentificarea utilizatorilor care se conectează într-un domeniu sau pe stații de lucru individuale. NTLM este succesorul protocolului de autentificare

Microsoft LAN Manager (LM) și a fost introdus în 1993 odată cu lansarea Windows NT 3.1. În continuare este prezentat modul în care protocolul NTLM este folosit pentru autentificarea unui utilizator care accesează un server dintr-un domeniu NT.

Un domeniu NT este o colecție de servicii (E-mail, File Sharing, Printing, etc) toate administrate prin intermediul unui Domain Controller (DC). Fiecare utilizator are un singur cont pentru un domeniu, gestionat de către Domain Controller. Autentificarea utilizatorilor care accesează un server din domeniu se face prin intermediul Domain Controller.

Fiecare utilizator are o parolă pentru autentificare. Din motive de securitate, la nivelul Domain Controller-ului nu se stochează în clar parola utilizatorului ci cheia acestuia obținută prin aplicarea unei funcții într-un singur sens asupra parolei. Cheile utilizatorilor sunt stocate de Domain Controller într-o bază de date internă denumită Security Account Manager database.

NTLM este un protocol de tip întrebare-răspuns (challenge-response). Pașii ce trebuie parcurși pentru autentificarea unui utilizator la un server din domeniu sunt următorii:

1. Utilizatorul transmite ID-ul său (numele de utilizator) către server;
2. Serverul trimite o valoare aleatoare (challenge) și așteaptă răspunsul utilizatorului;
3. Utilizatorul criptează valoarea aleatoare primită de la server cu cheia sa și trimite rezultatul către server (response);
4. Serverul trimite un mesaj către Domain Controller ce conține: ID-ul utilizatorului, valoarea aleatoare trimisă și răspunsul primit din partea utilizatorului;
5. Domain Controller criptează valoarea aleatoare trimisă de server cu cheia utilizatorului extrasă din baza de date internă și compară rezultatul obținut cu răspunsul furnizat de utilizator serverului. Dacă cele două valori coincid atunci înseamnă că utilizator cunoaște parola și prin urmare i se poate acorda accesul la server.

Întregul proces de autentificare este prezentat în Figura 3-8. Se observă că parolele nu circulă niciodată în clar prin rețea. Protocolul este eficient cât timp se folosesc parole sigure.

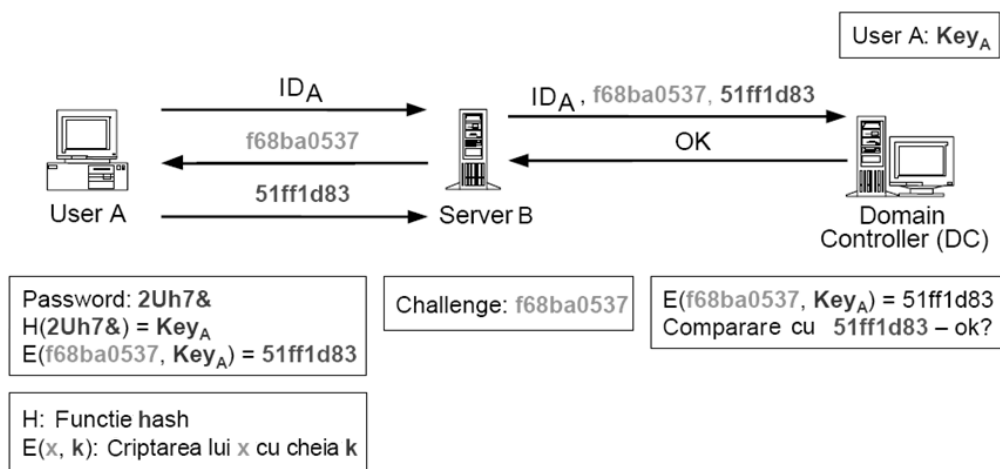


Figura 3-8. Protocolul de autentificare NTLM

Principalul dezavantaj al acestei scheme este faptul că protocolul de autentificare trebuie repetat pentru fiecare server în parte. Asta înseamnă că utilizatorul trebuie să introducă parola de fiecare dată când accesează un serviciu din rețea.

Un alt dezavantaj este faptul că parolele slabe sau scurte pot fi sparte offline prin diverse metode. Dacă un atacator, exploatând o breșă de securitate de pe server, obține o copie a Security Account Manager database, poate apoi să încerce să deducă parolele utilizatorilor prin atacuri de tip forță brută (brute force), dicționar sau rainbow tables.

Din motive de compatibilitate, pe sistemele Windows NT, 2000, XP și 2003, în mod implicit, se stochează atât hash-ul LM cât și hash-ul NTLM al parolei. Această opțiune se recomandă a fi dezactivată din motivele expuse mai jos.

Pentru calculul hash-ului LM se folosește un algoritm simetric (DES) în care parola este criptată cu o cheie fixă, astfel:

1. Se convertește parola la litere mari și se adaugă blank-uri până se obțin 14 caractere;
2. Se sparge valoarea rezultată în două părți și se criptează separat folosind algoritmul DES;
3. Cele două criptograme în format hexazecimal sunt concatenate pentru a forma hash-ul parolei.

Definiția funcției de calcul a hash-ului LM este următoarea:

```
Define LMOWFv1(Passwd, User, UserDom) as
    ConcatenationOf(DES(UpperCase(Passwd)[0..6], "KGS!@#$$%"),
        DES(UpperCase(Passwd)[7..13], "KGS!@#$$%"))
EndDefine
```

Este evident faptul că în cazul în care parola are mai puțin de 7 caractere, atunci cea de-a doua parte a valorii hash va fi aceeași: 0xAAD3B435B51404EE. De asemenea, prin convertirea parolei la litere mari, spațiu de căutare în cazul unui atac de tip dicționar se restrânge foarte mult.

În calcul unui hash NTLMv2 se folosește HMAC-MD5, iar caracterele parolei sunt în format Unicode. Definiția funcției de calcul a hash-ului NTLMv2 este următoarea:

```
Define NTOWFv2(Passwd, User, UserDom) as
    HMAC_MD5(MD4(UNICODE(Passwd)),
        UNICODE(ConcatenationOf(Uppercase(User), UserDom)))
EndDefine
```

Având în vedere dezavantajele NTLM și că algoritmi criptografici folosiți (DES și MD5) nu mai oferă protecția necesară, nu se mai recomandă folosirea acestui protocol în aplicații.

3.1.4. Kerberos

În mitologia greacă, Kérberos (sau Cerberus) a fost câinele de pază al lui Hades, zeul lumii de dincolo. În domeniul calculatoarelor, Kerberos este un protocol de autentificare celebru, dezvoltat la Massachusetts Institute of Technology (MIT). Kerberos a fost dezvoltat în cadrul proiectului Athena cu scopul de a permite entităților ce comunică folosind canale nesigure să își dovedească identitatea una celeilalte într-o manieră sigură. Altfel spus, protocolul asigură autentificare mutuală între entități, pentru controlul accesului la servicii distribuite în rețea.

Steve Miller și Clifford Neuman au publicat prima versiune a specificațiilor Kerberos la sfârșitul anilor '80 și anume versiunea 4. Primele trei versiuni au fost folosite intern în cadrul MIT și nu au fost publicate niciodată. Versiunea 5, ale cărei specificații au fost scrise de John Kohl și Clifford Neuman, a apărut în RFC 1510 în anul 1993 și a încercat să elimine limitările și problemele de securitate specifice vechii versiuni.

Kerberos este derivat din schema Needham-Schroeder pentru stabilirea unei chei de sesiune între două entități. Kerberos folosește algoritmi criptografici simetrici (DES) și, asemenea NTLM, nu transmite și nu stochează parolele în clar.

Kerberos se bazează pe serviciile de mediere oferite de un terț de încredere, numit *Centru de Distribuție al Cheilor (KDC - Key Distribution Center)*. Un KDC este format din două entități logice distincte: un *Server de Autentificare (AS - Authentication Server)* și un *Server ce Furnizează Tichete (TGS - Ticket Granting Server)*. În Kerberos, autentificarea entităților se face prin intermediul “tichetelor”.

Fiecare entitate (utilizator sau server) are o cheie master pe care o înregistrează la KDC. Conform terminologiei Kerberos, entitățile mai poartă și denumirea de *principals*. Cheia master este obținută prin aplicarea unei funcții într-un singur sens asupra parolei. Cheile entităților sunt stocate într-o bază de date internă, criptată cu cheia master a KDC. Un astfel de mediu poartă denumirea de *realm* (domeniu). Practic, un realm este o colecție de noduri care partajează aceeași bază de date Kerberos.

În procesul de autentificare din Kerberos se folosesc trei schimburi de mesaje, fiecare având un anumit rol:

1. Autentificarea Inițială (Authentication Service Exchange)
2. Obținere Tichet de Acces (Ticket-Granting Service Exchange)
3. Autentificarea Client-Server (Client-Server Authentication Exchange)

Autentificarea Inițială

Acest schimb de mesaje are loc o singură dată pe parcursul unei sesiuni de logon. În momentul în care utilizatorul de conectează la stația de lucru, introducând ID-ul (numele de utilizator) și parola, se transmite o cerere de autentificare către AS din cadrul KDC (AS_REQ). Această cerere de autentificare conține ID-ul utilizatorului (ID_A) și o criptogramă obținută prin criptarea timpului curent cu cheia master a utilizatorului ($MKey_A$).

KDC extrage cheia utilizatorului din baza de date internă ($MKey_A$), decriptează criptograma și compară timpul obținut cu timpul său curent. Dacă totul este în regulă, înseamnă că utilizatorul cunoaște parola și prin urmare i se poate acorda accesul.

Serverul generează o cheie de sesiune (S_A) pentru comunicare cu utilizatorul și un tichet de uz general (TGT – Ticket-Granting Ticket). TGT-ul se obține prin criptarea ID-ului utilizatorului (ID_A) și a cheii de sesiune (S_A) cu cheia master a KDC ($MKey_{KDC}$). Cheia de sesiune (S_A) și tichet-ul de uz general (TGT_A) sunt criptate cu cheia master a utilizatorului ($MKey_A$) și transmise acestuia ca răspuns la cererea de autentificare (AS_REP).

Utilizatorul decriptează mesajul primit de la KDC și stochează local cheia de sesiune (S_A) și tichet-ul de uz general (TGT_A). Pe baza TGT -ului, utilizatorul poate solicita în continuare tichete de acces la serverele din rețea. Întregul proces este prezentat schematic în Figura 3-9.

TGT -ul are o perioadă de viață configurabilă, de regulă între 8-24 ore. În tot acest interval, utilizatorul nu mai trebuie să introducă parola. În momentul în care TGT -ul expiră, autentificarea inițială trebuie reluată pentru a obține un nou TGT .

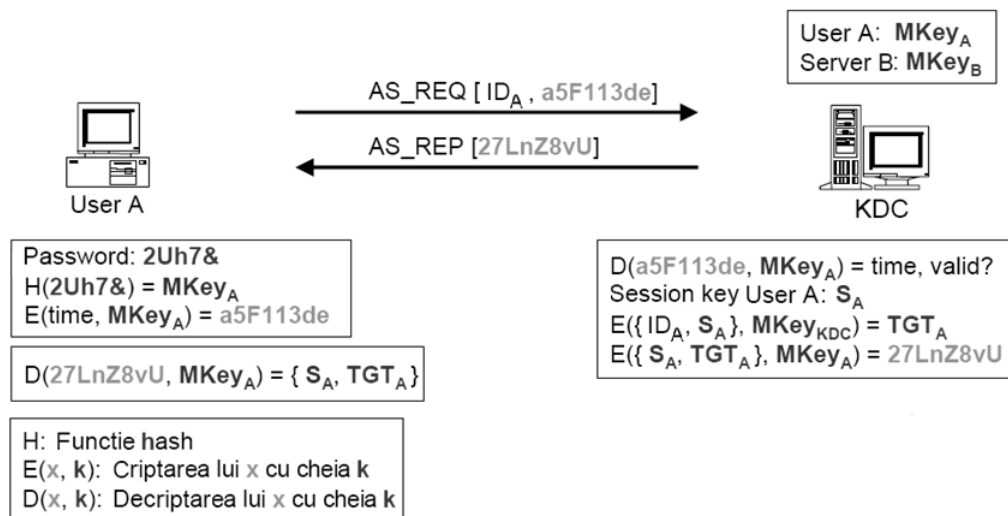


Figura 3-9. Autentificarea inițială în Kerberos

Obținere Tichet de Acces

În momentul în care utilizatorul dorește să acceseze un server din rețea, trebuie să obțină un tichet de acces la serverul respectiv. În acest sens, utilizatorul trimite o cerere către TGS din cadrul KDC (TGS_REQ). Această cerere conține ID-ul serverului (ID_B), TGT -ul utilizatorului (TGT_A) și o criptogramă obținută prin criptarea ID-ului utilizatorului (ID_A) și a timpului curent cu cheia de sesiune (S_A).

KDC decriptează tichetul utilizatorului (TGT_A) cu cheia sa master ($MKey_{KDC}$) și obține ID-ul utilizatorului (ID_A) și cheia de sesiune (S_A). Folosind cheia de sesiune (S_A), KDC decriptează criptograma primită de la client și compară ID-ul și timpul. Dacă totul este în regulă, înseamnă că cererea provine de la un utilizator autentificat.

În continuare, KDC generează o cheie de sesiune pentru comunicare între utilizator și server (S_{AB}) și un tichet pentru accesul utilizatorului la server (T_{AB}).

Tichetul de acces se obține prin criptarea ID-ului utilizatorului (ID_A) și a cheii de sesiune pentru comunicarea cu serverul (S_{AB}) cu cheia master a serverului ($MKey_B$). Cheia de sesiune pentru comunicarea cu serverul (S_{AB}) și tichet-ul de acces la server (T_{AB}) sunt criptate cu cheia de sesiune a utilizatorului (S_A) și transmise acestuia ca răspuns la solicitarea efectuată (TGS_REP).

Utilizatorul decriptează mesajul primit de la KDC și folosește în continuare cheia de sesiune (S_{AB}) și tichet-ul de acces (T_{AB}) pentru a se autentifica la server. Procesul de obținere a tichetului de acces este prezentat în Figura 3-10.

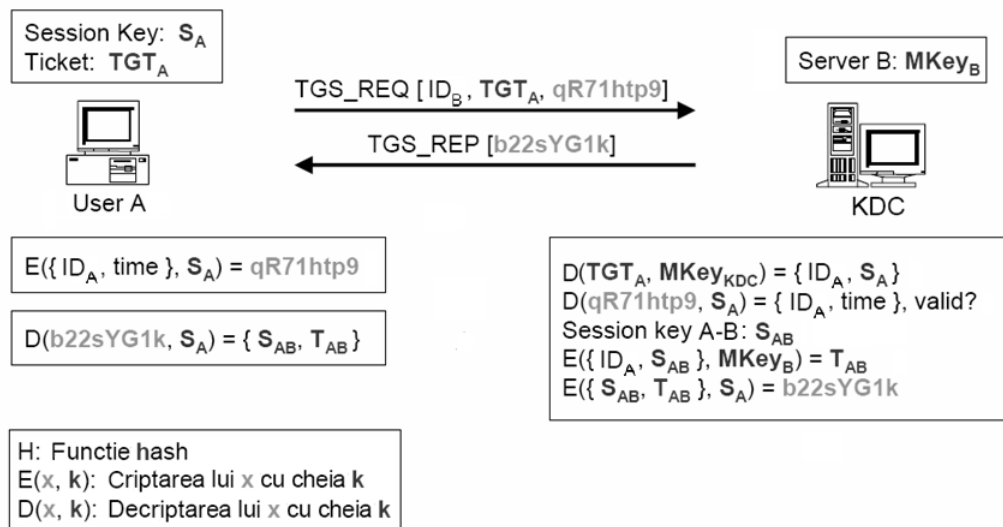


Figura 3-10. Obținerea tichetului de acces în Kerberos

Autentificarea Client-Server

Folosind tichetul de acces, utilizatorul se autentifică la server în vederea obținerii accesului la servicii. În acest sens, utilizatorul trimite o cerere către server (AP_REQ). Această cerere conține tichetul de acces (T_{AB}) și o criptogramă obținută prin criptarea ID-ului utilizatorului (ID_A) și a timpului curent cu cheia de sesiune pentru comunicarea cu serverul (S_{AB}).

Serverul decriptează tichetul de acces (T_{AB}) cu cheia sa master ($MKey_B$) și obține ID-ul utilizatorului (ID_A) și cheia de sesiune (S_{AB}). Folosind cheia de sesiune (S_{AB}), serverul decriptează criptograma primită de la client și compară ID-ul și timpul. Dacă totul este în regulă, înseamnă că cererea provine de la un utilizator autentificat.

În continuare, serverul criptează ID-ul propriu (ID_B) și timpul curent cu cheia de sesiune (S_{AB}) și transmite criptograma rezultată clientului (AP_REP).

Clientul decriptează criptograma primită de la server cu cheia de sesiune (S_{AB}) și verifică ID-ul serverului și timpul. Dacă totul este în regulă, înseamnă că clientul dialoghează cu un server autentic. Întregul proces este prezentat schematic în Figura 3-11.

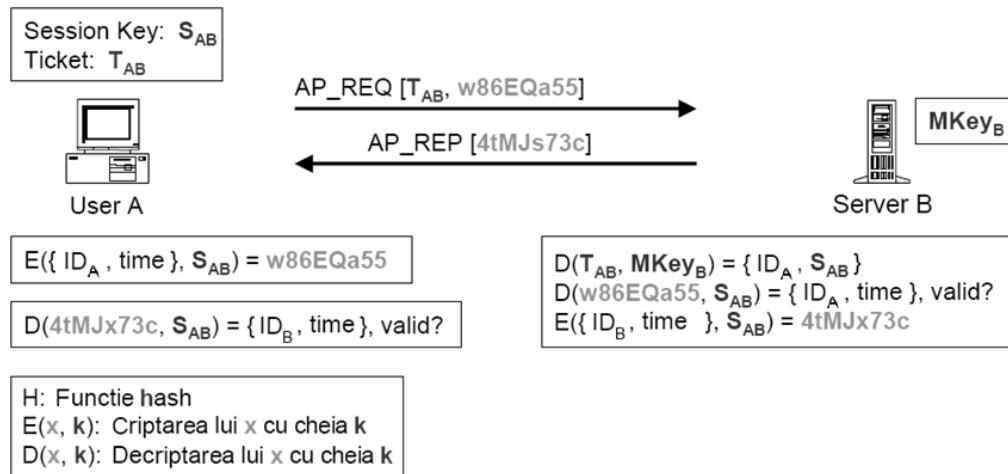


Figura 3-11. Autentificarea client-server în Kerberos

După cum se poate observa, Kerberos folosește amprente de timp în procesul de autentificare pentru a evita atacurile prin reluare. Pentru asta trebuie asigurată sincronizarea ceasurilor stațiilor de lucru și serverelor din rețea cu KDC. În acest scop, se poate folosi NTP (Network Time Protocol).

Kerberos poate fi extins pentru a suporta autentificare între realm-uri / organizații diferite astfel încât un utilizator dintr-o organizație să poată accesa un server din altă organizație. Pentru asta, trebuie ca între KDC-urile celor două realm-uri să se stabilească în comun o cheie secretă și o relație de încredere.

În momentul de față, Kerberos este implementat pe multe versiuni de UNIX: FreeBSD, Apple Mac OS X, Red Hat Enterprise Linux, Oracle Solaris, IBM AIX, HP-UX putând fi folosit cu succes pentru autentificarea utilizatorilor la servicii.

Kerberos a fost adoptat de Microsoft ca metodă principală de autentificare începând cu Windows 2000. Microsoft a propus și o extensie a protocolului pentru a permite autentificarea utilizatorilor pe bază de certificate digitale stocate pe smart card-uri criptografice. Această extensie este descrisă în RFC 4556 (Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)).

3.2. Autentificarea folosind generatoare de parole

Parolele reprezintă cea mai folosită metodă de autentificare. Din păcate, parolele, dacă nu sunt alese în mod corespunzător, sunt vulnerabile la o serie de atacuri. Parolele pot fi ghicite, pot fi interceptate atunci când sunt transmise prin rețea sau pot fi sparte prin atacuri de tip dicționar sau prin forță brută. Cu toate acestea, parolele au un mare avantaj: sunt ușor de folosit.

Generatoarele de parole de unică folosință păstrează avantajul în utilizare al parolelor și elimină riscurile de securitate specifice acestora. În cazul generatoarelor de parole de unică folosință, autentificarea se face de fiecare dată folosind altă parolă. Aceste parole nu pot fi prezise și nici reutilizate în cazul interceptării.

Pe piață sunt disponibile mai multe soluții de acest gen, dintre care cele mai cunoscute sunt:

- RSA SecurID (www.rsa.com)
- Vasco (www.vasco.com)
- SafeNet (www.safenet-inc.com)
- Gemalto (www.gemalto.com)



Figura 3-12. Generatoare de parole de unică folosință

Funcție de modul în care sunt generate parolele, soluțiile pot fi:

- bazate pe evenimente (event-based) – parola este generată în momentul producerii unui eveniment (apăsarea unui buton, de exemplu);
- bazate pe timp (time-based) – parola este generată la intervale regulate de timp;
- întrebare-răspuns (challenge-response) – parola este generată ca răspuns la o întrebare (challenge).

Există și o inițiativă pentru dezvoltarea unei arhitecturi de referință, bazată pe standarde deschise, cu scopul promovării folosirii acestei tehnologii în procesul de autentificare sigură a utilizatorilor (OAUTH - Initiative for Open Authentication). Efortul acestei inițiative s-a materializat prin elaborarea mai multor standarde de către grupul de lucru din cadrul IETF: RFC 4226 (An HMAC-Based One-Time Password Algorithm), RFC 6238 (TOTP: Time-Based One-Time Password Algorithm) sau RFC 6287 (OCRA: OATH Challenge-Response Algorithm).

În continuare este prezentată soluția **RSA SecurID**. Această soluție se bazează pe niște dispozitive speciale denumite token-uri care generează în mod aleator secvențe de numere, la interval de un minut, denumite *token code*. Autentificarea unui utilizator care accesează o resursă protejată cu RSA SecurID se face pe baza unui PIN pe care numai acesta îl cunoaște și a secvenței de numere generată de token-ul pe care acesta îl deține. PIN-ul utilizatorului concatenat cu numerele generate de token poartă denumirea de *passcode*. Putem spune deci că RSA SecurID este o soluție de autentificare bazată pe doi factori: ceea ce utilizatorul cunoaște (PIN-ul) și ceea ce utilizatorul deține (token-ul).

Validarea codurilor de autentificare se face de către serverul de autentificare, *RSA Authentication Manager*. Pe fiecare sistem din rețea protejat cu RSA SecurID se instalează un software special denumit *RSA Authentication Agent*, care substitue mecanismul de autentificare existent, bazat pe parole, cu cel descris mai sus, bazat pe doi factori. Arhitectura soluției RSA SecurID este prezentată în Figura 3-13.

Fiecare dispozitiv de autentificare este unic și este imposibil de prezis valoarea generată de acesta. Generarea codurilor de autentificare are la bază o schemă de generare numere aleatoare proprietară, bazată pe algoritmul AES și o sămânță (seed) unică pentru fiecare dispozitiv. Serverul RSA Authentication Manager cunoaște ce token se află în posesia fiecărui utilizator și implicit seed-ul asociat acestuia și decalajul de timp dintre ceasul intern și ceasul token-ului. Astfel, serverul este capabil să genereze aceeași secvență de numere ca și token-ul utilizatorului.

Atunci când un utilizator furnizează un passcode corect, putem fi siguri că el se află în posesia unui dispozitiv de autentificare RSA SecurID și cunoaște PIN-ul acestuia. Prin urmare, utilizatorul respectiv este un utilizator legitim al sistemului.

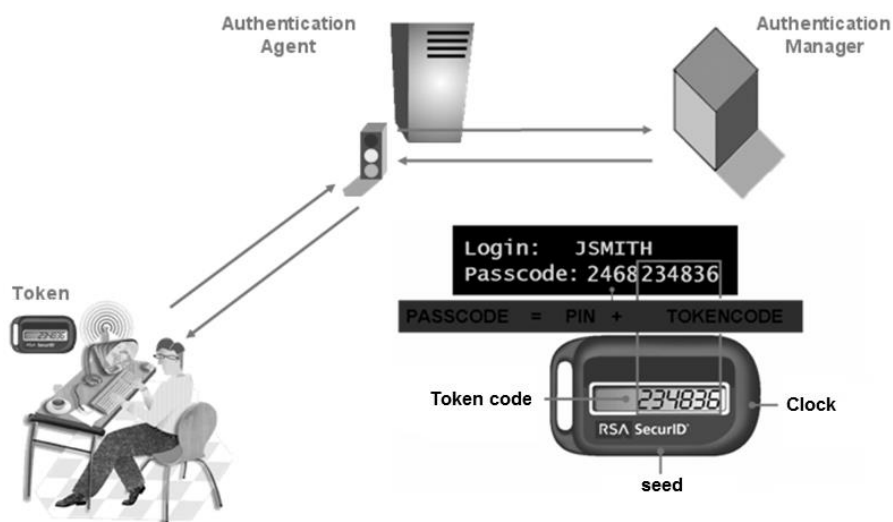


Figura 3-13. Arhitectura RSA SecurID

Soluția RSA SecurID este foarte versatilă putând fi folosită pentru autentificarea utilizatorilor în sisteme de operare Windows / Unix / Novell, la conectarea prin dial-up sau VPN sau la aplicații Web (IIS / Apache). Firma RSA Security oferă agenți de autentificare pentru sisteme de operare sau servere de aplicații larg răspândite. De asemenea, soluția poate fi integrată în aplicații proprii, în acest sens, firma RSA Security punând la dispoziția dezvoltatorilor API-uri și exemple de cod. În momentul de față, există peste 400 de produse care permit folosirea RSA SecurID ca și metodă de autentificare sigură a utilizatorilor.

Dispozitive de autentificare sunt diversificate și pot fi sub formă hardware sau software. Toate dispozitivele de autentificare funcționează pe baza aceluiași algoritm proprietar de generare a numerelor aleatoare și sunt protejate la deschidere.

3.3. Autentificarea folosind certificate digitale

Certificatele digitale X.509 conțin, pe lângă cheia publică, datele de identificare ale entității respective. Aceste date se află înscrise, de regulă, în câmpul *Subject Name* sau în extensia *Subject Alternative Name*. Prin urmare, am putea fi tentați să folosim direct aceste certificate pentru autentificarea unei entități. Din păcate, certificatul digital în sine nu constituie factor de autentificare. Certificatul este public și poate fi obținut de oricine din repository-ul Autorității de Certificare sau prin alte mijloace.

