

## chapter 1 introducere în criptografie

### 1.1 termeni de bază

**Criptografia** este o ramură a matematicii aplicate care este utilizată pentru securizarea și menținerea caracterului privat al informațiilor. În termeni practici, acest lucru implică conversia unui text (fișier, șir de caractere/biți) în clar (plain text) într-unul criptic (numit text cifrat).

Acest proces de conversie sau de codificare a textului clar se numește **criptare**. Procesul invers, de conversie a textului cifrat în text (în) clar se numește **decriptare**.

Ambele procese utilizează (sub o formă sau alta) o procedură de criptare, numită **algoritm de criptare**.

Majoritatea acestor algoritmi sunt de domeniul public, adică sunt cunoscuți, caracterul secret (privat) al comunicării fiind asigurat de utilizarea unei chei de criptare/decriptare, cheie cunoscută (în cazul ideal) doar de către entitățile care sunt îndreptățite să le cunoască, la ambele capete ale canalului de comunicare.

**Criptologia** este o ramură a matematicii care descrie fundamentele matematice ale metodelor criptografice precum și ale principiilor de autentificare și restricționare a accesului la informații.

Termenul de criptologie cuprinde atât **criptografia** (criptarea informațiilor) cât și **criptanaliza** (analiza criptografică), adică arta de a sparge sistemele de criptare.

### 1.2 servicii criptografice

Criptografia oferă următoarele servicii:

- autentificare
- integritatea datelor
- non-repudiere
- confidențialitate

Să detaliem puțin aceste concepte.

**Autentificarea** permite destinatarului mesajului să valideze identitatea expeditorului. Previne insinuarea unei entități neautorizate ca fiind expeditorul legitim al mesajului.

**Integritatea datelor** garantează că mesajul trimis nu a fost modificat sau alterat pe parcursul comunicării. Acest lucru este realizat, de obicei, prin atașarea la mesaj a unui șir de control (digest, hash) de lungime fixă. Acest șir de control poate fi recreat de către destinatar și permite identificarea unei modificări intenționate sau nu a conținutului mesajului trimis.

**Non-repudierea**, înseamnă, în securitatea digitală, două lucruri:

- un serviciu care oferă dovada integrității și (mai ales) a originii datelor
- o modalitate de autentificare cu un nivel înalt de încredere

**Confidențialitatea**, ca serviciu, previne accesul la conținutul real al mesajului din partea entităților neautorizate.

## 1.3 clasificarea algoritmilor criptografici

Există două tipuri principale de algoritmi de criptare cu cheie:

- cheie secretă – algoritmi cu cheie simetrică
- cheie publică – algoritmi cu cheie asimetrică

Algoritmii cu cheie simetrică se bazează pe caracterul privat (secret) al cheii de criptare/decriptare. Cheie este cunoscută de către expeditor și de către destinatar.

Acești algoritmi pot fi clasificați în continuare în algoritmi de tip **șir** (stream) și în algoritmi de tip **bloc** (block). Cei dintâi au ca unitate de codificare un caracter în timp ce ceilalți tip acționează asupra unui bloc de caractere, care este considerat drept unitate de codificare.

Viteza de execuție a algoritmilor simetrici este mult mai mare ca cea a celor asimetrici. Pe de altă parte, schimbul de chei dintre expeditor și destinatar implicat de utilizarea algoritmilor simetrici ridică noi probleme de securitate. În practică, se obișnuiește utilizarea criptării asimetrice pentru generarea și schimbul de chei ce urmează a fi utilizate pentru criptare propriu-zisă (cea simetrică) a mesajului.

## 1.4 algoritmi de criptare cu cheie simetrică

Algoritmii de criptare cu cheie simetrică utilizează o singură cheie (secretă) pentru a efectua atât criptarea cât și decriptarea mesajului. Din acest motiv, păstrarea caracterului secret a cheii comune este crucială pentru păstrarea caracterului secret al comunicării. Algoritmii cei mai utilizați la ora actuală sunt:

- **DES** – Data Encryption Standard (Standard de criptare a datelor) - a fost dezvoltat la mijlocul anilor 70. Este un standard NIST (US National Institute of Standards and Technology). DES este un algoritm de tip bloc (block) care utilizează blocuri de 64 de biți și o cheie de 56 de biți. Lungimea modestă a cheii îl face vulnerabil la atacuri de forță brută. Specificația inițială a acestui algoritm de criptare este FIPS (Federal Information Processing Standards) 46. Cea mai nouă versiune a DES-ului este Triple-DES (sau 3-DES) și se bazează pe utilizarea de trei ori a DES-ului, cu trei chei diferite, independente. Este, desigur, mai puternic decât DES, dar este și mai lent, dacă îl comparăm cu algoritmi mai noi de criptare. 3-DES este specificat în FIPS 46-3 (Octombrie 1999)
- **AES** – Advanced Encryption Standard (Standard de Criptare Avansată) - obiect al FIPS 197 (noiembrie 2001). AES este un cifru bloc care utilizează blocuri de 128 biți și o cheie de dimensiune 128 biți. Sunt specificate și variante utilizând chei de 192 și 256-biți. Ceea ce este specific pentru acest algoritm este faptul că procesează date la nivel de octet, spre deosebire de procesarea la nivel de biți care a fost folosită anterior. Algoritmii este eficient și considerat sigur.

## 1.5 distribuirea cheilor secrete

Utilizarea algoritmilor de criptare cu cheie simetrică presupune existența unui sistem robust de distribuție a cheilor. Desigur că aceste chei, fiind ele însele secrete, trebuie încryptate înainte de a fi trimise în mod electronic sau pot fi distribuite prin alte mijloca (prin curieri, de exemplu) pentru a preveni interceptarea cheilor de către o entitate neautorizată.

Există două standarde pentru distribuirea automată a cheilor secrete. Primul standard, numit X9.17 este definit de către ANSI (American National Standards Institute) iar cel de-al doilea este protocolul Diffie-Hellman.

În acest curs ne vom ocupa doar de cea de-a doua metodă, protocolul Diffie-Hellman.

## 1.6 algoritmi cu cheie asimetrică

Algoritmii cu cheie asimetrică se bazează pe două key distincte pentru implementarea celor două faze: criptarea și, respectiv, decriptarea:

- o cheie publică, care poate fi distribuită sau făcută publică la cerere
- o cheie privată (secretă) care corepunde unei anume chei publice, dar care este cunoscută doar de proprietarul cheii sau de către alte entități autorizate.

Fiecare din aceste două chei definește o funcție de transformare. Cele două transformări definite de către o pereche de chei publice/private sunt inverse una celeilalte și pot fi utilizate pentru criptarea/decriptarea unui mesaj. Nu este important care din cele două chei este folosită pentru criptare/decriptare.

Deși algoritmii de criptare cu cheie asimetrică sunt mai lenți decât cei cu cheie simetrică, aceștia au avantajul de a elimina procesul de distribuire a cheilor.

Algoritmii principali de criptare cu cheie asimetrică:

- **RSA** – (Rivest-Shamir-Aldeman) este cel mai utilizat algoritm cu cheie asimetrică. Este utilizat în principal pentru distribuirea cheilor și pentru semnături digitale. Toate calculele sunt făcute modulo  $n$  (de regulă de peste 2048 biți, adică peste 617 cifre zecimale)  $n = p \cdot q$ , unde numărul  $n$  este public, dar  $p$  și  $q$  sunt numere prime secrete. Pornind de la reprezentarea numerică a unui mesaj  $m$  și de la un exponent public  $e$ , este creat un text cifrat  $c = m^e \pmod{n}$ . Destinatarul utilizează exponentul invers al lui  $e$  modulo  $(p-1)(q-1)$  (i.e.  $d = e^{-1} \pmod{(p-1)(q-1)}$ ) pentru decriptare, adică  $c^d = m^{(e \cdot d)} = m \pmod{n}$ . Cheia privată este  $(n, p, q, e, d)$  (sau doar  $p, q, d$ ) în timp ce cheia publică este  $(n, e)$ . Mărimea lui  $n$  trebuie să fie cel puțin de 1024 biți (peste  $10^{300}$ ), dar cerințele curente pretind o valoare de 2048 biți.
- **Rabin** – acest sistem de criptare este echivalent cu procesul de factorizare a unui număr, problemă care este în sine dificilă. Punctul de plecare îl reprezintă (ca și în cazul RSA) un număr mare  $n = p \cdot q$ , unde  $p$  și  $q$  sunt numere prime secrete. Mesajul original  $m$  (văzut ca și număr) este ridicat la pătrat ( $c = m^2 \pmod{n}$ ) și trimis astfel la destinație. Destinatarul calculează rădăcina pătrată a lui  $c \pmod{p}$  și  $c \pmod{q}$ , rezultând 4 variante distincte ale mesajului original. Una din variante e cea corectă. Deși nu este obiectul unui standard (RSA este standardizat), algoritmul este bine explicat în câteva cărți și prezentări. Cheile de 1024 de biți și peste sunt considerate sigure.

## 1.7 funcții hash

Funcțiile hash au ca intrare un mesaj (fișier, șir) de lungime arbitrară și generează o sumă/șir de control (numit și hash sau digest) de lungime fixă. Lungimea acestui șir de control depinde de funcția utilizată, dar în general este între 128 și 512 biți.

Funcțiile hash sunt utilizate în special în trei domenii:

- garantarea integrității unui mesaj (sau a unui fișier descărcat) prin atașarea la mesaj a hash-ului generat. Destinatarul poate recalcula hash-ul mesajului/fișierului primit și acesta poate fi comparat cu hash-ul atașat de către expeditor. Dacă acestea coincid, avem garanția (cu un foarte înalt grad de încredere) că mesajul/fișierul nu a fost alterat.
- Sunt parte a procesului de creare a semnăturii digitale
- stocarea parolelor – parolele nu sunt aproape niciodată (sau nu ar trebui să fie) stocate în forma lor originală. Ceea ce este stocat, în general, este un hash al parolei. Atunci când un utilizator introduce o parolă, este calculat hash-ul acesteia, care este apoi comparat cu hash-ul stocat. Dacă ele coincid, parola este considerată ca fiind cea corectă.

Cele mai utilizate funcții hash sunt cele din familia MD și SHA - și anume MD5 și SHA-1, cele mai noi fiind SHA-2 și SHA-3. SHA înseamnă Secure Hash Algoritm. O altă funcție hash de interes este RipeMD -

## chapter 1

160. Funcțiile MD generează un hash de 128 biți și au fost proiectate de compania RSA Security. În timp ce MD5 este încă răspândit, MD4 a fost spart și este considerat nesigur. SHA-1 și RipeMD-160 sunt considerate sigure pentru moment. În timp ce SHA-2 este o extensie a SHA-1, SHA-3 oferă un nou algoritm pentru calculul hash-ului.

Începând cu cele mai noi funcții, iată o listă de funcții hash de interes practic.

- SHA-3 utilizează algoritmul Keccak, o construcție de burete (sponge) în care blocurile de mesaje sunt XOR-ed într-un subset al stării, care este apoi transformat ca un întreg. În versiunea utilizată în SHA-3, starea constă într-o matrice de  $5 \times 5$  cuvinte pe 64 de biți, în total 1600 de biți. Procesul de standardizare a fost finalizat în august 2015 ca FIPS 202 - <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>.
- SHA-2 include schimbări semnificative față de predecesorul său, SHA-1. Familia SHA-2 constă din șase funcții hash cu digest (valori hash) care sunt de lungime 224, 256, 384 sau 512 biți, cunoscute sub denumirile: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 și SHA-512/256.
- SHA-1 - Algoritm Hash Securizat. Publicat de Guvernul Statelor Unite. Specificația sa este obiectul FIPS 180-1 (aprilie 1995). FIPS reprezintă Federal Information Processing Standard (Standardele Federale de Procesare a Informațiilor). Produce un hash de 160 de biți (5 cuvinte pe 32 de biți).
- RipeMD-160 - conceput ca înlocuitor pentru seria MD. Produce un hash de 160 de biți (sau 20 de octeți, dacă doriți).
- MD5 - Message Digest 5. Dezvoltat de laboratoarele RSA. Produce un hash de 128 biți. Încă în uz, mai ales pentru verificarea integrității mesajelor (sau a fișerelor descărcate).
- MD2, MD4 - Algoritmi de hash mai vechi dezvoltate de către RSA Data Security. Deoarece au defecte cunoscute, ele sunt doar de interes istoric.

## 1.8 semnătura digitală

Unii algoritmi cu cheie publică pot fi utilizați pentru a genera semnături digitale. O semnătură digitală este o cantitate mică de date care poate fi creată utilizând o cheie privată și există o cheie publică care poate fi utilizată pentru a verifica dacă semnătura a fost generată cu adevărat utilizând cheia privată corespunzătoare. Algoritmii utilizați pentru a genera semnătura trebuie să fie astfel încât, fără a cunoaște cheia privată, nu este posibilă crearea unei semnături care să se confirme ca fiind validă.

Semnăturile digitale sunt folosite pentru a verifica dacă un mesaj vine de la expeditorul revendicat (presupunând că numai expeditorul cunoaște cheia privată corespunzătoare cheii publice). Aceasta se numește autentificare (a originii datelor). Ele pot fi, de asemenea, folosite pentru a marca temporal (timestamp) documente: o entitate de încredere semnează documentul și marca temporală cu cheia privată, certificând astfel că documentul a existat la momentul menționat.

Semnăturile digitale pot fi, de asemenea, folosite pentru a certifica faptul că o cheie publică aparține unei anumite entități. Aceasta se face prin semnarea combinației cheii publice și a informațiilor despre proprietarul acesteia printr-o cheie de încredere. Structura de date rezultată este adesea numită un certificat de cheie publică (sau pur și simplu, un certificat). Certificatele pot fi considerate ca fiind similare cu pașapoartele care garantează identitatea purtătorilor lor.

Entitatea de încredere care eliberează certificate entităților identificate se numește o autoritate de certificare (CA – Certificate Authority). Autoritățile de certificare pot fi considerate ca fiind similare cu guvernele care eliberează pașapoarte pentru cetățenii lor.

O autoritate de certificare poate fi administrată de un furnizor extern de servicii de certificare sau chiar de un guvern sau CA poate aparține aceleiași organizații ca entitățile. De asemenea, CA pot emite certificate altor (sub-) CA. Acest lucru duce la o ierarhie de certificare asemănătoare arborilor. Cel mai înalt CA de încredere din arbore este numit CA rădăcină. Ierarhia de încredere formată de entitățile finale, sub-CA-uri și rădăcină CA este numită infrastructură cu cheie publică (PKI – Public Key Infrastructure).

O infrastructură cu cheie publică nu necesită neapărat o ierarhie sau rădăcini acceptate universal și fiecare parte poate avea puncte de încredere diferite. Acesta este conceptul web de încredere utilizat, de

exemplu, în PGP (Pretty Good Privacy).

O semnătură digitală a unui document arbitrar este în mod tipic creată prin calcularea unui hash (digest) al mesajului din document și prin concatenarea acestuia cu informații despre semnatar, un marcaj temporal (de timp) etc. Aceasta se poate face prin aplicarea unei funcții hash criptografice pe date. Șirul rezultat este apoi criptat utilizând cheia privată a semnatarului utilizând un algoritm adecvat. Blocul de biți criptat rezultat este semnătura. Acesta este adesea distribuit împreună cu informații despre cheia publică care a fost utilizată pentru semnarea acestuia.

Pentru a verifica o semnătură, destinatarul determină mai întâi dacă are încredere că cheia aparține persoanei de care ar trebui să aparțină (folosind un certificat sau o cunoaștere a priori) și apoi decriptează semnătura utilizând cheia publică a persoanei. Dacă semnătura se decriptează în mod corespunzător și informațiile se potrivesc cu cea a mesajului (digest (hash) corect al mesajului etc.), semnătura este acceptată ca valabilă. Pe lângă autentificare, această tehnică asigură și integritatea datelor, ceea ce înseamnă că se detectează modificări neautorizate sau accidentale ale datelor în timpul transmiterii.

Câteva metode pentru realizarea și verificarea semnăturilor digitale sunt disponibile în mod liber. Cel mai cunoscut algoritm este RSA.

## 1.9 protocoale și standarde criptografice

- **DNSSEC** – Domain Name Server Security. Protocol pentru servicii de numire distribuite
- **SSL** – Secure Socket Layer – principalul protocol pentru conexiuni WWW securizate. Increasing importance due to higher sensitive information traffic. The latest version of the protocol is called **TLS** – Transport Security Layer. Was originally developed by Netscape as an open protocol standard.
- **SHTTP** – un protocol mai nou, mai flexibil decât SSL/TKS. Specificat în RFC 2660.
- **PKCS** – Public Key Encryption Standards – dezvoltat de către RSA Data Security și definește modalități sigure de utilizare a RSA.

## 1.10 vulnerabilitatea algoritmilor criptografici

Sistemele criptografice bune trebuie întotdeauna proiectate astfel încât să fie cât mai greu de spart. Este posibil să se construiască sisteme care nu pot fi sparte în practică (deși acest lucru nu poate fi, de obicei, dovedit). Acest lucru nu crește în mod semnificativ efortul de implementare a sistemului; cu toate acestea, este necesară atenție și expertiză. Nu există nici o scuză pentru un proiectant de sistem să facă sistemul vulnerabil din start. Orice mecanisme care pot fi folosite pentru a ocoli securitatea trebuie să fie explicate, documentate și aduse în atenția utilizatorilor finali.

În teorie, orice metodă criptografică cu o cheie poate fi spartă prin încercarea succesivă a tuturor cheilor. Dacă utilizarea forței brute pentru a încerca toate cheile este singura opțiune, puterea de calcul necesară crește exponențial cu lungimea cheii. O cheie pe 32 de biți are nevoie de  $2^{32}$  (aproximativ  $10^9$ ) pași. Acesta este ceva ce poate face oricine pe computerul său de acasă. Un sistem cu cheie pe 56 de biți, cum ar fi DES, necesită un efort substanțial, dar utilizarea unor sisteme distribuite masive necesită doar câteva ore de calcul. În 1999, o căutare de forțe brute folosind un supercomputer special conceput și o rețea mondială de aproape 100.000 de PC-uri pe Internet a găsit o cheie DES în 22 de ore și 15 minute. Se crede că cheile cu cel puțin 128 de biți (cum ar fi AES, de exemplu) vor fi suficiente împotriva atacurilor de forță brută în viitorul apropiat.

Cu toate acestea, lungimea cheii nu este singura problemă relevantă. Multe cifruri pot fi sparte fără a încerca toate cheile posibile. În general, este foarte dificil să se proiecteze metode de cifrare care nu ar putea fi sparte mai eficient folosind alte metode.

Modelele nepublicate sau secrete ar trebui, în general, să fie privite cu suspiciune. Destul de des designerul nu este sigur de siguranța algoritmului, sau securitatea acestuia depinde de secretul algoritmului.

## chapter 1

În general, nici un algoritm care depinde de secretul algoritmului nu este sigur. Pentru profesioniști, este ușor să dezassembleze și să inverseze algoritmul. Experiența a arătat că marea majoritate a algoritmilor secrete care au devenit cunoscute mai târziu au fost în realitate slabe.

Cheile folosite în algoritmi cu chei publice sunt de obicei mult mai lungi decât cele utilizate în algoritmi simetrici. Acest lucru este cauzat de structura suplimentară care este disponibilă pentru cript-analizator. Problema nu este aceea de a ghici cheia corectă, ci de a deduce cheia privată potrivită din cheia publică. În cazul RSA, acest lucru se poate face prin factorizarea unui număr întreg, care are doi factori principali mari. În cazul altor sisteme criptosisteme, acest lucru este echivalent cu calculul logaritmului discret modulo un număr întreg (care se consideră a fi aproximativ comparabil cu factorizarea atunci când moduli este un număr mare mare). Există criptosisteme cu cheie publică bazate pe alte principii.

Pentru a da o idee despre complexitatea criptosistemului RSA, un modul de 256 de biți este ușor de luat în considerare la domiciliu, iar cheile de 512 biți pot fi sparte de către grupurile de cercetare universitare în câteva luni. Cheile cu 768 de biți probabil că nu sunt sigure pe termen lung. Cheile cu 1024 de biți și mai mult ar trebui să fie în siguranță pentru moment, dacă nu se fac progrese criptografice majore împotriva RSA. RSA Security pretinde că cheile de 1024-biți sunt echivalente cu rezistența la cheile simetrice de 80 de biți și recomandă utilizarea acestora până în 2010. Cheile RSA de 2048-biți sunt susținute a fi echivalente cu cheile simetrice de 112 biți și pot fi utilizate cel puțin până în 2030 .

Trebuie subliniat faptul că puterea unui sistem criptografic este de obicei egală cu cea mai slabă verigă. Nici un aspect al designului sistemului nu ar trebui să fie trecut cu vederea, de la alegerea algoritmilor la politicile de distribuție și utilizare a cheilor.

### 1.11 analiza criptografică și spargerea sistemelor de criptare

Criptanaliza este arta descifrării comunicațiilor criptate fără cunoașterea cheilor corecte. Există multe tehnici criptanalitice. Unele dintre cele mai importante pentru un implementator de sistem sunt descrise mai jos.

- atac de tip **text cifrat** (ciphertext): aceasta este situația în care atacatorul nu știe nimic despre conținutul mesajului și trebuie să lucreze doar cu textul cifrat. În practică, este destul de des posibil să se facă presupuneri despre textul original, deoarece multe tipuri de mesaje au anteturi de format fix. Chiar și scrisorile și documentele obișnuite încep într-un mod foarte previzibil. De exemplu, multe atacuri clasice folosesc analiza de frecvență a textului cifrat, însă acest lucru nu funcționează bine împotriva algoritmilor moderni.

Sistemele de criptare moderne nu sunt vulnerabile la atacurile numai de tip text cifrat, deși uneori acestea sunt considerate cu presupunerea suplimentară că mesajul conține unele particularități statistice.

- atac de tip **text original parțial** (known plaintext): Atacantul știe sau poate ghici textul original pentru anumite părți ale textului cifrat. Sarcina este de a decripta restul blocurilor de tip text cifrat utilizând aceste informații. Acest lucru se poate face prin determinarea cheii folosite pentru criptarea datelor sau prin intermediul unei scurtături.

Una dintre cele mai cunoscute metode cunoscute de text original parțial este criptanaliza liniară împotriva cifrurilor de tip bloc.