Autentificarea pe bază de certificate digitale se realizează, de fapt, făcând dovada posesiei cheii private asociate cheii publice din certificat. Cheia privată este cunoscută numai de către entitatea autentificată și prin urmare poate fi considerată factor de autentificare. Pentru a face dovada posesiei cheii private asociate cheii publice din certificat, există două alternative posibile:

- Entitatea autentificată semnează digital un mesaj conținând date arbitrare. Verificarea semnăturii se face folosind cheia publică din certificat;
- Se trimite un mesaj criptat cu cheia publică a entității autentificate iar aceasta trebuie să demonstreze că este capabilă să-l decripteze. Cheia publică se obține și în acest caz tot din certificatul digital al entității autentificate.

Ambele variante sunt întâlnite în practică. În continuare vom prezenta protocoale de autentificare descrise în standardul ITU-T X.509 (Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks). Seria de standarde ITU-T X.500 se referă la serviciile de directoare. ITU-T X.509 definește un framework pentru implementarea de servicii de autentificare folosind directoare X.500.

În cadrul acestui framework s-a propus folosirea certificatelor digitale pentru autentificare și s-a standardizat pentru prima dată formatul acestor certificate. În procesul de autentificare, directorul X.500 joacă rol de repository pentru certificatele de chei publice. Atunci când o entitate are nevoie de cheia publică a altei entități, caută în repository certificatul digital al acesteia.

ITU-T X.509 definește trei protocoale pentru autentificarea sigură a entităților pe bază de certificate digitale:

- One-way authentication
- Two-way authentication
- Three-way authentication

În toate aceste protocoale, dovada posesiei cheii private asociate cheii publice din certificat se face prin semnarea digitală a unor date aleatoare. Se presupune că fiecare entitate dispune de certificatul digital al celeilalte entități cu care desfășoară un protocol de autentificare. Așa cum am precizat anterior, certificatul se poate obține dintr-un server de directoare X.500 sau printr-un schimb inițial de mesaje.

3.3.1. Autentificarea într-un singur sens

Protocolul de autentificare într-un singur sens (one-way authentication) constă într-un singur mesaj, transmis de utilizatorul A către utilizatorul B, care garantează:

1. identitatea lui A și faptul că mesajul a fost generat de către A;
2. mesajul îi este adresat lui B;
3. integritatea și originalitatea mesajului (protecție la modificări neautorizate și la atacuri prin reluare).

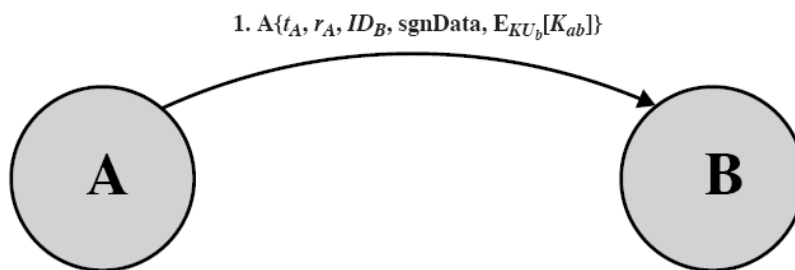


Figura 3-14. Autentificarea într-un singur sens

Protocolul de autentificare într-un singur sens este prezentat în Figura 3-14. Mesajul transmis de A către B conține:

- o amprentă de timp (t_A) formată din două date: data generării mesajului (opțională) și data expirării mesajului;
- o valoare generată în mod aleator (r_A) folosită pentru a preveni atacurile prin reluare;
- identitatea lui B (ID_B);
- semnătura digitală a lui A asupra mesajului (singData);
- o cheie de sesiune (K_{ab}) criptată cu cheia publică a lui B. Acest câmp este opțional și este transmis numai dacă A urmează să comunice criptat cu B.

3.3.2. Autentificarea în două sensuri

În plus față de protocolul anterior, protocolul de autentificare în două sensuri (two-way authentication) garantează:

4. identitatea lui B și faptul că mesajul de răspuns a fost generat de către B;
5. mesajul de răspuns îi este adresat lui A;
6. integritatea și originalitatea mesajului de răspuns (protecție la modificări neautorizate și la atacuri prin reluare).

Protocolul de autentificare în două sensuri, prezentat în Figura 3-15, asigură așadar autentificare mutuală între entități.

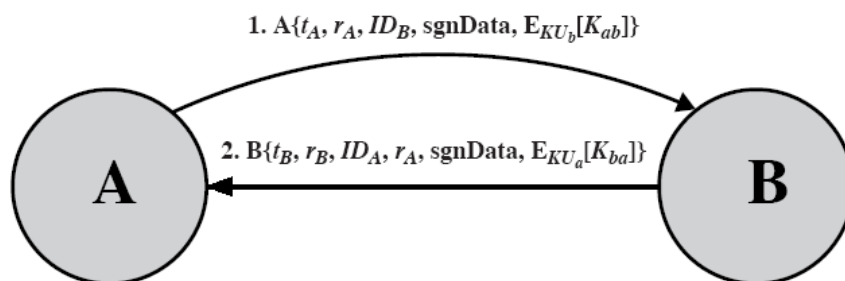


Figura 3-15. Autentificarea în două sensuri

Mesajul de răspuns transmis de la B la A include în plus valoarea aleatoare primită de la A (r_A) și opțional, o cheie de sesiune (K_{ba}) criptată cu cheia publică a lui A, dacă B urmează să comunice criptat cu A.

3.3.3. Autentificarea în trei sensuri

Protocolul de autentificare în trei sensuri (three-way authentication) asigură, de asemenea, autentificare mutuală între entități și se folosește atunci când ceasurile celor două entități nu se pot sincroniza și prin urmare, amprente de timp nu se pot verifica. Cel de-al treilea mesaj, transmis de A către B, conține o copie semnată a valorii aleatoare primită de la B (r_B). Procesul de autentificare în trei sensuri este prezentat în Figura 3-16.

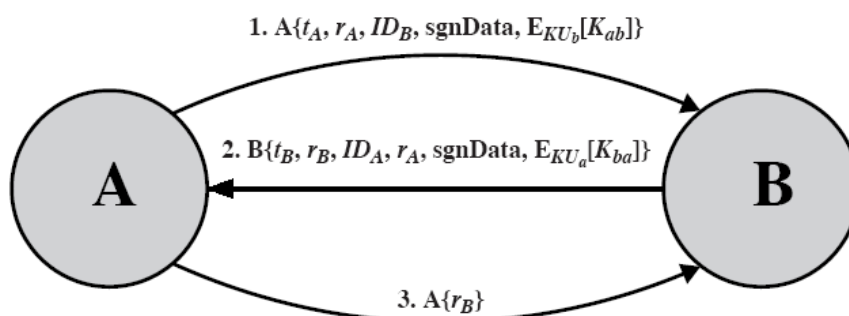


Figura 3-16. Autentificarea în trei sensuri

Autentificarea sigură a entităților pe bază de certificate digitale este folosită pentru logon în domenii Windows (Windows smart card logon), pentru autentificarea capetelor unui tunel IPSec sau pentru autentificarea entităților în aplicații Web, folosind SSL / TLS. Pentru a asigura protecția cheilor private, se recomandă generarea și stocarea acestora pe smart card-uri criptografice sau module criptografice hardware (HSM – Hardware Security Module).

3.4. Autentificarea folosind caracteristici biometrice

Recunoașterea biometrică se referă la folosirea caracteristicilor distinctive de natură fiziologică (ampretele, fața, irisul, de exemplu) sau comportamentală (semnătura olografă, de exemplu) pentru recunoașterea automată a indivizilor. Aceste caracteristici poartă denumirea de identificatori biometrici și având în vedere că nu pot fi ușor falsificați sau împrumutați, sunt considerați a fi mai siguri decât metodele tradiționale, atunci când sunt folosiți pentru autentificarea unei persoane.

Un *sistem biometric* este în esență un sistem de recunoaștere de șabloane (patterns) care recunoaște o persoană prin determinarea autenticității identificatorilor biometrici ai acesteia. Funcție de context, un sistem biometric poate opera în *modul verificare* sau în *modul identificare*.

Indiferent de modul de operare, recunoașterea biometrică implică înainte de toate înregistrarea persoanei în sistem. Procesul de înregistrare al unei persoane într-un sistem biometric este prezentat în Figura 3-17. În faza de înregistrare, identificatorul biometric al persoanei respective este mai întâi scanat cu un cititor biometric pentru a se obține o reprezentare digitală primară a acestuia. Apoi este verificată calitatea datelor achiziționate pentru a garanta faptul că eșantionul biometric obținut poate fi prelucrat corespunzător în etapele

următoare. Pentru a facilita compararea, reprezentarea digitală primară este prelucrată mai departe de către un extractor de caracteristici pentru a genera o reprezentare compactă numită *șablon biometric (template)*. În funcție de aplicație, șablonul biometric poate fi stocat în baza de date internă a sistemului biometric sau poate fi înregistrat pe un smart card emis pentru persoana respectivă.

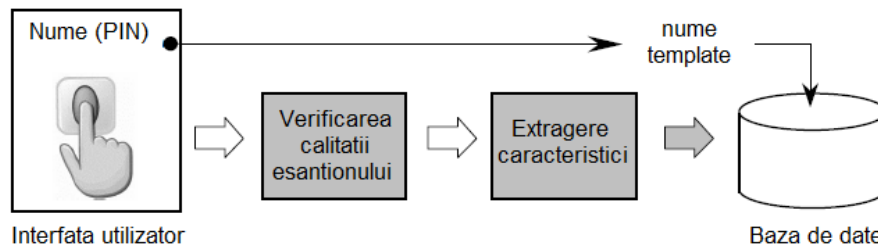


Figura 3-17. Procesul de înregistrare al unui utilizator într-un sistem biometric

În modul verificare, o persoană trebuie să demonstreze că este într-adevăr cine pretinde a fi. În acest sens, persoana respectivă introduce numele sau codul PIN cu ajutorul unei tastaturi. Cititorul biometric scanează identificatorul și îl convertește într-un format digital care este mai departe procesat de extractorul de caracteristici pentru a produce o reprezentare digitală compactă. Reprezentarea rezultată este furnizată comparatorului de caracteristici, care o compară cu șablonul persoanei extras din baza de date. Procesul de comparare este de tipul unu-la-unu și este prezentat generic în Figura 3-18. După cum se poate constata, procesul de verificare folosit în cadrul sistemelor biometrice este identic cu cel de autentificare descris în capitolele anterioare.

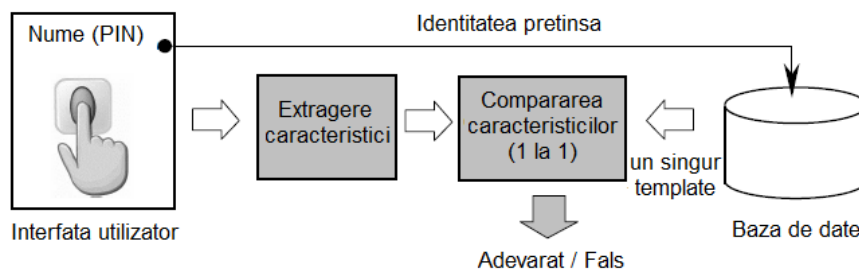


Figura 3-18. Procesul de verificare al unei persoane

În modul identificare sistemul nu știe cu cine are de-a face și compară reprezentarea rezultată în urma eșantionării cu toate șabloanele din baza de date pentru a determina identitatea persoanei. În cazul bazelor mari de date acest proces de comparare este unul intensiv din punct de vedere computațional. Pentru a crește performanța, sunt utilizate tehnici de clasificare și indexare care limitează numărul de șabloane biometrice cu care trebuie comparată intrarea.

Procesul de comparare este de tipul unu-la-mai-mulți și este prezentat generic în Figura 3-19.

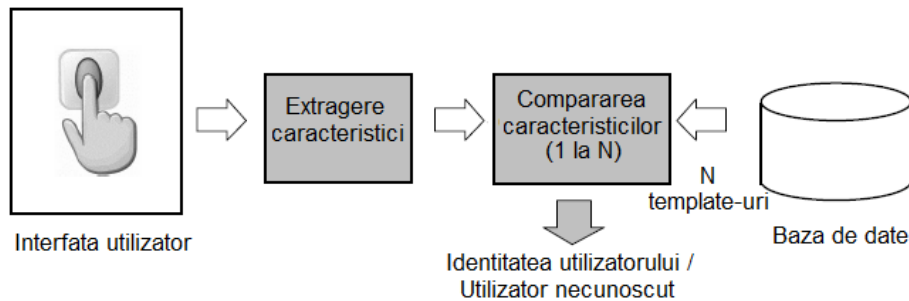


Figura 3-19. Procesul de identificare al unei persoane

Principalii identificatori biometrici care pot fi folosiți în procesul de recunoaștere al unei persoane sunt prezentați sumar în continuare:

- **ADN-ul.** Acidul dezoxiribonucleic reprezintă poate cel mai sigur identificator biometric. ADN-ul poate fi considerat un cod unic al fiecărei persoane, cu excepția gemenilor identici (monoziagoți) care au tipare ADN identice. ADN-ul este folosit în domeniul judiciar însă aplicarea lui în domeniul calculatoarelor este dificilă în momentul de față.
- **Amprenta.** Amprenta degetelor este cea mai veche metodă de recunoaștere a unei persoane, fiind folosită în domeniul criminalisticii încă din 1896. Structura amprente cuprinde două tipuri de trăsături. La nivel global, trăsăturile amprente sunt date de modele de creste și văi de pe suprafața degetului. Acestea sunt, în principal, folosite pentru clasificarea amprentelor și nu pentru recunoașterea persoanelor. La nivel local, trăsăturile amprente sunt privite ca detalii precise (minutae). Ele corespund punctelor în care crestele se bifurcă sau se termină brusc. Aceste trăsături definesc unicitatea amprentelor și sunt folosite în procesul de recunoaștere al unei persoane. Datorită avantajelor pe care le oferă, amprenta este, în momentul de față, cea mai folosită metodă pentru autentificarea utilizatorilor. Senzorii pentru scanarea amprentelor sunt performanți și accesibili ca preț.
- **Dinamica tastării.** Se presupune că fiecare persoană operează la tastatură într-un ritm diferit. Acest identificator biometric comportamental, dacă este măsurat, ar putea fi folosit, cu o precizie satisfăcătoare, pentru recunoașterea unei persoane.

- **Fața.** Fața este poate cel mai ușor de acceptat factor biometric deoarece aceasta este metoda folosită pentru recunoașterea persoanelor în viața de zi cu zi. Metoda de achiziție a imaginii feței necesită o cameră video și nu este intruzivă. Pentru recunoașterea unei persoane pe baza feței sunt disponibile două abordări și anume: abordarea pe bază de caracteristici și abordarea holistică sau globală. În abordarea pe bază de caracteristici se încearcă localizarea relativă a diferitelor atribute faciale (ochi, nas, buze, bărbie, sprâncene, etc). Procesul de localizare și măsurare a acestor atribute este unul foarte complex iar acuratețea nu este foarte ridicată. În abordarea holistică se procesează întreaga imagine a feței simultan, fără a se încerca să se localizeze punctele individuale, folosind analiza statistică, rețele neuronale sau diverse tipuri de transformări. Acuratețea recunoașterii este mai bună în acest caz. Indiferent de abordarea folosită, performanța metodei poate fi afectată de o serie de factori cum ar fi: îmbătrânirea, machiajul, ochelarii, poziția și condițiile de iluminare. În plus, această metodă nu poate face distincție între gemeni sau persoane deghizate.
- **Geometria mâinii.** Forma și dimensiunile degetelor mâinii permit recunoașterea destul de precisă a unei persoane. Template-urile biometrice ocupă puțin spațiu iar sistemul este simplu de implementat. Geometria mâinii este prima tehnologie biometrică folosită în practică, dispozitive performante de acest gen fiind produse încă de la începutul anilor 1980. Ea rămâne și în continuare foarte populară, fiind întâlnită în sistemele de control al accesului sau de pontaj.
- **Mersul.** Modul în care cineva se deplasează poate reprezenta un identicator biometric pentru persoana respectivă. Acesta nu este însă foarte distinctiv fapt pentru care mersul poate fi folosit numai în aplicații cu nivel de securitate scăzut. În plus, mersul se poate modifica pe măsură ce o persoană înaintea în vârstă, își schimbă semnificativ greutatea, etc.
- **Retina.** Structura vasculară a retinei se consideră a fi o caracteristică a fiecărei persoane și a fiecărui ochi. Retina se presupune a fi un identicator biometric sigur din moment ce nu se poate schimba sau copia. Captarea imaginii este însă foarte dificilă și necesită efort din partea utilizatorului, fapt pentru care metoda este rar folosită în practică.
- **Semnătura olografă.** Fiecare persoană se semnează în mod diferit. Această semnătură este folosită pentru autentificarea documentelor tipărite pe hârtie însă ar putea fi utilizată și pentru autentificarea utilizatorilor în cadrul sistemelor informatice.

- **Termograma facială.** Cantitatea de căldură emanată de diferitele zone ale feței este o caracteristică a fiecărei persoane. Șablonul biometric este obținut prin intermediul unei camere în infraroșu. Metoda este neinvazivă și poate fi folosită și pe timp de noapte. Pe lângă termograma facială se mai poate folosi și **termograma mâinii** sau **a venelor**.
- **Urechea.** Este cunoscut faptul că forma urechii și structura țesutului cartilaginos sunt specifice fiecărei persoane. Abordările de recunoaștere a persoanelor folosind urechea se bazează pe compararea distanțelor dintre o serie de puncte de referință localizate pe ureche.
- **Vocea.** Vocea este folosită în mod natural pentru recunoaștere între indivizi și la fel de bine poate fi utilizată și în cadrul sistemelor de calcul. Un sistem biometric bazat pe voce poate fi foarte mic și compact din moment ce pentru înregistrarea vocii este nevoie doar de un microfon. Cu toate acestea, vocea variază în timp, depinde de starea de sănătate și emoțională a persoanei, este influențată de acustica încăperii sau de zgomotul de fond.

Într-un sistem bazat pe parole, se compară datele introduse de utilizator cu cele stocate în baza de date. Algoritmul de comparare este simplu de realizat iar rezultatul nu poate fi decât „adevărat” sau „fals”. În cazul sistemelor biometrice, datele extrase de senzor nu sunt niciodată identice cu cele stocate în baza de date datorită factorilor variabili care intervin în acest proces. Prin urmare, algoritmul de comparare este mult mai complex iar rezultatul nu poate fi unul exact.

Pentru a măsura performanța unui sistem biometric, se folosesc, de regulă, următorii parametrii:

- **Rata de Acceptare Falsă (FAR - False Acceptance Rate)**, reprezintă procentul de impostori acceptați în mod greșit de către sistem. Ideal ar fi ca această rată să fie cât mai mică.
- **Rata de Rejecție Falsă (FRR - False Reject Rate)**, reprezintă procentul de utilizatori valizi rejecți în mod greșit de către sistem. Ideal ar fi ca și această rată să fie cât mai mică.

FAR și FRR sunt interdependente. Dacă se impune o valoare prag cât mai mică pentru FAR, cum ar fi normal, FRR va crește inevitabil. Graficele din Figura 3-20 exemplifică tocmai acest lucru. Punctul în care FAR și FRR se intersectează poartă numele de **Rata Erorilor Egale (EER - Equal Error Rate)** și reprezintă un indicator al acurateții sistemului biometric.

Acești parametrii sunt estimați pentru majoritatea sistemelor biometrice. Trebuie menționat faptul că măsurătorile sunt făcute, de regulă, în condiții de laborator iar calitatea depinde de baza de date utilizată. În acest sens, au fost create și sunt disponibile pe Internet, baze de date de test pentru fiecare tip de sistem biometric în parte.

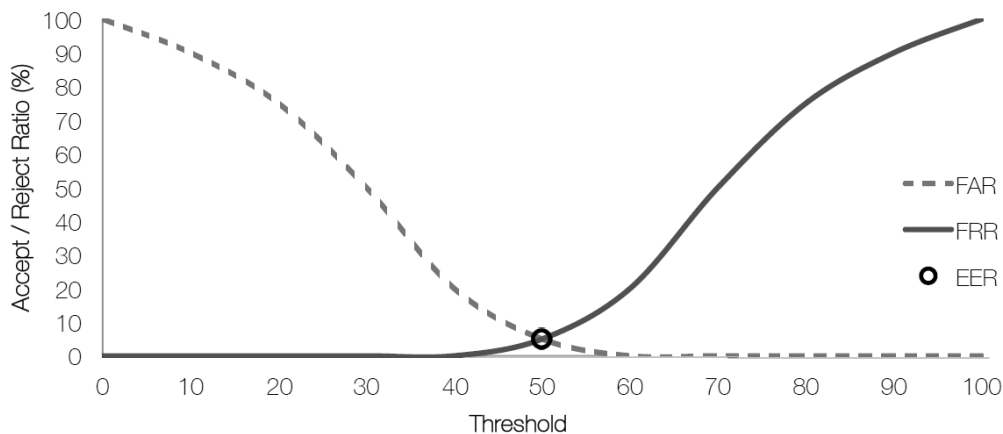


Figura 3-20. Performanța sistemelor biometrice

În afară de acești parametrii, atunci când se analizează un sistem biometric trebuie ținut cont și de cât de bine separă sistemul o persoană de alta și dacă poate face distincție între gemeni, dacă metoda rezistă la efectele îmbătrânirii persoanelor, dacă este sau nu intruzivă pentru utilizatori, etc.

Pentru aplicațiile cu cerințe de securitate foarte ridicate, se pot folosi mai mulți identificatori biometrici, obținându-se astfel un *sistem biometric multimodal*.

Biometria este un domeniu care se dezvoltă rapid, fiind folosită cu succes în multe domenii. Printre acestea se numără și autentificarea sigură a utilizatorilor care accesează date confidențiale sau care se conectează de la distanță prin VPN.

4. Protocoale de securitate la nivel rețea

La baza Internetului stă protocolul IPv4 care asigură transmiterea pachetelor între sursă și destinație. Atunci când a fost proiectat, în cadrul IPv4 nu a fost inclus nici un mecanism de securitate astfel că pachetele pot fi citite sau modificate relativ ușor. Pentru a asigura securitatea pachetelor transmise în rețea a fost dezvoltat protocolul IPSec care adaugă facilități de securitate la IPv4. IPSec este de fapt o colecție de protocoale ce permit securizarea comunicațiilor în rețelele IPv4:

- Protocolul *Authentication Header (AH)* poate fi folosit pentru asigurarea integrității și autentificarea originii pachetelor IP;
- Protocolul *Encapsulating Security Payload (ESP)* poate fi folosit pentru asigurarea confidențialității și integrității datelor din cadrul pachetelor IP;
- Protocolul *Internet Key Exchange (IKE)* asigură negocierea algoritmilor și parametrilor criptografici, autentificarea entităților și stabilirea de chei se de sesiune între entitățile comunicante.

Specificațiile IPSec se găsesc descrise în mai multe RFC-uri, cele mai importante fiind: RFC 4301 (Security Architecture for the Internet Protocol), RFC 4302 (IP Authentication Header), RFC 4303 (IP Encapsulating Security Payload), RFC 4308 (Cryptographic Suites for IPsec), RFC 7296 (Internet Key Exchange Protocol Version 2 (IKEv2)).

În continuare este prezentat în detaliu fiecare din aceste protocoale iar la finalul capitolului este explicat modul în care IPSec poate fi folosit pentru implementarea de rețele private virtuale.

4.1. Protocolul Authentication Header (AH)

Așa cum am menționat anterior, AH asigură integritatea și autentificarea originii datelor pentru întregul pachet IP. În acest sens AH calculează pentru fiecare pachet câte un MAC (Message Authentication Code). Valoarea MAC este calculată atât peste informațiile din antetul IP cât și peste payload. Prin urmare, în cazul folosirii AH, conținutul pachetelor poate fi în continuare citit însă nu poate fi modificat.

AH adaugă un antet în cadrul fiecărui pachet transmis în rețea. Acest antet conține șase câmpuri după cum se poate observa în Figura 4-1.

Next Header	Payload Length	Reserved
Security Parameters Index		
Sequence Number		
Authentication Information		

Figura 4-1. Formatul antetului AH

Câmpul *Următorul Antet (Next Header)* indică protocolul către care trebuie livrat payload-ul din pachet după procesarea antetului AH. Semnificația acestui câmpul este similară cu a câmpului *Protocol* din antetul IP. În modul tunel, payload-ul reprezintă un pachet IP, așadar valoarea câmpului Next Header este 4 (IP-in-IP). În modul transport, payload-ul este, de regulă, un protocolul de nivel transport, cel mai adesea TCP, caz în care valoarea câmpului Next Header este 6, sau UDP, caz în care valoarea este 17.

Câmpul *Lungimea Payload-ului (Payload Length)* reprezintă lungimea payload-ului în cuvinte de 32 biți.

Câmpul *Rezervat (Reserved)* nu este folosit fiind este rezervat pentru dezvoltări ulterioare. Valoarea sa trebuie setată la 0.

Câmpul *Indexul Parametrilor de Securitate (Security Parameters Index)* are rolul de identificator unic al conexiunii. Receptorul folosește valoarea SPI împreună cu adresa IP destinație și tipul protocolului IPSec (AH în acest caz) pentru a determina Asocierea de Securitate (SA). Prin intermediul SA receptorul știe ce algoritmi și ce chei de sesiune să utilizeze pentru efectuarea procesărilor criptografice.

Câmpul *Număr de Secvență (Sequence Number)* este folosit pentru a evita atacurile prin reluare, pachetele duplicate putând astfel fi ușor identificate deoarece au același număr de secvență.

Câmpul *Informație de Autentificare (Authentication Information)* reprezintă valoarea MAC calculată de emițător. La recepție se calculează din nou valoarea MAC pentru pachetul primit, folosind același algoritm hash și aceiași cheie secretă și se compară valoarea calculată cu valoarea recepționată. Dacă cele două valori sunt identice, pachetul nu a fost modificat în tranzit.

În cazul AH, valoarea MAC este calculată peste întregul pachet IP (antet plus payload) cu excepția câmpurilor din antet care pot suferi modificări în tranzit (TTL, de exemplu). Modul de calcul al MAC este conform cu RFC 2104 (HMAC: Keyed-Hashing for Message Authentication), AH suportând funcții hash moderne (SHA-1, SHA-256, SHA-384 sau SHA-512).

4.2. Protocolul Encapsulating Security Payload (ESP)

ESP este cel de-al doilea protocol din cadrul IPSec. Rolul acestuia este de a asigura confidențialitatea și integritatea datelor din cadrul pachetelor IP. Spre deosebire de AH, ESP nu protejează decât datele din pachet (payload-ul) nu și antetul IP.

ESP folosește algoritmi criptografici simetrici de tip cifru bloc pentru criptarea datelor cum ar fi AES-CBC, AES-GCM sau 3DES-CBC. Pentru asigurarea integrității datelor se folosesc MAC-uri calculate conform RFC 2104 (HMAC: Keyed-Hashing for Message Authentication), ESP suportând aceleași funcții hash ca și AH (SHA-1, SHA-256, SHA-384 sau SHA-512).

ESP adaugă un antet și un trailer în jurul payload-ului din pachetul IP, după cum se poate vedea în Figura 4-2.