- atac de tip **text original ales** (chosen plaintext): Atacatorul este capabil să aibă orice text pe convenabil criptat cu cheia necunoscută. Sarcina este de a determina cheia utilizată pentru criptare.

Un bun exemplu al acestui atac este criptanaliza diferențială care poate fi aplicată împotriva cifrurilor (algoritmilor) de tip bloc (și, în unele cazuri, și împotriva funcțiilor de hash).

Unele sisteme de criptare, în special RSA, sunt vulnerabile la atacurile de text original ales. Atunci când se utilizează astfel de algoritmi, trebuie să se aibă grijă să se proiecteze aplicația (sau protocolul) astfel încât un atacator să nu aibă niciodată posibilitatea de a cripta un text ales.

- atac de tip **man-in-the-middle** (intermediar): acest atac este relevant pentru comunicarea criptografică și pentru protocoalele de distribuție a cheilor. Ideea este că atunci când două părți, A și B, schimbă cheile pentru o comunicare securizată (de exemplu, folosind protocolul Diffie-Hellman), un adversar se poziționează

între A și B pe linia de comunicație. Adversarul interceptează apoi semnalele pe care A și B le trimit reciproc și efectuează un schimb de chei cu A și B separat. A și B vor avea în final chei diferite, fiecare dintre ele fiind cunoscută adversarului. Intermediarul poate apoi decifra orice mesaj de la A cu cheia pe care o împarte cu A și apoi să retransmită mesajul la B prin criptarea din nou cu cheia pe care o împarte cu B. Atât A cât și B vor crede că comunică în siguranță, dar de fapt adversarul aude totul.

Modalitatea obișnuită de a preveni atacul de tip "man-in-the-middle" este utilizarea unui sistem de criptare cu cheie publică capabil să furnizeze semnături digitale. Pentru configurare, părțile trebuie să-și cunoască în avans cheile publice. După ce secretul partajat a fost generat, părțile trimit semnăturile digitale între ele. Intermediarul (man-in-the-middle) nu reușește în atacul său, deoarece nu reușește să creeze aceste semnături fără cunoașterea cheilor private folosite pentru semnare.

Această soluție este suficientă dacă există și o modalitate de a distribui în siguranță cheile publice. Un astfel de mod este o ierarhie de certificare, cum ar fi X.509. Este folosit de exemplu în IPSec.

- **Corelația** dintre cheia secretă și ieșirile sistemului de criptare este sursa principală de informații pentru criptanalizator. În cel mai simplu caz, informațiile despre cheia secretă sunt scurgeri directe ale sistemului de criptare. Cazurile mai complicate necesită studierea corelației (în esență, a oricărei relații care nu ar fi de așteptat doar pe baza hazardului) între informațiile observate (sau măsurate) despre sistemul de criptare și informațiile esențiale ghicite.

De exemplu, în atacurile liniare (sau diferențiale) împotriva cifrurilor de tip bloc, cript-analizatorul studiază textele cunoscute (respectiv alese) și textul cifrat observat. Ghicind câțiva biți ai cheii sistemului de criptare, analistul determină prin corelarea dintre textul plat și textul cifrat dacă a ghicit corect. Acest lucru poate fi repetat și are multe variații.

Criptanaliza diferențială introdusă de Eli Biham și Adi Shamir la sfârșitul anilor 1980 a fost primul atac care a folosit pe deplin această idee împotriva cifrurilor de tip bloc (mai ales împotriva DES). Mai târziu, Mitsuru Matsui a venit cu criptanaliza liniară, care era și mai eficientă împotriva DES. Mai recent, s-au dezvoltat noi atacuri folosind idei similare.

Poate că cea mai bună introducere la acest material este procesul de EUROCRYPT și CRYPTO pe parcursul anilor 1990. Se poate găsi discuția lui Mitsuru Matsui despre criptanaliza liniară a DES și despre ideile unor diferențe trunchiate de Lars Knudsen (de exemplu, criptanaliza IDEA). Cartea lui Eli Biham și Adi Shamir despre criptanalizarea diferențiată a DES este lucrarea "clasică" pe acest subiect.

Ideea de corelare este fundamentală pentru criptografie și câțiva cercetători au încercat să construiască criptosisteme care sunt promițător de protejate împotriva acestor atacuri. De exemplu, Knudsen și Nyberg au studiat securitatea dovedită împotriva criptanalizei diferențiate.

- **Atacul împotriva sau utilizarea hardware-ului subiacent:** În ultimii ani, pe măsură ce tot mai multe dispozitive mobile de criptare mobile au intrat în uz larg, o nouă categorie de atacuri a devenit relevantă și urmărește direct implementarea hardware a sistemului de criptare.

Atacurile utilizează datele de la măsurătorile foarte fine ale dispozitivului crypto, care fac, de exemplu, criptarea și calculează informațiile cheie din aceste măsurători. Ideile de bază sunt apoi strâns legate de cele din alte atacuri de corelație. De exemplu, atacatorul ghicește niște biți ai cheii și încearcă să verifice corectitudinea presupunerii studiind corelația cu măsurătorile ei.

S-au propus mai multe atacuri, cum ar fi utilizarea temporizării fine a dispozitivului, măsurători fine ale consumului de energie și modele de radiații. Aceste măsurători pot fi utilizate pentru a obține cheia secretă sau alte tipuri de informații stocate pe dispozitiv.

Acest atac este, în general, independent de algoritmi criptografici utilizați și poate fi aplicat oricărui dispozitiv care nu este protejat în mod explicit împotriva acestuia.

Mai multe informații despre analiza puterii diferențiale sunt disponibile la <http://www.cryptography.com>.

- **Defectele în criptosisteme** pot duce la criptanaliză și chiar la descoperirea cheii secrete. Interesul pentru dispozitivele criptografice conduce la descoperirea faptului că unii algoritmi s-au comportat foarte prost odată cu introducerea de mici defecte în calculul intern.

De exemplu, implementarea obișnuită a operațiunilor cu cheie privată RSA este foarte susceptibilă la atacurile de eroare. S-a demonstrat că prin producerea unui bit de eroare la un punct adecvat se poate dezvălui factorizarea modulului (adică dezvăluie cheia privată).

Idei similare au fost aplicate la o gamă largă de algoritmi și dispozitive. Este deci necesar ca dispozitivele

## chapter 1

criptografice să fie proiectate pentru a fi foarte rezistente împotriva defectelor (și împotriva introducerii a defectelor de către criptanalizatori).

- **Calculul cuantic:** lucrarea lui Peter Shor despre factorizare în timp polinomial și algoritmi pentru logaritmul discret, utilizând computerele cuantice a provocat un interes tot mai mare în calculul cuantic. Calculul cuantic este un domeniu recent de cercetare care utilizează mecanica cuantică pentru a construi computere care, în teorie, sunt mai puternice decât computerele seriale moderne. Puterea derivă din paralelismul inerent al mecanicii cuantice. Deci, în loc să efectuăm sarcini una câte una, cum fac mașinile de serie, computerele cuantice le pot efectua pe toate odată (în paralel). Astfel, se speră că, cu ajutorul calculatoarelor cuantice, putem rezolva problemele imposibile pentru computerele seriale.

Rezultatele lui Shor implică faptul că, dacă calculatoarele cuantice ar putea fi implementate eficient, atunci majoritatea criptografiei cu cheie publică vor deveni istorie. Cu toate acestea, ele sunt mult mai puțin eficiente împotriva criptografiei cu cheie secretă.

Starea actuală a computerelor cuantice nu pare alarmantă, deoarece au fost implementate doar mașini foarte mici. Teoria calculului cuantic oferă promisiuni pentru performanțe mai bune decât computerele seriale, cu toate acestea, dacă se vor realiza în practică, este o întrebare deschisă.

Mecanica cuantică este, de asemenea, o sursă pentru noi modalități de secretizare a datelor și o comunicare sigură cu potențialul de a oferi o securitate nevulnerabilă, acesta fiind domeniul criptografiei cuantice. Spre deosebire de calculul cuantic, multe implementări experimentale de succes ale criptografiei cuantice au fost deja realizate. Cu toate acestea, criptografia cuantică este încă departe de a fi realizată în aplicații comerciale.

- **criptografia ADN:** Leonard Adleman (unul dintre inventatorii RSA) a venit cu ideea de a folosi ADN-ul ca calculatoare. Moleculele de ADN ar putea fi văzute ca un calculator foarte mare capabil de execuție paralelă. Această natură paralelă ar putea da computerelor ADN o creștere a performanțelor exponențială față de computerele seriale moderne.

Există, din păcate, probleme cu calculatoarele ADN, una fiind faptul că accelerarea exponențială necesită și o creștere exponențială a volumului de material necesar. Astfel, în practică, computerele ADN ar avea limite asupra performanței lor. De asemenea, nu este foarte ușor să construiești unul.



## chapter 2 funcții de hash

### 2.1 funcții de hash

O **funcție criptografică de hash** este o transformare care ia o intrare și returnează un șir de dimensiuni fixe, care se numește fie **valoare hash**, fie **sumă de control** sau **digest**. Funcțiile hash cu această proprietate sunt utilizate pentru o varietate de scopuri computaționale, inclusiv în criptografie. Valoarea hash este o reprezentare concisă a mesajului sau documentului mai lung din care a fost calculată. Sau, poate fi văzută ca un fel de "amprentă digitală" a documentului mai mare. Funcțiile hash criptografice sunt utilizate pentru a efectua verificări ale integrității mesajelor și pentru generarea semnăturii digitale precum și în alte aplicații de securitate a informațiilor, cum ar fi autentificarea și integritatea mesajelor.

O funcție de hash poate avea ca intrare un fișier, un mesaj, un șir de caractere, etc. de lungime arbitrară (deși un limită, greu de atins în practică, de genul  $2^{64}-1$  biți, poate fi impusă în multe situații) și are ca ieșire un șir de biți, cu o lungime fixă, de exemplu: 128, 160 sau 256 de biți), șir care este numit fie suma de control, hash, digest sau amprentă digitală. O valoare hash (numită și "digest" sau "checksum") este un fel de "semnătură" pentru fluxul de date care reprezintă conținutul fișierului sau mesajului. O analogie care explică rolul funcției hash ar fi sigiliile de detectare a modificării folosite pe un pachet software.

În diferite standarde și aplicații, cele două funcții de hash utilizate cel mai frecvent sunt MD5 și familia SHA. În 2005, defecte de securitate au fost identificate în ambii algoritmi. În 2007, Institutul Național de Standarde și Tehnologie a anunțat un concurs pentru proiectarea unei funcții de tip hash, care a fost numit SHA-3 și care face obiectul FIPS 202 – <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.202.pdf>.

Pentru o funcție hash  $h$  cu domeniul  $D$  și codomeniul (mulțimea în care ia valori)  $R$ , următoarele cerințe sunt obligatorii:

1. Rezistența la pre-imagini - dat  $y$  în  $R$ , este computațional imposibil să găsim  $x$  în  $D$  astfel încât  $h(x)=y$ .
2. A doua rezistență la pre-imagini – dat fiind  $x$  în  $D$ , este computațional imposibil să găsim un alt  $z$  în  $D$  astfel încât  $h(x)=h(z)$ .
3. Rezistența la coliziune - este computațional imposibil să găsim  $x, z$  în  $D$  astfel încât  $h(x)=h(z)$

### 2.2 aplicații ale funcțiilor de hash

Un caz tipic de utilizare a unui hash criptografic ar putea fi următorul: Alice propune o problemă de matematică dificilă lui Bob și susține că a rezolvat-o. Bob ar vrea să încerce el însuși, dar ar vrea să fie sigur că Alice nu blufează. Prin urmare, Alice își scrie soluția, adaugă un nonce random (un număr arbitrar care poate fi utilizat o singură dată), își calculează hash-ul și îi spune Bob valoarea hash (păstrând în același timp soluția și secretul nonce). În acest fel, când Bob vine cu soluția el însuși câteva zile mai târziu, Alice poate dovedi că a avut soluția mai devreme prin aceea că dezvăluie nonce-ul lui Bob. (Acesta este un exemplu de schemă simplă de angajament, în practica actuală, Alice și Bob vor fi adesea programe de calculator, iar secretul ar fi o entitate mai dificil de falsificat).

O altă aplicație importantă a hash-urilor securizate este verificarea integrității mesajelor. Determinarea faptului dacă au fost făcute schimbări la un mesaj (sau un fișier), de exemplu, poate fi realizată prin compararea hash-urilor mesajelor calculate înainte și după transmitere (sau după orice alt eveniment).

Un hash al mesajului poate servi și ca mijloc de identificare fiabilă a unui fișier; mai multe sisteme de gestionare a codurilor sursă, inclusiv Git, Mercurial și Monotone, utilizează sume de control SHA-1 pentru diferite tipuri de conținut (conținutul fișierelor, arborii de directoare, informațiile de origine etc.) pentru a le identifica în mod unic.

O aplicație înrudită este verificarea parolei. Parolele nu sunt de obicei stocate în text clar, din motive

## chapter 2

evidente, ci în formă de hash. Pentru a autentifica un utilizator, parola prezentată de utilizator este convertită în hash și comparată cu hash-ul stocat. Acest lucru este denumit uneori criptare unidirecțională.

Din motive de securitate și de performanță, majoritatea algoritmilor de semnătură digitală specifică faptul că numai digestul mesajului este "semnat", nu întregul mesaj. Funcțiile Hash pot fi de asemenea folosite în generarea de biți pseudorandom.

## 2.3 MD5 – generalități

MD5 este o funcție de hash de tip bloc (dimensiunea blocului este de 512 biți) care a fost dezvoltată de către Rivest în 1991. Intrarea pentru MD5 este un mesaj sau fișier de lungime arbitrară, în timp ce ieșirea este un șir de biți de lungime fixă. Lungimea acestui hash este de 128 biți sau de 4 cuvinte (a 32 de biți fiecare). Specificația formală a acestui algoritm hash este conținută în RFC 1321.

## 2.4 descrierea algoritmului MD5

Începem cu presupunerea că avem un mesaj de lungime  $b$  biți ca intrare și că dorim să-i aflăm hash-ul. În cazul nostru,  $b$  este un întreg pozitiv sau zero, de mărime arbitrară. Presupunem că biții acestui mesaj sunt scriși astfel:

$$m_0 m_1 \dots m_{\{b-1\}}$$

Pentru calcularea hash-ului (digest-ului) acestui mesaj, sunt parcurși următorii cinci pași.

### 2.4.1 pasul 1 – adăugarea biților de umplură

Mesajul este "**umplut**" (completat, extins) astfel ca lungimea sa (în biți) să fie congruentă cu 448, modulo 512. Cu alte cuvinte, mesajul este extins pentru a-i mai lipsi doar 64 de biți ca să fie multiplu de 512. Acesată completare este efectuată întotdeauna, chiar dacă lungimea mesajului este deja congruentă cu 448 modulo 512.

Completarea se face după cum urmează: un singur bit "1" este atașat la mesaj și apoi biți cu valoarea "0" sunt atașați astfel încât lungimea în biți a mesajului completat să devină congruentă cu 448, modulo 512. În total, cel puțin un bit și cel mult 512 de biți sunt atașați.

### 2.4.2 pasul 2 – adăugarea lungimii

O reprezentare pe 64 de biți a lui  $b$  (lungimea mesajului înainte de adăugarea biților de umplere) este atașată la rezultatul etapei anterioare. În cazul puțin probabil că  $b$  este mai mare de  $2^{64}$ , atunci se utilizează doar cei 64 de biți de ordin mic din  $b$ . (Acești biți sunt atașați ca două cuvinte pe 32 de biți și mai întâi este atașat cuvântul cu cifrele mai puțin semnificative, în conformitate cu convențiile anterioare.)

În acest moment mesajul rezultat (după umplutura cu biți și cu lungimea atașată) are o lungime care este un multiplu exact de 512 biți. În mod echivalent, acest mesaj are o lungime care este un multiplu exact de 16 cuvinte (32 de biți). Fie  $M [0 \dots N-1]$  cuvintele mesajului rezultat, unde  $N$  este un multiplu de 16.

### 2.4.3 pasul 3 – inițializarea bufferului MD

Un buffer de patru cuvinte (A, B, C, D) este folosit pentru a calcula digestul mesajului. Aici fiecare din A, B, C, D este un registru pe 32 de biți. Aceste registre sunt inițializate cu următoarele valori în octeți hexazecimali, în ordine descrescătoare):

```
cuvântul A: 01 23 45 67
cuvântul B: 89 ab cd ef
cuvântul C: fe dc ba 98
cuvântul D: 76 54 32 10
```

### 2.4.4 pasul 4 - procesarea mesajului în blocuri de 16-cuvinte

Mai întâi definim patru funcții auxiliare, fiecare având ca intrări trei cuvinte pe 32 de biți și producând ca ieșire un cuvânt pe 32 de biți.

$$F(X, Y, Z) = (X \& Y) | (\sim X \& Z)$$

$$G(X, Y, Z) = (X \& Z) | (Y \& \sim Z)$$

$$H(X, Y, Z) = X \wedge Y \wedge Z$$

$$I(X, Y, Z) = Y \wedge (X | \sim Z)$$

Pentru fiecare bit, F acționează ca o condiție: dacă X atunci Y altfel Z. Funcția F ar fi putut fi definită folosind + în locul lui **sau** din moment ce X și Y și not(X) și Z nu vor avea niciodată 1 în aceeași poziție de. Interesant de observat că dacă biții X, Y și Z sunt independenți și imparțiali, fiecare bit al F (X, Y, Z) va fi independent și imparțial.

Funcțiile G, H și I sunt similare cu funcția F, prin aceea că acționează în "paralel la nivel de biți" pentru a produce ieșirea lor de la biții X, Y și Z, astfel încât dacă biții corespunzători ai lui X, Y și Z sunt independenți și imparțiali, atunci fiecare bit al lui G (X, Y, Z), H (X, Y, Z) și I (X, Y, Z) va fi independent și imparțial. Rețineți că funcția H este funcția "xor" sau "paritate" a biților de intrare.

Acest pas utilizează un tabel cu 64 de elemente T [1 ... 64] construit din funcția sinus. Fie T [i] elementul i al tabelului, care este egal cu partea întreagă a lui  $4294967296 \cdot \sin(i)$ , unde i este în radiani. Elementele tabelului sunt date în anexă.

Mai jos, N este numărul de cuvinte. Deoarece ultimul bloc de 512 de biți (16 cuvinte) a fost completat (umplut), N este un multiplu de 16, în timp ce N / 16 este numărul de blocuri din mesajul completat.

Se execută următoarele instrucțiuni:

```
/* Proceșează fiecare bloc de 16 cuvinte. */
For i = 0 to N/16 - 1 do

  /* Copiază blocul i în X. */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* of loop on j */

  /* Salvează A ca AA, B ca BB, C ca CC și D ca DD. */
  AA = A
  BB = B
  CC = C
  DD = D

  /* Runda 1. */
  /* Notează cu [abcd k s i] operația
```

## chapter 2

```
        a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
/* efectuează următoarele 16 operații. */
[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]

/* Runda 2. */
/* Notează cu [abcd k s i] operația
   a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
/* efectuează următoarele 16 operații. */
[ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]
[ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]
[ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]
[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]

/* Runda 3. */
/* Notează cu [abcd k s t] operația
   a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
/* efectuează următoarele 16 operații. */
[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]
[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]

/* Runda 4. */
/* Notează cu [abcd k s t] operația
   a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
/* efectuează următoarele 16 operații. */
[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]
[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]
[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]

/* Apoi, efectuează următoarele adunări. (Adică incrementează fiecare
   din cei 4 regiștri cu valoarea pe care au avut-o înainte de începerea
   acestui bloc.) */
A = A + AA
B = B + BB
C = C + CC
D = D + DD

end /* bucla indexată de i */
```

### 2.4.5 pasul 5 - rezultat

Hashul mesajului (digest-ul, suma de control) este format din cuvintele A, B, C, D. Se începe cu octetul cel mai puțin semnificativ al lui A și se termină cu octetul cel mai semnificativ al lui D.

Descrierea lui MD5 este acum completă.

## 2.5 seria de teste pentru MD5 – conform RFC 1321

Valorile de hash ale unor șiruri sunt specificate în RFC 1321, astfel:

```

MD5("") = d41d8cd9 8f00b204 e9800998 ecf8427e
MD5("a") = 0cc175b9 c0f1b6a8 31c399e2 69772661
MD5("abc") = 90015098 3cd24fb0 d6963f7d 28e17f72
MD5("message digest") = f96b697d 7cb7938d 525a2f31 aaf161d0
MD5("abcdefghijklmnopqrstuvwxyz") = c3fcd3d7 6192e400 7dfb496c ca67e13b
MD5("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
    d174ab98 d277d9f5 a5611c2c 9f419d9f
MD5 ("123456789012345678901234567890123456789012345678901234567890123456
78901234567890") = 57edf4a2 2be3c955 ac49da2e 2107b67a

```

## 2.6 analiza criptografică (cript-analiza) a funcției MD5

În 1993, Den Boer și Bosselaers au anunțat o pseudo-coliziune a funcției de compresie a MD5, adică au găsit doi vectori de inițializare diferiți care au produs un hash identic.

În 1996, Dobbertin a anunțat o coliziune a funcției de compresie a MD5. Chiar dacă nu a fost o coliziune a întregii funcții MD5, a fost suficient ca criptograful să recomande trecerea la alte funcții de hash, precum WHIRLPOOL, SHA-1 or RIPEMD-160.

Totodată, lungime relativ scurtă a hash-ului - 128 de biți – sugerează posibilitatea unui atac de tip birthday (ziua de naștere). MD5CRK a fost un proiect distribuit care a început în martie 2004 cu scopul de a demonstra vulnerabilitatea MD5 prin găsirea unei coliziuni utilizând un atac de tip birthday.

MD5CRK s-a încheiat la puțin timp după 17 August 2004, atunci când a fost anunțată găsirea de coliziuni MD5 de către Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Atacul lor analitic a durat doar o oră pe un cluster IBM p690.

Pe 1 martie 2005, Arjen Lenstra, Xiaoyun Wang și Benne de Weger au demonstrat construirea a două certificate X.509 cu chei publice diferite dar cu același hash MD5, o coliziune care poate fi demonstrată practic. Această metodă de construire utilizează cheile private pentru ambele chei publice. Câteva zile mai târziu, Vlastimil Klima a descris un algoritm îmbunătățit, capabil să construiască coliziuni MD5 în câteva ore, utilizând un laptop. Pe 18 martie 2006, Klima a publicat un algoritm care poate găsi coliziuni într-un minut utilizând un singur laptop, prin folosirea unei metode numită tunneling.

O coliziune practică poate fi găsită la <http://www.mathstat.dal.ca/~selinger/md5collision/>. O reproducem aici, deoarece cele două mesaje utilizate sunt scurte. Mai mult decât atât, sunt aproape identice.

```

d131dd02c5e6eec4693d9a0698aff95c 2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a 085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6 dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1e c69821bcb6a8839396f9652b6ff72a70

```

```

d131dd02c5e6eec4693d9a0698aff95c 2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a 085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6 dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080280d1e c69821bcb6a8839396f965ab6ff72a70

```

Hash-ul comun al celor două mesaje este - 79054025-255fb1a2-6e4bc422-aef54eb4 .

## chapter 3 secure hash algorithm

Funcțiile hash SHA reprezintă un set de funcții hash criptografice proiectate de Agenția Națională de Securitate (NSA) și publicate de NIST ca standard federal de prelucrare a informațiilor din S.U.A. SHA înseamnă Secure Hash Algorithm. Cei trei algoritmi SHA inițiali sunt structurați diferit și se disting ca SHA-0, SHA-1 și SHA-2. Familia SHA-2 utilizează un algoritm identic cu o dimensiune a hashului variabilă, având denumirile distincte ca SHA-224, SHA-256, SHA-384 și SHA-512.

SHA-1 este cel mai utilizat dintre funcțiile hash existente SHA și este folosit în mai multe aplicații și protocoale de securitate utilizate pe scară largă. În 2005, au fost identificate deficiențe de securitate în SHA-1, și anume că ar putea exista o posibilă slăbiciune matematică, indicând faptul că ar fi de dorit o funcție mai puternică de tip hash. Deși nu s-au raportat încă atacuri asupra variantelor SHA-2, ele sunt asemănătoare algoritmilor cu SHA-1 și astfel s-au făcut eforturi pentru a dezvolta alternative îmbunătățite. Un nou standard hash, SHA-3, folosind un nou algoritm numit Keccak, a fost selectat printr-o competiție deschisă care a avut loc între toamna lui 2008 și 2012.

SHA-3 a fost standardizat în august 2015 ca FIPS 202.

### 3.1 SHA-0 și SHA-1

Specificația originală a algoritmului a fost publicată în 1993 ca standardul Secure Hash, FIPS PUB 180, de către agenția americană de standardizare NIST (Institutul Național de Standarde și Tehnologie). Această versiune este acum adesea denumită SHA-0. A fost retrasă de către NSA la scurt timp după publicare și a fost înlocuită de versiunea revizuită, publicată în 1995 în FIPS PUB 180-1 și denumită în mod obișnuit SHA-1. SHA-1 diferă de SHA-0 numai printr-o singură rotație la nivel de bit în rotația de mesaje a funcției sale de compresie; acest lucru s-a făcut, conform NSA, pentru a corecta un defect al algoritmului original care reducea securitatea sa criptografică. Cu toate acestea, ANS nu a furnizat nicio explicație suplimentară și nici nu a identificat defectul corectat. Puncte slabe au fost ulterior raportate atât la SHA-0, cât și la SHA-1. SHA-1 pare să ofere o mai mare rezistență la atacuri, susținând afirmația NSA că schimbarea a sporit securitatea.

SHA-1 (ca și SHA-0) produce un hash de 160 biți dintr-un mesaj cu o lungime maximă de  $(2^{64} - 1)$  biți. SHA-1 se bazează pe principii similare celor utilizate de Ronald L. Rivest de la MIT în proiectarea algoritmilor de generare a hash-ului mesajelor MD4 și MD5, dar are un design mai conservator.

### 3.2 SHA-2 family

NIST a publicat patru funcții hash suplimentare în familia SHA, denumite după lungimile hashului (în biți): SHA-224, SHA-256, SHA-384 și SHA-512. Algoritmii sunt denumiți în mod colectiv SHA-2.

Algoritmii au fost publicați pentru prima oară în 2001 în draftul FIPS PUB 180-2, moment în care revizuirea și comentariul au fost acceptate. FIPS PUB 180-2, care include și SHA-1, a fost lansat ca standard oficial în 2002. În februarie 2004 a fost publicată o notificare de schimbare pentru FIPS PUB 180-2, specificând o variantă suplimentară, SHA-224, lungimea cheii fiind cea a două chei Triple DES. Aceste variante sunt brevetate în brevetul US 6829355. Statele Unite au eliberat brevetul în baza unei licențe gratuite.

SHA-256 și SHA-512 sunt funcții noi de tip hash calculate cu cuvinte de 32 și, respectiv, 64 de biți. Ele folosesc cantități diferite de șifturi și constante aditive, dar structurile lor sunt altfel practic identice, diferențiind numai numărul de runde. SHA-224 și SHA-384 sunt pur și simplu variante trunchiate ale primelor două, calculate cu valori inițiale diferite.

Spre deosebire de SHA-1, funcțiile SHA-2 nu sunt utilizate pe scară largă, în ciuda securității lor mai bune. Motivele ar putea include lipsa suportului pentru SHA-2 pe sistemele care rulează Windows XP SP2

sau mai vechi, Iprecum și ipsa de motivație percepută din moment ce nu au fost găsite coliziuni SHA-1 sau dorița de a aștepta până la standardizarea SHA-3. SHA-256 este utilizat pentru a autentifica pachetele software Debian Linux și în standardul de semnătură digitală a mesajelor DKIM; SHA-512 face parte dintr-un sistem de autentificare a înregistrărilor video de arhivă de la Tribunalul Penal Internațional pentru genocidul din Ruanda. SHA-256 și SHA-512 sunt propuse pentru utilizarea în directiva DNSSEC NIST ce sugerează agențiilor guvernamentale americane să oprească cele mai multe utilizări ale SHA-1 după 2010 iar finalizarea SHA-3 poate accelera migrarea de la SHA-1.

În prezent, cele mai bune atacuri publice asupra SHA-2 au spart 24 din cele 64 (respectiv 80) de runde.

## 3.3 SHA-3

O competiție deschisă pentru o nouă funcție SHA-3 a fost anunțată în mod oficial în Registrul Federal al SUA în 2 noiembrie 2007. "NIST inițiază un efort de a dezvolta unul sau mai mulți algoritmi de hash suplimentari printr-o competiție publică, similar cu procesul de dezvoltare pentru Advanced Encryption Standard (AES)." Propunerile au avut loc pe 31 octombrie 2008, iar proclamarea unui câștigător și publicarea noului standard au avut loc în 2012.

NIST a selectat 51 de intrări pentru Runda 1, iar 14 dintre ele au avansat la Runda 2.

### 3.3.1 acceptate pentru runda doi

Următoarele propuneri de funcții de tip hash au fost acceptate pentru runda a doua.

- BLAKE
- Blue Midnight Wish
- [CubeHash](#)
- ECHO (France Telecom)
- [Fugue](#)
- [Grøstl](#) (Knudsen et al.)
- Hamsi
- JH
- [Keccak](#) (Keccak team, [Daemen](#) et al.)
- Luffa
- Shabal
- SHAvite-3
- SIMD
- [Skein](#) (Schneier et al.)

### 3.3.2 iar câștigătorul este ... Keccak

În octombrie 2012, algoritmul Keccak a fost declarat câștigător.

## 3.4 utilizare

SHA-1 este cea mai răspândită funcție hash din familia SHA. Acesta face parte din mai multe aplicații și protocoale de securitate utilizate pe scară largă, inclusiv TLS și SSL, PGP, SSH, S / MIME și IPsec. Aceste aplicații pot utiliza, de asemenea, MD5; atât MD5 cât și SHA-1 sunt descendenți din MD4. De asemenea, funcția de hash SHA-1 este utilizată în sistemele distribuite de control al versiunilor, cum ar fi Git, Mercurial și

## chapter 3

Monotone, pentru a identifica versiunile și pentru a detecta coruperea sau modificarea datelor.

SHA-256, SHA-384 și SHA-512 sunt algoritmi hash de securitate specificați prin lege pentru a fi utilizați în anumite aplicații ale guvernului SUA, inclusiv utilizarea în cadrul altor algoritmi și protocoale criptografice, pentru protecția informațiilor nesecrete sensibile. FIPS PUB 180-1 a încurajat, de asemenea, adoptarea și utilizarea SHA-1 de către organizațiile private și comerciale. SHA-1 urmează a fi retras pentru majoritatea utilizărilor guvernamentale; Institutul National de Standarde și Tehnologie din SUA spune ca "agențiile federale ar trebui să înceteze să utilizeze SHA-1 pentru ... aplicații care necesită rezistența la coliziune cât mai curând posibil și trebuie să utilizeze familia SHA-2 de funcții hash pentru aceste aplicații după 2010".

O primă motivație pentru publicarea algoritmului Secure Hash a fost DSS – Digital Signature Standard (Standardul de Semnătură Digitală), în care este încorporat.

Funcțiile hash SHA au fost folosite ca bază pentru algoritmi de cifrare de tip bloc SHACAL.

### 3.5 analiză criptografică și validare

Pentru o funcție hash în care  $L$  este numărul de biți din digestul mesajului, găsirea unui mesaj care corespunde unui hash de mesaj dat poate fi întotdeauna efectuată folosind o căutare de forță brută în  $2^L$  evaluări. Acesta este numit **atac de preimage** și poate sau nu poate fi practic în funcție de  $L$  și de mediul de calcul utilizat. Al doilea criteriu, găsirea a două mesaje diferite care produc același hash de mesaj, cunoscut sub numele de coliziune, necesită, în medie, numai  $2^{L/2}$  evaluări folosind un atac de tip ziua de naștere (**birthday attack**). Din acest din motiv, puterea unei funcții hash este de obicei comparată cu un cifru simetric cu cheia jumătate din lungimea digestului mesajului. Astfel, SHA-1 a fost considerat inițial că are o putere de 80 de biți.

Criptografiile au produs perechi de coliziuni pentru SHA-0 și au găsit algoritmi care ar trebui să producă coliziuni SHA-1 în mult mai puține decât cele  $2^{80}$  de evaluări inițial preconizate.

În ceea ce privește securitatea practică, o preocupare majoră cu privire la aceste noi atacuri este că acestea ar putea deschide calea către unele mai eficiente. Dacă se va întâmpla acest lucru, rămâne de văzut, dar se consideră că migrarea la hashuri mai puternice este prudentă. Unele aplicații care utilizează hash-urile criptografice, cum ar fi stocarea parolelor, sunt afectate minimal de un atac de coliziune. Construirea unei parole care să funcționeze pentru un anumit cont necesită un atac de preimage, precum și accesul la hash-ul parolei originale (stocate, în mod obișnuit, în fișiere umbră) ceea ce pare să nu să fie chiar așa de simplu. Inversarea criptării parolei (de exemplu, pentru a obține o parolă pentru a accesa contul unui utilizator în altă parte) nu este posibilă prin aceste atacuri. (Cu toate acestea, chiar și un hash de parole securizat nu poate împiedica atacurile de forță brută asupra parolelor slabe.)

În cazul semnării documentelor, un atacator nu poate falsifica pur și simplu o semnătură dintr-un document existent - atacatorul ar fi trebuit să producă o pereche de documente, unul inofensiv și unul dăunător și să determine titularul cheii private să semneze documentul inofensiv. Există circumstanțe practice în care acest lucru este posibil; până la sfârșitul anului 2008 a fost posibil să se creeze certificate SSL false folosind o coliziune MD5.

#### 3.5.1 SHA-0

La CRYPTO 98, doi cercetători francezi, Florent Chabaud și Antoine Joux, au prezentat un atac asupra SHA-0: coliziunile pot fi găsite cu complexitate  $2^{61}$ , mai puțin decât cele  $2^{80}$  pentru o funcție hash ideală de aceeași dimensiune.

În 2004, Biham și Chen au descoperit semi-coliziuni pentru SHA-0: două mesaje care au aproape același hash; în acest caz, 142 din cei 160 de biți sunt egali. De asemenea, au descoperit coliziunile complete ale SHA-0 redus la 62 din cele 80 de runde.

Ulterior, la 12 august 2004, o coliziune pentru algoritmul SHA-0 complet a fost anunțată de Joux, Carribault, Lemuet și Jalby. Acest lucru a fost făcut folosind o generalizare a atacului Chabaud și Joux. Găsirea coliziunii a avut complexitatea  $2^{51}$  și a durat aproximativ 80.000 de ore CPU pe un supercomputer



cu 256 procesoare Itanium 2. (Echivalent cu 13 zile de utilizare integrală a calculatorului.)

La 17 august 2004, în cadrul sesiunii Rump de la CRYPTO 2004, rezultate preliminare au fost anunțate de Wang, Feng, Lai și Yu, despre un atac asupra MD5, SHA-0 și a altor funcții hash. Complexitatea atacului lor asupra SHA-0 este  $2^{40}$ , semnificativ mai bună decât atacul lui Joux et al.

În februarie 2005, a fost anunțat un atac făcut de Xiaoyun Wang, Yiqun Lisa Yin și Hongbo Yu care au putut identifica coliziuni în SHA-0 în  $2^{39}$  de operațiuni.

### 3.5.2 SHA-1

În lumina rezultatelor pentru SHA-0, unii experți au sugerat că planurile de utilizare a SHA-1 în noile criptosisteme ar trebui reconsiderate. După ce au fost publicate rezultatele CRYPTO 2004, NIST a anunțat că intenționează să elimine treptat utilizarea SHA-1 până în 2010 în favoarea variantelor SHA-2.

La începutul anului 2005, Rijmen și Oswald au publicat un atac asupra unei versiuni reduse a SHA-1: 53 din 80 de runde - care găsește coliziuni cu un efort computațional de mai puțin de  $2^{80}$  operațiuni.

În februarie 2005 a fost anunțat un atac făcut de Xiaoyun Wang, Yiqun Lisa Yin, Bayarjargal și Hongbo Yu. Atacurile pot găsi coliziuni în versiunea completă a SHA-1, care necesită mai puțin de  $2^{69}$  operațiuni. (Căutarea cu forță brute ar necesita  $2^{80}$  operațiuni.)

Autorii scriu: "În particular, analiza noastră se bazează pe atacul diferențial original asupra SHA-0, atacul de semi-coliziune asupra SHA-0, tehnicile de coliziune multibloc, precum și pe tehnicile de modificare a mesajului folosite în atacul de căutare a coliziunilor pentru MD5. Spargere SHA-1 nu ar fi posibilă fără aceste tehnici analitice puternice." Autorii au prezentat o coliziune pentru SHA-1 cu 58 de runde, găsită cu  $2^{33}$  operații de tip hash. Lucrarea cu descrierea completă a atacului a fost publicată în august 2005 la conferința CRYPTO.

Într-un interviu, Yin afirmă: "În principiu, exploatăm următoarele două puncte slabe: Unul este că pasul de preprocesare a fișierelor nu este suficient de complicat, altul este că anumite operații matematice în primele 20 de runde au probleme de securitate neașteptate".

La 17 august 2005, o îmbunătățire a atacului SHA-1 a fost anunțată în numele Xiaoyun Wang, Andrew Yao și Frances Yao la sesiunea CRYPTO 2005, reducând complexitatea necesară pentru a găsi o coliziune în SHA-1 la  $2^{63}$ . La 18 Decembrie 2007 detaliile acestui rezultat au fost explicate și verificate de Martin Cochran.

Christophe De Cannière și Christian Rechberger au îmbunătățit și mai mult atacul asupra SHA-1 în "Finding SHA-1 Characteristics: General Results and Applications", primind premiul pentru cea mai bună lucrare la ASIACRYPT 2006. O coliziune a două blocuri pentru SHA-1 a fost prezentată, găsită folosind metode neoptimizate cu  $2^{35}$  de evaluări ale funcțiilor de comprimare. Deoarece acest atac necesită echivalentul a aproximativ  $2^{35}$  de evaluări, este considerat a fi un salt teoretic semnificativ. Pentru a găsi o coliziune reală în cele 80 de runde ale funcției hash, cu toate acestea, sunt necesare cantități masive de timp pentru calculator. În acest scop, o căutare de coliziune pentru SHA-1 utilizând platforma distribuită de calcul BOINC a început la 8 august 2007, organizată de Universitatea Tehnică din Graz. Efortul a fost abandonat pe 12 mai 2009 din cauza lipsei de progres.

La sesiunea CRYPTO 2006, Christian Rechberger și Christophe De Cannière au susținut că au descoperit un atac de coliziune asupra SHA-1, care ar permite atacatorului să selecteze cel puțin părți ale mesajului.

Cameron McDonald, Philip Hawkes și Josef Pieprzyk au prezentat un atac de coliziune a hash-ului cu complexitate revendicată de  $2^{52}$  la sesiunea Rump a Eurocrypt 2009. Cu toate acestea, lucrarea însoțitoare, "Differential Path for SHA-1 with complexity  $O(2^{52})$ " a fost retrasă din cauza descoperirii de către autori că estimarea lor a fost incorectă.

În 2017, CWI Amsterdam și Google au anunțat găsirea unei coliziuni (două fișiere pdf distincte care produc același hash) pentru SHA-1. [SHA1C]. Numărul de evaluări a fost de aproximativ  $2^{63.1}$ , adică de 100000 de ori mai rapid decât prin utilizarea forței brute printr-un atac al zilei de naștere (birthday attack).

### 3.5.3 SHA-2

Există două atacuri de preimagine împotriva SHA-2 cu un număr redus de runde. Primul atacă 41 de runde SHA-256 din cele 64 de runde cu complexitate de timp de  $2^{253,5}$  și complexitate spațială de  $2^{16}$  și 46 de runde SHA-512 din cele 80 de runde cu timpul  $2^{511,5}$  și spațiu  $2^3$ . Cel de-al doilea atacă 42 de runde SHA-256 cu complexitate de timp de  $2^{251,7}$  și complexitate spațială de  $2^{12}$  și 42 de runde SHA-512 cu timpul  $2^{502}$  și spațiu  $2^{22}$ .

## 3.6 generalități despre SHA-1

SHA-1 este obiectul specificației FIPS 180-1, un document NIST din 1993. Este o funcție hash de tip bloc, mărimea blocului fiind de 512 biți (64 octeți). Produce un hash de 160 de biți, de obicei sub forma a 5 cuvinte de 32 de biți, reprezentate în hexazecimal. Este considerată o funcție hash relativ sigură. Totuși, se preconizează înlocuirea sa treptată (începând cu 2010) cu variante mai sigure, precum SHA-2 și SHA-3.

## 3.7 preliminarii operaționale

### 3.7.1 șiruri de biți și întregi

Se va folosi următoarea terminologie referitoare la șiruri de biți și numere întregi:

a. O cifră hexazecimală este un element al mulțimii  $\{0, 1, \dots, 9, A, \dots, F\}$ . O cifră hexazecimală este reprezentarea unui șir de 4 biți. Exemple:  $7 = 0111$ ,  $A = 1010$ .

b. Un cuvânt este egal cu un șir de 32 de biți care poate fi reprezentat ca o succesiune de 8 cifre hexazecimale. Pentru a converti un cuvânt la 8 cifre hexazecimale fiecare șir de biți pe 4 biți este convertit în echivalentul său hexa, așa cum este descris la punctul (a) de mai sus. Exemplu:

$1010\ 0001\ 0000\ 0011\ 1111\ 1110\ 0010\ 0011 = A103FE23$ .

c. Un întreg între  $0$  și  $2^{32} - 1$  inclusiv poate fi reprezentat ca un cuvânt (de 32 de biți). Cei mai puțin semnificativi patru biți ai întregului sunt reprezentați ca cea mai din dreapta cifră hexa a reprezentării cuvântului. Exemplu: întregul  $291 = 2^8 + 2^5 + 2^1 + 2^0 = 256 + 32 + 2 + 1$  este reprezentat de cuvântul hexa:  $00000123$ .

Dacă  $z$  este un număr întreg,  $0 \leq z < 2^{64}$ , atunci  $z = 2^{32}x + y$  unde  $0 \leq x < 2^{32}$  și  $0 \leq y < 2^{32}$ . Deoarece  $x$  și  $y$  pot fi reprezentate ca și cuvinte  $X$  și  $Y$ ,  $z$  poate fi reprezentat ca o pereche de cuvinte  $(X, Y)$ .

d. blocul = șir de 512 biți. Un bloc (de exemplu,  $B$ ) poate fi reprezentat ca o secvență de 16 cuvinte.

### 3.7.2 operații pe cuvinte

Următorii operatori logici vor fi aplicați cuvintelor:

a. Operațiuni la nivel de bit cu cuvinte logice

$X \wedge Y = \text{"și"}$  (la nivel de bit) al lui  $X$  și  $Y$ .

$X \vee Y = \text{"sau inclusiv"}$  de  $X$  și  $Y$ .

$X \oplus Y = \text{"sau exclusiv"}$  al lui  $X$  și  $Y$

$\sim X$  = "complementul" logic la nivel de bit al lui X.

Exemplu:

```

      01101100101110011101001001111011
XOR  01100101110000010110100110110111
-----
=    00001001011110001011101111001100

```

b. Operația  $X + Y$  este definită după cum urmează: cuvintele X și Y reprezintă numerele întregi x și y, unde  $0 \leq x < 2^{32}$  și  $0 \leq y < 2^{32}$ . Pentru numerele întregi pozitive n și m, notăm cu  $(n \bmod m)$  restul împărțind lui n cu m. Calculăm:

$$z = (x + y) \bmod 2^{32}.$$

Atunci  $0 \leq z < 2^{32}$ . Convertim z la un cuvânt, Z și definim  $Z = X + Y$ .

c. Operația de șifrare circulară stânga  $S^n(X)$ , unde X este un cuvânt și n este un număr întreg cu  $0 \leq n < 2^{32}$ , este definită prin:

$$S^n(X) = (X \ll n) \text{ OR } (X \gg 32-n).$$

În cele de mai sus,  $X \ll n$  este obținut după cum urmează: desconsiderați cei mai din stânga n biți ai lui X și apoi introduceți n de zero în dreapta (rezultatul va fi în continuare de 32 de biți).  $X \gg n$  este obținut prin eliminarea celor mai mulți n biți ai lui X și apoi completarea rezultatului cu n de zero în stânga. Astfel,  $S^n(X)$  este echivalent cu o deplasare circulară a lui X cu n poziții spre stânga.