Antet ESP	Security Parameters Index		
	Sequence Number		
Payload	Data		
Trailer ESP	Padding	Padding Length	Next Header
Informatie de autentificare	Authentication Information		

Figura 4-2. Formatul pachetelor ESP

Câmpul *Indexul Parametrilor de Securitate (Security Parameters Index)* are rolul de identificator unic al conexiunii. Receptorul folosește valoarea SPI împreună cu adresa IP destinație și tipul protocolului IPSec (ESP în acest caz) pentru a determina Asocierea de Securitate (SA). Prin intermediul SA receptorul știe ce algoritmi și ce chei de sesiune să utilizeze pentru efectuarea procesărilor criptografice.

Câmpul *Număr de Secvență (Sequence Number)* este folosit pentru a evita atacurile prin reluare, pachetele duplicate putând astfel fi ușor identificate deoarece au același număr de secvență.

Urmează apoi payload-ul criptat. ESP folosind cifruri bloc pentru criptarea datelor, e posibil să fie necesar să se facă *padding* pentru extinderea lungimii totale a blocului de date până la un multiplu al lungimii blocului de criptare. De asemenea, padding-ul s-ar putea să fie necesar și pentru a completa dimensiunea trailerului ESP la multiplu de cuvinte de 32 biți. În plus padding-ul poate fi utilizat și pentru a schimba lungimea pachetelor pentru a evita atacurile bazate pe analiza traficului.

Câmpul *Lungime Padding (Padding Length)* reprezintă lungimea în octeți a zonei de padding.

Câmpul *Următorul Antet (Next Header)* indică protocolul către care trebuie livrat payload-ul din pachet după procesarea ESP. Valorile uzuale pentru acest câmp sunt 4 pentru IP-in-IP, 6 pentru TCP sau 17 pentru UDP.

Trailerul ESP este urmat de câmpul *Informație de Autentificare (Authentication Information)* în care este stocată valoarea MAC calculată de emițător. Această valoare se calculează doar peste payload nu și peste antetul IP.

4.3. Moduri de operare IPSec

IPSec poate fi utilizat în două moduri: *transport* sau *tunel*. Modul transport poate fi utilizat pentru stabilirea unui tunel IPSec între două calculatoare (host-uri) conectate direct cum sunt cele din Figura 4-3.



Figura 4-3. Scenariu de utilizare a modului transport

Dacă se folosește protocolul AH în modul transport, la pachetele transmise între cele două stații se adaugă un antet AH. Antetul IP original se păstrează (vezi Figura 4-4). În cazul în care pachetul transportă un payload de tip TCP, valoarea câmpului Protocol din antetul IP este 6. După aplicarea AH, valoarea câmpului Protocol din antetul IP devine 51 iar valoarea câmpului Next Header din antetul AH devine 6.

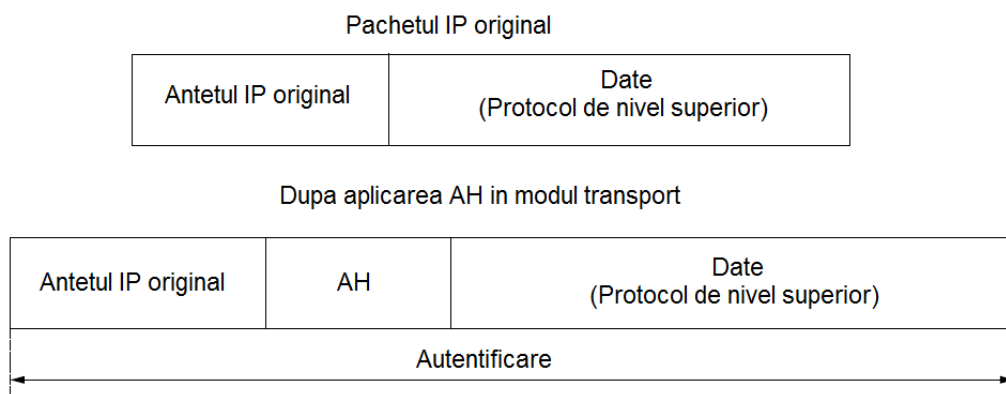


Figura 4-4. AH în modul transport

În cazul în care se folosește ESP în modul transport, lucrurile decurg în mod similar. Transformările criptografice se aplică asupra payload-ului, antetul IP original păstrându-se (vezi Figura 4-5). În cazul în care pachetul transportă un payload de tip TCP, valoarea câmpului Protocol din antetul IP este 6. După aplicarea ESP, valoarea câmpului Protocol din antetul IP devine 50 iar valoarea câmpului Next Header din trailer-ul ESP devine 6.

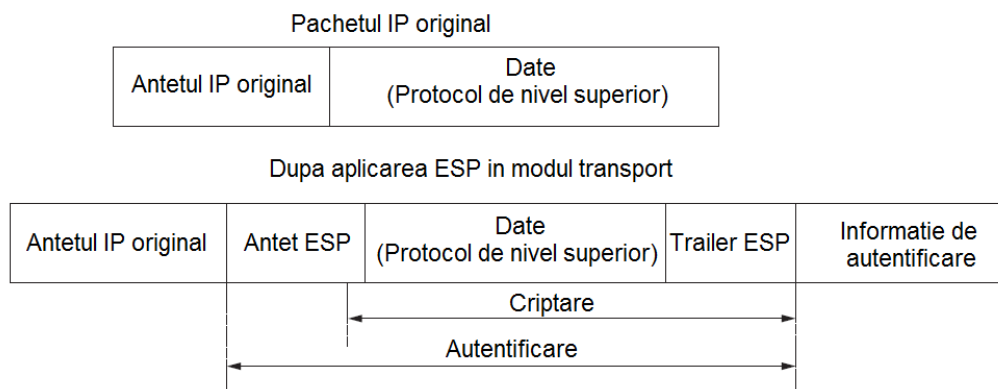


Figura 4-5. ESP în modul transport

Modul tunel este utilizat cu precădere pentru interconecarea a două subrețele între ele, tunelul IPSec stabilindu-se între gateway-uri ca în Figura 4-6. Modul tunel poate fi însă utilizat și pentru a conecta un calculator la o subrețea sau a două calculatoare între ele.

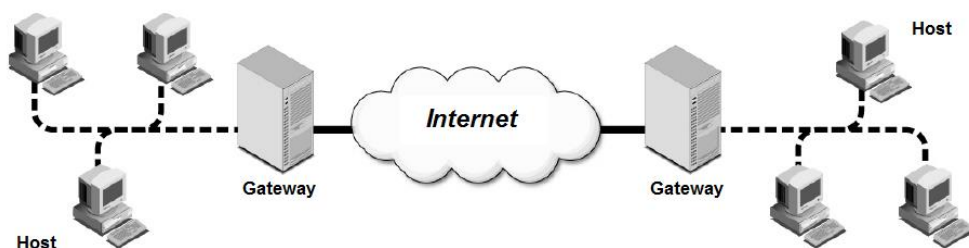


Figura 4-6. Scenariu de utilizare a modului tunel

Dacă un calculator din subrețeaua din stânga transmite un pachet pentru un calculator din subrețeaua din dreapta, iar la nivelul gateway-urilor este activat protocolul AH în modul tunel, atunci pachetul original este tratat de gateway ca payload și încapsulat într-un alt pachet IP (vezi Figura 4-7). Practic, prin folosirea AH în modul tunel se creează un nou antet IP pentru fiecare pachet. Valoarea câmpului Protocol din noul antet IP devine 51 iar valoarea câmpului Next Header din antetul AH devine 4.

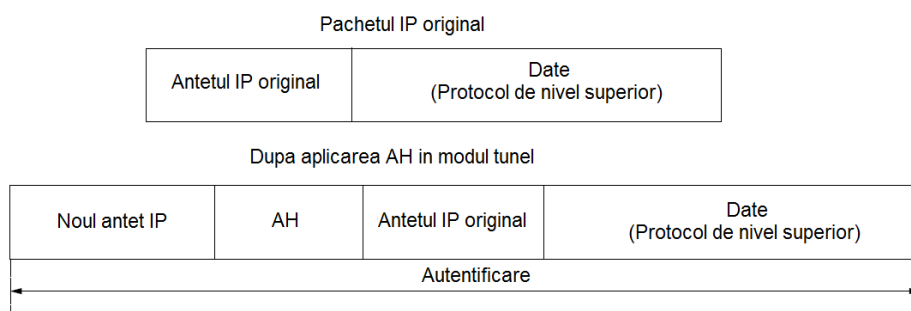


Figura 4-7. AH în modul tunel

Dacă se folosește ESP în modul tunel, lucrurile decurg în mod similar. Pachetul original este tratat ca payload și încapsulat într-un alt pachet IP (vezi Figura 4-8). Valoarea câmpului Protocol din noul antet IP devine 50 iar valoarea câmpului Next Header din trailerul ESP devine 4.

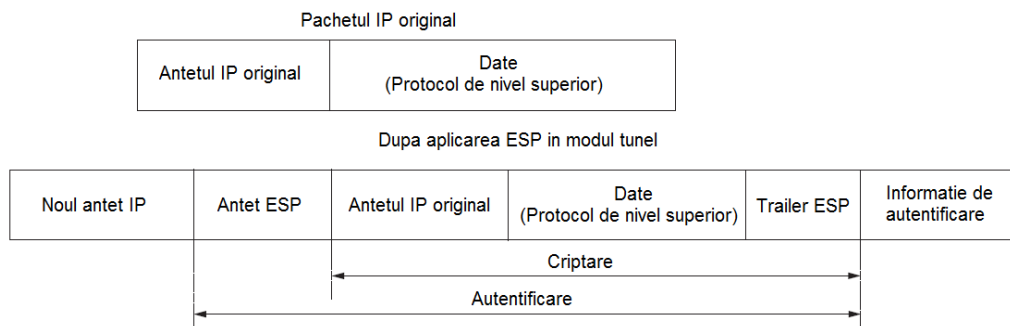


Figura 4-8. ESP în modul tunel

4.4. Protocolul Internet Key Exchange (IKE)

Rolul protocolului IKE este de a negocia, crea și gestiona asocieri de securitate. *Asocierile de Securitate (SA - Security Association)* reprezintă un termen generic pentru un set de valori ce definesc caracteristicile și transformările criptografice aplicate unei conexiuni IPsec. SA-urile pot fi create manual, folosind valori stabilite în avans între părți, însă aceste asocieri nu pot fi actualizate. Această metodă nu este scalabilă la scară largă. IKE permite crearea de SA-uri în mod automat, la stabilirea conexiunii IPsec și actualizarea în timp a acestora.

IKE este o combinație de două protocoale: Internet Security Association and Key Management Protocol (ISAKMP) și Oakley Key Exchange Protocol. Mesajele IKE sunt transmise prin intermediul UDP pe portul 500.

IKE operează în două faze. În prima fază între capetele tunelului IPsec se stabilește un SA inițial, care poartă denumirea de IKE SA. Rolul IKE SA este de a permite criptarea și autentificarea schimburilor ulterioare de mesaje din cadrul protocolului. IKE SA este bidirecțional ceea ce înseamnă că aceleași transformări criptografice se aplică atât pe fluxul de transmisie cât și pe cel de recepție. Pentru stabilirea IKE SA se poate folosi modul principal sau modul agresiv din IKE.

În faza a doua a protocolului se negociază un SA general care va fi folosit efectiv pentru protecția pachetelor transmise între capetele tunelului IPsec. Acest SA poartă denumirea de IPsec SA. Spre deosebire de IKE SA, IPsec SA este unidirecțional, ceea ce înseamnă că pentru o conexiune full duplex este nevoie de două SA-uri, câte una pentru fiecare flux. Pentru stabilirea IPsec SA se folosește modul rapid din IKE.

4.4.1. Modul principal

Modul principal constă din trei perechi de mesaje transmise între capetele tunelului (*inițiator* și *responder* conform terminologiei IPsec), după cum se poate observa în Figura 4-9.

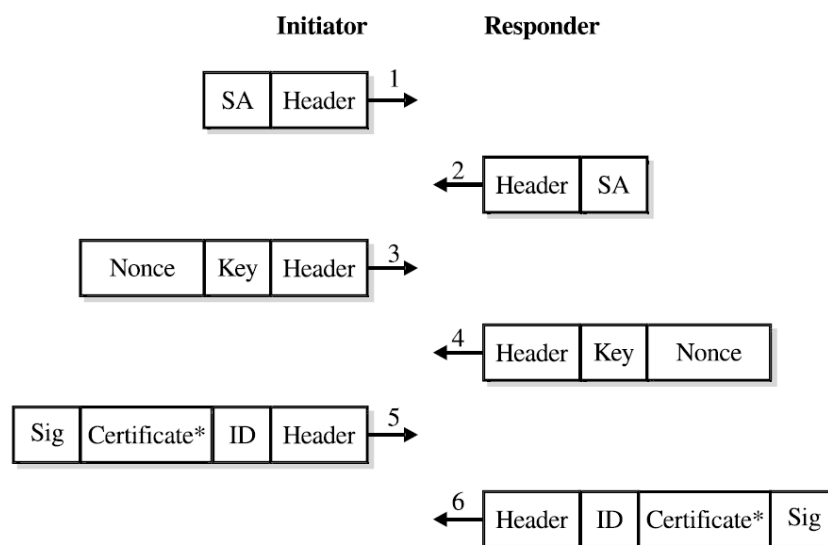


Figura 4-9. Modul principal al IKE

În primul schimb de mesaje se fac propuneri cu privire la parametrii ce vor fi folosiți pentru IKE SA. Următorii patru parametrii sunt obligatorii și poartă denumirea de suită de protecție (protection suite):

- *Algoritmul de criptare* (AES-CBC, AES-GCM sau 3DES-CBC);
- *Algoritmul pentru calculul MAC* (HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384 sau HMAC-SHA-512);
- *Metoda de autentificare*. În cadrul IPSec se pot folosi pentru autentificarea entităților parole partajate (pre-shared keys) sau certificate digitale. Anumite implementări pot suporta folosirea serviciilor externe de autentificare cum ar fi Kerberos.
- *Parametrii Diffie-Hellman (DH)*. Stabilirea cheii de sesiune în IPSec se face prin intermediul protocolului Diffie-Hellman.

Pe lângă acești parametri, între entități sunt transmise câte un cookie. Aceste cookie-uri oferă protecție împotriva atacurilor de tip blocare a serviciului (denial of service).

În al doilea schimb de mesaje are loc stabilirea cheii de sesiune prin intermediul Diffie-Hellman, folosind parametrii stabiliți în etapa anterioară. De asemenea, fiecare entitate transmite celeilalte o valoare aleatoare (nonce) folosită pentru a evita atacurile prin reluare.

În cel de-al treilea schimb de mesaje, entitățile se autentifică între ele. În Figura 4-9, autentificarea se face folosind certificate digitale. În acest sens, fiecare entitate trimite către cealaltă certificatul său digital și o semnătură pentru a face dovada posesiei cheii private asociate cheii publice din certificat. Semnătura se calculează peste o serie de valori specifice conexiunii și nonce.

În cazul în care autentificarea se face folosind parole partajate (pre-shared keys) atunci fiecare entitate transmite ID-ul propriu și hash de parolă și nonce.

Indiferent de metoda de autentificare folosită, ultimul schimb de mesaje este criptat folosind algoritmul și cheia de sesiune stabilite în etapele anterioare. În acest mod, identitatea entităților între care se stabilește tunelul IPSec este protejată.

4.4.2. Modul agresiv

Modul agresiv este mai rapid decât modul principal, fiind format din trei mesaje în loc de șase (vezi Figura 4-10). Din punct de vedere funcțional, modul agresiv asigură același lucru ca și modul principal: stabilirea suitei de protecție, stabilirea cheii de sesiune și autentificarea entităților.

Avantajul modului agresiv este viteza însă din punct de vedere al securității, modul agresiv nu asigură protecția identității capetelor tunelului IPSec. Dacă cineva monitorizează comunicația poate determina identitatea entităților între care se stabilește tunelul IPSec.

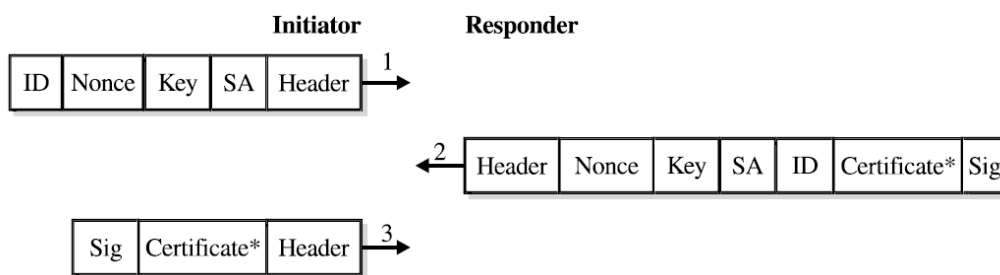


Figura 4-10. Modul agresiv al IKE

4.4.3. Modul rapid

Modul rapid presupune transmiterea a trei mesaje între capetele tunelului pentru a stabili IPSec SA (vezi Figura 4-11). Comunicația din modul rapid este criptată folosind IKE SA negociată în faza 1.

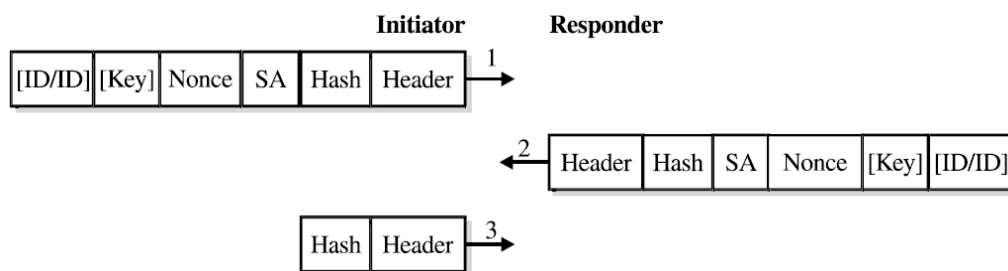


Figura 4-11. Modul rapid al IKE

IPSec SA fiind unidirecțional, este nevoie de două SA-uri, unul pentru fluxul de ieșire și unul pentru fluxul de intrare. Fiecare IPSec SA are asociat un timp de viață. În momentul în care acesta expiră, se negociază un nou IPSec SA folosind tot modul rapid.

Asocierile de securitate active sunt stocate într-o bază de date numită Security Association Database (SAD). SAD conține următoarele informații pentru fiecare conexiune IPSec:

- Adresa IP sursă
- Adresa IP destinație
- SPI
- Protocolul de securitate IPSec (AH sau ESP)
- Modul de operare (transport sau tunel)
- Algoritmul de criptare pentru ESP
- Algoritmul pentru calculul MAC
- Cheile secrete folosite în cadrul algoritmilor selectați
- Lungimea cheii, algoritmi selectați pot folosi lungimi de cheie diferite
- Durata de viață a SA
- Informații cu privire la numărul de secvență
- Tipurile de trafic protejate folosind acest SA

Un SA poate fi identificat în mod unic prin combinația a trei parametri: adresa IP destinație, SPI și protocolul de securitate IPsec. Atunci când o stație are nevoie să știe ce SA se aplică unui anumit pachet, caută în baza de date SA-ul cu parametrii corespunzători pachetului.

SA-ul descrie măsurile de securitate pe care IPSec ar trebui să le folosească pentru protejarea comunicației, însă nu specifică ce tip de trafic ar trebui protejat și în ce circumstanțe. Aceste informații sunt stocate în baza de date de politici de securitate (Security Policy Database). Această bază de date conține următoarele informații cu privire la traficul ce trebuie protejat:

- Adresele IP sursă și destinație
- Protocolul (TCP, UDP, etc)
- Porturile sursă și destinație (optional)
- Mecanismul de securitate aplicate
- Un pointer către SA-ul din SAD, dacă a fost negociat deja un SA pentru acest tip de trafic.

SA-urile au o durată de viață limitată, care nu poate fi prelungită. Când un SA se apropie de sfârșitul perioadei de viață, părțile ce comunică trebuie să creeze un nou SA și să îl folosească pe acesta. Procesul poartă denumirea de rekeying, deoarece scopul principal îl reprezintă generarea unei noi chei de sesiune. Durata de viață a SA-ului specifică cât de des trebuie actualizat un SA, bazat pe timpul scurs sau pe cantitatea de trafic.

4.5. Rețele private virtuale (VPN)

Cea mai importantă aplicație a IPSec o reprezintă implementarea de Rețele Private Virtuale (VPN - Virtual Private Network). Un VPN este o rețea virtuală, construită peste o rețea fizică, ce pune la dispoziție mecanisme sigure de comunicație între calculatoare. Deoarece VPN poate fi implementat peste rețelele existente, cum ar fi Internetul, acesta poate fi folosit pentru transferul securizat al datelor prin intermediul rețelilor publice. Prin urmare, VPN reprezintă o alternativă mai puțin costisitoare la liniile de comunicație dedicate folosite pentru interconectarea filialelor unei organizații sau pentru accesul utilizatorilor de la distanță. În plus, având în vedere aria globală de acoperire a Internetului, VPN permite interconectarea sistemelor indiferent de locație.

În ciuda acestor avantaje evidente, este important de înțeles faptul că VPN nu elimină toate riscurile de securitate specifice interconectării. O problemă potențială o reprezintă utilizarea incorectă a IPSec. După cum se poate observa din secțiunile anterioare, IPSec este un protocol complex și prin urmare este posibil să apară greșeli în momentul instalării și configurării soluției VPN. De asemenea, slăbiciunile algoritmilor criptografici, vulnerabilitățile din implementările software ale acestora sau folosirea de generatoare de numere aleatoare mai puțin sigure pot permite atacatorilor să intercepteze sau să modifice traficul. Managementul deficitar al cheilor criptografice poate, de asemenea, să creeze probleme. Compromiterea unei chei permite unui eventual atacator să decripteze traficul sau să se dea drept utilizator legitim. Disponibilitatea rețelei reprezintă un alt risc potențial din moment ce implementarea unui VPN presupune adăugarea de componente și servicii suplimentare la infrastructura de rețea existentă. Toate aceste riscuri trebuie controlate în mod corespunzător prin alegerea celui mai potrivit model arhitectural de VPN și implementarea corectă a acestuia.

Următoarele secțiuni prezintă cele trei arhitecturi VPN de bază care pot fi implementate prin intermediul IPSec: gateway-la-gateway, host-la-gateway și host-la-host.

4.5.1. Arhitectura gateway-la-gateway

Această arhitectură de VPN este folosită de regulă pentru securizarea comunicațiilor între două rețele. Acest lucru se realizează prin dispunerea câte unui gateway VPN în fiecare rețea și stabilirea unui tunel IPSec între cele două gateway-uri. Astfel, traficul dintre cele două rețele va fi protejat, fiind transmis prin intermediul tunelului IPSec stabilit între gateway-uri. Gateway-ul VPN poate fi un dispozitiv dedicat, ce îndeplinește exclusiv funcții specifice VPN sau poate fi un dispozitiv de rețea ce îndeplinește mai multe funcții, cum ar fi un router sau un firewall. Figura 4-12 prezintă un exemplu de arhitectură VPN de tipul gateway-la-gateway.



Figura 4-12. Exemplu de arhitectura de tip gateway-la-gateway

Pentru crearea tunelului, unul dintre gateway-urile trimite către celălalt o cerere pentru stabilirea conexiunii IPSec. Cele două gateway-uri vor face schimb de informații între ele și apoi stabilesc conexiunea IPSec. Odată stabilită conexiunea, tot traficul transmis între rețele este protejat. În momentul în care o stație dintr-o rețea dorește să comunice cu altă stație din cealaltă rețea, pachetele sunt rutate în mod automat prin conexiunea IPSec. Pentru asta, tabelele de rutare de pe stații trebuie să fie configurate în mod corespunzător (gateway-ul VPN trebuie setat ca și default gateway pe stații). Se poate opta pentru varianta în care se stabilește un singur tunel IPSec între gateway-uri care să suporte toate tipurile de trafic dintre cele două rețele, sau pentru tunele IPSec multiple, fiecare protejând tipuri diferite de trafic.

Arhitecturile VPN de tip gateway-la-gateway se caracterizează prin faptul că nu oferă protecția datelor pe întregul traseu parcurs de acestea, lucru scos în evidență și în Figura 4-12. Traficul circulă în clar de la stația sursă până la gateway-ul VPN, apoi criptat prin tunelul IPSec și din nou în clar de la gateway-ul VPN până la stația destinație.

Acest model arhitectural este folosit pentru interconectarea filialelor unei organizații cu sediul central sau pentru interconectarea a două organizații între ele. Din această cauză modelul mai poartă denumirea și de *site-to-site VPN*.

Din perspectiva managementului utilizatorilor și stațiilor de lucru, arhitectura de tip gateway-la-gateway este ușor de implementat și administrat. VPN-urile gateway-la-gateway sunt transparente pentru utilizatori, care nu trebuie să se autentifice separat pentru a putea folosi VPN-ul. În plus, utilizatorii nu trebuie să folosească vreun software special pentru a se conecta prin VPN.

4.5.2. Arhitectura host-la-gateway

Arhitectura de tip host-la-gateway este folosită tot mai des în ultima vreme în cadrul organizațiilor care doresc să ofere angajaților accesul securizat de la distanță la resursele informatice aflate în rețeaua internă a organizației. Pentru aceasta, organizația are nevoie de un gateway VPN, care poate fi un dispozitiv dedicat sau un echipament ce îndeplinește mai multe funcții de rețea. Stabilirea tunelului IPSec se face între stația angajatului aflat la distanță și gateway-ul VPN. Figura 4-13 prezintă un exemplu de arhitectură de tip host-la-gateway ce furnizează acces securizat pentru utilizatorii aflați la distanță.

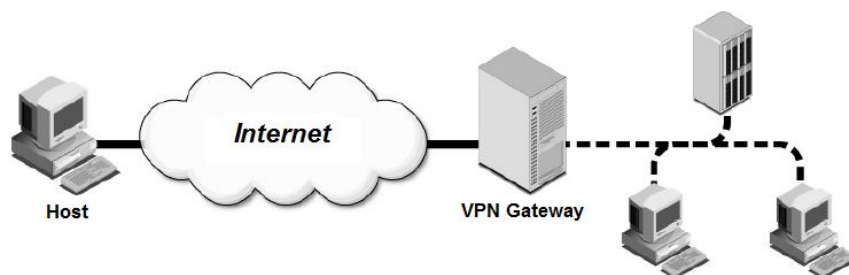


Figura 4-13. Exemplu de arhitectura de tip host-la-gateway

Conexiunile IPSec sunt stabilite la cererea fiecărui utilizator în parte. Stațiile utilizatorilor trebuie configurate astfel încât să se comporte ca și clienți în raport cu gateway-ul VPN. Când un utilizator inițiază o conexiune IPSec, acesta trebuie mai întâi să se autentifice. Autentificarea poate fi realizată de gateway-ul VPN sau de către un server de autentificare dedicat. Dacă datele de autentificare sunt valide, conexiunea IPSec este stabilită, iar traficul dintre stația utilizatorului și gateway-ul VPN este protejat. Din acest moment, utilizatorul are acces la resursele informatice din rețeaua internă a organizației.