## 3.8 descrierea funcției SHA-1

### 3.8.1 completarea mesajului (message padding)

SHA-1 este folosit pentru a calcula hash-ul unui mesaj sau al unui fișier de date care este furnizat ca intrare. Mesajul sau fișierul de date ar trebui considerat a fi un șir de biți. Lungimea mesajului este numărul de biți din mesaj (mesajul gol are lungimea 0). Dacă numărul de biți dintr-un mesaj este un multiplu de 8, pentru compactitate putem reprezenta mesajul în hexa. Scopul completării mesajului este de a face lungimea totală a unui mesaj să fie un multiplu de 512. SHA-1 procesează secvențial blocuri de 512 biți atunci când se compune digestul mesajului. Următoarele specifică modul în care se realizează această completare. Pe scurt, un "1" urmat de m de "0", urmat de un întreg pe 64 de biți sunt atașați la sfârșitul mesajului pentru a produce un mesaj completat (padded) cu lungimea de  $512 * n$ . Numărul întreg pe 64 de biți este l, lungimea mesajului original. Mesajul completat (căptușit) este apoi procesat de SHA-1 ca n blocuri de 512 biți.

Să presupunem că un mesaj are lungimea  $l < 2^{64}$ . Înainte de a intra în SHA-1, mesajul este completat (căptușit) în dreapta după cum urmează:

a. un bit cu valoarea "1" este atașat. Exemplu: dacă mesajul original este "01010000", acesta este căptușit la "010100001".

b. Se adaugă biți zero "0". Numărul de zerouri "0" va depinde de lungimea inițială a mesajului. Ultimii 64 de biți ai ultimului bloc de 512 biți sunt rezervați pentru lungimea l a mesajului original.

Exemplu: Să presupunem că mesajul original este șirul de biți

```
01100001 01100010 01100011 01100100 01100101.
```

După pasul (a) obținem:

## chapter 3

01100001 01100010 01100011 01100100 01100101 1.

Deoarece  $l = 40$ , numărul de biți de mai sus este 41 și 407 de "0" sunt atașați, făcând totalul acum 448. Aceasta dă (în hexa):

```
61626364 65800000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000.
```

c. Obțineți reprezentarea de 2 cuvinte a lui  $l$ , numărul de biți din mesajul original. Dacă  $l < 2^{32}$  atunci primul cuvânt este zero. Adăugați aceste două cuvinte la mesajul captușit.

Exemplu: Să presupunem că mesajul inițial este ca în (b). Apoi  $l = 40$  (rețineți că  $l$  este calculat înainte de orice umplutură). Reprezentarea cu două cuvinte a lui 40 este hex 00000000 00000028. Prin urmare, mesajul captușit final este în hexa:

```
61626364 65800000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000028.
```

Mesajul captușit va conține  $16 \cdot n$  cuvinte pentru un număr  $n > 0$ . Mesajul captușit este considerat ca o secvență de  $n$  blocuri  $M_1, M_2, \dots, M_n$ , unde fiecare  $M_i$  conține 16 cuvinte și  $M_1$  conține primele caractere (sau biți) ai mesajului.

### 3.8.2 funcții utilizate

SHA-1 utilizează o secvență de funcții logice  $f_0, f_1, \dots, f_{79}$ . Fiecare  $f_t, 0 \leq t \leq 79$ , operează pe trei cuvinte pe 32 de biți  $B, C, D$  și produce un cuvânt de 32 de biți ca ieșire.  $f_t(B, C, D)$  este definită după cum urmează: pentru cuvintele  $B, C, D$ ,

$$f_t(B, C, D) = (B \text{ AND } C) \text{ OR } ((\text{NOT } B) \text{ AND } D) \quad (0 \leq t \leq 19)$$

$$f_t(B, C, D) = B \text{ XOR } C \text{ XOR } D \quad (20 \leq t \leq 39)$$

$$f_t(B, C, D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D) \quad (40 \leq t \leq 59)$$

$$f_t(B, C, D) = B \text{ XOR } C \text{ XOR } D \quad (60 \leq t \leq 79).$$

### 3.8.3 constante utilizate

SHA-1 utilizează o secvență de cuvinte constante  $K(0), K(1), \dots, K(79)$ . În hexa, acestea sunt date de:

$$K_t = 5A827999 \quad (0 \leq t \leq 19)$$

$$K_t = 6ED9EBA1 \quad (20 \leq t \leq 39)$$

$$K_t = 8F1BBCDC \quad (40 \leq t \leq 59)$$

$$K_t = CA62C1D6 \quad (60 \leq t \leq 79)$$

### 3.9 pseudo-cod pentru SHA-1

O funcție hash criptografică este o transformare care ia o intrare și returnează un șir de dimensiuni fixe, care se numește valoarea hash.

```
// inițializare variabile
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0

// pre-procesare:
adăugați bitul "1" la mesaj
adăugați  $0 \leq k < 512$  biți cu valoarea "0", astfel încât lungimea mesajului
rezultat (în biți) este congruent cu  $448 \equiv -64 \pmod{512}$ 
adăugați lungimea mesajului (înainte de pre-procesare), în biți, ca întreg
întreg pe 64 de biți

// procesați mesajul în blocuri succesive de 512 de biți:
// spargeți mesajul în bucăți de 512 biți

for each bloc
    spargeți blocul în șaisprezece cuvinte pe 32 de biți  $w[i]$ ,  $0 \leq i \leq 15$ 

    // Extindeți cele 16 cuvinte pe 32 de biți în 80 de cuvinte pe 32 biți:
    for  $i$  from 16 to 79
         $w[i] = (w[i-3] \text{ xor } w[i-8] \text{ xor } w[i-14] \text{ xor } w[i-16])$  leftrotate 1

    // Inițializați valoarea hash pentru acest bloc:
    a = h0
    b = h1
    c = h2
    d = h3
    e = h4

    // bucla principală:
    for  $i$  from 0 to 79
        if  $0 \leq i \leq 19$  then
            f = (b and c) or ((not b) and d)
            k = 0x5A827999
        else if  $20 \leq i \leq 39$ 
            f = b xor c xor d
            k = 0x6ED9EBA1
        else if  $40 \leq i \leq 59$ 
            f = (b and c) or (b and d) or (c and d)
            k = 0x8F1BBCDC
        else if  $60 \leq i \leq 79$ 
            f = b xor c xor d
            k = 0xCA62C1D6

        temp = (a leftrotate 5) + f + e + k + w[i]
        e = d
        d = c
        c = b leftrotate 30
        b = a
        a = temp
```

## chapter 3

```
// adaugă hash-ul acestui bloc la rezultatul curent:
h0 = h0 + a
h1 = h1 + b
h2 = h2 + c
h3 = h3 + d
h4 = h4 + e

// obțineți valoarea de hash finală (big-endian):
digest = hash = h0 append h1 append h2 append h3 append h4
```

### 3.10 un mesaj simplu și valoarea sa de hash

Mesajul este codul binar ASCII al șirului "abc", i.e.,  
01100001 01100010 01100011.

Acest mesaj are lungimea  $l = 24$ . În pasul (a) din secțiunea 4, adăugăm "1". În pasul (b) adăugăm 423 "0" s. În pasul (c) se anexează hex 00000000 00000018, reprezentarea pe două cuvinte a lui 24. Astfel, mesajul final completat (căpțușit) constă dintr-un bloc, astfel încât  $n = 1$  în notația secțiunii 4.

Valorile inițiale ale regiștrilor  $\{H_i\}$  în hexa sunt:

```
H0 = 67452301
H1 = EFCDA89
H2 = 98BADCFE
H3 = 10325476
H4 = C3D2E1F0.
```

Începem procesarea blocului 1. Cuvintele blocului sunt:

```
W[0] = 61626380
W[1] = 00000000
W[2] = 00000000
W[3] = 00000000
W[4] = 00000000
W[5] = 00000000
W[6] = 00000000
W[7] = 00000000
W[8] = 00000000
W[9] = 00000000
W[10] = 00000000
W[11] = 00000000
W[12] = 00000000
W[13] = 00000000
W[14] = 00000000
W[15] = 00000018.
```

Valorile hexa ale lui A, B, C, D, E după pasul  $t$  al buclei "pentru  $t = 0$  la 79" (etapa (d) din secțiunea 7 sau etapa (c) din secțiunea 8) sunt:

	A	B	C	D	E
$t = 0$ :	0116FC33	67452301	7BF36AE2	98BADCFE	10325476
$t = 1$ :	8990536D	0116FC33	59D148C0	7BF36AE2	98BADCFE
$t = 2$ :	A1390F08	8990536D	C045BF0C	59D148C0	7BF36AE2

## secure hash algorithm

t = 3:	CDD8E11B	A1390F08	626414DB	C045BF0C	59D148C0
t = 4:	CFD499DE	CDD8E11B	284E43C2	626414DB	C045BF0C
t = 5:	3FC7CA40	CFD499DE	F3763846	284E43C2	626414DB
t = 6:	993E30C1	3FC7CA40	B3F52677	F3763846	284E43C2
t = 7:	9E8C07D4	993E30C1	0FF1F290	B3F52677	F3763846
t = 8:	4B6AE328	9E8C07D4	664F8C30	0FF1F290	B3F52677
t = 9:	8351F929	4B6AE328	27A301F5	664F8C30	0FF1F290
t = 10:	FBDA9E89	8351F929	12DAB8CA	27A301F5	664F8C30
t = 11:	63188FE4	FBDA9E89	60D47E4A	12DAB8CA	27A301F5
t = 12:	4607B664	63188FE4	7EF6A7A2	60D47E4A	12DAB8CA
t = 13:	9128F695	4607B664	18C623F9	7EF6A7A2	60D47E4A
t = 14:	196BEE77	9128F695	1181ED99	18C623F9	7EF6A7A2
t = 15:	20BDD62F	196BEE77	644A3DA5	1181ED99	18C623F9
t = 16:	4E925823	20BDD62F	C65AFB9D	644A3DA5	1181ED99
t = 17:	82AA6728	4E925823	C82F758B	C65AFB9D	644A3DA5
t = 18:	DC64901D	82AA6728	D3A49608	C82F758B	C65AFB9D
t = 19:	FD9E1D7D	DC64901D	20AA99CA	D3A49608	C82F758B
t = 20:	1A37B0CA	FD9E1D7D	77192407	20AA99CA	D3A49608
t = 21:	33A23BFC	1A37B0CA	7F67875F	77192407	20AA99CA
t = 22:	21283486	33A23BFC	868DEC32	7F67875F	77192407
t = 23:	D541F12D	21283486	0CE88EFF	868DEC32	7F67875F
t = 24:	C7567DC6	D541F12D	884A0D21	0CE88EFF	868DEC32
t = 25:	48413BA4	C7567DC6	75507C4B	884A0D21	0CE88EFF
t = 26:	BE35FBD5	48413BA4	B1D59F71	75507C4B	884A0D21
t = 27:	4AA84D97	BE35FBD5	12104EE9	B1D59F71	75507C4B
t = 28:	8370B52E	4AA84D97	6F8D7EF5	12104EE9	B1D59F71
t = 29:	C5FBAF5D	8370B52E	D2AA1365	6F8D7EF5	12104EE9
t = 30:	1267B407	C5FBAF5D	A0DC2D4B	D2AA1365	6F8D7EF5
t = 31:	3B845D33	1267B407	717EEBD7	A0DC2D4B	D2AA1365
t = 32:	046FAA0A	3B845D33	C499ED01	717EEBD7	A0DC2D4B
t = 33:	2C0EBC11	046FAA0A	CEE1174C	C499ED01	717EEBD7
t = 34:	21796AD4	2C0EBC11	811BEA82	CEE1174C	C499ED01
t = 35:	DCBBB0CB	21796AD4	4B03AF04	811BEA82	CEE1174C
t = 36:	0F511FD8	DCBBB0CB	085E5AB5	4B03AF04	811BEA82
t = 37:	DC63973F	0F511FD8	F72EEC32	085E5AB5	4B03AF04
t = 38:	4C986405	DC63973F	03D447F6	F72EEC32	085E5AB5
t = 39:	32DE1CBA	4C986405	F718E5CF	03D447F6	F72EEC32
t = 40:	FC87DEDF	32DE1CBA	53261901	F718E5CF	03D447F6
t = 41:	970A0D5C	FC87DEDF	8CB7872E	53261901	F718E5CF
t = 42:	7F193DC5	970A0D5C	FF21F7B7	8CB7872E	53261901
t = 43:	EE1B1AAF	7F193DC5	25C28357	FF21F7B7	8CB7872E
t = 44:	40F28E09	EE1B1AAF	5FC64F71	25C28357	FF21F7B7
t = 45:	1C51E1F2	40F28E09	FB86C6AB	5FC64F71	25C28357
t = 46:	A01B846C	1C51E1F2	503CA382	FB86C6AB	5FC64F71
t = 47:	BEAD02CA	A01B846C	8714787C	503CA382	FB86C6AB
t = 48:	BAF39337	BEAD02CA	2806E11B	8714787C	503CA382
t = 49:	120731C5	BAF39337	AFAB40B2	2806E11B	8714787C
t = 50:	641DB2CE	120731C5	EEBCE4CD	AFAB40B2	2806E11B
t = 51:	3847AD66	641DB2CE	4481CC71	EEBCE4CD	AFAB40B2
t = 52:	E490436D	3847AD66	99076CB3	4481CC71	EEBCE4CD
t = 53:	27E9F1D8	E490436D	8E11EB59	99076CB3	4481CC71
t = 54:	7B71F76D	27E9F1D8	792410DB	8E11EB59	99076CB3
t = 55:	5E6456AF	7B71F76D	09FA7C76	792410DB	8E11EB59
t = 56:	C846093F	5E6456AF	5EDC7DDB	09FA7C76	792410DB
t = 57:	D262FF50	C846093F	D79915AB	5EDC7DDB	09FA7C76
t = 58:	09D785FD	D262FF50	F211824F	D79915AB	5EDC7DDB
t = 59:	3F52DE5A	09D785FD	3498BFD4	F211824F	D79915AB
t = 60:	D756C147	3F52DE5A	4275E17F	3498BFD4	F211824F
t = 61:	548C9CB2	D756C147	8FD4B796	4275E17F	3498BFD4

## chapter 3

t = 62:	B66C020B	548C9CB2	F5D5B051	8FD4B796	4275E17F
t = 63:	6B61C9E1	B66C020B	9523272C	F5D5B051	8FD4B796
t = 64:	19DFA7AC	6B61C9E1	ED9B0082	9523272C	F5D5B051
t = 65:	101655F9	19DFA7AC	5AD87278	ED9B0082	9523272C
t = 66:	0C3DF2B4	101655F9	0677E9EB	5AD87278	ED9B0082
t = 67:	78DD4D2B	0C3DF2B4	4405957E	0677E9EB	5AD87278
t = 68:	497093C0	78DD4D2B	030F7CAD	4405957E	0677E9EB
t = 69:	3F2588C2	497093C0	DE37534A	030F7CAD	4405957E
t = 70:	C199F8C7	3F2588C2	125C24F0	DE37534A	030F7CAD
t = 71:	39859DE7	C199F8C7	8FC96230	125C24F0	DE37534A
t = 72:	EDB42DE4	39859DE7	F0667E31	8FC96230	125C24F0
t = 73:	11793F6F	EDB42DE4	CE616779	F0667E31	8FC96230
t = 74:	5EE76897	11793F6F	3B6D0B79	CE616779	F0667E31
t = 75:	63F7DAB7	5EE76897	C45E4FDB	3B6D0B79	CE616779
t = 76:	A079B7D9	63F7DAB7	D7B9DA25	C45E4FDB	3B6D0B79
t = 77:	860D21CC	A079B7D9	D8FDF6AD	D7B9DA25	C45E4FDB
t = 78:	5738D5E1	860D21CC	681E6DF6	D8FDF6AD	D7B9DA25
t = 79:	42541B35	5738D5E1	21834873	681E6DF6	D8FDF6AD.

Blocul 1 a fost procesat. Valorile  $\{H_i\}$  sunt:

$$H_0 = 67452301 + 42541B35 = A9993E36$$

$$H_1 = EFCDA89 + 5738D5E1 = 4706816A$$

$$H_2 = 98BADCFE + 21834873 = BA3E2571$$

$$H_3 = 10325476 + 681E6DF6 = 7850C26C$$

$$H_4 = C3D2E1F0 + D8FDF6AD = 9CD0D89D.$$

Hashul mesajului = A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D

## 3.11 familia de funcții burete Keccak pentru SHA-3

Prezentarea acestei funcții este conformă cu [KSF], scris de Gudo Bertoni, Joan Daemen, Michael Peeters și Gilles Van Assche.

Keccak (pronunțat [kɛtʃak], ca și "ketchak") este o familie de funcții hash care a fost adoptată ca algoritmul hash al NIST pentru SHA-3. Textul de mai jos este o descriere rapidă a lui Keccak folosind pseudo-cod. Nimic nu ar trebui ca acest text introductiv să fie considerat o descriere oficială și de referință a lui Keccak. În schimb, obiectivul este de a prezenta Keccak cu accent pe lizibilitate și claritate.

### 3.11.1 structura algoritmului Keccak

Keccak este o familie de funcții hash care se bazează pe construcția de tip burete și, prin urmare, este o familie de funcții de tip burete. În Keccak, funcția de bază este o permutare aleasă într-un set de șapte permutări Keccak-f, denumită Keccak-f[b], unde  $b \in \{25, 50, 100, 200, 400, 800, 1600\}$  este mărimea permutării. Mărimea permutării este, de asemenea, mărimea stării în construcția buretelui.

Starea este organizată ca o serie de  $5 \times 5$  benzi, fiecare având lungimea  $w \in \{1, 2, 4, 8, 16, 32, 64\}$  ( $b = 25w$ ). Când este implementată pe un procesor pe 64 de biți, o bandă de Keccak-f[1600] poate fi reprezentată ca un cuvânt CPU pe 64 de biți.

Obținem funcția de burete Keccak[r, c], cu parametrii de capacitate c și rata de biți r, dacă aplicăm construcția de burete la Keccak-f[r + c] și aplicăm o completare (umplere) specifică mesajului de intrare.



### 3.11.2 descrierea pseudo-codului

În primul rând începem cu descrierea lui Keccak-f în pseudo-codul de mai jos. Numărul de runde nr depinde de lățimea de permutare și este dat de  $nr = 12 + 2l$ , unde  $2l = w$ . Acest lucru dă 24 de runde pentru Keccak-f [1600].

```
Keccak-f[b](A) {
  forall i in 0...nr-1
    A = Round[b](A, RC[i])
  return A
}

Round[b](A,RC) {
  // θ step
  C[x] = A[x,0] xor A[x,1] xor A[x,2] xor A[x,3] xor A[x,4], forall x in 0...4
  D[x] = C[x-1] xor rot(C[x+1],1), forall x in 0...4
  A[x,y] = A[x,y] xor D[x], forall (x,y) in (0...4,0...4)

  // ρ and π steps
  B[y,2*x+3*y] = rot(A[x,y], r[x,y]), forall (x,y) in (0...4,0...4)

  // χ step
  A[x,y] = B[x,y] xor ((not B[x+1,y]) and B[x+2,y]), forall (x,y) in (0...4,0...4)
4)

  // ι step
  A[0,0] = A[0,0] xor RC

  return A
}
```

În pseudocodul de mai sus, se folosesc următoarele convenții. Toate operațiunile pe indici se fac modulo 5. A denotă matricea completă de stare a permutării, iar A [x, y] denotă o bandă specială în acea stare. B [x, y], C [x], D [x] sunt variabile intermediare. Constantele r [x, y] sunt indecșii de rotație (vezi tabelul 2), în timp ce RC [i] sunt constantele rundei (vezi tabelul 1). rot (W, r) este operația obișnuită de șift circular la biților, deplasând un bit din poziția i în poziția i + r (modulo dimensiunea benzii).

Apoi, prezentăm pseudocodul pentru funcția burete Keccak[r, c], cu parametrii de capacitate c și rata biților r. Descrierea de mai jos este limitată la cazul mesajelor care acoperă un număr întreg de octeți. Pentru mesajele cu un număr de biți care nu pot fi divizate cu 8, consultați specificațiile pentru mai multe detalii. De asemenea, presupunem pentru simplitate că r este un multiplu al mărimii benzii; acesta este cazul pentru parametrii candidați SHA-3.

```
Keccak[r,c](M) {
  // Inițializare și umplere
  S[x,y] = 0, forall (x,y) in (0...4,0...4)
  P = M || 0x01 || 0x00 || ... || 0x00
  P = P xor (0x00 || ... || 0x00 || 0x80)

  // faza de absorbție
  forall block Pi in P
    S[x,y] = S[x,y] xor Pi[x+5*y], forall (x,y) such that x+5*y < r/w
    S = Keccak-f[r+c](S)

  // faza de presarew
  Z = empty string
```

## chapter 3

```
while output is requested
  Z = Z || S[x,y], forall (x,y) such that x+5*y < r/w
  S = Keccak-f[r+c](S)

return Z
}
```

În pseudo-codul de mai sus, S reprezintă starea ca o serie de benzi. Mesajul căptușit (completat) P este organizat ca o serie de blocuri  $P_i$ , ele însele fiind organizate ca o serie de benzi. Operatorul  $||$  denotă concatenarea obișnuită a șirurilor de octeți.

## chapter 4 digital encryption standard

### 4.1 scurtă istorie a of DES

Originile DES datează de la începutul anilor 1970. În 1972, după încheierea unui studiu privind necesitățile de securitate informatică ale guvernului SUA, organismul american de standardizare NBS (denumit acum NIST) a identificat necesitatea unui standard guvernamental de criptare a informațiilor neclasificate, sensibile. În consecință, la 15 mai 1973, după consultarea cu NSA, NBS a solicitat propuneri pentru un cifru care să îndeplinească criteriile riguroase de proiectare. Niciuna dintre propunerile trimise nu a fost considerată potrivită. O a doua solicitare a fost emisă la 27 august 1974. De data aceasta, IBM a prezentat un candidat care a fost considerat acceptabil - un cifru dezvoltat în perioada 1973-1974 bazat pe un algoritm anterior, cifrul Lucifer al lui Horst Feistel. Echipa de la IBM implicată în proiectarea și analiza cifrului i-a inclus pe Feistel, Walter Tuchman, Don Coppersmith, Alan Konheim, Carl Meyer, Mike Matyas, Roy Adler, Edna Grossman, Bill Notz, Lynn Smith și Bryant Tuckerman.

#### 4.1.1 implicarea NSA în design

La 17 martie 1975, DES propus a fost publicat în Registrul federal. Au fost solicitate observații publice, iar în anul următor au avut loc două ateliere deschise pentru a discuta standardul propus. Au existat unele critici din partea unor participanți, inclusiv de la pionierii criptografiei publice Martin Hellman și Whitfield Diffie, specificând lungimea scurtă a cheii și misterioasele "S-boxes" ca dovadă a interferențelor incorecte din partea NSA. Suspiciunea era că algoritmul a fost slăbit în secret de către agenția de informații, astfel încât ei - dar nimeni altcineva - să citească cu ușurință mesajele criptate. Alan Konheim (unul dintre designerii DES) a comentat: "Am trimis boxele S la Washington, s-au întors și au fost cu totul altfel". Comitetul selectiv al Senatului Statelor Unite ale Americii privind serviciile de informații a revizuit acțiunile NSA pentru a determina dacă a existat o implicare necorespunzătoare. În rezumatul neclasificat al concluziilor lor, publicat în 1978, Comitetul a scris:

*"În dezvoltarea DES, NSA a convins IBM că o dimensiune redusă a cheii este suficientă, a asistat indirect la dezvoltarea structurilor S-box și a certificat că algoritmul final DES era, după cunoștințele sale, liber de orice slăbiciune statistică sau matematică."*

Cu toate acestea, a constatat, de asemenea, că:

*"NSA nu a manipulat în nici un fel proiectarea algoritmului. IBM a inventat și a proiectat algoritmul, a luat toate deciziile pertinente cu privire la acesta și a fost de acord că dimensiunea-cheii convenită este mai mult decât adecvată pentru toate aplicațiile comerciale pentru care DES este intenționat."*

Un alt membru al echipei DES, Walter Tuchman, este citat spunând: "Am dezvoltat algoritmul DES în întregime în IBM folosind IBMers. NSA nu a dictat un singur fir!" În schimb, o carte desecretizată a NSA privind istoria criptologică afirmă:

*"În 1973, NBS a solicitat industriei private un standard de criptare a datelor (DES), iar primele oferte au fost dezamăgitoare, așa că NSA a început să lucreze la algoritmul său propriu", a declarat Howard Rosenblum, director adjunct pentru cercetare și inginerie, că Walter Tuchman de la IBM pe o modificare a lui Lucifer pentru uz general. NSA ia dat lui Tuchman o autorizație și la adus să lucreze împreună cu Agenția pentru modificarea lui Lucifer."*

Unele suspiciuni cu privire la slăbiciunile ascunse în cutiile S au fost eliminate în 1990, cu descoperirea independentă și publicarea deschisă de Eli Biham și Adi Shamir a criptanalizei diferențiale, o metodă generală de spargere a cifrurilor de tip bloc. Casetă S a DES a fost mult mai rezistentă la atac decât dacă ar fi fost alese la întâmplare, sugerând puternic că IBM a știut despre tehnică din 1970. Acesta a fost într-adevăr

## chapter 4

cazul - în 1994, Don Coppersmith a publicat unele dintre criteriile originale de proiectare pentru casetele S. Potrivit lui Steven Levy, cercetătorii IBM Watson au descoperit atacuri criptanalitice diferențiate în 1974 și au fost rugați de către NSA să păstreze secretul tehnicii. Coppersmith explică decizia secretă a IBM prin a spune: "Aceasta a fost pentru că [criptanaliza diferențială] poate fi un instrument foarte puternic, folosit împotriva multor scheme, și există îngrijorare că astfel de informații în domeniul public ar putea afecta negativ securitatea națională". Levy citează pe Walter Tuchman: "Ne-au cerut să declarăm toate documentele confidențiale ... Am pus de fapt un număr pe fiecare și le-am încuiat în seifuri, pentru că erau considerate secrete ale guvernului american."

### 4.1.2 algoritmul ca standard

În ciuda criticilor, DES a fost aprobat ca standard federal în noiembrie 1976 și publicat la 15 ianuarie 1977 ca FIPS PUB 46, fiind autorizat pentru utilizare cu toate datele neclasificate. A fost ulterior reafirmat ca standard în 1983, 1988 (revizuit ca FIPS-46-1), 1993 (FIPS-46-2) și din nou în 1999 (FIPS-46-3), acesta din urmă descriind "Triple DES". La 26 mai 2002, DES a fost în cele din urmă înlocuită de Standardul de criptare avansată (AES), în urma unei competiții publice. La 19 mai 2005, FIPS 46-3 a fost retras oficial, însă NIST a aprobat Triple DES până în anul 2030 pentru informații guvernamentale sensibile.

Algoritmul este de asemenea specificat în ANSI X3.92, NIST SP 800-67 și ISO / IEC 18033-3 (ca o componentă a TDEA).

Un alt atac teoretic, criptanaliza liniară, a fost publicat în 1994, dar a fost un atac de forță brută în 1998 care a demonstrat că DES ar putea fi atacat foarte practic și a subliniat necesitatea unui algoritm de înlocuire.

## 4.2 descriere

DES este cifrul blocului arhetipal - un algoritm care ia un șir de lungime fixă de biți plaintext și îl transformă printr-o serie de operații complicate într-un alt bitstring de tip ciphertext de aceeași lungime. În cazul DES, dimensiunea blocului este de 64 de biți. DES utilizează, de asemenea, o cheie pentru personalizarea transformării, astfel încât decriptarea se presupune că este realizată numai de cei care cunosc cheia specială folosită pentru a cripta. Cheia aparent constă în 64 de biți; totuși, doar 56 dintre acestea sunt de fapt folosite de algoritm. Opt biți sunt utilizați exclusiv pentru verificarea parității și apoi sunt eliminați. Prin urmare, lungimea efectivă a cheii este de 56 de biți și este de obicei citată ca atare.

Ca și alte cipuri de blocuri, DES nu este un mijloc securizat de criptare, ci trebuie folosit într-un mod de operare. FIPS-81 specifică mai multe moduri de utilizare cu DES. Comentarii suplimentare privind utilizarea DES sunt cuprinse în FIPS-74.

Structura generală a algoritmului este prezentată în Figura 1: există 16 etape identice de prelucrare, numite runde. Există, de asemenea, o permutare inițială și finală, denumită IP și FP, care sunt inverse (IP "anulează" acțiunea FP și invers). IP și FP nu au aproape nici o semnificație criptografică, dar au fost aparent incluse pentru a facilita blocurile de încărcare în și din hardware-ul din anii 1970, precum și pentru a face DES să funcționeze mai lent în software-ul.

Înainte de rundele principale, blocul este împărțit în două jumătăți de 32 de biți și procesat alternativ; această intersecție este cunoscută sub numele de schema Feistel. Structura Feistel asigură că decriptarea și criptarea sunt procese foarte asemănătoare - singura diferență este că subcheiile sunt aplicate în ordine inversă atunci când se decriptează. Restul algoritmului este identic. Acest lucru simplifică foarte mult implementarea, în special în domeniul hardware, deoarece nu este nevoie de algoritmi separați de criptare și decriptare.

Simbolul  $\oplus$  desemnează operațiunea exclusiv-OR (XOR). Funcția F combină jumătate de bloc împreună cu o parte din cheie. Ieșirea din funcția F este apoi combinată cu cealaltă jumătate a blocului, iar jumătățile sunt schimbate înainte de următoarea rundă. După runda finală, jumătățile nu sunt schimbate; aceasta este o caracteristică a structurii Feistel care face similare procesele de criptare și decriptare.

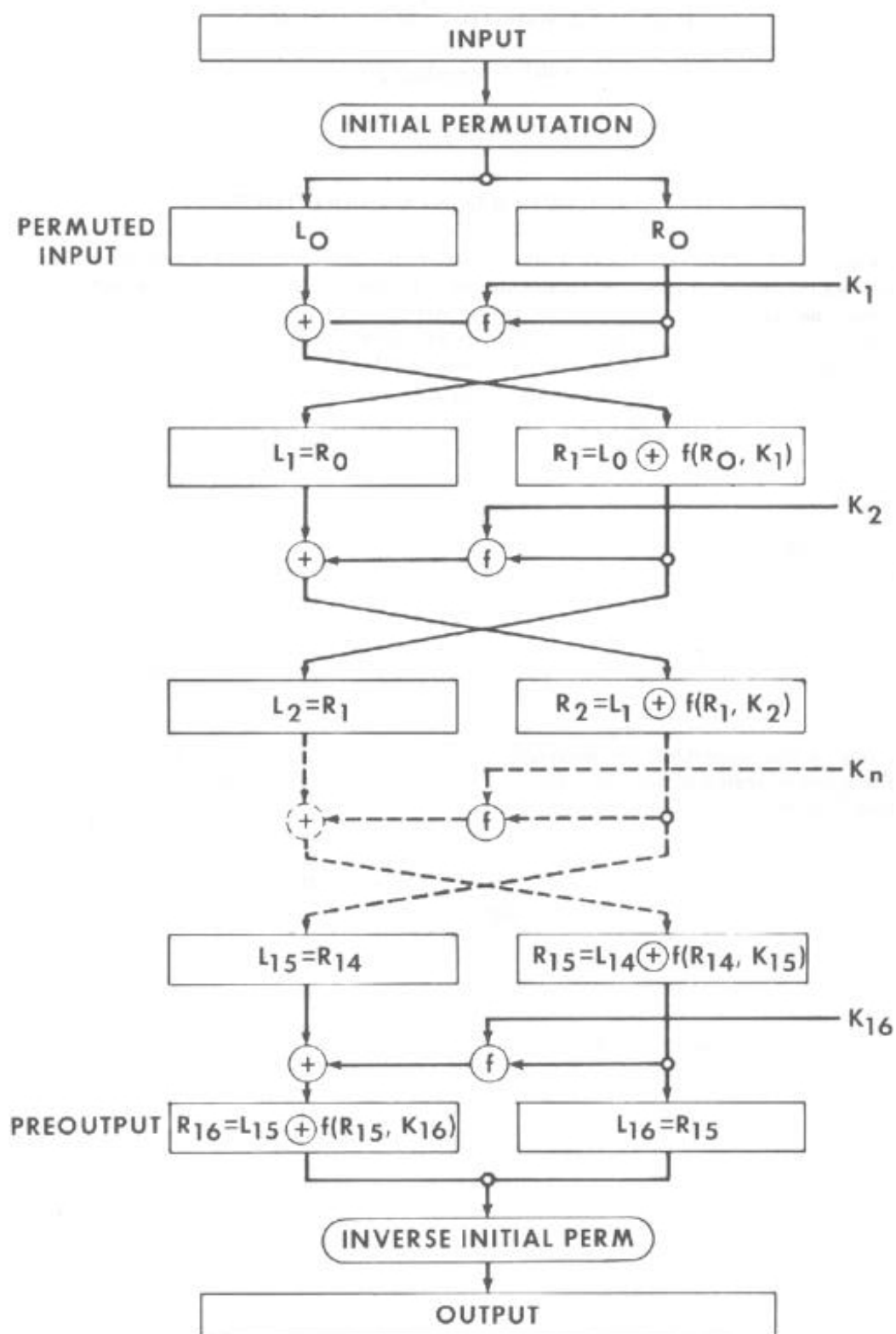


Figura 1 – Structura generală a algoritmului DES

## 4.3 funcția Feistel (F)

Funcția F, prezentată în figura 2, funcționează pe o jumătate de bloc (32 biți) la un moment dat și constă din patru etape:

1. Extindere - jumătatea de bloc pe 32 de biți este extinsă la 48 de biți folosind permutarea de expansiune, denumită E în diagramă, duplicând jumătate din biți. Ieșirea este compusă din 8 bucăți de 6 biți, fiecare conținând o copie de 4 biți de intrare corespunzătoare, plus o copie a bitului imediat adiacent din fiecare piesă de intrare în fiecare parte.

2. Mixarea cheii - rezultatul este combinat cu o subcheie folosind o operație XOR. Șasezeci de sub-chei de 48 de biți - câte una pentru fiecare rundă - sunt derivate din cheia principală folosind programul cheilor (descriș mai jos).

3. Înlocuire - după amestecare în subcheie, blocul este împărțit în opt bucăți de 6 biți înainte de prelucrare de către cutiile S sau casetele de substituție. Fiecare dintre cele opt casete S înlocuiește cei șase biți de intrare cu patru biți de ieșire, în conformitate cu o transformare neliniară, furnizată sub forma unei tabele de căutare. Casetele S furnizează nucleul securității DES - fără ele, cifrul ar fi liniar și trivial de spart.

4. Permutarea - în cele din urmă, cele 32 de ieșiri din casetele S sunt rearanjate în funcție de o permutare fixă, P. Aceasta este proiectată astfel încât, după expansiune, fiecare bit de ieșire al lui S-box să fie împărțit în 6 casete S diferite în runda următoare.

Alternarea substituției din cutiile S și permutarea biților din cutia P și E-expansiune oferă așa-numita "confuzie și difuzie" respectiv, un concept identificat de Claude Shannon în anii 1940 drept o condiție necesară pentru un cifru securizat și practic.

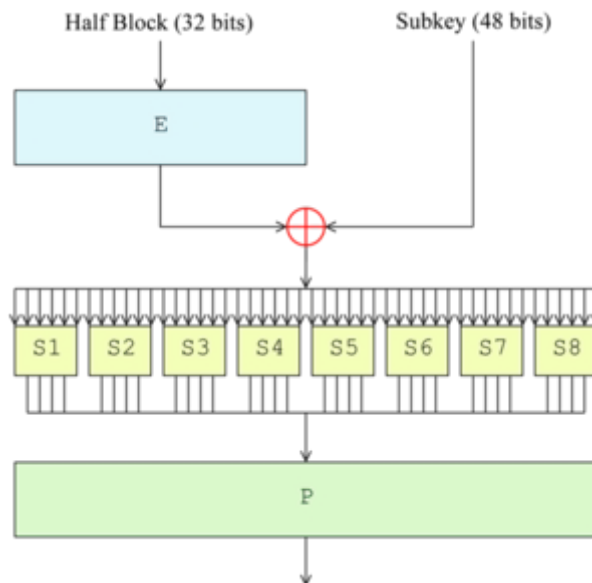


Figura 2 – funcția Feistel

## 4.4 programul cheilor

Figura 3 ilustrează programul cheilor pentru criptare - algoritmul care generează subcheile. Inițial, 56 de biți ai cheii sunt selectați din cei 64 inițiali cu Permuted Choice 1 (PC-1) - restul de 8 biți sunt fie eliminați, fie

utilizați ca biți de verificare a parității. Cei 56 de biți sunt apoi împărțiți în două jumătăți de 28 de biți; fiecare jumătate este apoi tratată separat. În runde succesive, ambele jumătăți sunt rotite la stânga cu unul sau doi biți (specificați pentru fiecare rundă) și apoi 48 biți de subcheie sunt selectați de Permuted Choice 2 (PC-2) - 24 de biți din jumătatea stângă și 24 de la dreapta. Rotațiile (indicate prin "<<<" în diagramă) înseamnă că un set diferit de biți este utilizat în fiecare subcheie; fiecare bit este utilizat în aproximativ 14 din cele 16 subchei.

Programul cheilor pentru decriptare este similar - subcheile sunt în ordine inversă față de criptare. În afară de această schimbare, procesul este același ca și pentru criptare.

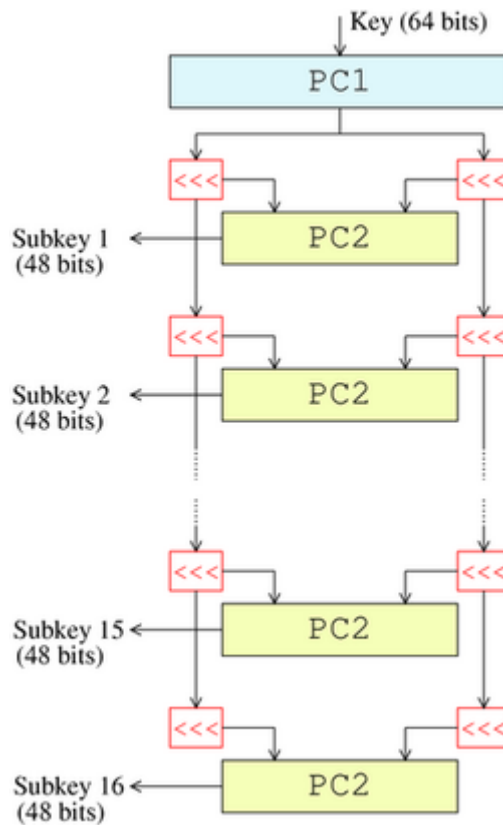


Figura 3

## 4.5 pașii algoritmului DES

Pozițiile celor 64 de biți ai blocului de intrare ce urmează a fi criptat sunt permutate, conform permutării inițiale **IP**:

**IP**

58 50 42 34 26 18 10 2  
 60 52 44 36 28 20 12 4  
 62 54 46 38 30 22 14 6  
 64 56 48 40 32 24 16 8  
 57 49 41 33 25 17 9 1

## chapter 4

59 51 43 35 27 19 11 3  
61 53 45 37 29 21 13 5  
63 55 47 39 31 23 15 7

Aceasta înseamnă bitul 58 din blocul de intrare va fi mutat în poziția 1, bitul din poziția 50 va fi mutat în poziția 2 iar bitul din poziția 7 va ajunge pe ultima poziție. The permuted input block is then the input to a complex key-dependent computation described below. The output of that computation, called the preoutput, is then subjected to the following permutation which is the inverse of the initial permutation:

**IP<sup>-1</sup>**

40 8 48 16 56 24 64 32  
39 7 47 15 55 23 63 31  
38 6 46 14 54 22 62 30  
37 5 45 13 53 21 61 29  
36 4 44 12 52 20 60 28  
35 3 43 11 51 19 59 27  
34 2 42 10 50 18 58 26  
33 1 41 9 49 17 57 25

Cu alte cuvinte, în urma aplicării algoritmului, bitul 40 al blocului de intrare devine primul bit în blocul de ieșire, bitul 8 devine al doilea bit din blocul de ieșire, etc., până ce bitul 25 din blocul de intrare devine ultimul bit din blocul de ieșire.

Calculul care folosește blocul de intrare permutat ca intrare pentru a produce blocul de pre-ieșire constă, cu excepția unui schimb final de blocuri, din 16 iterații ale calculului descris mai jos prin funcția de cifrare  $f$  care acționează pe două blocuri, unul de 32 de biți și unul de 48 de biți și produce un bloc de 32 de biți.

Să considerăm cei 64 de biți ai blocului de intrare la o iterație să constea dintr-un bloc  $L$  de 32 biți urmat de un bloc de 32 biți  $R$ . Folosind notația definită în introducere, blocul de intrare este atunci  $LR$ . Fie  $K$  un bloc de 48 de biți ales din cheia de 64 de biți. Atunci ieșirea  $L'R'$  a unei iterații cu intrarea  $LR$  este definită prin:

$$(1) L' = R$$

$$R' = L \wedge f(R, K)$$

unde  $\wedge$  denotă adunarea bit-cu-bit modulo 2.

După cum am specificat mai înainte, intrarea pentru prima iterație a calculului este blocul de intrare permutat. Dacă  $L'R'$  este ieșirea celei de-a 16 iterații, atunci  $R'L'$  este blocul de pre-ieșire. La fiecare iterație un bloc diferit  $K$  de biți ai cheii este ales din cheia de 64 de biți desemnată de  $KEY$ .

Utilizând notații noi putem descrie iterațiile calculului cu mai multe detalii. Fie  $KS$  funcția care are ca intrare un întreg  $n$  din domeniul 1 la 16 precum și un bloc  $KEY$  de 64 de biți iar ca ieșire are un bloc  $Kn$  de 48 de biți care este o selecție permutată a biților din  $KEY$ . Cu alte cuvinte

$$(2) Kn = KS(n, KEY)$$

unde  $Kn$  este determinat de 48 de biți aflați în poziții distincte din  $KEY$ .  $KS$  este numit programul de chei (key schedule) deoarece blocul  $K$  utilizat în iterația  $n$  din (1) este blocul  $Kn$  determinat în (2).

Ca și mai înainte, notăm cu  $LR$  blocul de intrare permutat. Fie  $L()$  și  $R()$  pe post de  $L$  and  $R$  iar  $L_n$  și  $R_n$  pe post de  $L'$  și  $R'$  în (1) atunci când  $L$  și  $R$  sunt respectiv  $L_{n-1}$  și  $R_{n-1}$  iar  $K$  este  $Kn$ ; adică atunci când  $n$  este în domeniul 1 - 16,

$$(3) L_n = R_{n-1}$$

$$R_n = L_{n-1} \wedge f(R_{n-1}, Kn)$$

Blocul de pre-ieșire este atunci  $R_{16}L_{16}$ .



Programul cheilor produce cei 16 ***Kn*** care sunt necesari algoritmului.

## 4.6 un exemplu

## 4.7 securitate și criptanaliză

Deși au fost publicate mai multe informații despre criptanaliza DES decât despre alt cifru de tip bloc, atacul cel mai practic până în prezent continuă să fie forța brută. Sunt cunoscute diverse proprietăți criptanalitice minore și sunt posibile trei atacuri teoretice care, deși au o complexitate teoretică mai mică decât un atac de forță brută, necesită o cantitate nerealistă de text sub formă de text cunoscut sau ales pentru a fi efectuat și nu reprezintă o problemă în practică.

### 4.7.1 atacurile de forță brută

Pentru orice cifru, cea mai de bază metodă de atac este forța brută - încercând pe rând toate cheile posibile. Lungimea cheii determină numărul de chei posibile și, prin urmare, fezabilitatea acestei abordări. În ceea ce privește DES, s-au ridicat întrebări cu privire la adecvarea dimensiunii cheii sale, încă înainte de a fi adoptată ca standard și a fost dimensiunea mică a cheii, mai degrabă decât criptanaliza teoretică, care a dictat necesitatea unui algoritm de înlocuire. Ca urmare a discuțiilor care au implicat consultanți externi, inclusiv NSA, dimensiunea cheii a fost redusă de la 128 biți la 56 biți pentru a se potrivi pe un singur cip.