Ca și în cazul VPN-urilor de tip gateway-la-gateway și în acest caz traficul nu este în întregime protejat. Datele circulă criptate de la stația sursă până la gateway-ul VPN, prin tunelul IPSec și în clar de la gateway-ul VPN până la serverul destinație prin rețeaua internă.

Modelul arhitectural host-la-gateway este folosit cel mai adesea pentru conectarea utilizatorilor de la distanță la resursele interne ale organizației, prin intermediul Internetului. Din această cauză modelul mai poartă denumirea și de *remote access VPN*.

Soluțiile de tip host-la-gateway sunt mai complexe de implementat și administrat. În plus, aceste soluții nu sunt transparente pentru utilizatori. Pe stația utilizatorilor trebuie instalat clientul de VPN iar aceștia trebuie să se autentifice pentru stabilirea unei conexiunii VPN.

4.5.3. Arhitectura host-la-host

Arhitectura de tip host-la-host este folosită mai rar, în situații speciale cum ar fi un administrator de sistem care trebuie să configureze de la distanță un server. În acest caz, serverul organizației trebuie să fie configurat să ofere servicii VPN iar stația administratorului are rolul de client VPN. Administratorul de sistem va folosi clientul VPN atunci când are nevoie să stabilească o conexiune

criptată cu serverul aflat la distanță. Figura 4-14 prezintă un exemplu de arhitectură de tip host-la-host care permite conectarea securizată a unui utilizator la un server.

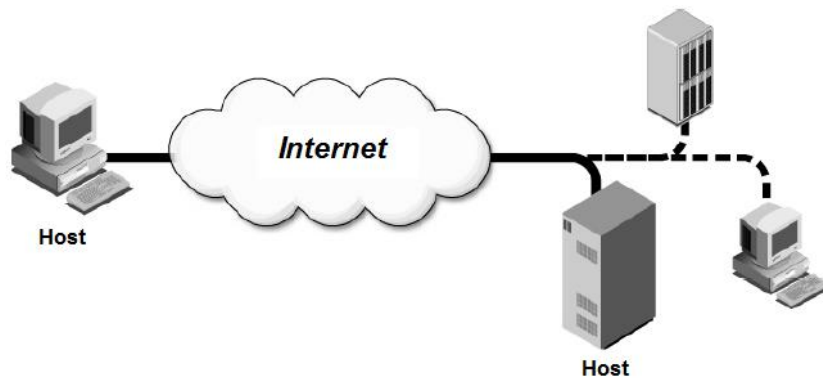


Figura 4-14. Exemplu de arhitectura de tip host-la-gateway

În această arhitectură, conexiunile IPSec sunt create pentru fiecare utilizator în parte. Stațiile utilizatorilor trebuie să fie configurate să funcționeze ca și clienți în raport cu serverul. Când un utilizator dorește să acceseze resurse aflate pe server, acesta trebuie să se autentifice pentru stabilirea conexiunii. Din acest moment, tot traficul dintre stația utilizatorului și server va fi protejat.

Traficul este protejat în întregime în acest caz. Acest lucru poate fi însă un dezavantaj deoarece sistemele de protecție de tip firewall sau IDS care acționează la nivel rețea nu pot inspecta pachetele transmise între stație și server.

Această arhitectură este folosită de utilizatori considerați de încredere de către organizație, care au nevoie să administreze sau să folosească de la distanță anumite echipamente.

5. Protocoloale de securitate la nivel transport

Serviciile standard de rețea (Web, E-mail, FTP, Telnet, etc) sunt dezvoltate folosind modelul client-server, bazat pe paradigma cerere-răspuns. Comunicația între entități se realizează prin intermediul socket-urilor și este în clar, putând fi ușor interceptată sau modificată.

Secure Sockets Layer (SSL) și varianta îmbunătățită a acestuia, Transport Layer Security (TLS), permit stabilirea unui canal de comunicație sigur între două entități (procesele client și server). Acest capitol prezintă detalii cu privire la arhitectura și modul de funcționare al celor două protocoale precum și o serie de aplicații ale acestora.

5.1. Secure Sockets Layer (SSL)

Protocolul SSL a fost dezvoltat de către compania Netscape Communications în 1994. La acea vreme, peste World Wide Web (WWW) începuse să se dezvolte o serie de aplicații de comerț electronic care necesitau mecanisme sigure de comunicație. Protocolul SSL avea ca scop asigurarea confidențialității, integrității și autenticității datelor transmise între browserele și serverele de Web.

SSLv1 a fost folosit intern în cadrul companiei Netscape și s-a dovedit a fi inefficient. Prin urmare, a fost dezvoltată o nouă versiune de protocol, SSLv2 care a fost încorporată în browser-ul de Web produs de către firmă, Netscape Navigator.

După lansarea versiunii 2.0, protocolul SSL a devenit standard de facto. De-a lungul timpului, SSL a fost îmbunătățit, iar în 1995 a fost lansat pe piață SSLv3. Aceasta a corectat problemele de securitate ale SSLv2. În plus, SSLv3 permite compresia datelor, folosirea mecanismelor Diffie-Hellman și Fortezza pentru stabilirea cheii de sesiune și adaugă suport pentru certificate non-RSA.

În 1996, în cadrul Internet Engineering Task Force (IETF) ia ființă grupul de lucru Transport Layer Security (TLS) având ca obiectiv standardizarea unui protocol de securitate la nivel transport. Prima versiune oficială a protocolului TLS a fost publicată în 1999, prin RFC 2246. În principiu TLS este asemănător cu SSLv3 cu câteva excepții și adăugiri.

SSL este un protocol independent de platformă, utilizat pentru securizarea aplicațiilor client-server bazate pe protocolul TCP. SSL se interpune între nivelul transport (TCP) și nivelul aplicație punând la dispoziția aplicațiilor socket-uri de comunicație securizate. SSL este format din mai multe protocoale, fiecare având roluri funcționale distincte. Figura 5-1 prezintă poziționarea SSL în stiva TCP/IP.

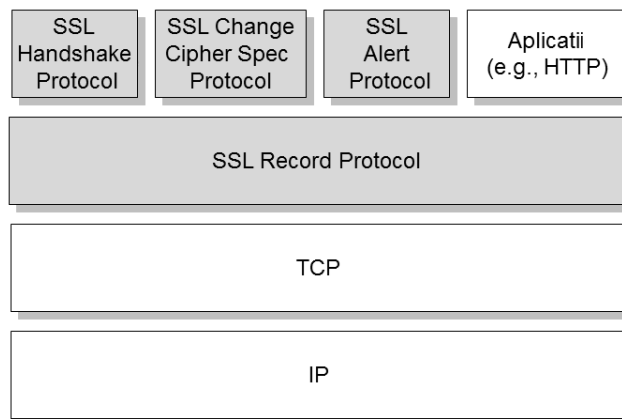


Figura 5-1. Suita de protocoale SSL

SSL Handshake Protocol (Protocolul de Negociere a Conexiunii) se ocupă cu negocierea algoritmilor și parametrilor criptografici, autentificarea entităților și stabilirea unei chei secrete de sesiune.

SSL Change Cipher Spec Protocol (Protocolul de Tranziție Criptografică) are rolul de a indica începutul comunicației securizate între client și server, folosind algoritmi și cheile negociate în faza de handshake.

SSL Record Protocol (Protocolul de Transfer al Datelor) este responsabil de securizarea efectivă a comunicației dintre client și server asigurând fragmentarea, compresia, integritatea, autenticitatea și criptarea datelor.

SSL Alert Protocol (Protocolul de Alertare) este folosit pentru transmiterea mesajelor de eroare apărute în cadrul protocolului.

SSL permite folosirea următorilor algoritmi criptografici simetrici pentru criptarea datelor de la nivel aplicație:

- Cifruri bloc: RC2_40, DES_40, DES_56, 3DES_168, IDEA_128, Fortezza_80. Pentru toți algoritmi, modul de criptare ales este CBC.
- Cifruri șir: RC4_40, RC4_128.

După cum se poate observa, în lista algoritmi criptografici simetrici suportați de SSL regăsim și algoritmi având lungimi de chei de 40 de biți. Acest lucru se datorează faptului că specificațiile protocolului SSL au fost elaborate înainte de anii 2000 când în SUA erau încă în vigoare restricțiile privind exportul de software criptografic.

Asigurarea integrității și autenticității datelor se face prin intermediul MAC-urilor (Message Authentication Code). Ca și funcție hash pentru calculul MAC se poate folosi atât MD5 cât și SHA-1.

Pentru stabilirea cheii de sesiune între client și server, SSL suportă următoarele metode:

- *RSA* – cheia de sesiune este criptată cu cheia publică RSA a serverului. Cheia publică RSA a serverului trebuie să fie disponibilă sub forma unui certificat digital emis de o Autoritate de Certificare.
- *Fixed Diffie-Hellman* – schimb de chei bazat pe Diffie-Hellman în care certificatul serverului conține parametri publici Diffie-Hellman. Dacă este necesară autentificarea clientului, certificatul digital al acestuia conține, de asemenea, parametri publici Diffie-Hellman. Altfel, parametri Diffie-Hellman ai clientului sunt generați ad-hoc și transmiși serverului.
- *Ephemeral Diffie-Hellman* – schimb de chei bazat pe Diffie-Hellman în care parametrii publici Diffie-Hellman sunt generați ad-hoc atât de către client cât și de către server. Serverul semnează parametrii săi Diffie-Hellman folosind cheia privată de semnătură RSA sau DSA. De asemenea, clientul poate semna parametrii săi Diffie-Hellman folosind tot o cheie privată de semnătură. Validarea semnăturilor se face folosind cheile publice din certificatele digitale emise de Autoritatea de Certificare.
- *Anonymous Diffie-Hellman* – schimb de chei bazat pe Diffie-Hellman în care parametri publici Diffie-Hellman sunt generați ad-hoc și transmiși fără autentificare. Această abordare este vulnerabilă la atacuri de tip man-in-the-middle, în care atacatorul poate face schimburi de chei anonime cu ambele părți.
- *Fortezza* – schemă proprietară folosită în domeniul guvernamental și militar.

În continuare, sunt prezentate detalii cu privire la fiecare componentă din cadrul protocolului SSL.

5.1.1. SSL Handshake Protocol

SSL Handshake Protocol reprezintă cea mai complexă parte a SSL-ului. Acest protocol asigură negocierea algoritmilor și parametrilor criptografici, autentificarea dintre server și client și stabilirea unei chei secrete de sesiune. SSL Handshake Protocol este utilizat în faza inițială, înaintea oricărui transfer de date și constă dintr-o serie de mesaje transmise între client și server. Formatul acestor mesaje este prezentat în Figura 5-2. Primul câmp reprezintă tipul mesajului, cel de-al doilea lungimea după care urmează conținutul propriu-zis.



Figura 5-2. Formatul mesajelor transmise în faza de negociere

Schimbul de mesaje inițial, necesar pentru stabilirea conexiunii SSL între client și server, este prezentat în Figura 5-3.

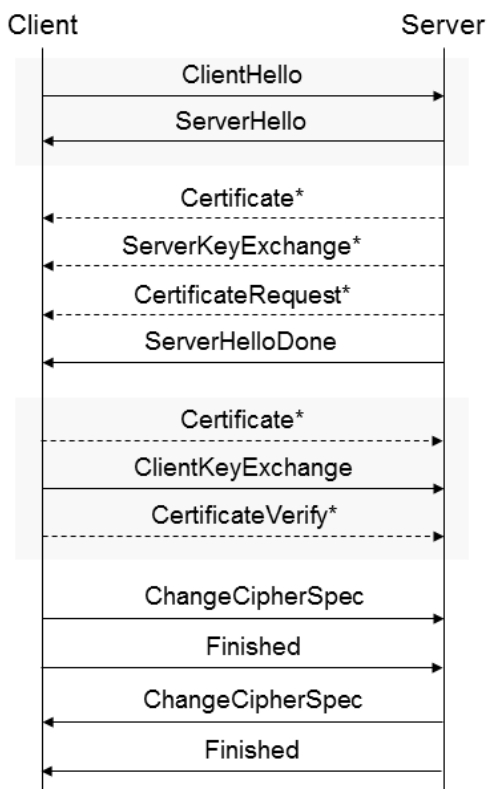


Figura 5-3. Schimbul de mesaje din cadrul fazei de negociere

ClientHello

SSL Handshake Protocol debutează prin transmiterea mesajului ClientHello de către client. Rolul acestui mesaj este de a informa serverul asupra versiunii de protocol și algoritmilor criptografici suportați de client. Mesajul ClientHello conține următoarele câmpuri:

- *Client_version* – cea mai mare versiune de SSL suportată de client.
- *Random* – o valoare aleatoare generată de client, constând dintr-o amprentă de timp de 4 octeți și 28 de octeți generați în mod aleator. Aceste valori servesc drept nonce-uri și sunt utilizate în timpul schimbului de chei pentru prevenirea atacurilor prin reluare.
- *Session_id* – un identificator al sesiunii având lungime variabilă. O valoare diferită de 0 indică intenția clientului de reluare a unei conexiuni existente, sau de creare a unei noi sesiuni, în cazul în care valoarea Session_id este 0.
- *Cipher_suites* – lista algoritmilor criptografici suportați de client, în ordine descrescătoare a preferinței. Fiecare element din listă definește o suită constând dintr-un algoritm de schimb de chei, un algoritm de criptare și o funcție hash pentru calculul MAC. De exemplu, prima suită din lista:

SSL_RSA_WITH_3DES_EDE_CBC_SHA

SSL_RSA_WITH_RC4_128_MD5

SSL_RSA_WITH_IDEA_CBC_SHA

semnifică faptul că schimbul de chei se va realiza prin anvelopare cu RSA, criptarea datelor se va face folosind algoritmul 3DES în varianta EDE, modul de criptare fiind CBC iar funcția hash pentru calculul MAC este SHA-1.

- *Compression_methods* – lista metodelor de compresie suportate de client, în ordine descrescătoare a preferinței. Toate implementările de SSL trebuie să suporte măcar metoda Null, adică fără compresie.

După transmiterea mesajului ClientHello, clientul așteaptă mesajul ServerHello de la server.

ServerHello

Mesajul ServerHello conține aceleași câmpuri ca și mesajul ClientHello. Diferența constă în faptul că mesajul ClientHello este folosit pentru a informa serverul despre capacitățile criptografice ale clientului pe când mesajul

ServerHello este folosit de server pentru a decide ce algoritmi criptografici vor fi folosiți efectiv.

- *Client_version* – cea mai mare versiune de SSL suportată atât de client cât și de server.
- *Random* – o valoare aleatoare generată de server, având aceeași structură cu cea generată de client. Ea trebuie să fie independentă și diferită de valoarea transmisă de client.
- *Session_id* – identificatorul sesiunii. Dacă clientul a cerut reluarea unei sesiuni mai vechi și acest lucru este posibil, atunci câmpul *Session_id* conține identificatorul acelei sesiuni. Altfel, câmpul *SessionID* conține o valoare asignată de server corespunzătoare noii sesiuni.
- *Cipher_suite* – suita selectată de server din lista algoritmilor criptografici propuși de către client.
- *Compression_method* – metoda selectată de server din lista metodelor de compresie propuse de către client.

Server Certificate

Imediat după stabilirea în comun a algoritmilor criptografici ce urmează a fi folosiți în cadrul protocolului, serverul transmite certificatul său pentru autentificare. Mesajul conține certificatul serverului în format X.509 sau un lanț de certificare. Mesajul este obligatoriu pentru oricare dintre metodele de schimb de chei, cu excepția Anonymous Diffie-Hellman.

ServerKeyExchange

Acest mesaj este transmis numai dacă mesajul anterior (Server Certificate) nu conține suficiente informații pentru a încheia schimbul de chei. De exemplu, dacă schimbul de chei se face prin Anonymous Diffie-Hellman, atunci acest mesaj conține parametri publici Diffie-Hellman ai serverului.

CertificateRequest

Funcție de modul de configurare sau implementare, serverul poate cere clientului să se autentifice. Mesajul *CertificateRequest* include două câmpuri: *certificate_types* și *certificate_authorities*. Câmpul *certificate_types* indică tipul de certificat cerut. Mesajul *certificate_authorities* conține o listă cu numele Autorităților de Certificare acceptate de server.

ServerHelloDone

Așa cum sugerează și numele, prin intermediul mesajului *ServerHelloDone* serverul notifică clientul că a încheiat faza de hello. După transmiterea acestui mesaj, serverul așteaptă răspunsul clientului. Mesajul *ServerHelloDone* nu are parametri.

După primirea mesajului *ServerHelloDone*, clientul trebuie să verifice că certificatul transmis de server este valid și că ceilalți parametri recepționați sunt în regulă. Dacă totul a decurs normal, clientul continuă protocolul transmițând răspunsurile sale către server.

Client Certificate

Dacă serverul a cerut clientului să se autentifice, atunci acesta transmite serverului certificatul său digital. Dacă clientul nu deține un certificat potrivit, se trimite o alertă de tipul *No_certificate* rămânând la latitudinea serverului să decidă dacă va continua sau va întrerupe comunicația.

ClientKeyExchange

Conținutul mesajului *ClientKeyExchange* depinde de metoda selectată pentru schimbul de chei. În cazul în care metoda este RSA, *ClientKeyExchange* conține cheia de sesiune criptată cu cheia publică RSA a serverului. Numai serverul care conține cheia privată asociată poate extrage cheia de sesiune. În acest fel, serverul face și dovada posesiei cheii private asociată cheii publice din certificat. În cazul Ephemeral Diffie-Hellman sau Anonymous Diffie-Hellman, clientul transmite parametrii publici Diffie-Hellman.

CertificateVerify

Prin intermediul acestui mesaj, clientul face dovada posesiei cheii private asociată cheii publice din certificat. În acest sens, clientul semnează o serie de parametri stabiliți în cadrul protocolului iar serverul verifică această semnătură. Mesajul este transmis numai dacă s-a cerut autentificarea clientului cu certificat digital.

ChangeCipherSpec

Pentru a semnala faptul că este gata să transfere date criptat, clientul transmite mesajul *ChangeCipherSpec*. Acest mesaj nu face parte din SSL Handshake Protocol ci este transmis prin intermediul SSL Change CipherSpec Protocol.

Finished

Mesajul Finished este transmis la sfârșitul fazei de negociere pentru a verifica dacă schimbul de chei și autentificarea au fost realizate cu succes. Mesajul Finished este primul mesaj protejat cu algoritmi negociați în faza de handshake.

Ca răspuns la aceste două mesaje, serverul transmite la rândul său ChangeCipherSpec și Finished. Din acest moment negocierea este completă, iar clientul și serverul pot începe transferul criptat al datelor folosind SSL Record Protocol.

După ce datele au fost transferate, sesiunea TCP este închisă. Deoarece nu există o legătură între TCP și SSL, starea SSL poate fi menținută. Pentru comunicații ulterioare între client și server, mulți dintre parametri negociați pot fi reutilizați. Dacă un client dorește să reia transferul de date, acesta nu trebuie să reia negocierea algoritmilor de criptare sau a cheilor de sesiune. Conform specificațiilor, informațiile de stare nu trebuie păstrate mai mult de 24 de ore. Dacă o sesiune nu e reluată în acest timp toate informațiile sunt șterse. Specificațiile recomandă ca nici clientul, nici serverul să nu rețină informațiile în cazul în care suspectează faptul că cheile de criptare ar fi compromise. Dacă clientul sau serverul nu este de acord cu reluarea sesiunii, din orice motiv, atunci negocierea trebuie reluată de la început.

La sfârșitul fazei de negociere, între client și server se stabilește o cheie secretă de sesiune numită *pre-master secret*. Din acest secret se derivează un *master secret* de 48 de octeți prin concatenarea a 3 hash-uri MD5, obținute folosind formula:

```
master_secret = MD5(pre_master_secret + SHA('A' +
pre_master_secret + ClientHello.random +
ServerHello.random)) + MD5(pre_master_secret + SHA('BB' +
pre_master_secret + ClientHello.random +
ServerHello.random)) + MD5(pre_master_secret + SHA('CCC' +
pre_master_secret + ClientHello.random +
ServerHello.random));
```

Din *master secret* se obține blocul de chei tot prin concatenarea de hash-uri MD5, conform formulei:

```
key_block = MD5(master_secret + SHA('A' + master_secret +
ServerHello.random + ClientHello.random)) +
MD5(master_secret + SHA('BB' + master_secret +
ServerHello.random + ClientHello.random)) +
MD5(master_secret + SHA('CCC' + master_secret +
ServerHello.random + ClientHello.random)) + [...];
```

Blocul de chei este apoi partiționat astfel:

```
client_write_MAC_secret[CipherSpec.hash_size]
server_write_MAC_secret[CipherSpec.hash_size]
client_write_key[CipherSpec.key_material]
server_write_key[CipherSpec.key_material]
client_write_IV[CipherSpec.IV_size]
server_write_IV[CipherSpec.IV_size]
```

Comunicația între client și server fiind în ambele sensuri simultan (full duplex) este nevoie de câte un set de chei pentru fiecare sens. Client_write_MAC_secret este folosit de client pentru a calcula MAC-ul datelor transmise către server și trebuie să fie identic cu cel folosit de server pentru a verifica MAC-ul datelor recepționate. În mod similar, Server_write_MAC_secret este folosit de server pentru a calcula MAC-ul datelor transmise către client și trebuie să fie identic cu cel folosit de client pentru a verifica MAC-ul datelor recepționate. Client_write_key este cheia secretă folosită de client pentru criptarea datelor transmise. Client_write_IV este vectorul de inițializare folosit de client pentru modul de criptare CBC.

5.1.2. SSL Change Cipher Spec Protocol

SSL Change Cipher Spec Protocol este cel mai simplu protocol din cadrul SSL având rolul de a semnaliza tranziția la comunicație criptată în cadrul conexiunii. SSL Change Cipher Spec Protocol constă într-un singur mesaj, un octet cu valoarea 1, transmis la încheierea fazei de negociere atât de client cât și de server pentru a indica faptul că în continuare, blocurile de date vor fi criptate folosind algoritmi și cheile negociate.

5.1.3. SSL Record Protocol

Datele transmise de nivelul aplicație sunt procesate de către SSL Record Protocol. Acesta oferă două servicii de securitate de bază:

- integritate și autenticitate prin utilizarea codurilor de autentificare a mesajelor (MAC – Message Authentication Code);
- confidențialitate prin criptare cu algoritmi simetrici.

Opțional, SSL Record Protocol poate asigura și compresia datelor în vederea reducerii costurilor de transmisie.

Figura 5-4 prezintă modul de operare al SSL Record Protocol. Acesta preia mesajele de nivel aplicație, le fragmentează, opțional comprimă datele, aplică MAC, criptează conținutul, adaugă un antet și apoi transmite rezultatul la destinație prin intermediul TCP.

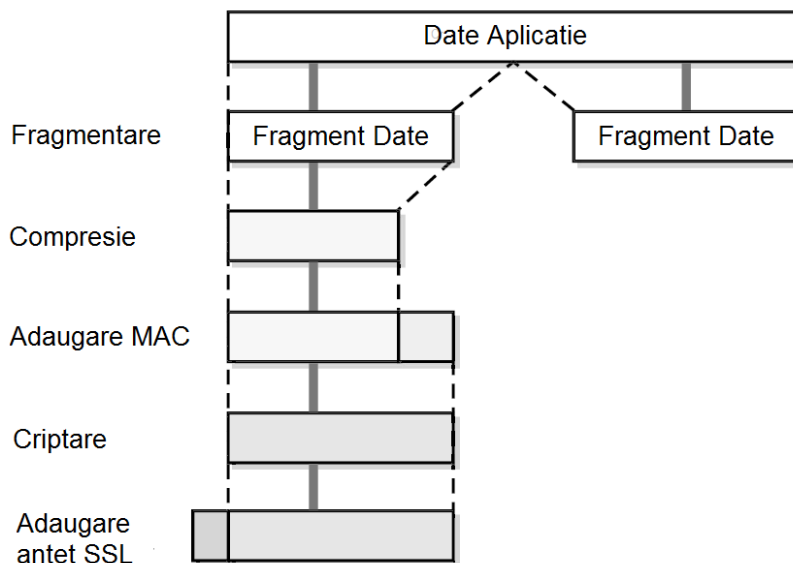


Figura 5-4. Modul de operare al SSL Record Protocol

După cum se poate observa în Figura 5-4 prima operație în procesarea mesajelor de nivel aplicație este fragmentarea. Fiecare mesaj este împărțit în blocuri de date de maxim 2^{14} octeți (16 Kocteți) numite înregistrări (records).

Opțional, blocurile de date sunt comprimate pentru a reduce costul transmisiei. Dacă se realizează, compresia trebuie să se facă fără pierderi de date și fără a mări lungimea blocului de date cu mai mult de 1024 octeți (deși se dorește ca acest proces să reducă dimensiunea blocului de date, este posibil ca, pentru blocuri de date mici, algoritmul de compresie să mărească lungimea blocului de date, datorită convențiilor de formatare).

La pasul următor este calculat un MAC pentru blocul de date. Pentru aceasta este utilizată o funcție hash și o cheie secretă, stabilită în faza de handshake. Codul hash este calculat peste o combinație formată din blocul de date, cheia secretă și un padding, conform formulei:

```
MAC = hash(MAC_write_secret + pad_2 +
hash(MAC_write_secret + pad_1 + seq_num + type + length +
content));
```

unde:

```
hash: MD5, SHA-1
pad_1 = 0x36 repetat de 48 de ori în cazul MD5 și de 40 de
ori în cazul SHA-1
pad_2 = 0x5C repetat de 48 de ori în cazul MD5 și de 40 de
ori în cazul SHA-1
seq_num = numărul de secvență al blocului de date
type = tipul blocului de date (vezi semnificația mai jos)
length = lungimea blocului de date
content = conținutul propriu-zis
```

Receptorul calculează în același mod MAC-ul și compară valoarea MAC primită cu cea calculată. Dacă cele două se potrivesc, receptorul este sigur că mesajul este autentic. Un atacator nu poate să modifice mesajul fără ca acest lucru să nu poată fi detectat deoarece nu deține cheia secretă necesară recalculării MAC-ului.

Blocul de date împreună cu MAC-ul aferent sunt criptate folosind un algoritm simetric de criptare și o cheie de sesiune. În cazul folosirii unui cifru bloc, e posibil să fie necesar să se facă padding. Prin criptare, lungimea blocului de date nu trebuie să crească cu mai mult de 1024 octeți.

La final, la blocul de date criptat este adăugat un antet, cu următoarele câmpuri:

- *Tip Continut (Type)* - protocolul de nivel superior folosit pentru procesarea conținutului. Valorile predefinite sunt: *change_cipher_spec*, *alert*, *handshake*, și *application_data*, primele trei fiind protocoale specifice SSL, iar ultimul reprezentând varietatea de protocoale de la nivel aplicație care pot folosi SSL, fără a se face distincție între ele.
- *Versiune (Version)* - versiunea de protocol SSL folosită. Acest câmp conține două valori: versiunea majoră și versiunea minoră. În cazul SSLv3, versiunea este 3.0.
- *Lungime (Length)* - lungimea, în octeți, a fragmentului. Valoarea maximă este $2^{14} + 2048$.