Mașina DES Cracker de 250.000 USD a EFF conținea 1.856 de jetoane personalizate și putea forța o forță cheie DES în câteva zile.

În mediul academic, au fost avansate diverse propuneri pentru o mașină de spargere a DES. În 1977, Diffie și Hellman au propus o mașină care costă aproximativ 20 milioane USD, care ar putea găsi o cheie DES într-o singură zi. Până în 1993, Wiener a propus o mașină de căutare cu cheie cu un milion de dolari SUA, care va găsi o cheie în 7 ore. Cu toate acestea, niciuna dintre aceste propuneri timpurii nu a fost vreodată pusă în aplicare - sau, cel puțin, nicio implementare nu a fost recunoscută public. Vulnerabilitatea DES a fost practic demonstrată la sfârșitul anilor '90. În 1997, RSA Security a sponsorizat o serie de concursuri, oferind un premiu de 10.000 USD primei echipe care a spart un mesaj criptat cu DES pentru concurs. Acest concurs a fost câștigat de proiectul DESCHALL, condus de Rocke Verser, Matt Curtin și Justin Dolske, folosind ciclurile inactive ale mii de computere de pe Internet. Fezabilitatea spargerii DES a fost rapid demonstrată în 1998, când a fost construit un cracker DES personalizat de către Electronic Frontier Foundation (EFF), un grup de drepturi civile din spațiul cibernetic, cu un cost de aproximativ 250.000 USD (a se vedea crackerul EFF DES). Motivația lor a fost aceea de a arăta că DES poate fi spart atât în practică, cât și în teorie: "Există multe persoane care nu vor crede un adevăr până nu îl vor vedea cu propriii ochi. Arătându-le o mașină fizică care poate sparge DES în câteva zile este singura modalitate de a convinge unii oameni că nu pot avea încredere în securitatea lor pentru DES." Mașina a gădit o cheie prin forță brută într-un timp puțin mai mare de 2 zile.

Mașina COPACOBANA, construită în 2006 pentru 10.000 USD de către universitățile din Bochum și Kiel, Germania, conține 120 de FPGA low-cost și ar putea efectua o căutare cheie exhaustivă pe DES în 9 zile în medie.

Singurul alt cracker DES confirmat a fost aparatul COPACOBANA construit în 2006 de echipele universităților din Bochum și Kiel, ambele din Germania. Spre deosebire de aparatul EFF, COPACOBANA constă din circuite integrate, reconfigurabile disponibile comercial. 120 din aceste matrice de canale programabile Field (FPGA) de tip XILINX Spartan3-1000 rulează în paralel. Sunt grupate în 20 de module

## chapter 4

DIMM, fiecare conținând 6 FPGA. Utilizarea hardware-ului reconfigurabil face ca aparatul să se aplice și altor sarcini de rupere a codului. Unul dintre aspectele mai interesante ale COPACOBANA este factorul său de cost. O mașină poate fi construită pentru aproximativ 10.000 USD. Reducerea costurilor cu aproximativ un factor de 25 pentru mașina EFF este un exemplu impresionant pentru îmbunătățirea continuă a hardware-ului digital. Ajustarea inflației pe 8 ani produce o îmbunătățire și mai mare de aproximativ 30x. Începând cu 2007, SciEngines GmbH, o companie spin-off a celor doi parteneri de proiect ai COPACOBANA, a consolidat și dezvoltat succesorii COPACOBANA. În 2008, COPACOBANA RIVYERA a redus timpul pentru a sparge DES la mai puțin de o zi, folosind 128 Spartan-3 5000. [DESW]

### 4.7.2 atacuri mai rapide decât forța brută

Se cunosc trei atacuri care pot sparge cele șaisprezece runde ale DES cu mai puțină complexitate decât o căutare cu forța brută: criptanaliza diferențială (DC), criptanaliza liniară (LC) și atacul lui Davies. Cu toate acestea, atacurile sunt teoretice și sunt imposibil de montat în practică; aceste tipuri de atac sunt uneori denumite puncte slabe certificationale.

- Analiza criptografică diferențială a fost redescoperită la sfârșitul anilor 1980 de către Eli Biham și Adi Shamir; aceasta a fost cunoscută mai devreme atât de IBM, cât și de NSA și păstrată secretă. Pentru a sparge cele 16 runde complete, criptanaliza diferențială necesită 247 de texte alese. DES a fost proiectat să fie rezistent la DC.

- Criptanaliza liniară a fost descoperită de Mitsuru Matsui și are nevoie de 243 de texte cunoscute; metoda a fost implementată și a fost prima criptanaliză experimentală a DES care a fost raportată. Nu există dovezi că DES a fost adaptat pentru a fi rezistent la acest tip de atac. În 1994 a fost sugerată o generalizare a LC - criptanaliză liniară multiplă și a fost rafinată în continuare de Biryukov și colab. În 2004; Analiza lor sugerează că mai multe aproximări liniare ar putea fi utilizate pentru a reduce cerințele de date ale atacului cu cel puțin un factor de 4 (adică  $2^{41}$  în loc de  $2^{43}$ ). O reducere similară a complexității datelor poate fi obținută într-o variantă aleasă în textul criptanalizei liniare (Knudsen și Mathiassen, 2000). Junod (2001) a efectuat mai multe experimente pentru a determina complexitatea timpului real al criptanalizei liniare și a raportat că a fost ceva mai rapid decât a fost prevăzut, necesitând un timp echivalent cu  $2^{39}$ - $2^{41}$  evaluări DES.

- Atacul îmbunătățit al lui Davies: în timp ce criptanaliza liniară și diferențială sunt tehnici generale și pot fi aplicate la o serie de scheme, atacul lui Davies este o tehnică specializată pentru DES, sugerată prima dată de Donald Davies în anii optzeci și îmbunătățită de Biham și Biryukov în 1997. Cea mai puternică formă de atac necesită 250 de planuri cunoscute, are o complexitate de calcul de 250 și are o rată de succes de 51%.

Au fost, de asemenea, atacuri propuse împotriva versiunilor reduse ale cifrului, adică versiuni ale DES cu mai puțin de șaisprezece runde. O astfel de analiză oferă o perspectivă asupra cât de multe runde sunt necesare pentru siguranță și cât de multă „marjă de securitate” păstrează versiunea completă. Langanal și Hellman au propus criptanaliza liniară diferențială și combină criptanaliza liniară diferențială și liniară într-un singur atac. O versiune îmbunătățită a atacului poate sparge 9-round DES cu 215.8 planuri de text cunoscute și are o complexitate de timp 229.2 (Biham și colab., 2002). [DESW]

## 4.8 triplu DES

### 4.8.1 algoritmul

Triple DES utilizează un set de 3 chei, K1, K2 și K3, fiecare de 56 biți. Algoritmul de criptare este:

ciphertext = EK3(DK2(EK1(plaintext)))

I.e., DES criptează cu K1, DES *decriptează* cu K2, apoi DES criptează with K3.

Decriptarea are loc în sens invers:

plaintext = DK1(EK2(DK3(ciphertext)))

I.e., DES decriptează cu K3, DES *criptează* cu K2, apoi DES decriptează with K1.

Fiecare tripletă criptează un bloc de 64 de biți de date.

În fiecare caz, operația de la mijloc este inversul primei și ultimei. Acest lucru îmbunătățește puterea algoritmului atunci când utilizați opțiunea de cheie de tip 2 compatibilitatea cu DES cu opțiunea de cheie 3.

## 4.8.2 opțiuni de cheie

Standardele definesc trei opțiuni de cheie:

- Opțiunea de cheie 1: Toate cele trei taste sunt independente.
- Opțiunea de cheie 2: K1 și K2 sunt independente, iar K3 = K1.
- Opțiunea de cheie 3: Toate cele trei taste sunt identice, adică K1 = K2 = K3.

Opțiunea de cheie 1 este cea mai puternică, cu  $3 \times 56 = 168$  biți de cheie independenți.

Opțiunea de cheie 2 oferă mai puțină securitate, cu  $2 \times 56 = 112$  biți de cheie. Această opțiune este mai puternică decât criptarea DES de două ori, de ex. cu K1 și K2, deoarece protejează împotriva atacurilor de întâlnire la mijloc.

Opțiunea de cheie 3 este echivalentă cu DES, cu doar 56 de biți cheie. Această opțiune oferă o compatibilitate cu DES, deoarece prima și a doua operație DES se anulează. Nu mai este recomandată de Institutul Național de Standarde și Tehnologie (NIST) și nu este susținută de ISO / IEC 18033-3.

## chapter 5 advanced encryption standard

### 5.1 istoric AES

Standardul avansat de criptare (AES) este un standard de criptare adoptat de guvernul S.U.A. Standardul cuprinde trei algoritmi de cifrare de tip bloc, AES-128, AES-192 și AES-256, adoptați dintr-o colecție mai mare publicată inițial sub numele Rijndael. Fiecare cifru AES are un bloc de 128 biți, cu dimensiuni cheie de 128, 192 și, respectiv, 256 de biți. Algoritmii de cifrare AES au fost analizați pe scară largă și sunt acum utilizați la nivel mondial, cum a fost cazul și cu predecesorul său, Standardul de criptare a datelor (DES).

AES a fost anunțat de Institutul Național de Standarde și Tehnologie (NIST) ca US FIPS PUB 197 (FIPS 197) pe 26 noiembrie 2001, după un proces de standardizare de 5 ani în care au fost prezentate și evaluate cincisprezece proiecte concurente înainte de a fi selectat Rijndael ca cel mai potrivit (pentru detalii, consultați procesul de adoptare a Advanced Encryption Standard). A devenit efectiv standard guvernamental federal la 26 mai 2002, după aprobarea de către Secretarul de Comerț. Acesta este disponibil în multe pachete de criptare. AES este primul cifru public accesibil și deschis, aprobat de ANS pentru informații de cel mai înalt nivel de secretizare (a se vedea secțiunea "Securitatea AES", mai jos).

Cifrul Rijndael a fost elaborat de doi criptografi belgieni, Joan Daemen și Vincent Rijmen, și a fost prezentat de aceștia la procesul de selecție AES. Rijndael (pronunțat [reinda:ɪ]) este o combinație a numelor celor doi inventatori.

### 5.2 descriere generală

AES se bazează pe un principiu de proiectare cunoscut sub numele de rețea de permutare de substituție. Este rapid atât în software, cât și în hardware. Spre deosebire de predecesorul său, DES, AES nu utilizează o rețea Feistel.

AES are o dimensiune a blocului fixă de 128 biți și o cheie de lungime 128, 192 sau 256 de biți, deși Rijndael poate fi specificat cu blocuri și dimensiuni de cheie orice multiplu de 32 de biți, cu un minim de 128 biți și un maxim de 256 biți.

AES funcționează pe o matrice de octeți  $4 \times 4$ , denumită stare (versiunile lui Rijndael cu o dimensiune mai mare a blocului au coloane suplimentare în stare). Majoritatea calculelor AES se fac într-un câmp finit special.

Cifrul AES este specificat ca un număr de repetări ale rundelor de transformare care convertesc textul de intrare în rezultatul final al textului cifrat. Fiecare rundă constă din mai mulți pași de procesare, inclusiv unul care depinde de cheia de criptare. Se aplică un set de runde inverse pentru a transforma textul criptat înapoi în textul original, folosind aceeași cheie de criptare.

### 5.3 fundamente algebrice

#### 5.3.1 reprezentarea octeților

Unitatea de procesare din AES este octetul. Acest lucru este destul de diferit de DES, unde operațiile de bază au fost efectuate la nivel de biți. Un octet este o secvență de opt biți, numiți  $b_0, b_1, \dots, b_7$  -  $b_0$  fiind

cel mai puțin semnificativ. Fiecărui octet (sau secvență de opt biți) îi asociem un polinom de grad cel mult 7, cu coeficienți în  $Z_2$ , inelul de întregi modulo 2, după cum urmează. Dacă valoarea octetului nostru este  $b_7 * 2^7 + b_6 * 2^6 + b_5 * 2^5 + b_4 * 2^4 + b_3 * 2^3 + b_2 * 2^2 + b_1 * 2^1 + b_0 * 2^0$  cu toți coeficienții în  $Z_2$ , atunci polinomul asociat este  $b_7 * x^7 + b_6 * x^6 + \dots + b_2 * x^2 + b_1 * x + b_0$ .

De exemplu, {01100011} identifică elementul de câmp finit  $x^6 + x^5 + x + 1$ . În afară de reprezentările lor binare sau polinomiale, octeții vor fi de asemenea reprezentați în notație hexazecimală fie ca {63} (octetul de mai sus), fie ca 0x63.

Toți octeții din algoritmul AES sunt interpretați ca elemente de câmp finit. Elementele de câmp finite pot fi adăugate și multiplicare, dar aceste operații sunt diferite de cele utilizate pentru numere. Următoarele subsecțiuni introduc conceptele matematice de bază necesare pentru descrierea algoritmului.

### 5.3.2 adunarea

Adăugarea a două elemente într-un câmp finit se obține prin "adunarea" coeficienților pentru puterile corespunzătoare în polinoamele pentru cele două elemente. Adunarea se face cu operația XOR (notată cu  $\wedge$ ) - modulo 2 - astfel încât  $1 \wedge 1 = 0$ ,  $1 \wedge 0 = 1$  și  $0 \wedge 0 = 0$ .

În consecință, scăderea polinomială este identică cu adunarea polinomială.

Alternativ, adăugarea de elemente de câmp finit poate fi descrisă ca adăugarea modulo 2 a biților corespunzători în byte. Pentru doi octeți  $\{a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0\}$  and  $\{b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0\}$ , suma este

$\{c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0\}$ , unde pentru fiecare  $i$ ,  $c_i = a_i \wedge b_i$  (i.e.,  $c_7 = a_7 \wedge b_7$ ,  $c_6 = a_6 \wedge b_6$ , ...,  $c_0 = a_0 \wedge b_0$ ).

De exemplu, următoarele expresii sunt echivalente una cu cealaltă:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 \text{ (notație polinomială);}$$

$$\{01010111\} \wedge \{10000011\} = \{11010100\} \text{ (notație binară);}$$

$$\{57\} \wedge \{83\} = \{d4\} \text{ (notație hexazecimală).}$$

### 5.3.3 înmulțirea (produsul)

În reprezentarea polinomială, înmulțirea în  $GF(2^8)$  (notată cu  $\cdot$ ) corespunde cu înmulțirea polinoamelor modulo un polinom ireductibil de gradul 8. Un polinom este ireductibil dacă singurii săi divizori sunt 1 și el însuși. Pentru algoritmul AES, acest polinom ireductibil este

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

sau {01}{1b} în hexazecimal.

De exemplu,  $\{57\} \cdot \{83\} = \{c1\}$ , deoarece

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^7 +$$

$$x^7 + x^5 + x^3 + x^2 + x +$$

$$x^6 + x^4 + x^2 + x + 1$$

$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

și

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \text{ modulo } (x^8 + x^4 + x^3 + x + 1)$$

$$= x^7 + x^6 + 1.$$

Reducerea modulo  $m(x)$  ne asigură că rezultatul înmulțirii va fi întotdeauna un polinom de grad mai mic decât 8 și de aceea, acesta poate fi reprezentat de către un octet. Spre deosebire de adunare, nu există nicio

## chapter 5

operație simplă la nivelul octeților care corespunde acestei multiplicări.

Înmulțirea definită mai sus este asociativă, iar elementul  $\{01\}$  este identitatea la operația de înmulțire. Pentru orice polinom binar non-zero  $b(x)$  de grad mai mic de 8, inversul multiplicativ al  $b(x)$ , notat  $b^{-1}(x)$ , poate fi găsit după cum urmează: algoritmul euclidian extins este folosit pentru a calcula polinoamele  $a(x)$  și  $c(x)$  astfel încât

$$b(x)a(x) + m(x)c(x) = 1$$

Deci,  $a(x) \cdot b(x) \bmod m(x) = 1$ , ceea ce înseamnă

$$b^{-1}(x) = a(x) \bmod m(x)$$

Mai mult decât atât, pentru orice  $a(x)$ ,  $b(x)$  și  $c(x)$  din câmp, avem

$$a(x) \cdot (b(x) + c(x)) = a(x) \cdot b(x) + a(x) \cdot c(x).$$

Rezultă că setul de 256 valori posibile, cu XOR folosit ca adunare și multiplicarea definită ca mai sus, are structura câmpului finit  $GF(2^8)$ .

### 5.3.4 înmulțirea cu x

Multiplicarea unui polinom binar cu polinomul  $x$  rezultă în  $x \cdot b(x)$  și se obține prin reducerea rezultatului de mai sus modulo  $m(x)$ . Dacă  $b_7 = 0$ , rezultatul este deja în formă redusă. Dacă  $b_7 = 1$ , reducerea este realizată prin scădere (adică, prin XOR) a polinomului  $m(x)$ . Rezultă că multiplicarea cu  $x$  (adică,  $\{00000010\}$  sau  $\{02\}$ ) poate fi implementată la nivel de octet ca o deplasare la stânga și un XOR cu  $\{1b\}$ . Această operație pe octeți este notată cu  $xtime()$ .

Înmulțirea prin puteri mai mari a lui  $x$  poate fi implementată prin aplicarea repetată a operației  $xtime()$ .

Prin adăugarea unor rezultate intermediare, poate fi implementată înmulțirea cu orice constantă.

De exemplu,  $\{57\} \cdot \{13\} = \{fe\}$  deoarece

$$\{57\} \cdot \{02\} = xtime(\{57\}) = \{ae\}$$

$$\{57\} \cdot \{04\} = xtime(\{ae\}) = \{47\}$$

$$\{57\} \cdot \{08\} = xtime(\{47\}) = \{8e\}$$

$$\{57\} \cdot \{10\} = xtime(\{8e\}) = \{07\},$$

astfel,

$$\{57\} \cdot \{13\} = \{57\} \cdot (\{01\} \wedge \{02\} \wedge \{10\})$$

$$= \{57\} \wedge \{ae\} \wedge \{07\}$$

$$= \{fe\}.$$

## 5.4 programul cheilor

Cheia privată utilizată în procesul de criptare (care poate fi de 4, 6 sau 8 cuvinte de 32 de biți) este utilizată pentru generarea unui set de cuvinte care va fi utilizat în fiecare din rundele algoritmului pentru a fi adăugat (utilizând operația XOR) la cuvintele din matricea de stare. Primele  $N_k$  cuvinte din acest set sunt formate din chiar cheia privată. Restul sunt generate de ea, conform procedurii de mai jos.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
    word temp
    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))
        temp = w[i-1]

        if (i mod Nk = 0)
            temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if

        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end

```

Programul cheilor pentru decriptare este similar - subcheiile sunt în ordine inversă față de criptare. În afară de această schimbare, procesul este același ca și pentru criptare.

## 5.5 funcții utilizate

### 5.5.1 funcția SubBytes()

În pasul `SubBytes()`, fiecare octet din stare este înlocuit cu intrarea acestuia dintr-un tabel de căutare fix pe 8 biți,  $S$ ;  $b_{ij} = S(a_{ij})$ .

În pasul `SubBytes()`, fiecare octet din matrice este actualizat utilizând o cutie de substituție pe 8 biți, caseta S-Rijndael. Această operație asigură non-liniaritatea în cifru. S-boxul folosit este derivat din inversul multiplicativ peste  $GF(2^8)$ , cunoscut ca având proprietăți bune de neliniaritate. Pentru a evita atacurile bazate pe proprietăți simple algebrice, S-box-ul este construit prin combinarea funcției inverse cu o transformare afină inversabilă. S-box-ul este, de asemenea, ales pentru a evita orice puncte fixe (și astfel poate fi considerat un deranjament), precum și orice puncte fixe opuse.

### 5.5.2 funcția ShiftRows()

În pasul `ShiftRows`, octeții din fiecare linie al stării sunt deplasați ciclic spre stânga. Numărul de locuri pe care fiecare octet este mutat diferă pentru fiecare rând.

Pasul `ShiftRows` funcționează pe liniile stării; deplasează (șiftează) ciclic octeții din fiecare rând (linie) cu un anumit număr de poziții. Pentru AES, primul rând este lăsat neschimbat. Fiecare octet al celui de-al doilea rând este deplasat cu o poziție spre stânga. În mod similar, rândurile al treilea și al patrulea sunt deplasate cu două și, respectiv, trei poziții. Pentru blocul de mărimi de 128 biți și 192 de biți, modelul de schimbare este același. În acest fel, fiecare coloană a stării de ieșire a pasului `ShiftRows` este compusă din octeți din fiecare coloană a stării de intrare. (Variantele Rijndael cu o dimensiune mai mare a blocului au offseturi ușor diferite). În cazul blocului de 256 de biți, primul rând este neschimbat, iar schimbarea pentru al doilea, al treilea și al patrulea rând este de 1 octet, 3 octeți și respectiv 4 octeți - această modificare se aplică numai cifrului Rijndael când este utilizat cu un bloc de 256 de biți, deoarece AES nu utilizează blocuri de 256 biți.

### 5.5.3 funcția MixColumns()

În pasul `MixColumns`, fiecare coloană a stării se înmulțește cu un polinom fix  $c(x)$ .

În pasul `MixColumns`, cei patru octeți din fiecare coloană a stării sunt combinați folosind o transformare liniară inversabilă. Funcția `MixColumns` are patru octeți ca intrări și ieșiri de patru octeți, în care fiecare octet de intrare afectează toți cei patru octeți de ieșire. Împreună cu `ShiftRows`, `MixColumns` oferă difuzie în cifru. Fiecare coloană este tratată ca un polinom peste  $GF(2^8)$  și apoi este multiplicată modulo  $x^4 + 1$  cu un polinom fix  $c(x) = 0x03x^3 + x^2 + x + 0x02$ . (Coeficienții sunt afișați în echivalentul lor hexazecimal al reprezentării binare a polinomialor de biți din  $GF(2)[x]$ .) Pasul `MixColumns` poate fi de asemenea vizualizat ca o multiplicare de către o matrice specială MDS în câmpul finit.

### 5.5.4 funcția AddRoundKey()

În pasul `AddRoundKey`, fiecare octet al stării este combinat cu un octet al subcheiei de rundă utilizând operația XOR ( $\oplus$ ).

În pasul `AddRoundKey`, subcheia este combinată cu matrice de stare. Pentru fiecare rundă, o subcheie este derivată din cheia principală utilizând programul de chei al algoritmului Rijndael; fiecare subcheie are aceeași dimensiune ca și matricea de stare. Subcheia este adăugată prin combinarea fiecărui octet al stării cu octetul corespunzător al subcheiei folosind operația XOR la nivel de bit.

## 5.6 pseudo-cod