Structura unui bloc de date SSL este prezentată în Figura 5-5.

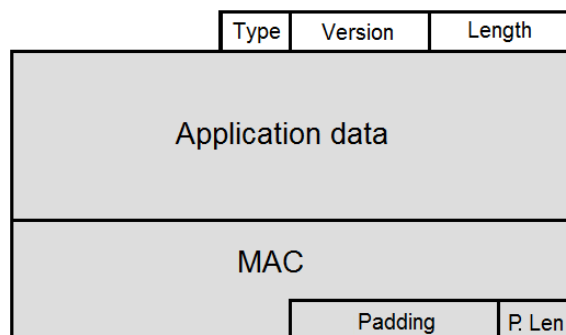


Figura 5-5. Formatul blocurilor de date SSL

La destinație se execută operațiile în sens invers: datele primite sunt decriptate, verificate din punct de vedere al autenticității, decomprimate, reasamblate și apoi livrate protocoalelor de nivel superior.

Pentru efectuarea procesărilor criptografice, SSL Record Protocol folosește algoritmi și cheile negociate prin intermediul SSL Handshake Protocol în faza de stabilire a conexiunii.

5.1.4. SSL Alert Protocol

Rolul SSL Alert Protocol este de a transmite eventualele alerte entităților care comunică prin intermediul SSL. Ca și mesajele de nivel aplicație, alertele sunt transmise comprimat și criptat prin intermediul SSL Record Protocol.

Fiecare mesaj din cadrul SSL Alert Protocol este format din doi octeți. Primul octet poate lua valoarea 1 (warning) sau 2 (fatal) pentru a sugera gravitatea mesajului. Dacă nivelul erorii este fatal, conexiunea trebuie închisă imediat. Ambele entități trebuie să șteargă identificatorul sesiunii și cheile secrete asociate unei conexiuni eșuate. Cel de-al doilea octet conține un cod specific alertei.

Principalele mesaje de alertă din cadrul protocolului SSL sunt:

- *Unexpected_message (fatal)* – această alertă este returnată atunci când este recepționat un mesaj inadecvat.
- *Bad_record_mac (fatal)* – această alertă este returnată atunci când este recepționat bloc de date cu MAC-ul incorrect.

- *Decompression_failure (fatal)* – această alertă este returnată atunci când datele recepționate nu pot fi decompimate.
- *Handshake_failure (fatal)* – această alertă este returnată atunci când negocierea parametrilor de securitate eșuează.
- *Illegal_parameter (fatal)* – această alertă apare atunci când un câmp din mesajul de negociere este invalid.
- *No_certificate (warning)* – această alertă poate fi transmisă ca răspuns la o cerere de autentificare cu certificat digital dacă entitatea respectivă nu deține un certificat corespunzător.
- *Bad_certificate (warning)* – această alertă este returnată atunci când certificatul digital recepționat este invalid.
- *Unsupported_certificate (warning)* – această alertă este returnată atunci când certificatul digital recepționat are un format nesuportat.
- *Certificate_revoked (warning)* – această alertă este returnată atunci când certificatul digital recepționat a fost revocat de către Autoritatea de Certificare emitentă.
- *Certificate_expired (warning)* – această alertă este returnată atunci când certificatul digital recepționat este expirat.
- *Certificate_unknown (warning)* – această alertă este returnată atunci când certificatul digital recepționat nu poate fi acceptat din alte motive.
- *Close_notify (warning)* – această alertă este folosită pentru a notifica entitatea parteneră că nu mai sunt date de transmis. Fiecare entitate este obligată să transmită o astfel de alertă înainte de a închide pentru scriere o conexiune.

5.2. Transport Layer Security (TLS)

Protocolul TLS reprezintă rezultatul standardizării SSL de către IETF. De-a lungul timpului TLS a evoluat de la versiunea 1.0 apărută în 1999 prin publicarea RFC 2246 la versiunea 1.2 publicată în 2008 prin RFC 5246. Versiunea 1.3 este în stadiu de draft în momentul de față. Din punct de vedere funcțional, SSL și TLS sunt similare astfel că în secțiunile următoare se vor puncta doar diferențele dintre cele două.

Câmpul Versiune

Formatul blocurilor de date (înregistrărilor) TLS este același cu cel întâlnit la SSL (vezi Figura 5-5), iar câmpurile din antet au rămas neschimbate. Singura diferență constă în valorile câmpului versiune. Pentru TLS versiunea 1.0, versiunea majoră este 3, iar cea minoră este 1. Pentru TLS versiunea 1.2, versiunea majoră este 3, iar cea minoră este 2.

Algoritmii criptografici suportați

TLS v1.2 suportă următorii algoritmi criptografici simetrici pentru criptarea datelor:

- Cifruri bloc: 3DES_168, AES_128, AES_256. Pentru toți algoritmii, modul de criptare ales este CBC (Cipher Block Chaining).
- Cifruri șir: RC4_128.

Pentru calculul MAC se pot folosi: MD5, SHA-1 sau SHA-256.

Mecanismele de stabilire a cheii de sesiune sunt aceleași cu cele din SSLv3, cu excepția Fortezza.

O suită de algoritmi din cadrul TLS are forma:

TLS_RSA_WITH_AES_128_CBC_SHA

Codul de autentificare a mesajelor (MAC)

Modul de calcul al MAC în TLS este conform cu RFC 2104 (HMAC: Keyed-Hashing for Message Authentication) și poartă denumirea de HMAC_hash. În plus, în calculul MAC se ține cont și de câmpul versiune.

Derivarea cheilor

TLS folosește o funcție pseudo aleatoare (PRF) pentru a extinde valoarea secretă stabilită în faza de handshake la un bloc de biți din care se aleg cheile de sesiune. La început se folosește o funcția de expandare care are formula:

$$P_hash(secret, seed) = \begin{array}{l} HMAC_hash(secret, A(1) + seed) + \\ HMAC_hash(secret, A(2) + seed) + \\ HMAC_hash(secret, A(3) + seed) + \dots \end{array}$$

unde A() se definește ca fiind:

$$\begin{array}{l} A(0) = \text{sămânță} \\ A(i) = HMAC_hash(secret, A(i - 1)) \end{array}$$

P_hash poate fi iterată ori de câte ori este necesar, pentru a produce cantitatea dorită de biți de date.

Funcția pseudo aleatoare (PRF) este creată prin aplicarea P_hash asupra valorii secrete, astfel:

$$\text{PRF}(\text{secret}, \text{label}, \text{seed}) = \text{P_hash}(\text{secret}, \text{label} + \text{seed})$$

Eticheta (label) este un string ASCII.

Padding

În cadrul SSL, padding-ul reprezintă cantitatea minimă de date necesară extinderii lungimii totale a blocului de date până la un multiplu al lungimii blocului de criptare.

Pentru TLS, padding-ul poate avea dimensiuni de până la 255 octeți, cu condiția ca lungimea totală a blocului de date să fie multiplu al lungimii blocului de criptare. O lungime variabilă pentru padding permite evitarea atacurilor bazate pe analiza lungimii mesajelor schimbate.

Mesajele de alertă

Protocolul TLS suportă toate mesajele de alertă definite în SSLv3, cu excepția mesajului no_certificate. În plus, în TLS sunt definite un număr de mesaje de alertă adiționale cum ar fi:

- *Decryption_failed (fatal)* – această alertă este returnată atunci când decriptarea unui mesaj a eșuat. Fie lungimea acestuia nu a fost multiplu al blocului de decriptare, fie padding-ul a fost incorect, la verificare.
- *Record_overflow (fatal)* – această alertă este returnată atunci când înregistrarea TLS primită conține un text criptat a cărui lungime depășește $2^{14} + 2048$ octeți.
- *Unknown_ca (fatal)* – această alertă este returnată atunci când s-a primit un lanț de certificare valid, dar certificatul nu a putut fi acceptat deoarece Autoritatea de Certificare nu a fost declarată ca fiind de încredere.
- *Decode_error (fatal)* – această alertă este returnată atunci când un mesaj nu a putut fi decodificat deoarece valoarea unui câmp este în afara domeniului de valori sau lungimea mesajului este invalidă.
- *Protocol_version (fatal)* – această alertă este returnată de server atunci când versiunea de protocol pe care clientul încearcă să o negocieze este recunoscută dar nu este suportată.

- *Decrypt_error (fatal)* – această alertă semnalează eșecul unei operații criptografice, cum ar fi, de exemplu, imposibilitatea verificării unei semnături.
- *User_canceled (warning)* – această alertă este transmisă pentru a semnală întreruperea handshake-ului în urma intervenției utilizatorului.

5.3. Aplicații ale SSL / TLS

Protocolul SSL/TLS este folosit cu precădere pentru securizarea traficului Web. În cazul în care browser-ul comunică cu serverul de Web peste SSL/TLS protocolul folosit este *https (HTTP over SSL/TLS)* iar portul de comunicație standard este *443/tcp*. Pentru a indica faptul că traficul se desfășoară securizat, browser-ul de Web afișează un lăcățel în dreptul URL-ului (vezi Figura 5-6)



Figura 5-6. Accesul securizat la un site Web

Configurarea SSL/TLS la nivelul serverelor de Web presupune emiterea unui certificat digital pentru serverul de Web respectiv. Acest certificat trebuie emis de o Autoritate de Certificare de încredere pentru utilizatori. Folosirea de certificate digitale autosemnate sau semnate de o autoritate nerecunoscută, permite desfășurarea de atacuri de tip man-in-the-middle prin intermediul cărora se poate intercepta sau modifica traficul.

Cel mai simplu este să se achiziționeze certificatele de server de la o Autoritate de Certificare recunoscută la nivel global, preinstalată în browser-ele utilizatorilor, cum ar fi: DigiCert, GeoTrust, GlobalSign, Verizon, etc. De regulă, acestea comercializează trei tipuri de certificate SSL:

- *Extended Validation (EV)*. Aceste certificate oferă cel mai ridicat grad de încredere. În momentul în care un utilizator accesează un site Web securizat cu un certificat de tipul EV, bara de adresă din browser devine verde. Emiterea unui certificat de tipul EV de către o Autoritate de Certificare trebuie să se facă cu respectarea strictă a regulilor de validare specificate în EV Guidelines, ratificate de către CA/Browser Forum în 2007, ce includ:
 - Verificarea existenței organizației, din punct de vedere legal, fizic și operațional;
 - Verificarea identității organizației pe baza documentelor oficiale;

- Verificarea dreptului organizației de a folosi numele de domeniu ce urmează a fi înscris în certificat;
- Verificarea faptului că cererea de emitere a certificatului a fost autorizată de managementul organizației.

Autoritățile de Certificare care emit certificate de tipul EV trebuie să fie auditate în conformitate cu EV Audit Guidelines emise de același forum. Auditul trebuie repetat anual, pentru a garanta integritatea procesului de emitere al certificatelor.

- *Organization Validation (OV).* În cazul certificatelor de tip OV, Autoritatea de Certificare verifică identitatea organizației și dreptul acesteia de a folosi numele de domeniu ce urmează a fi înscris în certificat. Acest tip de certificate oferă un grad rezonabil de încredere.
- *Domain Validation (DV).* În cazul certificatelor de tip DV, Autoritatea de Certificare verifică doar dreptul organizației de a folosi numele de domeniu ce urmează a fi înscris în certificat. Nu se fac nici un fel de verificări asupra identității organizației. Acest tip de certificate oferă cel mai scăzut grad de încredere.

Dacă în momentul accesării unui site securizat prin SSL/TLS browser-ul afișează un mesaj de avertizare (warning) cu privire la validitatea certificatului sau dacă numele înscris în certificat nu corespunde cu adresa site-ului, atunci este indicat să se renunțe la conexiune, mai ales atunci când site-ul respectiv este unul de internet banking sau dacă prin intermediul site-ului respectiv urmează să se facă plăți on-line.

SSL/TLS poate fi folosit și în conjuncție cu alte protocoale de nivel aplicație care rulează în mod normal peste TCP. Tabelul următor prezintă o listă a acestor aplicații împreună cu porturile asociate.

Serviciu	Port	Descriere
https	443/tcp	HTTP over TLS/SSL
smtps	465/tcp	SMTP over TLS/SSL
sshell	614/tcp	SSLShell
ldaps	636/tcp	LDAP over TLS/SSL
ftps-data	989/tcp	FTP Data over TLS/SSL
ftps	990/tcp	FTP Control over TLS/SSL
telnets	992/tcp	Telnet over TLS/SSL
imaps	993/tcp	IMAP over TLS/SSL
pop3s	995/tcp	POP3 over TLS/SSL

Tabelul 5-1. Porturi SSL/TLS standard

În cazul în care se dorește „sslizarea” unei aplicații client-server proprii, se poate folosi una din bibliotecile de mai jos:

- RSA BSAFE (<http://www.emc.com/security/rsa-bsafe.htm>)
- OpenSSL (<http://www.openssl.org/>)
- Cryptlib (<https://www.cs.auckland.ac.nz/~pgut001/cryptlib/>)
- Java Secure Socket Extension

În Anexa 1 este prezentat un exemplu de aplicație client-server în care comunicația este securizată prin intermediul SSL.

Un alt domeniu în care SSL/TLS poate fi utilizat cu succes îl reprezintă implementarea de rețele private virtuale (VPN). Soluțiile de tip SSL VPN sunt foarte populare, permițând utilizatorilor de la distanță ca prin intermediul unui browser Web să acceseze aplicațiile Web sau client-server din cadrul organizației sau să se conecteze în rețeaua internă a organizației. În ciuda popularității, SSL VPNs nu își propune să înlocuiască IPsec VPN. Cele două tehnologii VPN sunt complementare și adresează cerințe arhitecturale și de business diferite.

Cele mai cunoscute soluții de tip SSL VPN sunt cele comerciale provenind de la Cisco, Citrix, F5 Networks, Juniper, SonicWALL, precum și soluția open-source, OpenVPN (<https://openvpn.net/>), dezvoltată folosind librăria OpenSSL.

De-a lungul timpului, SSL/TLS a fost analizat intens din punct de vedere al securității, identificându-se o serie de atacuri posibile, cum ar fi: atacul POODLE, atacul BEAST, atacurile CRIME și BREACH, atacul prin renegociere, atacul prin trunchiere, atacurile de tip downgrade (FREAK și Logjam), etc. Lista atacurilor cunoscute asupra SSL/TLS a fost publicată de către IETF într-un RFC informativ (RFC 7457). La aceste atacuri se adaugă erorile de implementare, cea mai gravă dintre acestea fiind Heartbleed bug. În plus, o serie de algoritmi au devenit nesiguri (DES, RC2, RC4, MD5) și prin urmare trebuie dezactivați.

În momentul de față, se recomandă folosirea doar a TLS v1.2, cu lungimi de chei de 2048 pentru algoritmi asimetrice și 128 de biți pentru cei simetrici iar ca funcții hash SHA-256.

6. Protocoale de securitate la nivel aplicație

Și la nivel aplicație au fost dezvoltate o serie de protocoale de securitate pentru a rezolva cerințe specifice. Dintre acestea, cele mai cunoscute sunt:

- Pretty Good Privacy (PGP) și Secure/Multipurpose Internet Mail Extensions (S/MIME) - protocoale pentru securizarea poștei electronice;
- Secure Shell (SSH) – protocol pentru securizarea sesiunilor de lucru la distanță;
- Secure Electronic Transactions (SET), 3-D Secure – protocoale pentru securizarea tranzacțiilor cu cărți de credit peste Internet.

În acest capitol ne vom îndrepta atenția numai asupra protocoalelor folosite pentru securizarea poștei electronice. Poșta electronică este, în momentul de față, cea mai folosită aplicație Internet. Odată cu adoptarea pe scară largă a acestui mijloc de comunicare, a apărut nevoia asigurării confidențialității și autenticității mesajelor transmise între utilizatori. Dintre soluțiile propuse, două s-au impus pe scară largă: PGP și S/MIME pe care le vom prezenta în detaliu în continuare.

6.1. Pretty Good Privacy (PGP)

Pretty Good Privacy (PGP) este un pachet software, dezvoltat inițial de către Phil Zimmermann, care furnizează servicii de confidențialitate și autenticitate pentru mesajele transmise prin e-mail sau fișierele stocate local. Phil Zimmermann a integrat cei mai moderni algoritmi criptografici într-un program care să poată rula pe mai multe platforme, cu scopul de a permite utilizatorilor obișnuiți să-și protejeze datele confidențiale.

Prima versiune de PGP (v1.0) a apărut în 1991. Versiunile ulterioare v2.x și v3.x au fost implementate printr-un efort de colaborare al mai multor voluntari, sub coordonarea lui Phil Zimmermann. În 1996, Zimmermann și echipa sa au format o companie pentru a dezvolta și comercializa noi versiuni de PGP. Această companie a fuzionat cu ViaCrypt care ulterior a fost redenumită PGP Internațional. De atunci și până în prezent, PGP a aparținut mai multor companii: Network Associates, PGP Corporation și cel mai recent, Symantec Corporation.

În 1997, având în vedere comunitatea mare de utilizatori, Zimmermann a propus standardizarea PGP de către IETF, sub denumirea OpenPGP. Astfel, a fost format OpenPGP Working Group care a elaborat mai multe RFC-uri, dintre care cele mai importante sunt:

- RFC 4880 - OpenPGP Message Format
- RFC 3156 - MIME Security with OpenPGP

Free Software Foundation a dezvoltat un program compatibil OpenPGP, intitulat GNU Privacy Guard (GnuPG), distribuit sub licență GPL (GNU General Public License). Acesta nu include, bineînțeles, algoritmi criptografici protejați prin patente cum ar fi IDEA, de exemplu.

Încă de la început, codul sursă al PGP a fost disponibil free. Acest lucru a conferit încredere utilizatorilor, oricine putând verifica modul de implementare al algoritmilor sau modul de gestiune al cheilor criptografice.

PGP folosește o combinație de algoritmi simetrici (cu chei secrete) și asimetrice (cu chei publice) pentru a implementa serviciile de confidențialitate și autenticitate pentru e-mail și fișierele de date. În plus, PGP asigură și servicii de compresie și compatibilitate e-mail.

Implementările actuale de PGP suportă algoritmi criptografici de ultimă generație cum ar fi:

- algoritmi criptografici simetrici: IDEA, 3DES, CAST-128, Blowfish, AES, Twofish
- algoritmi criptografici asimetrice: RSA, ElGamal
- scheme de semnături digitale: RSA, DSA, ECDSA
- funcții hash: SHA-1, SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512), RIPEMD-160

În secțiunile următoare este prezentat în detaliu modul de funcționare al PGP. Semnificația notațiilor din scheme este următoarea:

K_s = cheie de sesiune

K_{Ua} = cheia publică a utilizatorului A

K_{Ra} = cheia privată a utilizatorului A

EC = criptare cu chei secrete

EP = criptare cu chei publice

Z = compresie

|| = concatenare

H = funcție hash

K_{Ub} = cheia publică a utilizatorului B

K_{Rb} = cheia privată a utilizatorului B

DC = decriptare cu chei secrete

DP = decriptare cu chei publice

Z^{-1} = decompresie

6.1.1. Serviciul de autenticitate

Pentru asigurarea autenticității mesajelor, PGP folosește conceptul de semnătură digitală, prezentat schematic în Figura 6-1. Etapele acestui proces sunt următoarele:

1. Se calculează rezumatul mesajului folosind o funcție hash (SHA-1);
2. Rezumatul este criptat folosind RSA cu cheia privată a semnatarului obținându-se astfel semnătura digitală care este atașată la mesajul original;
3. Mesajul și semnătura atașată sunt comprimate folosind un algoritm de compresie (zip);

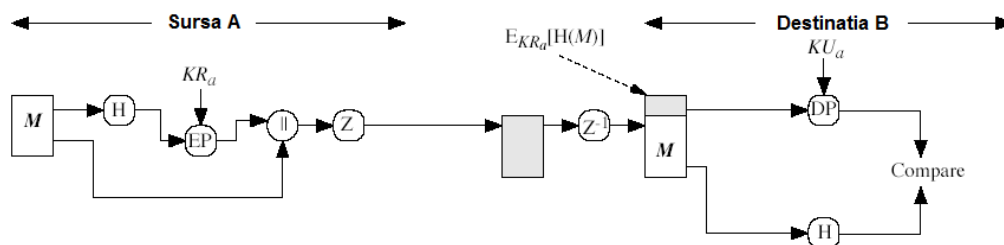


Figura 6-1. Serviciul de autenticitate în PGP

La destinație, pentru verificarea semnăturii, se execută următoarele operații:

1. Se decomprimă mesajul (unzip), dacă este cazul;
2. Folosind aceeași funcție hash (SHA-1) se calculează rezumatul mesajului recepționat;
3. Se decriptează semnătura folosind RSA cu cheia publică a semnatarului și se obține valoarea hash din momentul semnării mesajului;
4. Se compară cele două valori hash. Dacă sunt identice, atunci semnătura este validă ceea ce înseamnă că mesajul este autentic.

PGP permite crearea de semnături atașate sau detașate. În primul caz, în urma semnării rezultă un singur obiect ce conține atât mesajul propriu-zis cât și semnătura aferentă. În cel de-al doilea caz, semnătura este stocată separat de mesaj. Crearea de semnături detașate este utilă în mai multe cazuri. De exemplu, verificarea integrității unui program executabil se poate face prin intermediul unei semnături detașate, generată de autorul programului respectiv.

6.1.2. Serviciul de confidențialitate

Asigurarea confidențialității mesajelor în PGP se realizează prin criptare. Etapele acestui proces, prezentate schematic în Figura 6-2, sunt următoarele:

1. Mesajul creat de expeditor este opțional comprimat folosind un algoritm de compresie (zip);
2. Criptarea datelor se realizează folosind un algoritm simetric (AES) și o cheie secretă generată în mod aleator (K_s). Cheia secretă este folosită o singură dată și din această cauză poartă denumirea de cheie de sesiune.
3. Cheia de sesiune este criptată folosind un algoritm asimetric (RSA), cu cheia publică a destinatarului iar rezultatul este atașat la criptogramă.

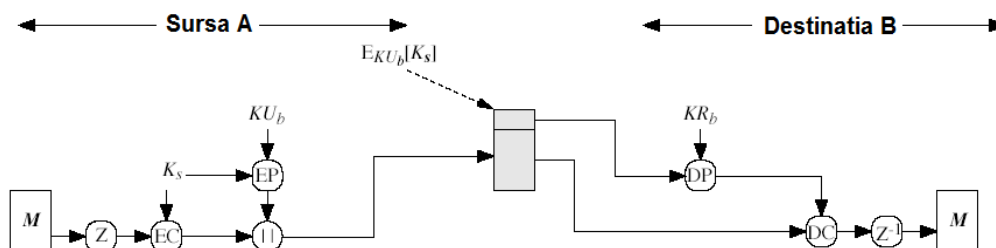


Figura 6-2. Serviciul de confidențialitate în PGP

La destinație se execută aceleași operații în ordine inversă:

1. Se extrage cheia de sesiune folosind același algoritm asimetric (RSA), cu cheia privată a destinatarului;
2. Se decriptează mesajul folosind același algoritm simetric (AES) și cheia de sesiune obținută la pasul anterior;
3. Se decomprimă mesajul (unzip) obținându-se textul original.

Serviciile de autenticitate și confidențialitate pot fi folosite separat sau combinat. În cazul în care sunt folosite combinat, se aplică mai întâi operația de semnare și apoi cea de criptare.

6.1.3. Serviciul de compresie

Pentru a reduce timpul/costul de transmisie sau spațiul de stocare pe disc, PGP permite compresia mesajelor. Compresia se aplică după semnare dar înainte de criptare.

Aplicarea compresiei după operația de semnare este justificată de faptul că algoritmi de compresie nu sunt determinați. Rata de compresie depinde de algoritmul de compresie folosit precum și de alți factori cum ar fi memoria alocată. Aplicarea funcției de semnare după compresie ar constrânge toate implementările de PGP să folosească aceeași versiune de algoritm de compresie.

Aplicarea compresiei înainte de criptare se face din motive de eficiență. Algoritmi de compresie sunt eficienți în cazul în care redundanța la nivelul mesajului este mare ori, prin criptare, această redundanță dispare.

PGP oferă suport pentru ZIP (RFC 1951), ZLIB (RFC 1950) și BZip2.

6.1.4. Serviciul de compatibilitate e-mail

În urma semnării sau criptării unui mesaj, o parte din date sunt în format binar. Sistemul de poștă electronică (e-mail) permite doar transmiterea de text ASCII. Pentru a rezolva această limitare, PGP pune la dispoziție un serviciu de conversie a datelor binare într-un șir de caractere ASCII. Schema folosită în acest scop este Radix-64.

Datele binare de la intrare sunt împărțite în blocuri de câte 6 biți. Prin intermediul unei tabele de conversie, valoarea pe 6 biți este convertită într-un caracter ASCII pe 8 biți. Tabela de conversie are 64 de intrări (2^6) de unde și denumirea schemei - Radix-64. Setul de caractere din tabelă este format din litere mari (A-Z), litere mici (a-z), cifre (0-9) și caracterele '+' și '/'. În cazul în care dimensiunea datelor de intrare nu este multiplu de 6 biți, se adaugă octeți suplimentari (padding). Numărul de caractere '=' de la sfârșit semnifică câți octeți de padding s-au folosit. Pentru detectarea erorilor de transmisie, formatul Radix-64 include un CRC pe 24 de biți (Cyclic Redundancy Check).

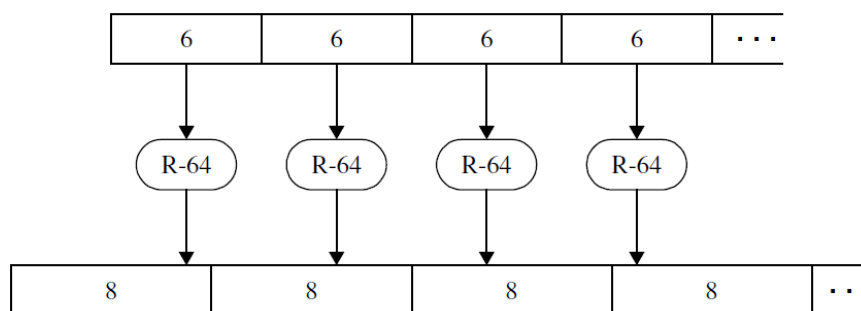


Figura 6-3. Conversia Radix-64

6 bit value	Character encoding	6 bit value	Character encoding	6 bit value	Character encoding	6 bit value	Character encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/
						(pad)	=

Figura 6-4. Tabela Radix-64

De exemplu, să presupunem că la intrare avem următorul șir de biți:

10110010 01101011 01101001

Aranjând în blocuri de câte 6 biți acest șir, vom obține:

101100 100110 101101 101001

Valorile zecimale corespunzătoare sunt: 44, 38, 45, 41 care corespund următoarelor caractere din tabela Radix-64:

smt p

Dacă aceste caractere sunt codificate ASCII pe 8 biți, se obțin următorii octeți:

01110011 01101101 01110100 01110000

După cum se poate observa, prin aplicarea conversiei Radix-64, dimensiunea mesajelor crește cu 4/3 (33%).

6.1.5. Formatul mesajelor PGP

Pentru a putea efectua procesările criptografice atât la expeditor cât și la destinatar, este nevoie de standardizarea formatului mesajelor PGP. Acest standard trebuie să permită identificarea algoritmilor criptografici folosiți, a cheilor publice, a momentelor de timp la care s-au efectuat operațiile precum și stabilirea poziției acestor informații în cadrul mesajului.

O problemă foarte importantă o reprezintă identificarea cheii publice a semnatarului în cazul folosirii serviciului de autenticitate sau a cheilor publice ale destinatarilor în cazul serviciului de confidențialitate. Fără această facilitate, operațiile de verificare a semnăturilor sau de decriptare a mesajelor nu pot fi efectuate. Soluția adoptată în PGP pentru rezolvarea acestei probleme a constat în atribuirea unui identificator (ID) fiecărei chei publice care, cu o probabilitate foarte mare, este unic pentru fiecare utilizator. Acest identificator reprezintă de fapt ultimii 64 de biți din cheia publică.

Formatul general al mesajelor PGP este prezentat în Figura 6-5. După cum se poate observa în această figură, un mesaj PGP include trei componente: mesajul propriu-zis, semnătura (opțional) și cheia de sesiune (opțional).

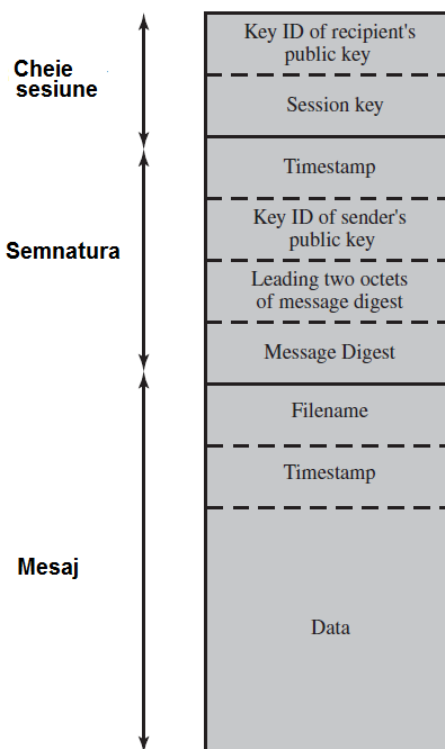


Figura 6-5. Formatul general al unui mesaj PGP

Mesajul propriu-zis include datele ce urmează a fi transmise, numele fișierului și o amprentă de timp pentru a determina momentul creării fișierului.

Componenta de semnătură include următoarele câmpuri:

- *amprenta de timp (timestamp)* folosită pentru a determina momentul la care a fost creată semnătura;
- *rezumatul mesajului (message digest)* reprezintă valoarea hash criptată cu cheia privată a semnatarului. Rezumatul este calculat peste amprenta de timp a semnăturii și zona de date din mesaj;
- *primii doi octeți ai rezumatului mesajului*. Acești octeți permit destinatarului să determine dacă a selectat corect cheia publică pentru verificarea semnăturii mesajului. Pentru asta, destinatarul compară cei doi octeți cu octeții obținuți în urma decriptării rezumatului mesajului;
- *identificatorul cheii publice a semnatarului (key ID)*. Acest câmp este folosit de destinatar pentru a determina cheia publică ce trebuie folosită în procesul de validare a semnăturii.

Mesajul propriu-zis și componenta de semnătură pot fi comprimate folosind un algoritm de compresie (zip) sau criptate folosind un algoritm criptografic simetric și o cheie de sesiune.

Componenta cheii de sesiune include:

- *cheia de sesiune (session key)* criptată cu cheia publică a destinatarului;
- *identificatorul cheii publice a destinatarului (key ID)*. Acest câmp este folosit de destinatar pentru a determina cheia privată ce trebuie folosită pentru extragerea cheii de sesiune.

În cazul în care sunt mai mulți destinatari, componenta cheii de sesiune se repetă pentru fiecare destinatar în parte.

Întregul bloc este, de regulă, convertit în format text folosind Radix-64. Pentru a informa utilizatorul asupra tipului de conținut, PGP adaugă o serie de etichete. Mai jos este prezentat un exemplu de mesaj criptat și codificat Radix-64 în care se pot observa etichetele de început și de sfârșit introduse de PGP.

-----BEGIN PGP MESSAGE-----

Version: PGP 8.1 - not licensed for commercial use: www.pgp.com

qANQR1DBwU4DpuSsW91TIWMQB/9hh+iMBjSbdCSG/4B57qRWonkMTM/a+iS8kj7S
McOBx34nvbdQflF7AlMy2rLz5365nEn9MI1LugioDOY4K0Vv2B2VmnvuhAcwvSJ4
XhoMh9Wa3a9RmIw9PIhE+TtE2BgVJTr5HSI91a90JcpN0Vx0m2wZTtyy9+r8MpaN
vfB3ae0oAr4wnsiO3k4G7vTwituxg+F/d7i3E5FC1zbW1VoatCYQPEcySMWFzffm
C4T3uAHJXIKewGrZ8HQPOGrDi0fXzH+wWv0OUacsl+JFSs2YKriYvCFCvyRvfP0H
N3IbpHk0hpuejLxlC3qGkXu0W6FFpFEkrgh0oLDHMjLcxGAAB/4hfXLx6/IkqOX0
mukD5Qt+CdX3r8CZZ5c9+6n150Z901zq/QrDlZ/xMNjED/ifCNcgvVK0IWRMG6I
UPRSd9saWS4QPoWUV2IQCMmv3aWOergrzYXR4uy+v9Wtbd26esFGKUoksuM7ZTjb
dOdrv05vRixrzQJnLg3OMvosOLNWBu4+UDWl1Q66B+krRoz9x6LhM/v0xcKUzQvN
YmDzZOE6BdI5XDmK+eDma8tgV9galnO+lIpK3sl37yfqOahvENImnnOYFm+BcoCh
JyXfA+eAxCV4P3qedwzEso4Wz30edwH0TuDN4N5ipAOaWI+Q1ENSH7QPBVF5nGO4
BS4izmCG0ocBLBhgMIobaU15S9KAObMP2mZlxLgknVBtqk1Qeg6VgZJVcmzer5zu
L/kXVeLQx/yBWoj6gVgZsBabyzVUA/7zy9p/zdwwp8lIvXtj6vUgkzCp70LGzO6
EL3jWoAKiHTGP3rjw4hETqqIN0krFdDTHH6NSKx0JQ8vq0/S0bj/LhqKelwDV14=
=YiYK

-----END PGP MESSAGE-----

6.1.6. Managementul cheilor publice

PGP folosește criptografia cu chei publice pentru a semna sau cripta mesaje. Garantarea autenticității cheilor publice reprezintă cea mai mare problemă a acestor sisteme criptografice. Fără un mecanism care să ne permită să stabilim dacă o cheie publică aparține într-adevăr unui individ nu putem folosi criptografia cu chei publice.

Pentru Phil Zimmermann, apărător al dreptului la viață privată (privacy) al indivizilor, ideea de terț (Autoritate de Certificare) în care toată lumea să aibă încredere și care să garanteze că o cheie publică aparține unei entități, era de neconceput. Ca urmare, Zimmermann a propus un nou mecanism numit generic **“pânză de încredere” (web of trust)**. Modelul Web of Trust este un model distribuit bazat pe recomandări. Conform acestui model, o cheie publică poate fi considerată validă de către un utilizator dacă este semnată (are recomandări) de cât mai multe persoane de încredere pentru utilizatorul respectiv.

Cheile publice sunt distribuite sub formă de certificate digitale. Un certificat PGP conține numele utilizatorului, adresa de e-mail, cheia sa publică și o serie de semnături (recomandări) de la persoane care certifică faptul că cheia publică din certificat aparține utilizatorului respectiv. În mod firesc, prima recomandare este de la el însuși. Formatul generic al certificatelor PGP este prezentat în Figura 6-6.


Nume: Alice
E-mail: alice@abc.com
 Cheia Publica
Semnatura lui Alice
Semnatura lui Bob
Semnatura lui Robert

Figura 6-6. Certificat digital PGP

Referitor la nivelul de încredere pe care un utilizator îl acordă altui utilizator în a da recomandări, acesta poate fi:

- ***untrusted*** - recomandările date de aceste persoane nu au valoare (0)
- ***marginal*** - recomandările date de aceste persoane au pondere relativ scăzută ($1/Y$)
- ***complete*** - recomandările date de aceste persoane au pondere ridicată ($1/X$)
- ***ultimate*** – recomandarea este dată chiar de către utilizator (1)

Validitatea unei chei se calculează prin însumarea ponderilor fiecărei recomandări. Dacă suma depășește un anumit prag, atunci cheia publică poate fi considerată ca fiind validă. De regulă, se folosesc ponderile $X=1$, $Y=2$ și pragul minim 1. Asta înseamnă că o cheie este validă dacă este semnată de o persoană cu nivelul de încredere complete sau de către două persoane cu nivelul marginal.

În Figura 6-7 este prezentat modelul Web of Trust folosit în cadrul PGP. Toate calculele se fac relativ la utilizator. De exemplu, cheile publice ale lui A, B, D și E sunt considerate ca fiind valide deoarece sunt semnate chiar de către utilizator, care are nivelul de încredere ultimate, cu pondere 1. Cheia publică a lui L este semnată de două persoane necunoscute și de către E pe care utilizatorul îl consideră ca având nivelul de încredere complete, cu pondere 1. Prin urmare, cheia publică a lui L poate fi considerată ca fiind validă chiar dacă L, ca persoană, este considerat untrusted de către utilizator. Cheia publică a lui C este, de asemenea, validă deoarece este semnată de către B și D pe care utilizatorul îi consideră ca având nivelul de încredere marginal, cu ponderile $1/2$ fiecare.

Conform modelului Web of Trust, validitatea unei cheie publice este ceva relativ și nu absolut ca în cazul modelului bazat pe Autorități de Certificare. O cheie PGP poate fi considerată ca fiind validă de către un utilizator sau invalidă de către alt utilizator, în funcție de încrederea pe care cei doi o acordă semnatarilor certificatului cheii publice. În cazul modelului bazat pe Autorități de Certificare, o cheie este validă pentru toți utilizatorii dacă certificatul său digital este semnat de către o autoritate de încredere pentru utilizatorii respectivi.

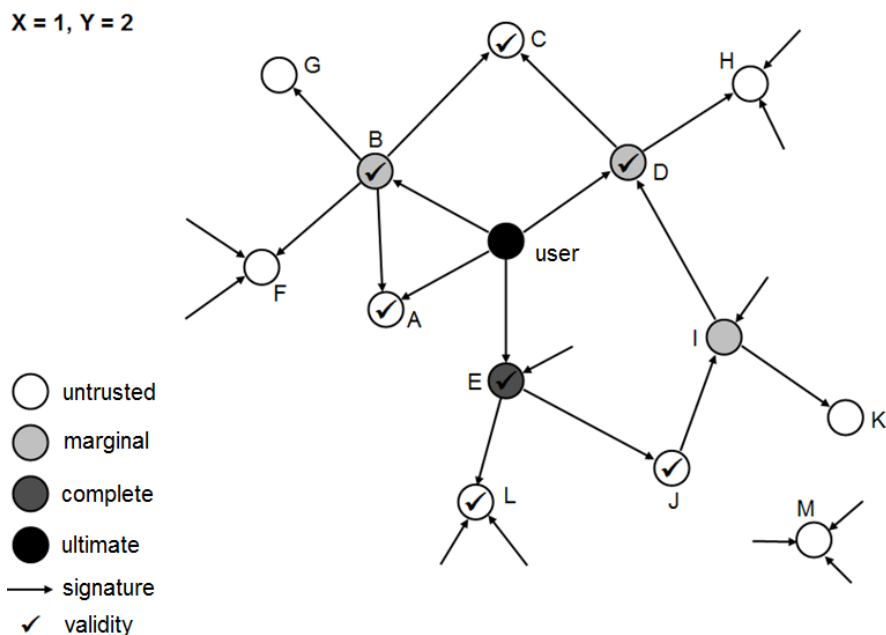


Figura 6-7. Modelul Web of Trust

Pentru managementul cheilor, fiecare utilizator dispune de două *inele de chei*:

- **inelul de chei private (private-key ring)** în care se află cheile private ale utilizatorului. Aceste chei sunt stocate pe disc într-un fișier criptat cu o parolă (passphrase);
- **inelul de chei publice (public-key ring)** reprezintă un fișier în care se află cheile publice ale persoanelor cu care utilizatorul corespundează. În momentul în care utilizatorul modifică nivelul de încredere pentru o persoană, PGP recalculează automat validitatea cheilor din inelul de chei publice.

O cheie publică poate fi exportată în format Radix-64 și distribuită altor persoane. Mai jos se află un exemplu de astfel de cheie.

-----BEGIN PGP PUBLIC KEY BLOCK-----

Version: PGP 8.1 - not licensed for commercial use: www.pgp.com

```
mQGIBFXQvJcRBADy4olYMcG4at7SLbSkYwGlqbOoNtRMZt1/ShoG1LIEB6UE5rrD
6elCmRLoZuH4MiQnvNOM7b6X++pPJ6+A4v9oS1612DU4ywU08EQ1vrGilmFNSPhp
efTO2f5evJvlfCAIrhD5mIXwqvPANWLAVXPm8bBrSxNS4u8740ehsLZazQCg/46v
1wyg3wnL5KBQdRhXreIIIg8D+wX27PhvxQn7BYwnfIDVacj3miPFXlvEGm5JWBF8
Nb6PSDXRbAyt3uHYeh343oG+IV0+u2hFYeodRnVR6/fBGqeSgJcPHFXriKb0Ccw+
f5+3diQDwnGzaSJb1Z4QfdgN0vtnoWMGJCeihA18UcfuWuwAlBcBt2B768CDybwD
WEuIBACEy+1mtpI6GQM9HFke0NQFt2Jx9IMePNgfheSJ9HQPHjG5iAtiKKiSNw4T
5UbGo/pBwcc0O12LVV93S4h0d0pQkwTMnA6+qUTNbGg0TrJkQXTMlEaphV+Bjlnq
ZGGLoqY71ApFSGRbjgybhtjQNSHrJELUDHRVRTZPghQx4TdY8bQXVGvzdCBVc2Vy
IDx1c2VyQGFiYy5ybz6JAF0EEBECAB0FAlXQvJcHCwkIBwMCCgIZAQUBAwAAAAUe
AQAAAAAKCRAtOWlmG7rnphmGAKCo+15yPy2oY23h+clyt4oLuAKXgCcDiqh9bI/
IHibaoGumb69mELjZ3e5Ag0EVdC8lxAIAPZCV7cIfwgXcqK61qlC8wXo+VMROU+2
8W65Szzg2gGnVqMU6Y9AVfPQB8bLQ6mUrfdMZIZJ+AyDvWXPf9Sh01D49V1f3HZS
Tz09jdvOmeFXklN/biude/F/Ha8g8VHMGHOfMlm/xX5u/2RXscBqtNbno2gpXI6
1Brwv0YAWCv19Ij9WE5J280gtJ3kkQc2azNsOA1FHQ98iLMcfFstjvbySPAQ/C1
WxiNjrtVjLhdONM0/XwXV0OjHRhs3jMhLLUq/zzhS1AGBGNfISnCNLWhsQDGCgH
KXrKlQzZlp+r0ApQmwJG0wg9ZqRdQZ+cfL2JSyIZJrqr0l7DVeKyCzsAAgIH/0HJ
VirXfhC4WN710ehq99cB7n9+AcrG6UvmLUSODMe/5CQWT5hDrblbRHM0njufYdnS
NP3jL4c7oixZ4Ii5V6/BHojbJAQ9zM+0jfi1VKwKcFzLUpbk46sUAE+/Ebeme2A0
tEVPmWIST6ijyh1+hSbHrPIr88/gnu9J2UtJ/0I7bWy88RMtzDN1HWEVPPZ/EGnj
cYbPK5V404La0bNJIkhPXqc8+sz2dbdzeIBIh8oulsDwUAFfSr1DZ45dYnl/qp0s
8dQFC803OXJ0C1wf+6x1Thk/Oo3nQ0odx3RBgpXtJ9tIJfYU3nKdLW63TTPp9pR
ZCkqwU8c/EgEmB2n1XuJAEwEGBECAAwFAlXQvJcFGwwAAAAACgkQLTlpZhu656ZS
DwCfcDiAQKc1Thg03EA5deTXM6HScJ0An1mLoRvqlgrXyZIXX0OQrLJ1YCG
=97Ze
```

-----END PGP PUBLIC KEY BLOCK-----

Distribuirea certificatelor de cheilor publice se poate face:

- *direct (față-în-față).* Aceasta este cea mai sigură metodă însă este limitată din punct de vedere practic. Utilizatorul poate importa în inel și semna el însuși cheia publică astfel obținută.
- *peste Internet cu validarea hash-ului prin intermediul unui canal sigur.* Certificatul digital poate fi transmis prin e-mail urmând ca utilizatorii să verifice în mod obligatoriu, printr-un canal sigur cum este telefonul, că hash-ul cheii recepționate este identic cu cel al cheii transmise. Și în acest caz, utilizatorul poate importa în inel și semna el însuși cheia publică astfel obținută.
- *prin intermediul serverelor de chei.* Certificatul digital este publicat pe un server de chei PGP și importat de către cine are nevoie. În acest caz, validitatea cheii publice se stabilește numai pe baza recomandărilor.

Prin urmare, este esențial ca un utilizator să obțină recomandări de la cât mai multe persoane de încredere înainte de a-si publica certificatul pe server. Pentru implementarea serverelor de chei PGP se folosește, de regulă un server LDAP (Lightweight Directory Access Protocol).

La un moment dat, s-ar putea să fie nevoie ca o cheie publică să trebuiască să fie revocată din diferite motive (cheia privată asociată a fost compromisă, de exemplu). În PGP, convenția este ca proprietarul să emită un certificat de revocare a cheii publice, semnat de către el însuși. Acest certificat are aceeași formă ca un certificat normal însă include un indicator care specifică faptul că scopul certificatului este de a revoca cheia publică din certificat. Proprietarul trebuie să distribuie cât mai repede acest certificat de revocare a cheii publice către toți cei cu care corespundează în mod uzual.

Modelul Web of Trust este relativ simplu și ușor de pus în aplicare pentru o comunitate restrânsă de utilizatori care se cunosc între ei, cum este cazul utilizatorilor de e-mail. Din păcate, soluția nu este scalabilă la nivel mai mare și nu poate fi folosită în alte categorii de aplicații cum ar fi cele de comerț electronic.

PGP este folosit astăzi de diverși utilizatori, de la studenți și oameni de afaceri, până la organizații naționale, internaționale și agenții guvernamentale. Librăriile criptografice care stau la baza PGP au fost folosite și în alte categorii de aplicații cum ar fi:

- criptarea în întregime a discurilor de pe calculatoarele portabile (laptop);
- criptarea fișierelor sau a folderelor partajate de pe serverele de fișiere;
- criptarea comunicațiilor prin sistemele de mesagerie instantanee (Instant Messaging);

În plus, au apărut versiuni compatibile OpenPGP pentru telefoane inteligente (smartphones), cum ar fi iPGMail pentru iOS și OpenKeychain pentru Android, care permit criptarea e-mail-urilor transmise de pe aceste platforme.

6.2. Secure/Multipurpose Internet Mail Extensions (S/MIME)

Secure/Multipurpose Internet Mail Extensions (S/MIME) este un protocol proiectat și utilizat pe scară largă pentru securizarea mesajelor de e-mail. Serviciile de securitate asigurate sunt: confidențialitate, autenticitate, integritate și non-repudiare. Protocolul precizează formatul mesajelor și stabilește modul de folosire a certificatelor digitale în contextul protocolului.

Prima versiune S/MIME a fost dezvoltată în 1995 de un grup de lucru condus de RSA Data Security și format din mai mulți reprezentanți ai industriei de profil. Rezultatele obținute în urma activității grupului au fost transferate ulterior către organismul IETF în vederea standardizării.

În acest moment, standardul S/MIME se afla la versiunea 3.2 și este definit de următoarele specificații:

- RFC 5751 – Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification
- RFC 5750 – Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Certificate Handling

Mesajele folosite în cadrul protocolului S/MIME sunt definite ca extensie a formatelor MIME. Formatele MIME au fost propuse ca o suită de standarde IETF și permit includerea de informație non-text (binară) în cadrul mesajelor transmise prin Internet folosind protocoale ce suportă doar comunicații text. Folosind formatarea MIME este posibilă, de exemplu, transmiterea prin e-mail a datelor multimedia (imagini, conținut video, audio, programe, etc.).

MIME nu oferă însă nici-un fel de suport de securitate. Folosind extensia S/MIME, sunt adăugate facilități de securitate care permit criptarea și semnarea mesajelor de e-mail. S/MIME poate fi utilizat, teoretic, cu orice protocol de nivel aplicație ce suportă transportul datelor MIME (de exemplu HTTP).

În acest moment S/MIME este folosit cu preponderență pentru securizarea serviciului de e-mail, fiind asociat cu acest tip de serviciu. Majoritatea aplicațiilor client de e-mail suportă funcționalități S/MIME fiind asigurată astfel o interoperabilitate foarte bună la nivelul acestor aplicații.

S/MIME implementează serviciile de securitate folosind următoarele mecanisme criptografice:

- Criptarea simetrică și asimetrică pentru confidențialitatea mesajelor.
- Semnături digitale și funcții hash pentru autenticitatea, integritatea și non-repudierea mesajelor.

S/MIME combină schemele de criptare simetrică, criptografia cu chei publice, funcțiile hash și certificatele digitale. Motivația acestei diversități este câștigul de eficiență. Folosind fiecare din metodele respective în contextul potrivit, se obține o schemă hibridă ce oferă în același timp securitate și eficiență.

Standardul este restrictiv în ceea ce privește lista algoritmilor acceptați pentru a realiza operațiile criptografice necesare. În versiunea 3.2, aceștia sunt:

- Algoritmi de criptare simetrică, folosiți pentru criptarea datelor: AES-CBC (128/192/256), 3DES-CBC.
- Algoritmi de criptare asimetrică, folosiți pentru criptarea cheilor de sesiune: Diffie-Hellman, RSA.
- Algoritmi de semnătură, folosiți pentru semnarea mesajelor: DSA, RSA.
- Algoritmi de hash, folosiți în procesul de creare a semnăturilor: SHA-1, SHA-256, MD5 (nerecomandat).

După realizarea operațiilor criptografice, pentru formatarea mesajelor S/MIME finale se folosește codificarea de tip Base-64. Aceasta este similară cu conversia Radix-64 folosită de PGP, cu excepția CRC, și realizează conversia în format text a mesajelor obținute.

6.2.1. Formatul mesajelor criptografice (CMS)

Protocolul S/MIME folosește formate de mesaje definite de standardul *CMS – Cryptographic Message Syntax*. Acesta este urmașul specificației tehnice PKCS#7 propusă de RSA Laboratories încă din 1993. Standardul CMS este elaborat și menținut de IETF, versiunea actuală fiind definită în RFC 5262. Standardul definește 6 tipuri (formate) de mesaje criptografice și structurile de date necesare pentru implementarea acestora:

- Mesajul general de tip *Data* – tip generic folosit pentru a descrie date sub forma unui șir arbitrar de octeți a căror semnificație este gestionată de fiecare aplicație în parte. Aplicația este cea care va genera și apoi va interpreta conținutul respectiv.

- Mesajul de tip *SignedData* – folosit pentru semnarea datelor. Datele pot fi semnate de unul sau mai mulți semnatori în paralel.
- Mesajul de tip *EnvelopedData* – definit pentru formatarea datelor criptate folosind forma anvelopată. Permite criptarea datelor și asigură managementul cheii de criptare pentru unul sau mai mulți destinatari.
- Mesajul de tip *DigestedData* – definit pentru formatarea rezumatelor hash calculate asupra datelor.
- Mesajul de tip *EncryptedData* – definit pentru formatarea datelor criptate. Conține doar datele criptate, iar spre deosebire de *EnvelopedData* nu asigură managementul cheii de criptare a datelor.
- Mesajul de tip *AuthenticatedData* – definit pentru autentificarea datelor. Poate conține orice tip de date, un cod de autentificare (MAC), și cheia folosită la generarea codului MAC, criptată asimetric pentru unul sau mai mulți destinatari.

Structurile de date folosite pentru implementarea acestor mesaje sunt definite folosind notația ASN.1. În acest sens, pentru fiecare din mesajele prezentate mai sus, standardul precizează câte o structură ASN.1 specifică. În plus, este definită o structură ASN.1 globală care încapsulează orice tip de mesaj CMS. Această structură are următorul format:

```
ContentInfo ::= SEQUENCE {
    contentType ContentType,
    content [0] EXPLICIT ANY DEFINED BY contentType }
```

Primul câmp al structurii (*contentType*) conține un identificator pentru tipul de mesaj încapsulat. Al doilea câmp (*content*) conține codificarea BER a structurii de date corespunzătoare tipului precizat de câmpul *contentType*.

În continuare sunt prezentate pe larg structurile de date corespunzătoare mesajelor de tip *EnvelopedData* și *SignedData*. Aceste formate sunt folosite în cadrul protocolului S/MIME.

Mesajul de tip *EnvelopedData*

Acesta conține datele criptate simetric pe baza unei chei de sesiune și cheia de sesiune criptată folosind cheia publică a destinatarului. Încapsularea la comun a datelor criptate și a cheii folosită la criptarea lor este cunoscută și sub numele de anvelopă digitală. Un mesaj anvelopat poate fi construit pentru a permite decriptarea de către unul sau mai mulți destinatari. Mesajul conține o singură copie a datelor criptate și cheia de sesiune criptată pentru fiecare destinatar în parte.

Structura de date *EnvelopedData* este prezentată schematic în Figura 6-8, fiind definită astfel:

```
EnvelopedData ::= SEQUENCE {  
    version CMSVersion,  
    originatorInfo [0] IMPLICIT OriginatorInfo OPTIONAL,  
    recipientInfos RecipientInfos,  
    encryptedContentInfo EncryptedContentInfo,  
    unprotectedAttrs [1] IMPLICIT UnprotectedAttributes  
OPTIONAL }
```

Câmpurile structurii *EnvelopedData* au următoarele semnificații:

- *Versiunea (version)* indică versiunea mesajului. Standardul de CMS impune anumite reguli stricte de versionare în funcție de câmpurile folosite.
- *Informațiile despre transmitătorul mesajului (originatorInfo)* conține detalii cu privire la entitatea care a creat mesajul. Este un câmp opțional și de regulă nu este utilizat. Când este folosit, se poate implementa un protocol de management de chei pentru a transmite către destinatar certificatul transmitătorului. Transmitătorul poate primi apoi mesaje criptate cu cheia publică din certificat.
- *Informațiile pentru destinatari (recipientInfos)* este o listă de structuri *recipientInfo*. Fiecare structură *recipientInfo* conține informațiile necesare unui destinatar pentru decriptarea mesajului: cheia de sesiune criptată cu cheia publică a destinatarului, identificatorul algoritmului utilizat la criptarea cheii și informații despre cheia publică folosită.
- *Conținutul criptat (encryptedContentInfo)* conține datele criptate simetric și detalii privind algoritmul și parametrii folosiți la criptarea datelor.
- *Alte attribute (unprotectedAttrs)* este un câmp opțional ce poate conține o listă suplimentară de attribute necriptate. Pot fi folosite pentru a completa structura *EnvelopedData* cu informații suplimentare, iar de regulă aplicațiile nu-l folosesc.