```

Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]
  state = in
  AddRoundKey(state, w[0, Nb-1])

  for round = 1 step 1 to Nr-1
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
  end for

  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  out = state
end

```

## 5.7 funcțiile inverse

Funcțiile prezentate în paragrafele precedente pot fi inversate și aplicate în ordine inversă, permițând astfel decriptarea mesajului cifrat, având ca rezultat mesajul inițial.

Continuăm cu o scurtă descriere a celor patru funcții inverse. Pentru mai multe detalii, consultați FIPS 197 - <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>.



- `InvShiftRows` – liniile (rândurile) din matricea de stare sunt șifrate (deplasate) spre dreapta cu același număr de octeți, spre deosebire de funcția inițială, unde deplasarea se făcea la stânga.
- `InvSubBytes` – funcția folosește caseta de substituție care este inversa casetei de substituție a funcției `SubBytes`
- `InvMixColumns` – polinomul utilizat pentru multiplicare este  $0x0bx^3 + 0x0dx^2 + 0x09x + 0x0e$
- `AddRoundKey` – este chiar funcția inițială, deoarece este propria sa inversă

## 5.8 pseudo-cod pentru decriptare

```

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

  for round = Nr-1 step -1 downto 1
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    InvMixColumns(state)
  end for

  InvShiftRows(state)
  InvSubBytes(state)
  AddRoundKey(state, w[0, Nb-1])
  out = state
end

```

## 5.9 pseudo-cod echivalent pentru decriptare

Algoritmul AES are două proprietăți care permit elaborarea unei versiuni echivalente pentru pseudo-codul procedurii de decriptare. Aceste proprietăți sunt:

- funcțiile `SubBytes` și `ShiftRows` comută, la felși inversele lor, `InvSubBytes` și `InvShiftRows`
- Operațiile de mixare a coloanelor – `MixColumns` și `InvMixColumns` - sunt liniare în raport cu intrările de la coloane

Aceste proprietăți conduc la o versiune mai eficientă a procedurii de decriptare. Listăm mai jos pseudo-codul pentru procedura echivalentă de decriptare:

```

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

  for round = Nr-1 step -1 downto 1
    InvSubBytes(state)
    InvShiftRows(state)
    InvMixColumns(state)
    AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
  end for

```

## chapter 5

```
end for

InvSubBytes(state)
InvShiftRows(state)
AddRoundKey(state, dw[0, Nb-1])
out = state
end
```

Matricea `dw[]` care conține programul de chei se obține adăugînd pseudo-codul următor la sfârșitul procedurii de program al cheilor:

```
for i = 0 step 1 to (Nr+1)*Nb-1
  dw[i] = w[i]
end for

for round = 1 step 1 to Nr-1
  InvMixColumns(dw[round*Nb, (round+1)*Nb-1])
end for
```

### 5.10 un exemplu

```
PLAINTEXT:    00112233445566778899aabbccddeeff
KEY:          000102030405060708090a0b0c0d0e0f
```

CIPHER (ENCRYPT):

```
round[ 0].input 00112233445566778899aabbccddeeff
round[ 0].k_sch 000102030405060708090a0b0c0d0e0f
round[ 1].start 00102030405060708090a0b0c0d0e0f0
round[ 1].s_box 63cab7040953d051cd60e0e7ba70e18c
round[ 1].s_row 6353e08c0960e104cd70b751bacad0e7
round[ 1].m_col 5f72641557f5bc92f7be3b291db9f91a
round[ 1].k_sch d6aa74fdd2af72fadaa678f1d6ab76fe
round[ 2].start 89d810e8855ace682d1843d8cb128fe4
round[ 2].s_box a761ca9b97be8b45d8ad1a611fc97369
round[ 2].s_row a7be1a6997ad739bd8c9ca451f618b61
round[ 2].m_col ff87968431d86a51645151fa773ad009
round[ 2].k_sch b692cf0b643dbdf1be9bc5006830b3fe
round[ 3].start 4915598f55e5d7a0daca94fa1f0a63f7
round[ 3].s_box 3b59cb73fcd90ee05774222dc067fb68
round[ 3].s_row 3bd92268fc74fb735767cbe0c0590e2d
round[ 3].m_col 4c9c1e66f771f0762c3f868e534df256
round[ 3].k_sch b6ff744ed2c2c9bf6c590cbf0469bf41
round[ 4].start fa636a2825b339c940668a3157244d17
round[ 4].s_box 2dfb02343f6d12dd09337ec75b36e3f0
round[ 4].s_row 2d6d7ef03f33e334093602dd5bfb12c7
round[ 4].m_col 6385b79ffc538df997be478e7547d691
round[ 4].k_sch 47f7f7bc95353e03f96c32bcfd058dfd
round[ 5].start 247240236966b3fa6ed2753288425b6c
round[ 5].s_box 36400926f9336d2d9fb59d23c42c3950
round[ 5].s_row 36339d50f9b539269f2c092dc4406d23
round[ 5].m_col f4bcd45432e554d075f1d6c51dd03b3c
round[ 5].k_sch 3caaa3e8a99f9deb50f3af57adf622aa
round[ 6].start c81677bc9b7ac93b25027992b0261996
round[ 6].s_box e847f56514dadde23f77b64fe7f7d490
```

```

round[ 6].s_row e8dab6901477d4653ff7f5e2e747dd4f
round[ 6].m_col 9816ee7400f87f556b2c049c8e5ad036
round[ 6].k_sch 5e390f7df7a69296a7553dc10aa31f6b
round[ 7].start c62fe109f75eedc3cc79395d84f9cf5d
round[ 7].s_box b415f8016858552e4bb6124c5f998a4c
round[ 7].s_row b458124c68b68a014b99f82e5f15554c
round[ 7].m_col c57e1c159a9bd286f05f4be098c63439
round[ 7].k_sch 14f9701ae35fe28c440adf4d4ea9c026
round[ 8].start d1876c0f79c4300ab45594add66ff41f
round[ 8].s_box 3e175076b61c04678dfc2295f6a8bfc0
round[ 8].s_row 3e1c22c0b6fcbf768da85067f6170495
round[ 8].m_col baa03de7a1f9b56ed5512cba5f414d23
round[ 8].k_sch 47438735a41c65b9e016baf4aebf7ad2
round[ 9].start fde3bad205e5d0d73547964ef1fe37f1
round[ 9].s_box 5411f4b56bd9700e96a0902fa1bb9aa1
round[ 9].s_row 54d990a16ba09ab596bbf40ea111702f
round[ 9].m_col e9f74eec023020f61bf2ccf2353c21c7
round[ 9].k_sch 549932d1f08557681093ed9cbe2c974e
round[10].start bd6e7c3df2b5779e0b61216e8b10b689
round[10].s_box 7a9f102789d5f50b2beffd9f3dca4ea7
round[10].s_row 7ad5fda789ef4e272bca100b3d9ff59f
round[10].k_sch 13111d7fe3944a17f307a78b4d2b30c5
round[10].output 69c4e0d86a7b0430d8cdb78070b4c55a

```

INVERSE CIPHER (DECRYPT):

```

round[ 0].iinput 69c4e0d86a7b0430d8cdb78070b4c55a
round[ 0].ik_sch 13111d7fe3944a17f307a78b4d2b30c5
round[ 1].istart 7ad5fda789ef4e272bca100b3d9ff59f
round[ 1].is_row 7a9f102789d5f50b2beffd9f3dca4ea7
round[ 1].is_box bd6e7c3df2b5779e0b61216e8b10b689
round[ 1].ik_sch 549932d1f08557681093ed9cbe2c974e
round[ 1].ik_add e9f74eec023020f61bf2ccf2353c21c7
round[ 2].istart 54d990a16ba09ab596bbf40ea111702f
round[ 2].is_row 5411f4b56bd9700e96a0902fa1bb9aa1
round[ 2].is_box fde3bad205e5d0d73547964ef1fe37f1
round[ 2].ik_sch 47438735a41c65b9e016baf4aebf7ad2
round[ 2].ik_add baa03de7a1f9b56ed5512cba5f414d23
round[ 3].istart 3e1c22c0b6fcbf768da85067f6170495
round[ 3].is_row 3e175076b61c04678dfc2295f6a8bfc0
round[ 3].is_box d1876c0f79c4300ab45594add66ff41f
round[ 3].ik_sch 14f9701ae35fe28c440adf4d4ea9c026
round[ 3].ik_add c57e1c159a9bd286f05f4be098c63439
round[ 4].istart b458124c68b68a014b99f82e5f15554c
round[ 4].is_row b415f8016858552e4bb6124c5f998a4c
round[ 4].is_box c62fe109f75eedc3cc79395d84f9cf5d
round[ 4].ik_sch 5e390f7df7a69296a7553dc10aa31f6b
round[ 4].ik_add 9816ee7400f87f556b2c049c8e5ad036
round[ 5].istart e8dab6901477d4653ff7f5e2e747dd4f
round[ 5].is_row e847f56514dadde23f77b64fe7f7d490
round[ 5].is_box c81677bc9b7ac93b25027992b0261996
round[ 5].ik_sch 3caaa3e8a99f9deb50f3af57adf622aa
round[ 5].ik_add f4bcd45432e554d075f1d6c51dd03b3c
round[ 6].istart 36339d50f9b539269f2c092dc4406d23
round[ 6].is_row 36400926f9336d2d9fb59d23c42c3950
round[ 6].is_box 247240236966b3fa6ed2753288425b6c
round[ 6].ik_sch 47f7f7bc95353e03f96c32bcfd058dfd
round[ 6].ik_add 6385b79ffc538df997be478e7547d691
round[ 7].istart 2d6d7ef03f33e334093602dd5bfb12c7

```

## chapter 5