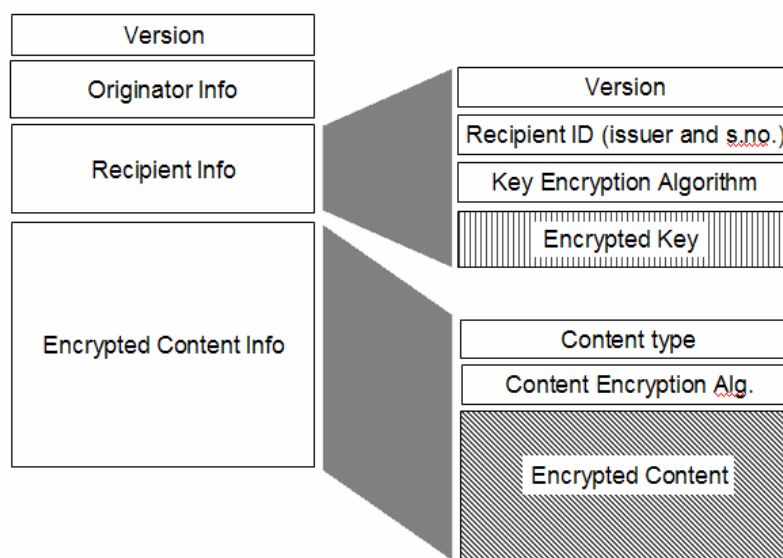


Figura 6-8. Structură de tip EnvelopedData

Mesajul de tip *SignedData*

Structura de date *SignedData* este prezentată schematic în Figura 6-9, fiind definită de standard astfel:

```
SignedData ::= SEQUENCE {
    version CMSVersion,
    digestAlgorithms DigestAlgorithmIdentifiers,
    encapContentInfo EncapsulatedContentInfo,
    certificates [0] IMPLICIT CertificateSet OPTIONAL,
    crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,
    signerInfos SignerInfos }
```

Câmpurile din cadrul structurii *SignedData* au următoarele semnificații:

- *Versiunea (version)* indică versiunea mesajului.
- *Algoritmii de hash (digestAlgorithms)* conține lista identificatorilor de tip OID corespunzători algoritmilor de hash folosiți la generarea semnăturilor din mesaj.
- *Datele semnate (encapContentInfo)* conține tipul și conținutul datelor asupra cărora se aplică semnătura. Tipul datelor este un identificator (OID) și permite aplicației de verificare să aplice reguli particulare de interpretare a datelor după verificarea semnăturii. Conținutul efectiv al datelor semnate este opțional. Dacă datele nu sunt incluse, sunt generate semnături detașate (fără datele semnate). Acest tip de mesaje semnate

sunt utile în aplicații unde datele sunt, de exemplu, semnate multiplu folosind instanțe diferite de structuri *SignedData*, sau când datele sunt procesate de aplicații care nu sunt compatibile cu structurile CMS.

- *Setul de certificate de verificare (certificates)* conține o listă opțională de certificate ale semnatarilor. Lista poate conține de asemenea și lanțul de certificare. Adăugarea completă a certificatelor necesare este utilă pentru aplicațiile de validare însă conduce la creșterea în dimensiune a mesajului final. De regulă aplicațiile sunt configurabile cu privire la acest aspect.
- *Setul de CRL-uri (crls)* reprezintă o listă opțională de CRL-uri care pot fi utilizate la validarea certificatelor folosite la semnarea datelor.
- *Lista informațiilor de semnare (signerInfos)* este o listă de structuri *signerInfo*. Fiecare *signerInfo* conține datele privind semnătura realizată de un semnatar: semnătura efectivă realizată folosind un algoritm de semnare precum și alte informații suplimentare: identificatorul algoritmului (ex: DSA, RSA) folosit de semnatar la generarea semnăturii, identificatorul algoritmului de hash folosit la generarea semnăturii, identificatorul (numărul serial și numele CA-ului emitent) certificatului de cheie publică corespunzătoare cheii private folosite în procesul de semnare. În plus, fiecare semnatar poate completa conținutul semnat cu informații suplimentare adăugate sub forma unor atribute. În acest sens există o serie de atribute standard care pot fi semnate împreună cu datele (*signedAttributes*) și atribute care nu sunt incluse în procesul de semnare (*unsignedAttributes*). Aceste atribute oferă detalii suplimentare cu privire la semnătură, cum ar fi de exemplu: momentul semnării (*signingTime*), certificatul folosit la semnare (*signingCertificate*), o contra-semnătură (*counterSignature*) sau o marcă temporală obținută de la o Autoritate de Marcare Temporală de încredere (*signatureTimestamp*).

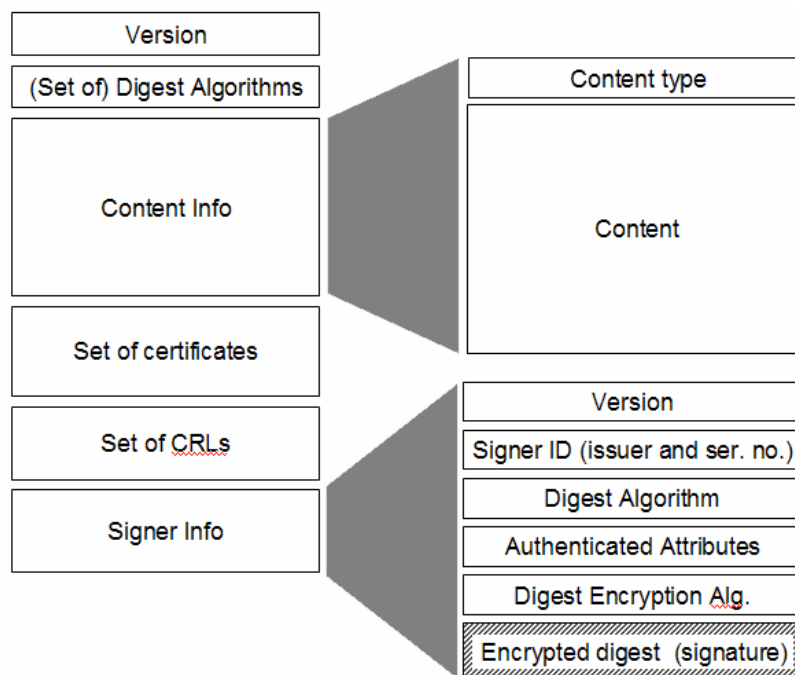


Figura 6-9. Structură de tip SignedData

6.2.2. Criptarea mesajelor

Pentru criptarea mesajelor protocolul S/MIME folosește formatul mesajului criptografic *EnvelopedData*. Procesul de generare a unui mesaj S/MIME criptat este format din următoarele etape:

1. Se codifică mesajul de e-mail într-un format de mesaj MIME. Acesta este conținutul care urmează să fie criptat.
2. Se alege un algoritm simetric de criptare (AES, 3DES) și se generează aleator o cheie pentru el. Această cheie se numește și cheie de sesiune, iar pentru generarea ei aplicația folosește, de regulă, un algoritm pseudo aleator.
3. Cheia de sesiune este criptată pentru fiecare destinatar al mesajului, folosind un algoritm asimetric (RSA, DH).
4. Pentru fiecare destinatar se crează câte un bloc *RecipientInfo*. Acesta conține cheia de sesiune criptată asimetric (*Encrypted Key*), împreună cu alte informații necesare în procesul de decriptare: identificatorul algoritmului folosit la criptarea cheii de sesiune (*Key Encryption Algorithm*), identificatorul cheii publice a destinatarului (*Recipient ID*).

5. Conținutul mesajului de e-mail formatat MIME este criptat simetric folosind cheia de sesiune. Rezultatul obținut (*Encrypted Content*) este adăugat la structura finală împreună cu identificatorul algoritmului simetric folosit (*Content Encryption Alg.*).
6. Este construită structura *EnvelopedData*, care în final este încapsulată în structura de uz general *ContentInfo*. Aceasta este codificată BER, iar rezultatul obținut este transformat în informație text folosind codificarea Base-64.
7. Rezultatul obținut după codificarea Base-64 este formatat apoi într-un nou mesaj MIME final de tip *application/pkcs7-mime*.

Pentru a obține conținutul în clar al mesajului inițial trebuie parcursă în sens invers secvența de pași prezentată mai sus. Conținutul mesajului S/MIME este decodificat Base-64. Apoi, cheia de sesiune este decriptată asimetric folosind cheia privată a destinatarului. În cazul în care destinatarul are mai multe chei private, identificatorul din câmpul *Recipient ID* este folosit pentru selectarea cheii private potrivite. În final, datele mesajului sunt decriptate simetric folosind cheia de sesiune.

Mai jos este prezentat un exemplu de mesaj MIME care încapsulează un mesaj S/MIME criptat. Se observă că tipul MIME folosit este *application/pkcs7-mime*, iar acesta conține încapsulat un mesaj S/MIME de tip *enveloped-data*.

```
MIME-Version: 1.0
Content-Disposition: attachment; filename="smime.p7m"
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
name="smime.p7m"
Content-Transfer-Encoding: base64
```

```
MIICchwYJKoZIhvcNAQcDoIIICeDCCAnQCAQAxggFcmIIBWAIBADBAMDSxCzAJBgNV
BAYTAlJPMQwwCgYDVQQKDANNVEExDDAKBgNVBAsMA1NUSTEQMA4GA1UEAwwHVGVz
dCBDDQQIBAJANBgkqhkiG9w0BAQEFAASCACQWiw7H03XCjAeCp2lKF79OFHg8eM
TmZs7/yXQJmDuKpiPe6lR3cLYG4/hqqEueqp/0U/bYYXWokfk8JBZt0KdUvb7dSJ
HkVpCBT5H5a3RAAGoMZUEGq12YtmpE5+egCo35TmYTclp8BFtnfcr/wVexirzmwr
eonkbonbdViQqdCMpk1hDZAm5hUK8W0555XnUIgMA78MsKhqdNxxP82UGcpLwt/d
XfCaUkrrxtZXB6QzJkjBuCidSqWrW+fkup4SlBCC9d0fFiCsFYze2QiWat0NdSkJ
vRUNLq0GU+r2m0gdxM+lZUFcABOjduockd2vfzMP9yRHGpZ5ON88RF7uMIIBDQYJ
KoZIhvcNAQcBMB0GCWCGSAFlAwQBAGQQVW10ySRkqf8ZvNoWlBfMe4CB4MzXdXXG
5R5iD/Bha8L3l17Pnbk+rmEdk00jWSHDXPHtn9RWwOERb5LVbKJis6v3k1P+XLgQ
HKQa7K5vs0S6jppn8cphu6fjVaXrsOsuB5FgbwjR9JMpIXeTZ1s+SuDibah8gcX
3DhbFOAHovMhJCA2bxfZklVcroSAYWsi7l5lcmeh83Lr9z6vWl2lfnVDMeJW4/dM
GMze2sMOy9fyCu3lcbfMhZrVMbf4ZdiFQXXqop4C5Gju7l7RlFctmqngM/iEA/S
uuRHds4D2jG3lo/vsqwSmyscBbTsplSqnwwZ
```

6.2.3. Semnarea mesajelor

Protocolul S/MIME pune la dispoziție două metode pentru semnarea mesajelor:

- Metoda de tip ***application/pkcs7-mime***. În acest caz este folosită structura *SignedData* în formatul de semnătură cu date incluse (datele semnate sunt încapsulate împreună cu semnătura generată). Metoda este suportată însă numai la nivelul unora din aplicațiile existente de e-mail (doar cele care au funcționalități S/MIME).
- Metoda de tip ***multipart/signed***. În acest caz este folosită, de asemenea, structura *SignedData*, împreună cu semnături detașate (datele semnate nu sunt incluse în formatul de semnătură). Această metodă este suportată de majoritatea aplicațiilor de pe piață.

Standardul S/MIME nu obligă la folosirea uneia dintre cele două metode, însă semnarea de tip ***multipart/signed*** este preferată datorită ariei mai largi de implementare la nivelul aplicațiilor.

Semnarea de tip *application/pkcs7-mime*

La construcția unui mesaj S/MIME semnat folosind această metodă sunt realizate următoarele operații:

1. Se codifică mesajul de e-mail într-un format de mesaj MIME. Acesta este conținutul care urmează să fie semnat.
2. Se calculează hash-ul (rezumatul) mesajului MIME ce urmează să fie semnat, folosind un algoritm de hash ales de semnatar.
3. Se criptează hash-ul mesajului folosind cheia privată a semnatarului. Se obține astfel semnătura calculată asupra conținutului semnat.
4. Se construiește structura *signerInfo* ce conține informațiile de semnare: semnătura calculată, identificatorul algoritmului de semnare folosit, un identificator al cheii publice corespunzătoare cheii private folosite la calculul semnăturii.
5. Un mesaj poate fi semnat de mai mulți semnatori. Fiecare nou semnatar parcurge pașii 2, 3 și 4.
6. Conținutul semnat este adăugat împreună cu informațiile de semnare într-o structură *SignedData*. Aceasta este încapsulată în structura de uz general *ContentInfo* care este codificată BER și apoi este transformată la forma text folosind o codificare Base-64.

7. Rezultatul obținut după codificarea Base-64 este formatat apoi într-un nou mesaj MIME final de tip *application/pkcs7-mime*.

La primirea unui mesaj semnat acesta este mai întâi decodificat Base-64 și parsat ASN.1. Se face apoi verificarea semnăturilor și validarea certificatelor de semnare. Verificarea unei semnături implică folosirea cheii publice pentru a decripta hash-ul calculat la momentul semnării. Acesta este comparat cu hash-ul actual al datelor calculat de asemenea independent de verificator.

Mai jos este prezentat un exemplu de mesaj MIME care formează un mesaj S/MIME semnat folosind metoda *application/pkcs7-mime*. Se observă că tipul MIME folosit este *application/pkcs7-mime*, iar acesta conține încapsulat un mesaj S/MIME de tip *signed-data*.

```
MIME-Version: 1.0
Content-Disposition: attachment; filename="smime.p7m"
Content-Type: application/pkcs7-mime; smime-type=signed-data;
name="smime.p7m"
Content-Transfer-Encoding: base64
```

MIIG1AYJKoZIhvcNAQcCoIIGxTCCBsECAQExCTAHBgUrDgMCGjCB6WYJKoZIhvcNAQcBoIhDBIHaRnJvbToJTWlwyY2VhIENlcm5hdA0KU2VudDoJU2F0dXJkYXksIFNlchRlbWJlciAwNSwgMjAxNSA0OjA2IFBNDQpUbzoJSW9uIEJpY2ENCln1YmplY3Q6CW5vdGhmaWNhcmUgb3JhcG0KDQpWYSBpbmZvcmlhbSAgY2EgYSBmb3N0IGVtaXMgb3JhcnVsIHByaXZpbmQgZGVzZmFzdXJhcmVhIGN1cnN1cm1sb3IgbGEgbWVzdGVvYXQuDQoNC1RvYXRlIGNlbGUGyYnVuZSwNcK1pcmNlYQ0KDQqgggN9MIIDeTCCAmGgAwIBAgIBAjANBgkqhkiG9w0BAQUFADA7MQswCQYDVQQGEWJSTzEMMAoGA1UECgwDTVRBMQwwCgYDVQQZLDANTVEkxEDA0BgNVBAMMB1Rlc3QgQ0EwHhcNMTUwNTE2MTk0OTE4WhcNMTYwNTE1MTk0OTE4WjBpMQswCQYDVQQGEWJSTzEMMAoGA1UECgwDQVRNMQwwCgYDVQQZLDANTVEkxFjAUBGQNVBAMMDU1pcmN1YSBDZXJ1YXQxJjAkBgkqhkiG9w0BCQEWF2lpcmN1YS5jZXJ1YXRAZ21haWwuyY29tMIIIBiJANBgkqhkiG9w0BAQEFAAOCAQ8AMIcBgKCAQEASfHJZ1Hhn8Sqw5lE8rf8Szcpei4RSu4MzRIOBFjnrQ07XQZ399B/xxd90Ec5fmbIFdn81yzoMKJVVSG76pqUsojrsGGfV4YytrJ3P3cptKVQYtd4DfKE0clUhpXixbjrTrzepVNIAjktJacaeKrV9QoP0nCs0ECg28ZEM9dekFfStfGQRXp8+T3vzeUVGTWQt4gEVib+PTLc5NzTZRHMsuCD/8l++4B7aN6yc+HwB3nY/+XfEl3nhbd+QgX/6rgLw4v1xbfupDiQ7Go5KWI1VsFB09fhYSihwh3dYgJgZLamxafpWFFndZnl8XISvrbKv1alvMpA2uIENpI1bthFQIDAQABolowWDAJBgNVHRMEAjAAMAsGA1UdDwQEAwIF4DAdBgNVHQ4EFgQU3RplFgj4crz1f563hnLY+giStTwwHwYDVR0jBBgwFoAuOT50TyOGjBrkt/N3610mFtp8pEgwdQYJKoZIhvcNAQEFBQADggEBAJLP9UE6p5N4dRXrT2mxholkXemyU292CqUxaSVkE51cirKQbXPxdAM2kYPT6Jv7s3LSPp5nePdcTlgrN7dTe+mLlhGD/7qtTrbZB788xvORFuB/U8BGAvgSKNZBUsvcxkNvI5tMncfBXxKQy0JHvFvny/OGoiQRiA3kTwrPlx2nOgSJc8+Vq4BPbqNw/4LfyLC9dDTJL/2gh8h2Y130iGmr8bsck0Ws/Ca/qgvMEFLjGjAKJr0dhG/9MQKaODGXbKc+XgrTEqNzCTjuYTJuqPHqkL5bDGH9T0tHeAp9FhKdFsqWxbF38K7RlWJ/0mkiJpayb2dQcQDuBaJmJq4414tcxvgJAMICPAIBATBAMDSxCzAJBgNVBAYTA1JPMQwwCgYDVQQKDANNVEEeDDAKjENBGNVBSA5MA1NUSTEQMA4GA1UEAwwHVGVzdCBBDQqIIBAjahBgUrDgMCGqCB2DAYBgkqhkiG9w0BCQMxCwYJKoZIhvcNAQcBMBwGCSqGSIb3DQEJBTEPFw0xNTA5MDUxMzA2NDRAmCMGCSqGSIb3DQEJBDEWBBCRCc55KbSk+aH9

7dPugxROZbuHODB5BgkqhkiG9w0BCQ8xbDBqMAsGCWCGSAFlAwQBKjALBglghkgB
 ZQMEARYwCwYJYIZIAWUDBAECMAoGCCqGSIB3DQMHMA4GCCqGSIB3DQMCAGIAgDAN
 BggqhkiG9w0DAgIBQDAHBgUrDgMCBzANBggqhkiG9w0DAgIBKDANBgkqhkiG9w0B
 AQEFAASCAQAn3b3FUb46n18Q9NsEiYH2JQ+EIJ6cNBvHcJkat5GiJlvPyTHZBEjC
 ElPGKbj3NortbDFx+/0BoY2Mgq1nR5AXJtdF5HvblIcAvZaV8/RO3lORmgz3hDfF
 LmuPA+xL9ahmdhViLSpD6ZFGI8TD1Poy5rI4cCSQ8cnvp6975Rb/bp52KUkiYVuP
 s2YRVBlQ3QYi8gK3KobhEwVPM6gRxYt1o4zAydzKz5w2LT8Rk1/zSEiYHqL5ytaf
 74IMuiDJsc/TIirq3d6tRMZD58UfKiGeMQNpwbTvex7l+UYDK4bf+No1YLLckiQ8x
 /+HsxOs10nBq7T4yjmMBNTx1+EG4cXde

Semnarea de tip *multipart/signed*

Această metodă folosește aceeași structură *SignedData*. Față de prima metodă de semnare există însă următoarele două diferențe:

- Structura *SignedData* nu conține mesajul semnat. În acest caz, câmpul *encapContentInfo* din *SignedData* conține doar tipul datelor nu și datele. Acest tip de semnătură se numește semnătură detașată.
- Mesajul MIME final este format din două părți separate. Prima parte conține mesajul de e-mail. A doua parte conține semnătura S/MIME adică codificarea Base-64 a structurii *SignedData*. Aceasta este generată ca în cazul metodei *application/pkcs7-mime* cu excepția câmpului cu datele semnate lăsat acum necompletat.

În cazul metodei *multipart/signed* datele sunt independente de semnătură. Avantajul acestui format constă în faptul că un client de e-mail care nu este compatibil S/MIME poate accesa și procesa conținutul mesajului de e-mail inclus separat în mesajul S/MIME primit. În acest caz clientul nu poate utiliza funcția de securitate S/MIME însă are acces la datele mesajului de e-mail.

Mai jos este prezentat un exemplu de mesaj MIME care formatează un mesaj S/MIME semnat folosind metoda *multipart/signed*. Se observă mesajul MIME este format din 2 părți: prima parte conține mesajul semnat, iar cea de a doua conține semnătura CMS de tip *SignedData* codificată Base-64.

```
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-
signature"; micalg="sha1"; boundary="----
18E6332C520072EE6094C1ACD68234A9"
```

This is an S/MIME signed message

-----18E6332C520072EE6094C1ACD68234A9

Content-Type: text/plain

Va informam ca a fost emis orarul privind desfasurarea cursurilor la masterat.

Toate cele bune,
Mircea

-----18E6332C520072EE6094C1ACD68234A9

Content-Type: application/pkcs7-signature; name="smime.p7s"

Content-Transfer-Encoding: base64

Content-Disposition: attachment; filename="smime.p7s"

MIIF8wYJKoZIhvcNAQcCoIIIF5DCCBeACAQExCTAHBgUrDgMCGjALBgkqhkiG9w0B
BwGgggN9MIIDeTCCAmGgAwIBAgIBAjanBgkqhkiG9w0BAQUFADA7MQswCQYDVQQG
EwJSTzEMMAoGA1UECgwDTVRBMQwwCgYDVQQQLDANTVEkxEDAObgNVBAMMB1Rlc3Qg
Q0EwHhcNMTUwNTE2MTk0OTE4WhcNMTYwNTE1MTk0OTE4WjBpMQswCQYDVQQGEwJS
TzEMMAoGA1UECgwDQVRNMQwwCgYDVQQQLDANTVEkxZFjAUBgNVBAMMDU1pcmNlYSBD
ZXJuYXQxJjAkBgkqhkiG9w0BCQEF21pcmNlYS5jZXJuYXRhZ21haWwY29tMIIB
IjanBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAShJZ1Hhn8SqkW51E8Rf8Szc
pei4RSu4MzRIOBFjnrQQ7XOz399B/xxD9OEc5fmBIFdn81yzoMJKVVSgb76pqUoSj
rSGGfV4YYtrJ3P3cptKVQYtd4Dfke0clUhpXixbjrTrzepVNIAjktJacaeKrV9Qo
POnCs0ECg28ZEM9dekFfsTfGQrXp8+T3vzeUVGtWQt4gEVib+PTLc5NzTZRHMSu
C/8l++4B7an6yc+HwB3nY/+XfE13nhbd+QgX/6rgLw4vlxbfupDiQ7Go5KWI1VsF
B09fhYSihwh3dYgJgZLamxafpWFFndZnl8XISvrbKvlalvMpA2uIENpI1bthFQID
AQABolowWDAJBgNVHRMEAIAAMASGA1UdDwQEAwIF4DAdBgNVHQ4EFgQU3RplFgj4
crz1f563hnLY+giStTwwHwYDVR0jBBgwFoAUoT50TyOGjBrkt/N3610mFtp8pEgw
DQYJKoZIhvcNAQEFBQADggEBAJLP9UE6p5N4dRXrT2mxholkXemyU292CqUxaSVk
E51cirKQbXPxdAM2kYPT6Jv7s3LSPp5nePDcT1grN7dTe+mLlhGD/7qtTrbZB788
xvORFuB/U8BGAvgSKNZBUSvcxkNvI5tMNcfBXxKQy0JHvFvny/OGOiQRiA3kTwrP
1x2nOgSJC8+Vq4BPbqNw/4LfyLC9dDTJL/2gh8h2Yl30iGMr8bsck0Ws/Ca/qgvM
EFLjGjAKJr0dhG/9MQKaoDGXbKc+XgrTEqNzCTjuyTJuqPHqkL5bDGh9T0thEAp9
FhKdFsQwxbF38K7RlWJ/0mki jpayb2dCtQuBaJmq4j414tcxggJAMII CPAIBATBA
MDsx CzAJBgNVBAYTALJPMQwwCgYDVQQKDANNVEExDDAKBgNVBAsMA1NUSTEQMA4G
A1UEAwwHVGVzdCBDQQIBAJAHBgUrDgMCGqCB2DAYBgkqhkiG9w0BCQMxCwYJKoZI
hvcNAQcCBMBwGCSqGSIb3DQEJBTEPFw0xNTA5MDUxMjU4NDVaMCMGCSqGSIb3DQEJ
BDEWBBTnOmJo4GdyxlJuWts+FSjBhqEQCjB5BgkqhkiG9w0BCQ8xbDBqMASGCWCG
SAFlAwQBKjALBglghkgBZQMEARYwCwYJYIZIAWUDBAECMAoGCCqGSIb3DQMHMA4G
CCqGSIb3DQMCAGIAgDANBgghkiG9w0DAGIBQDAHBGUrDgMCBzANBgghkiG9w0D
AgIBKDANBgkqhkiG9w0BAQEFAASCAQBUq/f016Xk/DQuX364n5wmOMbL80IFnWgn
rB7FBllE npLlgMy2mpTCMvU267yLiTidvbVEQQKumv8FQHeiGEDyQ5D3ygIyAq5v
DGpJxtypMQN8KCd/Y7AbmI8YFsMk7vEY4QUod6vWuRitBQStNzNmGon7WIA6KoVI
FhwtrNXpauQU62XxhM+vwEB38xM9Msx3QX61+4wR7yzTN1tgudmf4rTVgmj743V5
DP8SgxO6ily55W9Ec4gFCvrDb/QhyaQPk0RC63ay86o6rv6j+mzuwficRoEf5RWG
sWcNmKiWMJHMKegIJFOSDjWCDA04wP4k2XnvERUrvAzEzwryGwoU

-----18E6332C520072EE6094C1ACD68234A9--

6.2.4. Semnarea și criptarea mesajelor

Protocolul permite combinarea celor două operații descrise anterior pentru a putea transmite un singur mesaj S/MIME care să fie și semnat și criptat. Acest tip de mesaj asigură toate cele patru servicii de securitate: confidențialitate, autenticitate, integritate și non-repudiare.

Conținutul MIME poate fi mai întâi semnat, iar rezultatul obținut poate fi apoi criptat. Standardul permite și realizarea celor două operații în ordine inversă (mai întâi criptarea și apoi semnarea rezultatului). Ordinea operațiilor de criptare și semnare este lăsată la nivelul aplicațiilor, ca opțiune de implementare și/sau folosire.

6.2.5. Extensiile ESS

La nivelul protocolului S/MIME au fost adăugate câteva servicii suplimentare de securitate cunoscute sub numele de extensii ESS (*Enhanced Security Services*). Acestea sunt definite în RFC 2634.

Recipise semnate (*Signed receipts*)

Este un serviciu opțional care permite confirmarea livrării unui mesaj de e-mail. Acesta poate fi folosit numai în cazul mesajelor semnate. Transmițătorul unui e-mail semnat poate fi informat printr-o recipisă semnată că e-mailul său a fost primit la destinație și, în plus, că semnătura din mesaj a fost verificată cu succes. Recipisa este transmisă sub forma unui e-mail de răspuns semnat. În cazul când semnătura nu poate fi verificată, recipisa nu este transmisă.

Expeditorul mesajului de e-mail trebuie să opteze explicit pentru primirea unei recipise semnate. În acest caz mesajul său S/MIME va conține un atribut special numit *receiptRequest*. Acesta este adăugat la semnătură sub formă de atribut semnat în setul de informații de semnare (câmpul *signerInfo*) din structura *SignedData*. După verificarea semnăturii mesajului primit, dacă acesta conține atributul *receiptRequest*, aplicația destinatarului va emite și transmite automat un e-mail de răspuns care conține recipisa semnată. Acest răspuns este un mesaj S/MIME semnat, semnătura fiind creată pe un set de date ce conțin informații din mesajul inițial printre care și valoarea semnăturii.

Sunt mai multe opțiuni de folosire pentru acest tip de serviciu. Nu este obligatoriu ca recipisele să fie generate de toți destinatarii care primesc și procesează un e-mail semnat. La nivelul atributului *recipientRequest* este precizată lista destinatarilor care trebuie să răspundă cu recipise. De asemenea, pot fi precizate adrese alternative (altele decât cea a expeditorului) către care să

fie trimisă recipisa semnată. O recipisă trebuie expediată o singură dată la o adresă, iar aplicațiile mențin o evidență strictă a recipiselor expediate.

Etichete de securitate (*Security labels*)

Etichetele de securitate sunt informații opționale, adăugate la mesajele S/MIME pentru a preciza sensibilitatea conținutului protejat. Etichetele de securitate pot fi incluse doar în cazul mesajelor semnate scopul fiind acela de a controla accesul la conținutul mesajului. O aplicație de e-mail interpretează eticheta de securitate existentă la nivelul unui mesaj numai după verificarea semnăturii și decide apoi dacă permite sau nu accesul la conținutul mesajului.

Etichetele de securitate sunt incluse la nivelul structurii *signerInfo*, folosind câmpul atributelor semnate. Standardul definește atributul denumit *ESSSecurityLabel*. Acesta conține o politică de securitate și nivelul de clasificare al mesajului. Sunt definite următoarele nivele: unmarked (0), unclassified (1), restricted (2), confidential (3), secret (4), top-secret (5), însă la nivelul unei organizații pot fi definite și alte nivele. Etichetele pot fi utilizate de asemenea pentru echivalarea politicilor de securitate folosite la nivelul a două sau mai multe organizații. Aplicația expeditorului poate adăuga la mesaj un set de nivele de securitate considerate ca fiind echivalente. Acestea sunt incluse în cadrul unui atribut semnat numit *Equivalent-Labels*. La destinație, atributul este procesat după verificarea cu succes a semnăturii și numai cu condiția să poată fi stabilit un nivel de încredere în semnatarul mesajului.

Certificatul de semnare (*Signing certificate*)

Formatul *SignedData*, folosit pentru formatarea mesajelor S/MIME semnate, este vulnerabil la atacul de substituție a certificatului de semnare. Certificatul semnatarului este foarte important pentru că identifică contextul și dreptul de semnătură realizată cu cheia privată asociată. Folosind formatarea CMS, aceasta nu „leagă” în nici un fel semnătura calculată asupra datelor de acest certificat. Algoritmul criptografic de calcul al semnăturii ia în considerare doar valoarea hash obținută peste datele care sunt semnate sau peste atributele semnate, iar identificatorul certificatului semnatarului din structura *SignerInfo* nu este inclus în acest algoritm. Din această cauză sunt posibile câteva variante de atac asupra mesajului semnat, toate având la bază înlocuirea certificatului semnatarului după semnarea mesajului. Un prim exemplu de atac: utilizatorul poate semna un mesaj de e-mail folosind unul din certificatele sale, iar ulterior altcineva (sau chiar utilizatorul) poate schimba certificatul utilizat, cu alt certificat emis pentru un scop diferit. Un alt exemplu poate fi un certificat expirat sau revocat, folosit la semnarea unui e-mail iar ulterior înlocuit în mesajul S/MIME rezultat cu unul valid, semnătura fiind astfel validată și ea.

Pentru a contracara acest tip de atac, a fost propus atributul semnat *ESSSigning-Certificate* (cu varianta *ESSSigning-Certificate-v2*) Acesta conține un rezumat calculat printr-o funcție hash peste valoarea certificatului semnatarului. Prezența atributului respectiv în cadrul atributelor semnate va conduce la „legarea” certificatului semnatarului de semnătură și implicit la protejarea semnăturii împotriva atacurilor de substituție. Orice încercare de substituie a certificatului semnatarului va duce automat la invalidarea semnăturii.

6.2.6. Managementul cheilor publice

Ca și în cazul PGP, protocolul S/MIME folosește criptografia cu chei publice pentru a semna sau cripta mesajele. Pentru garantarea autenticității cheilor publice, S/MIME utilizează însă infrastructurile bazate pe Autorități de Certificare (CA-uri) ierarhice și certificate X.509v3. Certificatele sunt necesare în următoarele cazuri:

- La construcția mesajelor S/MIME criptate. Aplicația expeditorului trebuie să aibe acces la certificatele ce conțin cheile publice ale destinatarilor mesajului.
- La construcția mesajelor S/MIME semnate. Aplicația expeditorului trebuie să aibe configurat certificatul (și cheia privată asociată) semnatarului.
- La validarea semnăturilor incluse în mesajele S/MIME semnate. Aplicațiile destinatarilor trebuie să aibe acces la certificatele folosite la semnare.

Obținerea certificatelor necesare și a informațiilor de validare pentru acestea constituie una din problemele mai dificil de rezolvat sau automatizat la nivelul aplicațiilor S/MIME. Pentru obținerea certificatelor, de regulă sunt folosite următoarele metode:

- Extragerea certificatelor din mesajele semnate primite anterior. De regulă, mesajele semnate conțin certificatul semnatarului, uneori și lanțul de certificate necesar validării căii. O aplicație care procesează un email semnat utilizează aceste certificate pentru validarea semnăturii și salvează certificatele respective pentru tranzacțiile viitoare.
- Aplicația expeditorului trimite un mesaj unui destinatar pentru a primi de la acesta un răspuns semnat care să conțină și certificatul său.

- Accesul și căutarea automată în bazele de tip repository de certificate. De regulă, aplicațiile au suport și configurări pentru accesul la servicii LDAP. Căutarea certificatelor într-un astfel de repository se face folosind numele X.500 din subiectul certificatului.

Protocolul recomandă validarea certificatelor folosite. Procesul de validare presupune validarea căilor de certificare în raport cu punctele de încredere (certificate rădăcină) configurate la nivelul aplicațiilor. Opțional, are loc apoi verificarea stării de revocare a certificatelor folosind liste de certificate revocate (CRL-uri).

Standardul S/MIME stabilește un profil pentru certificatele folosite în cadrul protocolului. Acesta este bazat pe următoarele extensii:

- *Basic Constraints* – certificatele de CA trebuie indicate folosind corect extensia respectivă, iar certificatele entităților client nu trebuie să conțină această extensie.
- *Key Usage* – aplicațiilor S/MIME trebuie să invalideze o semnătură realizată pe baza unui certificat ce conține această extensie dar nu include valorile biților corespunzători pentru digitalSignature sau nonRepudiation.
- *Subject Alternative Name* – trebuie să includă adresa de e-mail a posesorului certificatului folosind câmpul *rfc822Name*.
- *Extended Key Usage* – trebuie să includă valoarea pentru *emailProtection*.

Anexe

Anexa 1. Aplicație client-server securizată prin SSL

Componenta server

```
/*
Acesta este un exemplu de program server care comunica cu
programele client folosind protocolul SSL.
Serverul este configurat cu certificat si cheie privata si
primește conexiuni SSL pe portul 4443.
Canalul de comunicare este cu autentificare mutuala (clientul
trebuie sa se autentifice la server cu certificat).
Verificarea certificatului de client se face pe baza unei liste
de certificate de CA, configurata la nivelul serverului.
Pentru comunicare, serverul nu folosește interfata BIO din
OpenSSL. In acest program, sunt folositi direct
socketi (implementarea winsock 2.0).
Dupa stabilirea unei conexiuni TCP, conexiunea este securizata
SSLv2, SSLv3 sau TLSv1.

```

Acest program folosește funcții din biblioteca OpenSSL.

```
*****/
#include <openssl/ssl.h>
#include <openssl/x509v3.h>
#include <openssl/bio.h>
#include <openssl/err.h>
#include <openssl/applink.c>
#include <winsock.h>

/* Portul pe care asculta serverul */
#define SERVER_PORT 4443

void print_certinfo (const char * label, X509 *cert);

void main ()
{
    int ret;

    /* Descriptor socket server */
    int server;
    /* Descriptor socket client */
    int client;
    /* Detalii adresa server (local)*/
    struct sockaddr_in server_addr;
```

```

/* Detalii adresa client (remote)*/
struct sockaddr_in client_addr;
unsigned int client_len;

/*****
Pasul 1: Se face alocarea si initializarea unui context SSL.
Acesta va contine configuratia SSL folosita.
*****/

/* Secventa necesara pentru initializarea bibliotecii SSL */
SSL_library_init();
SSL_load_error_strings();

/* Alocare context SSL.
   Accepta versiunile de protocol: SSLv2, SSLv3 sau TLSv1 */
SSL_CTX *ctx = SSL_CTX_new(SSLv23_server_method());

/* Se incarca certificatul pentru
   server (fisierul server-atm.crt) */
SSL_CTX_use_certificate_file(ctx,
    "server-atm.crt", SSL_FILETYPE_PEM);

/* Se incarca cheia privata pentru
   server (fisierul server-atm.key) */
SSL_CTX_use_PrivateKey_file(ctx,
    "server-atm.key", SSL_FILETYPE_PEM);

/* Verifica corespondenta dintre cheia privata si certificat */
if (!SSL_CTX_check_private_key(ctx)) {
    printf("Eroare,
           cheia privata nu corespunde certificatului\n");
    exit(1);
}

/* Incarca lista de CA -uri pentru autentificare clienti */
if (!SSL_CTX_load_verify_locations(ctx, "ATMCA.crt", NULL)) {
    ERR_print_errors_fp(stderr);
    exit(1);
}

/* Autentificarea va fi mutuala. Certificatul clientului este
verificat pe un lant de de adancime maxima de 4 certificate */
SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, NULL);
SSL_CTX_set_verify_depth(ctx, 4);

/* Aloca structura SSL (contine informatii sesiune SSL).
   Este configurata si suita de cifruri acceptate */
SSL* ssl = SSL_new(ctx);
SSL_set_cipher_list(ssl, "ALL");

```



```

/*****
Pasul 2. Se configureaza partea specifica de comunicatie
pentru server folosind interfete sockets.

Este folosita implementarea winsock 2.0 pentru
*****/

/* Necesar pentru winsock 2.0*/
WSADATA wsa;
WSAStartup(MAKEWORD(2, 0), &wsa);

/* Alocare socket server */
server = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

/* Configurare parametrilor server */
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
server_addr.sin_port = htons(SERVER_PORT);

ret = bind(server,
            (struct sockaddr *) &server_addr, sizeof(server_addr));
if(ret < 0) {
    printf("Eroare la socket bind");
    exit(1);
}

ret = listen(server, 5);
if (ret < 0) {
    printf("Eroare la socket listen");
    exit(1);
}

/*****
Pasul 3. Secventa clasica (la server) de preluare a
conexiunilor client. Dupa stabilirea unei conexiuni TCP,
aceasta va fi securizata SSL
*****/

for (;;)
{
    client_len = sizeof(client_addr);

    /* Serverul asteapta conexiuni de la clienti */
    client = accept(server, (struct sockaddr *)
                    &client_addr, (int *) &client_len);

    /* Exista o conexiune noua */
    printf("Conexiune noua de la IP: %s\n",
           inet_ntoa(client_addr.sin_addr));
}

```

```

/* Se asigneaza conexiunea TCP la structura SSL */
SSL_set_fd(ssl, client);

/* Se realizeaza handshake -ul SSL la server */
if (SSL_accept(ssl) == -1) {

    ERR_print_errors_fp(stderr);
    continue;
}

/*****
Pasul 4. Etapa de verificare a certificatului
clientului conectat
*****/

/* Preia certificatul clientului
   din sesiunea SSL pentru verificari la server */
X509 *cert = SSL_get_peer_certificate(ssl);

if (cert == NULL) {

    printf("Eroare, clientul nu are certificat\n");

    /* Serverul inchide sesiunea SSL cu clientul
       si trece mai departe pentru alta conexiune */
    SSL_shutdown(ssl);
    continue;
}
else {
    /* Afisare informatii certificat */

    print_certinfo ("Info certificat client: ", cert);
    X509_free(cert);
}

/*****
Pasul 5. Clientul este autentificat, se face schimbul de date.
*****/

char buf[4096];

/* Datele transmise de client sunt citite
   de pe conexiunea SSL si sunt afisate pe ecran */
ret = SSL_read(ssl, buf, sizeof(buf) - 1);
buf[ret] = '\0';
printf ("\nAm primit de la client: '%s'", buf);

/* Serverul transmite un raspuns catre client */
const char *data = "Datele serverului";
SSL_write(ssl, data, strlen(data));

```

```

    printf ("\nAm transmis la client: '%s'\n", data);

    /* Inchide sesiunea SSL cu clientul */
    SSL_shutdown(ssl)
}

}

/*****
Functie de afisare informatii certificat (<ISSUER> <SUBJECT>)
Nota: Functia este folosita si in aplicatia client
*****/
void print_certinfo (const char *label, X509 *cert)
{
    printf ("%s<ISSUER: %s>, <SUBJECT: %s>", label,
        X509_NAME_oneline (X509_get_issuer_name(cert), NULL, 0),
        X509_NAME_oneline (X509_get_subject_name(cert), NULL, 0));
}

```

Componenta client

```
/******  
Exemplu de aplicatie client care comunica cu serverul peste SSL.  
Clientul este configurat cu certificat si cheie privata si se  
conecteaza la server pe portul 4443.  
Canalul de comunicatie este cu autentificare mutuala (clientul  
se autentifica la server cu certificat).  
Verificarea certificatului de server se face pe baza unei liste  
de certificate de CA, configurata la nivelul clientului.  
Clientul foloseste interfata BIO (OpenSSL) pentru realizarea  
comunicatiei.
```

Acest program foloseste functii din biblioteca OpenSSL.

```
*****/  
#include <openssl/ssl.h>  
#include <openssl/x509v3.h>  
#include <openssl/bio.h>  
#include <openssl/err.h>  
#include <openssl\applink.c>  
  
/* Adresa serverului */  
#define SERVER_URL "server.atm.ro:4443"  
  
int verify_callback(int preverify, X509_STORE_CTX* x509_ctx);  
void print_certinfo (const char * msg, X509 *cert);  
  
void main ()  
{  
    int ret;  
  
    /* Initializare biblioteca SSL */  
    SSL_library_init();  
    SSL_load_error_strings();  
  
    /******  
    Pasul 1: Alocare si configurare context SSL.  
    *****/  
  
    /* Alocare context SSL */  
    SSL_CTX* ctx = SSL_CTX_new(SSLv23_method());  
  
    /* Clientul suporta numai protocol TLSv1 */  
    const long flags = SSL_OP_NO_SSLv2 |  
                      SSL_OP_NO_SSLv3 | SSL_OP_NO_COMPRESSION;  
    SSL_CTX_set_options(ctx, flags);
```

```

/* Incarca certificatul clientului (fisierul client-atm.crt) */
SSL_CTX_use_certificate_file(ctx,
                             "client-atm.crt", SSL_FILETYPE_PEM);

/* Incarca cheia privata (fisierul client-atm.key) */
SSL_CTX_use_PrivateKey_file(ctx,
                             "client-atm.key", SSL_FILETYPE_PEM);

/* Verifica corespondenta dintre cheia privata si certificat */
if (!SSL_CTX_check_private_key(ctx)) {
    printf("Eroare,
           cheia privata nu corespunde certificatului\n");
    exit(1);
}

/* Configurare verificare certificat server
(si functie de callback pentru afisare detalii certificate) */

SSL_CTX_set_verify (ctx, SSL_VERIFY_PEER, verify_callback);
SSL_CTX_set_verify_depth(ctx, 4);

/* Se incarca lista de CA -uri trusted
    pentru autentificarea serverului (fisierul ATMCA.crt)*/
SSL_CTX_load_verify_locations(ctx, "ATMCA.crt", NULL);

/*****
Pasul 2: Stabilire conexiune SSL la server (se foloseste
interfata BIO)
*****/

/* Se alocă un stream BIO pentru comunicatia SSL */
BIO *server = BIO_new_ssl_connect(ctx);
if(server == NULL) {
    printf ("\nEroare la initializare stream de comunicatie");
    exit (1);
}

/* Configurare stream */
ret = BIO_set_conn_hostname(server, SERVER_URL);
if(ret != 1) {
    printf ("\nEroare de conexiune la server");
    exit (1);
}

SSL *ssl = NULL;
BIO_get_ssl(server, &ssl);
if(ssl == NULL) {
    printf ("\nEroare pe streamul SSL");
    exit (1);
}

```

```

/* Configurare suita cifruri acceptate */
ret = SSL_set_cipher_list(ssl, "ALL");

/* Conectare la server */
ret = BIO_do_connect(server);
if(ret != 1) {
    printf ("\nEroare de conectare la server");
    exit (1);
}

/* Se realizeaza handshake -ul SSL */
ret = BIO_do_handshake(server);
if(ret != 1) {
    printf ("\nEroare handshake SSL");
    exit (1);
}

/*****
Pasul 3: Verificare certificat server
*****/

/* Daca se obtine certificat server, se afiseaza detalii,
altfel eroare */
X509* cert = SSL_get_peer_certificate(ssl);

if (cert != NULL) {
    print_certinfo ("Info certificat server: ", cert);
    X509_free(cert);
}
else {
    printf ("\nEroare certificat server (nu exista)");
    exit (1);
}

/* Verifica rezultatul validarii lantului
           pentru certificatul serverului */
ret = SSL_get_verify_result(ssl);
if(ret != X509_V_OK) {
    printf("\nEroare de validare certificat server");
    exit (1);
}

/*****
Nota: Aici pot fi realizate si alte verificari pentru
certificat:
- validare hostname server
- validitate
- stare de revocare
- etc.
*****/

```

```

/*****
Pasul 4. Severul este autentificat, urmeaza schimbul de date.
*****/

/* Transmite date la server pe stream -ul SSL */
const char *data = "Datele clientului";
BIO_puts(server, data);
printf ("\nAm transmis la server: '%s'", data);

/* Citeste datele transmise de server pe stream -ul SSL,
si le scrie la consola stdout */
BIO *out = BIO_new_fp(stdout, BIO_NOCLOSE);
BIO_puts (out, "\nAm primit de la server: '");
int len = 0;
do {
    char buff [2048];
    len = BIO_read(server, buff, sizeof(buff));
    if(len > 0)
        BIO_write(out, buff, len);

} while (len > 0 || BIO_should_retry(server));

BIO_puts(out, "'\n");

/* Se inchide sesiunea SSL cu serverul */
SSL_shutdown(ssl);

/* Se elibereaza stream -uri si context SSL */
if(server != NULL)
    BIO_free_all(server);

if(out != NULL)
    BIO_free_all(out);
if(ctx != NULL)
    SSL_CTX_free(ctx);
}

/*****
Functia de callback folosita in aplicatia client. Este apelata
pentru fiecare certificat din lantul de certificate.

Implementarea curenta afiseaza detalii din certificatul curent
*****/
int verify_callback(int preverify, X509_STORE_CTX* x509_ctx)
{
    X509* cert = X509_STORE_CTX_get_current_cert(x509_ctx);
    int depth = X509_STORE_CTX_get_error_depth(x509_ctx);
    printf("\nverify_callback (depth = %d):", depth);
    print_certinfo ("", cert);
}

```

```

    /* Pot fi realizate si alte tipuri de operatii pentru
    certificatul procesat

    return preverify;
}

/*****
Functia afisare info certificat (<ISSUER> <SUBJECT>)
Nota: functia este folosita si in aplicatia server
*****/
void print_certinfo (const char * label, X509 *cert)
{
    printf ("%s<ISSUER: %s>, <SUBJECT: %s>", label,
            X509_NAME_oneline (X509_get_issuer_name(cert), NULL, 0),
            X509_NAME_oneline (X509_get_subject_name(cert), NULL, 0));
}

```


Bibliografie

- [AL02] C. Adams, S. Lloyd. „Understanding PKI: Concepts, Standards, and Deployment Considerations – Second Edition”. Addison Wesley, 2002
- [Atan07] A. Atanasiu. „Securitatea informației, Vol. I (Criptografie)”. Editura INFODATA Cluj, 2007
- [BBPS12] E. Barker, W. Burr, W. Polk, M. Smid. „Recommendation for Key Management – Part 1: General”. NIST Special Publication 800-57, 2012
- [Bica05] I. Bica. „Securitatea documentelor electronice în rețele de calculatoare”. Editura Academiei Tehnice Militare, 2005
- [Bidg06] H. Bidgoli. „Handbook of Information Security”. John Wiley & Sons, 2006
- [BP01] S. Burnett, S. Paine. „RSA Security’s Official Guide to Cryptography”. McGraw-Hill, 2001
- [Burr98] W.E. Burr. “Public Key Infrastructure (PKI) Technical Specifications: Part A - Technical Concept of Operations”. NIST TWG-98-59, 1998
- [Carm06] J. H. Carmouche. „IPSec Virtual Private Network Fundamentals”. Cisco Press, 2006
- [Davi01] C. Davis. „IPSec: Securing VPNs”, RSA Press, 2001
- [FHOP08] S. Frankel, P. Hoffman, A. Orebaugh, R. Park. „Guide to SSL VPNs”. NIST Special Publication 800-113, 2008
- [FKLO05] S. Frankel, K. Kent, R. Lewkowski, A. D. Orebaugh, R. W. Ritchey, S. R. Sharma. „Guide to IPSec VPNs”. NIST Special Publication 800-77, 2005
- [FIPS46] National Institute of Standards and Technology (NIST) – FIPS PUB 46-3. „Data Encryption Standard (DES)”, 1999
- [FIPS140] National Institute of Standards and Technology (NIST) – FIPS PUB 140-2. „Security Requirements for Cryptographic Modules”, 2001
- [FIPS180] National Institute of Standards and Technology (NIST) – FIPS PUB 180-4. „Secure Hash Standard (SHS)”, 2015
- [FIPS186] National Institute of Standards and Technology (NIST) – FIPS PUB 186-4. „Digital Signature Standard (DSS)”, 2013
- [FIPS197] National Institute of Standards and Technology (NIST) – FIPS PUB 197. „Advanced Encryption Standard (AES)”, 2002
- [FS03] N. Ferguson, B. Schneier. „Practical Cryptography”. Wiley Publishing, 2003
- [Hals05] F. Halsall, „Computer Networking and the Internet, 5/E”, Addison-Wesley, 2005
- [HP01] R. Housley, T. Polk. „Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure”. Wiley Computer Publishing, 2001

- [Kauf02] C. Kaufman. "Network Security, 2/E". Prentice Hall, 2002
- [MABC99] M. W. Murhammer, O. Atakan, Z. Badri, B. Cho, H. J. Lee, A. Schmid. „A Comprehensive Guide to Virtual Private Networks”. IBM Redbooks, 1999
- [MMJP03] D. Maltoni, D. Maio, A.K. Jain, S. Prabhakar. „Handbook of Fingerprint Recognition”. Springer, 2003
- [MOV96] A. Menezes, P. van Oorschot, S. Vanstone. „Handbook of Applied Cryptography”. CRC Press, 1996
- [NDJB01] A. Nash, W. Duane, C. Joseph, D. Brink. „PKI: Implementing and Managing E-Security”. RSA Press, 2001
- [PBD06] L. Parziale, D. Britt, C. Davis. „TCP/IP Tutorial and Technical Overview”. IBM Redbooks, 2006
- [PD12] L. Peterson, B. Davie. „Computer Networks: A Systems Approach 5/E”. Morgan Kaufmann, 2012
- [PKCS#1] RSA Laboratories. “PKCS #1: RSA Cryptography Standard”, 1998
- [PKCS#7] RSA Laboratories. “PKCS #7: Cryptographic Message Syntax Standard”, 1997
- [PPBV01] V. Patriciu, M. Pietroșanu, I. Bica, C. Văduva, N. Voicu. „Securitatea Comerțului Electronic”. Editura All, 2001
- [PPBP06] V. Patriciu, M. Pietroșanu, I. Bica, I. Priescu. „Semnături electronice și securitate informatică”. Editura All, 2006
- [Rhee03] Man Young Rhee. „Internet Security - Cryptographic Principles, Algorithms and Protocols”. John Wiley & Sons, 2003
- [Schn96] B. Schneier. „Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C”. Wiley Computer Publishing, 1996
- [Stall10] W. Stallings. „Cryptography and Network Security: Principles and Practice, 5/E”. Prentice Hall, 2010
- [Stall07] W. Stallings. „Data and Computer Communications, 8/E”. Prentice Hall, 2007
- [Stam06] M. Stamp. „Information Security - Principles and Practice”. John Wiley & Sons, 2006
- [Stin95] D. Stinson. „Cryptography: Theory and Practice”. CRC Press, 1995.
- [Tane11] A. Tanenbaum. „Computer Networks, 5/E”. Prentice Hall, 2011
- [TK04] H. Tipton, M. Krause. „Security Management Handbook, 5/E”. CRC Press, 2004
- [X.509] ITU-T Recommendation X.509. „Information Technology – Open Systems Interconnection – The Directory: Public Key and Attribute Certificate Frameworks”, 2012
- [Yun03] Yau Wei Yun. „The ‘123’ of Biometric Technology”, 2003
- [Zimm95] P. Zimmermann. “The Official PGP User's Guide”. MIT Press, 1995