```
round[ 7].is_row 2dfb02343f6d12dd09337ec75b36e3f0
round[ 7].is_box fa636a2825b339c940668a3157244d17
round[ 7].ik_sch b6ff744ed2c2c9bf6c590cbf0469bf41
round[ 7].ik_add 4c9c1e66f771f0762c3f868e534df256
round[ 8].istart 3bd92268fc74fb735767cbe0c0590e2d
round[ 8].is_row 3b59cb73fcd90ee05774222dc067fb68
round[ 8].is_box 4915598f55e5d7a0daca94fa1f0a63f7
round[ 8].ik_sch b692cf0b643dbdf1be9bc5006830b3fe
round[ 8].ik_add ff87968431d86a51645151fa773ad009
round[ 9].istart a7be1a6997ad739bd8c9ca451f618b61
round[ 9].is_row a761ca9b97be8b45d8ad1a611fc97369
round[ 9].is_box 89d810e8855ace682d1843d8cb128fe4
round[ 9].ik_sch d6aa74fdd2af72fadaa678f1d6ab76fe
round[ 9].ik_add 5f72641557f5bc92f7be3b291db9f91a
round[10].istart 6353e08c0960e104cd70b751bacad0e7
round[10].is_row 63cab7040953d051cd60e0e7ba70e18c
round[10].is_box 00102030405060708090a0b0c0d0e0f0
round[10].ik_sch 000102030405060708090a0b0c0d0e0f
round[10].i_outp 00112233445566778899aabbccddeeff
```

### 5.11 atacuri de tip canal lateral (side-channel)

Până în mai 2009, singurele atacuri publicate cu succes împotriva algoritmului complet AES au fost atacurile canalului lateral (side-channel) asupra unor implementări specifice. Agenția Națională de Securitate (NSA) a revizuit toți finaliștii AES, inclusiv Rijndael, și a declarat că toți au fost suficient de siguri pentru datele ne-clasificate ale guvernului Statelor Unite. În iunie 2003, guvernul american a anunțat că AES poate fi utilizat pentru a proteja informațiile clasificate:

"Designul și rezistența tuturor lungimilor cheie ale algoritmului AES (adică 128, 192 și 256) sunt suficiente pentru a proteja informațiile clasificate până la nivelul SECRET. Informațiile TOP SECRET vor necesita utilizarea cheilor de lungime de 192 sau 256 de biți. Implementarea AES în produsele destinate să protejeze sistemele și / sau informațiile naționale de securitate trebuie revizuită și certificată de NSA înainte de achiziționarea și utilizarea acestora."

AES are 10 runde pentru chei de 128 biți, 12 runde pentru chei de 192 de biți și 14 runde pentru chei de 256 de biți. Până în 2006, cele mai cunoscute atacuri au fost cele 7 runde pentru chei de 128 biți, 8 runde pentru chei de 192 biți și 9 runde pentru chei de 256 de biți.

Pentru criptografi, o "spargere" criptografică este ceva mai rapid decât o căutare exhaustivă. Astfel, un atac XSL împotriva unui AES pe 128 de biți care necesită  $2^{100}$  de operații (în comparație cu  $2^{128}$  de chei posibile) ar fi considerat o spargere. Cel mai mare atac de forță brute, cunoscut în public, a fost împotriva unei chei RC5 de 64 de biți de către distribuite.net.

Spre deosebire de majoritatea celorlalte blocuri de cifru, AES are o descriere algebrică foarte curată. În 2002, un atac teoretic, denumit "atac XSL", a fost anunțat de Nicolas Courtois și Josef Pieprzyk, pretinzând că prezintă o slăbiciune a algoritmului AES datorită descrierii sale simple. De atunci, alte documente au arătat că atacul prezentat inițial nu este posibil; a se vedea atacul XSL asupra cifrurilor de tip bloc.

În timpul procesului de adoptare a AES, dezvoltatorii de algoritmi concurenți au scris despre Rijndael: "... suntem preocupați de utilizarea sa în aplicații critice de securitate". Cu toate acestea, la finalul procesului AES, Bruce Schneier, un dezvoltator al algoritmului competitiv Twofish, a scris că, deși el credea că atacurile academice reușite asupra lui Rijndael ar putea fi dezvoltate într-o zi, "Nu cred că cineva va descoperi vreodată un atac care permite cuiva să citească traficul de la Rijndael. "

Pe 1 iulie 2009, Bruce Schneier a dezvăluit un atac cu privire la versiunile de 192 biți și 256 de biți ale AES descoperite de Alex Biryukov și Dmitry Khovratovich; atacul cheie asociat cu versiunea de 256 de biți a AES exploatează simplitatea programei de chei a programului AES și are o complexitate de  $2^{119}$ . Acesta este o continuare a unui atac descoperit anterior în 2009 de Alex Biryukov, Dmitry Khovratovich și Ivica Nikolic, cu o complexitate de  $2^{96}$  pentru una din fiecare  $2^{35}$  chei.

Un alt atac a fost anunțat pe blog de Bruce Schneier la 30 iulie 2009 și publicat pe 3 august 2009. Acest nou atac de către Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich și Adi Shamir este împotriva lui AES-256 care folosește doar două chei asociate și timp de  $2^{39}$  pentru a recupera cheia completă de 256 biți a unei versiuni cu 9 runde sau timp de  $2^{45}$  pentru o versiune cu 10 runde, cu un tip mai puternic de atac de subcheie asociat, sau cu timp de  $2^{70}$  pentru o versiune cu 11 runde. AES cu chei de 256-biți folosește 14 runde, deci aceste atacuri nu sunt eficiente împotriva AES complet.

În noiembrie 2009 a fost publicat primul atac împotriva versiunii cu 8 runde a lui AES-128. Acest atac distinctiv cu cheie cunoscută este o îmbunătățire a reculului sau a atacurilor de la început de la mijloc pentru permutările asemănătoare AES, care văd două runde consecutive de permutare ca fiind aplicarea așa-numitei Super-Box. Funcționează pe versiunea cu 8 runde a modelului AES-128, cu o complexitate de calcul de  $2^{48}$  și o complexitate de memorie de  $2^{32}$ .



## chapter 7 algoritmul diffie-hellman de schimb chei

### 7.1 istoria protocolului

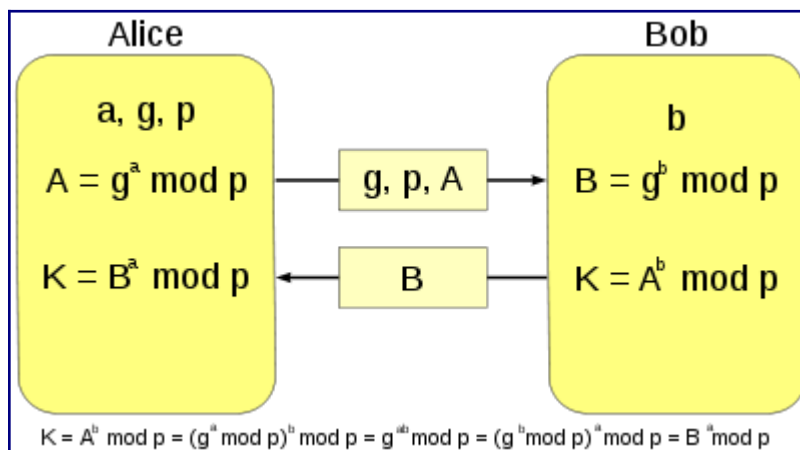
Algoritmul de schimb al cheilor Diffie-Hellman-Merkle a fost inventat în anul 1976 în timpul unei colaborări între Whitfield Diffie și Martin Hellman, aceasta fiind prima metodă practică pentru stabilirea unui secret comun peste un canal de comunicații neprotejat. Munca lui Ralph Merkle în distribuirea cheilor publice a fost o influență. John Gill a sugerat aplicarea problemei logaritmului discret. Prima dată a fost inventat de Malcolm Williamson de la GCHQ în Marea Britanie, cu câțiva ani înainte, dar GCHQ a ales să nu îl facă public până în anul 1997, dar atunci nu a avut nici o influență în domeniul academic de cercetare.

Metoda care a urmat în scurt timp a fost RSA, o altă implementare a criptografiei cu cheie publică ce folosește algoritmi asimetrici. Patentul U.S. 4,200,770, acum expirat, descrie algoritmul și îi acreditează pe Hellman, Diffie și Merkle ca inventatori

### 7.2 descriere

Diffie-Hellman stabilește un secret comun care poate fi folosit în comunicare secretă prin schimbul de date peste o rețea publică.

Exemplu care include elementele matematice folosite în această criptare:



Schimbul de chei Diffie-Hellman

Cea mai simplă și originală implementare a protocolului care folosește grupul multiplicativ al numerelor întregi modulo  $p$ , unde  $p$  este prim și  $g$  este o rădăcină primitivă mod  $p$ . Exemplu de schimb, cu valorile publice în verde și valorile secrete în roșu:

	ALICE			BOB		
Secret	Public	Calcul		Calcul	Public	Secret
	p, g				p, g	

## chapter 7

a						b
		$g^a \pmod p$				
				$g^b \pmod p$		
	$(g^a)^b \pmod p$		→		$(g^b)^a \pmod p$	
			←			

- Alice și Bob cad de acord să folosească numărul prim  $p=23$  și baza  $g=5$ .
- Alice alege un număr întreg secret  $a=6$ , iar apoi îl trimite lui Bob  $A = g^a \pmod p$ 
  - $A = 5^6 \pmod{23} = 8$ .
- Bob alege un număr întreg secret  $b=15$ , iar apoi îl trimite lui Alice  $B = g^b \pmod p$ 
  - $B = 5^{15} \pmod{23} = 19$ .
- Alice calculează  $s = B^a \pmod p$ 
  - $19^6 \pmod{23} = 2$ .
- Bob calculează  $s = A^b \pmod p$ 
  - $8^{15} \pmod{23} = 2$ .

Afît Alice cît și Bob au ajuns la aceeași valoare, pentru că  $g^{ab}$  and  $g^{ba}$  sunt egale mod  $p$ . De remarcat că doar  $a$ ,  $b$  și  $g^{ab} = g^{ba} \pmod p$  sunt secrete. Toate celelalte valori –  $p$ ,  $g$ ,  $g^a \pmod p$ , și  $g^b \pmod p$  – nu sunt secrete. Atunci când Alice și Bob calculează cheia secretă comună, ei o pot folosi ca cheie de criptare, cunoscută doar de ei, pentru transmiterea de mesaje peste un canal de comunicații deschis. Desigur, trebuie folosite valori mult mai mari pentru  $a$ ,  $b$  și  $p$  pentru ca acest exemplu să fie securizat, pentru că este ușor să se încerce toate valorile posibile  $g^{ab} \pmod{23}$  (sunt doar 22 de valori, chiar dacă  $a$  și  $b$  sunt mari). Dacă  $p$  ar fi un număr prim de cel puțin 300 de cifre, iar  $a$  și  $b$  de cel puțin 100 cifre, atunci nici cei mai buni algoritmi cunoscuți în ziua de azi nu ar putea găsi  $a$  dacă se cunoaște doar  $g$ ,  $p$ , și  $g^a \pmod p$ , chiar dacă se folosește toată puterea de calcul a întregii omeniri.

Problema este cunoscută ca problema logaritmului discret. Observați că nu este nevoie ca  $g$  să fie mare și în practică este 2 sau 5.

O descriere generală a protocolului:

- Alice și Bob aleg în comun un grup ciclic finit  $G$  și un element generator  $g$  în  $G$ . (Acest lucru se face de obicei cu mult înaintea restului de pași ai protocolului;  $g$  se presupune a fi cunoscut de toți atacatorii). Noi vom scrie grupul  $G$  multiplicativ.
- Alice alege un număr natural aleatoriu  $a$  și trimite  $g^a$  lui Bob.
- Bob alege un număr natural aleatoriu  $b$  și trimite  $g^b$  lui Alice.
- Alice calculează  $(g^b)^a$ .
- Bob calculează  $(g^a)^b$ .

Ambii Alice și Bob sunt acum în posesia unui element al grupului,  $g^{ab}$ , care pot servi ca și cheie secretă comună. Valorile lui  $(g^b)^a$  și  $(g^a)^b$  sunt egale deoarece ridicarea la putere este asociativă.



## 7.3 chart

Exemplu de diagramă care ajută la prezentarea a cine știe ce informație. (Eve este o persoană care „trage cu urechea” – ea urmărește ce este trimis între Alice și Bob, dar Eve nu poate altera conținutul informațiilor aflate pe parcursul acestei comunicări).

- Fie  $s$  = cheia secretă comună.  $s = 2$
- Fie  $a$  = cheia privată a lui Alice.  $a = 6$
- Fie  $A$  = cheia publică a lui Alice.  $A = g^a \text{ mod } p = 8$
- Fie  $b$  = cheia privată a lui Bob.  $b = 15$
- Fie  $B$  = cheia publică a lui Bob.  $B = g^b \text{ mod } p = 19$
- Fie  $g$  = baza publică.  $g=5$
- Fie  $p$  = număr prim public.  $p = 23$

<b>Alice</b>		<b>Eve</b>		<b>Bob</b>	
cunoaște	nu cunoaște	cunoaște	nu cunoaște	cunoaște	nu cunoaște
$p = 23$	$b = 15$	$p = 23$	$a = 6$	$p = 23$	$a = 6$
base $g = 5$		base $g = 5$	$b = 15$	base $g = 5$	
$a = 6$			$s = 2$	$b = 15$	
$A = 5^6 \text{ mod } 23 = 8$		$A = 5^a \text{ mod } 23 = 8$		$B = 5^{15} \text{ mod } 23 = 19$	
$B = 5^b \text{ mod } 23 = 19$		$B = 5^b \text{ mod } 23 = 19$		$A = 5^a \text{ mod } 23 = 8$	
$s = 19^6 \text{ mod } 23 = 2$		$s = 19^a \text{ mod } 23 = 2$		$s = 8^{15} \text{ mod } 23 = 2$	
$s = 8^b \text{ mod } 23 = 2$		$s = 8^b \text{ mod } 23 = 2$		$s = 19^a \text{ mod } 23 = 2$	
$s = 19^6 \text{ mod } 23 = 8^b \text{ mod } 23$		$s = 19^a \text{ mod } 23 = 8^b \text{ mod } 23$		$s = 8^{15} \text{ mod } 23 = 19^a \text{ mod } 23$	
$s = 2$				$s = 2$	

Notă: Ar trebui să fie dificil pentru Alice să rezolve cheia privată a lui Bob sau pentru Bob să rezolve cheia privată a lui Alice. Dacă nu ar fi dificile cazurile acestea, Eve ar putea foarte ușor să substituie cu perechea de cheie să publică / privată, ea poate pune cheia publică a lui Bob în cheia sa privată, produce o cheie secretă comună falsă, și să rezolve cheia privată a lui Bob (și să o folosească pentru a rezolva cheia secretă comună. Eve ar putea încerca să aleagă o pereche de cheie publică / privată care îi va ușura calcularea cheii private a lui Bob).

## 7.4 securitate

Protocolul este considerat securizat / sigur împotriva celor care ar putea „asculta” dacă  $G$  și  $g$  sunt aleși cum se cuvine. Acești „ascultători” (Eve) ar trebui să rezolve problema Diffie-Hellman de a obține  $g^{ab}$ . Acest lucru este deocamdată considerată dificil. Un algoritm eficient de a rezolva problema logaritmului discret ar ușura calcularea a sau  $b$  și ar rezolva problema Diffie-Hellman, făcând acest și multe alte criptosisteme cu chei publice nesigure.

## chapter 7

Ordinul lui  $G$  ar trebui să fie prim sau să aibă un factor prim larg pentru a preveni folosirea algoritmului Pohlig-Hellman pentru obținerea lui  $a$  sau  $b$ . Pentru acest motiv, un prim  $q$  de tip Sophie-Germain este câteodată folosit la calcularea lui  $p=2q+1$ , considerat a fi un prim sigur, deoarece ordinul lui  $G$  este atunci doar divizibil cu  $2$  și  $q$ .  $g$  este câteodată ales pentru a genera subgrupul de ordin  $q$  al lui  $G$ , mai degrabă decât  $G$ , astfel încât simbolul Legendre a lui  $g^a$  niciodată nu dezvăluie bit-ul de ordin mic a lui  $a$ .

Dacă Alice și Bob folosesc generatoare de numere aleatorii al căror output nu este complet aleatoriu și poate fi prezis într-o oarecare măsură, atunci munca lui Eve este mult mai ușoară.

Numerele întregi secrete  $a$  și  $b$  sunt înlăturate la sfârșitul sesiunii. Așadar schimbul de chei Diffie-Hellman realizează în mod trivial „secretul direcționat” perfect deoarece nici un material cu cheie privată de lungă durată nu există pentru a putea fi dezvăluit.

## 7.5 autentificare

În descrierea originală, schimbul Diffie-Hellman-Merkel propriu-zis nu oferă autentificarea pentru persoanele care comunică, ceea ce îl face vulnerabil la atacuri man-in-the-middle. O persoană în mijloc poate iniția două schimbări de chei Diffie-Hellman diferite, una cu Alice și alta cu Bob, mascându-se eficient în Alice sau Bob, și vice versa, permițând atacatorului să decripteze (și să citească sau să stocheze) și după să creeze din nou mesajele trimise între ei. O metodă pentru autentificarea comunicării dintre persoane este, în general, necesară pentru a preveni acest tip de atac

O varietate de soluții de autentificare criptografice includ un schimb Diffie-Hellman. Când Alice și Bob au o infrastructură pentru cheia publică, ei pot să semneze digital cheia la care au căzut de acord, sau  $g^a$  și  $g^b$ , precum în componentele MQV, STS și IKE din protocolul IPsec folosite pentru securizarea comunicațiilor din Protocolul de Internet (IP).

Când Alice și Bob împart o parolă, ei pot folosi un acord de cheie autentificată prin parolă din Diffie-Hellman, cum este cel descris în recomandarea ITU-T X.1035, care este folosită de către standardul de rețea de domiciliu G.hn.

## chapter 8 criptare asimetrică - RSA

În criptografie, RSA este un algoritm pentru criptografia cu chei publice. A fost primul algoritm cunoscut a fi potrivit pentru semnare și criptare și unul dintre primele mari progrese în criptografia cu chei publice. RSA este utilizat pe scară largă în protocoalele de comerț electronic și se crede că este sigur, dacă se utilizează chei suficient de lungi și utilizarea implementărilor actualizate.

Criptografia cu cheie publică, cunoscută și sub denumirea de criptografie asimetrică, este o formă de criptografie în care un utilizator are o pereche de chei criptografice - o cheie publică și o cheie privată. Cheia privată este păstrată secretă, în timp ce cheia publică poate fi distribuită pe scară largă. Cheile sunt legate matematic, dar cheia privată nu poate fi practic derivată din cheia publică. Un mesaj criptat cu cheia publică poate fi decriptat doar cu cheia privată corespunzătoare.

În schimb, criptografia cu chei secrete, cunoscută și sub denumirea de criptografie simetrică, folosește o singură cheie secretă atât pentru criptare cât și pentru decriptare.

### 8.1 istoric

The algorithm was publicly described in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT; the letters RSA are the initials of their surnames.

Clifford Cocks, a British mathematician working for the UK intelligence agency GCHQ, described an equivalent system in an internal document in 1973, but given the relatively expensive computers needed to implement it at the time, it was mostly considered a curiosity and, as far as is publicly known, was never deployed. His discovery, however, was not revealed until 1997 due to its top-secret classification, and Rivest, Shamir, and Adleman devised RSA independently of Cocks' work.

MIT was granted US patent 4405829 for a "Cryptographic communications system and method" that used the algorithm in 1983. The patent expired on 21 September 2000. Since a paper describing the algorithm had been published in August 1977, prior to the December 1977 filing date of the patent application, regulations in much of the rest of the world precluded patents elsewhere and only the US patent was granted. Had Cocks' work been publicly known, a patent in the US might not have been possible either.

### 8.2 operation

RSA involves a public key and a private key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key. The keys for the RSA algorithm are generated the following way:

- Choose two distinct large random prime numbers  $p$  and  $q$
- Compute  $n = p \cdot q$
- $n$  is used as the modulus for both the public and private keys
- Compute the totient:  $\phi(n) = (p - 1)(q - 1)$ . (The totient  $\phi(n)$  of a positive integer  $n$  is defined to be the number of positive integers less than or equal to  $n$  that are coprime to  $n$ . For example,  $\phi(9) = 6$  since the six numbers 1, 2, 4, 5, 7 and 8 are coprime to 9)
- Choose an integer  $e$  such that  $1 < e < \phi(n)$ , and  $e$  and  $\phi(n)$  share no factors other than 1 (i.e.  $e$  and  $\phi(n)$  are coprime)
- $e$  is released as the public key exponent

## chapter 8

- Compute  $d$  to satisfy the congruence relation  $de \equiv 1 \pmod{\phi(n)}$ ; i.e.  $de = 1 + k\phi(n)$  for some integer  $k$ .
- $d$  is kept as the private key exponent

Notes on the above steps:

- Step 1: Numbers can be probabilistically tested for primality.
- Step 3: changed in PKCS#1 v2.0 Public Key Cryptography Standards to  $\lambda(n) = \text{lcm}(p-1, q-1)$ , where lcm is the least common multiple, instead of  $\phi(n) = (p-1)(q-1)$ .
- Step 4: A popular choice for the public exponents is  $e = 2^{16} + 1 = 65537$ . Some applications choose smaller values such as  $e = 3, 5, 17$  or  $257$  instead. This is done to make encryption and signature verification faster on small devices like smart cards but small public exponents may lead to greater security risks.
- Steps 4 and 5 can be performed with the extended Euclidean algorithm; see modular arithmetic.

The extended Euclidean algorithm is an extension to the Euclidean algorithm for finding the greatest common divisor (GCD) of integers  $a$  and  $b$ : it also finds the integers  $x$  and  $y$  in Bézout's identity

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

(Typically either  $x$  or  $y$  is negative).

The extended Euclidean algorithm is particularly useful when  $a$  and  $b$  are coprime, since  $x$  is the modular multiplicative inverse of  $a$  modulo  $b$ .

The public key consists of the modulus  $n$  and the public (or encryption) exponent  $e$ .

The private key consists of the modulus  $n$  and the private (or decryption) exponent  $d$  which must be kept secret.

- For efficiency a different form of the private key can be stored:
- $p$  and  $q$ : the primes from the key generation,
- $d \pmod{p-1}$  and  $d \pmod{q-1}$ ,
- $q^{-1} \pmod{p}$ .
- All parts of the private key must be kept secret in this form.  $p$  and  $q$  are sensitive since they are the factors of  $n$ , and allow computation of  $d$  given  $e$ . If  $p$  and  $q$  are not stored in this form of the private key then they are securely deleted along with other intermediate values from key generation.
- Although this form allows faster decryption and signing by using the Chinese Remainder Theorem, it is considerably less secure since it enables side channel attacks. This is a particular problem if implemented on smart cards, which benefit most from the improved efficiency. (Start with  $y = x^e \pmod{n}$  and let the card decrypt that. So it computes  $y^d \pmod{p}$  or  $y^d \pmod{q}$  whose results give some value  $z$ . Now, induce an error in one of the computations. Then  $\text{gcd}(z - x, n)$  will reveal  $p$  or  $q$ .)

In cryptography, a side channel attack is any attack based on information gained from the physical implementation of a cryptosystem, rather than theoretical weaknesses in the algorithms (compare cryptanalysis). For example, timing information, power consumption, electromagnetic leaks or even sound can provide an extra source of information which can be exploited to break the system. Many side-channel attacks require considerable technical knowledge of the internal operation of the system on which the cryptography is implemented.

Attempts to break a cryptosystem by deceiving or coercing people with legitimate access are not typically called side-channel attacks: see social engineering and rubber-hose cryptanalysis. For attacks on computer systems themselves (which are often used to perform cryptography and thus contain cryptographic keys or plaintexts), see computer security.

## 8.3 examples

A *timing attack* watches data movement into and out of the CPU, or memory, on the hardware running the cryptosystem or algorithm. Simply by observing how long it takes to transfer key information, it is sometimes possible to determine how long the key is in this instance (or to rule out certain lengths which can also be cryptanalytically useful). Internal operational stages in many cipher implementations provide information (typically partial) about the plaintext, key values and so on, and some of this information can be inferred from observed timings. Alternatively, a timing attack may simply watch for the length of time a cryptographic algorithm requires -- this alone is sometimes enough information to be cryptanalytically useful.

A *power monitoring attack* can provide similar information by observing the power lines to the hardware, especially the CPU. As with a timing attack, considerable information is inferable for some algorithm implementations under some circumstances.

As a fundamental and inevitable fact of electrical life, fluctuations in current generate radio waves, making whatever is producing the currents subject -- at least in principle -- to a *van Eck* (aka, TEMPEST) attack. If the currents concerned are patterned in distinguishable ways, which is typically the case, the radiation can be recorded and used to infer information about the operation of the associated hardware. According to former MI5 officer Peter Wright, the British Security Service analysed emissions from French cipher equipment in the 1960s[1]. In the 1980s, Soviet eavesdroppers were known to plant bugs inside IBM Selectric typewriters to monitor the electrical noise generated as the type ball rotated and pitched to strike the paper; the characteristics of those signals could determine which key was pressed<sup>[citation needed]</sup>.

If the relevant currents are those associated with a display device (ie, highly patterned and intended to produce human readable images), the task is greatly eased. CRT displays use substantial currents to steer their electron beams and they have been 'snooped' in real time with minimum cost hardware from considerable distances (hundreds of meters have been demonstrated). LCDs require, and use, smaller currents and are less vulnerable -- which is not to say they are invulnerable. Some LCDs have been proven that they are vulnerable too, see [2].

Also as an inescapable fact of electrical life in actual circuits, flowing currents heat the materials through which they flow. Those materials also continually lose heat to the environment due to other equally fundamental facts of thermodynamic existence, so there is a continually changing thermally induced mechanical stress as a result of these heating and cooling effects. That stress appears to be the most significant contributor to low level acoustic (i.e. *noise*) emissions from operating CPUs (about 10 kHz in some cases). Recent research by Shamir et al. has demonstrated that information about the operation of cryptosystems and algorithms can be obtained in this way as well. This is an acoustic attack; if the surface of the CPU chip, or in some cases the CPU package, can be observed, infrared images can also provide information about the code being executed on the CPU, known as a *thermal imaging attack*.

## 8.4 encrypting messages

Alice transmits her public key  $(n, e)$  to Bob and keeps the private key secret. Bob then wishes to send message  $M$  to Alice.

He first turns  $M$  into a number  $m < n$  by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext  $c$  corresponding to:

$$c = m^e \pmod n$$

## chapter 8

This can be done quickly using the method of exponentiation by squaring. Bob then transmits  $c^d$  to Alice.

### 8.5 decrypting messages

Alice can recover  $m$  from  $c^d$  by using her private key exponent  $e$  by the following computation:

$$m \equiv c^d \pmod{n}.$$

Given  $c^d$ , she can recover the original message  $M$ .

The above decryption procedure works because first

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

Now,  $ed \equiv 1 \pmod{(p-1)(q-1)}$ , and hence

$$ed \equiv 1 \pmod{p-1} \quad \text{and}$$

$$ed \equiv 1 \pmod{q-1}$$

which can also be written as

$$ed = k(p-1) + 1 \quad \text{and}$$

$$ed = h(q-1) + 1$$

for proper values of  $k$  and  $h$ . If  $m$  is not a multiple of  $p$  then  $m$  and  $p$  are coprime because  $p$  is prime; so by Fermat's little theorem

$$m^{(p-1)} \equiv 1 \pmod{p}$$

and therefore, using the first expression for  $ed$ ,

$$m^{ed} = m^{k(p-1)+1} = (m^{p-1})^k m \equiv 1^k m = m \pmod{p}.$$

If instead  $m$  is a multiple of  $p$ , then

$$m^{ed} \equiv 0^{ed} = 0 \equiv m \pmod{p}.$$

Using the second expression for  $ed$ , we similarly conclude that

$$m^{ed} \equiv m \pmod{q}.$$

Since  $p$  and  $q$  are distinct prime numbers, applying the Chinese remainder theorem to these two congruences yields

$$m^{ed} \equiv m \pmod{pq}.$$

Thus,

$$c^d \equiv m \pmod{n}.$$

## 8.6 a worked example

Here is an example of RSA encryption and decryption. The parameters used here are artificially small, but you can also use OpenSSL to generate and examine a real keypair.

- Choose two prime numbers

$$p = 61 \text{ and } q = 53$$

- Compute

$$n = 61 * 53 = 3233$$

- Compute the totient  $\phi(n) = (p - 1)(q - 1)$

$$\phi(n) = (61 - 1)(53 - 1) = 3120$$

- Choose  $e > 1$  coprime to 3120

$$e = 17$$

- Compute  $d$  such that  $de \equiv 1 \pmod{\phi(n)}$  e.g., by computing the modular multiplicative inverse of  $e$  modulo  $\phi(n)$  :

$$d = 2753$$

$$17 * 2753 = 46801 = 1 + 15 * 3120.$$

The public key is  $(n = 3233, e = 17)$ . For a padded message  $m$  the encryption function is:

$$c = m^e \pmod n = m^{17} \pmod{3233}.$$

The private key is  $(n = 3233, d = 2753)$ . The decryption function is:

$$m = c^d \pmod n = c^{2753} \pmod{3233}.$$

For example, to encrypt  $m = 123$ , we calculate

$$c = 123^{17} \pmod{3233} = 855.$$

To decrypt  $c = 855$ , we calculate

$$m = 855^{2753} \pmod{3233} = 123.$$

Both of these calculations can be computed efficiently using the square-and-multiply algorithm for modular exponentiation.

## 8.7 padding schemes

When used in practice, RSA is generally combined with some padding scheme. The goal of the padding scheme is to prevent a number of attacks that potentially work against RSA without padding (In cryptography, padding refers to a number of distinct practices):

- When encrypting with low encryption exponents (e.g.,  $e = 3$ ) and small values of the  $m$ , (i.e.  $m < n^{1/e}$ ) the result of  $m^e$  is strictly less than the modulus  $n$ . In this case, ciphertexts can be easily decrypted by taking the  $e$ th root of the ciphertext over the integers.
- If the same clear text message is sent to  $e$  or more recipients in an encrypted way, and the receiver's shares the same exponent  $e$ , but different  $p$ ,  $q$ , and  $n$ , then it is easy to decrypt the original clear text

## chapter 8

message via the Chinese remainder theorem (Chinese remainder theorem refers to a result about congruences in number theory and its generalizations in abstract algebra). Johan Håstad noticed that this attack is possible even if the cleartexts are not equal, but the attacker knows a linear relation between them. This attack was later improved by Don Coppersmith.

- Because RSA encryption is a deterministic encryption algorithm – i.e., has no random component – an attacker can successfully launch a chosen plaintext attack against the cryptosystem, by encrypting likely plaintexts under the public key and test if they are equal to the ciphertext. A cryptosystem is called semantically secure if an attacker cannot distinguish two encryptions from each other even if the attacker knows (or has chosen) the corresponding plaintexts. As described above, RSA without padding is not semantically secure.
- RSA has the property that the product of two ciphertexts is equal to the encryption of the product of the respective plaintexts. That is  $m_1^e m_2^e \equiv (m_1 m_2)^e \pmod{n}$ . Because of this multiplicative property a chosen-ciphertext attack is possible. E.g. an attacker, who wants to know the decryption of a ciphertext  $c = m^e \pmod{n}$  may ask the holder of the secret key to decrypt an unsuspecting-looking ciphertext  $c' = cr^e \pmod{n}$  for some value  $r$  chosen by the attacker. Because of the multiplicative property  $c'$  is the encryption of  $mr \pmod{n}$ . Hence, if the attacker is successful with the attack, he will learn  $mr \pmod{n}$  from which he can derive the message  $m$  by multiplying  $mr$  with the modular inverse of  $r$  modulo  $n$ .

To avoid these problems, practical RSA implementations typically embed some form of structured, randomized padding into the value  $m$  before encrypting it. This padding ensures that  $m$  does not fall into the range of insecure plaintexts, and that a given message, once padded, will encrypt to one of a large number of different possible ciphertexts.

Standards such as PKCS have been carefully designed to securely pad messages prior to RSA encryption. Because these schemes pad the plaintext  $m$  with some number of additional bits, the size of the un-padded message  $M$  must be somewhat smaller. RSA padding schemes must be carefully designed so as to prevent sophisticated attacks which may be facilitated by a predictable message structure. Early versions of the PKCS standard (i.e. PKCS #1 up to version 1.5) used a construction that turned RSA into a semantically secure encryption scheme. This version was later found vulnerable to a practical adaptive chosen ciphertext attack. Later versions of the standard include Optimal Asymmetric Encryption Padding (OAEP), which prevents these attacks. The PKCS standard also incorporates processing schemes designed to provide additional security for RSA signatures, e.g., the Probabilistic Signature Scheme for RSA (RSA-PSS).

## 8.8 practical considerations

### 8.8.1 key generation

Finding the large primes  $p$  and  $q$  is usually done by testing random numbers of the right size with probabilistic primality tests which quickly eliminate virtually all non-primes.

$p$  and  $q$  should not be 'too close', lest the Fermat factorization for  $n$  be successful, if  $p \approx q$ , for instance is less than  $2n^{1/4}$  (which for even small 1024-bit values of  $n$  is  $3 \times 10^{77}$ ) solving for  $p$  and  $q$  is trivial. Furthermore, if either  $p-1$  or  $q-1$  has only small prime factors,  $n$  can be factored quickly by Pollard's  $p-1$  algorithm, and these values of  $p$  or  $q$  should therefore be discarded as well.

It is important that the secret key  $d$  be large enough. Michael J. Wiener showed in 1990 that if  $p$  is between  $q$  and  $2q$  (which is quite typical) and  $d < n^{1/4}/3$ , then  $d$  can be computed efficiently from  $n$  and  $e$ . There is no known attack against small public exponents such as  $e=3$ , provided that proper padding is used. However, when no padding is used or when the padding is improperly implemented then small public exponents have a greater risk of leading to an attack, such as for example the unpadded plaintext vulnerability listed above. 65537 is a commonly used value for  $e$ . This value can be regarded as a compromise between avoiding potential small exponent attacks and still allowing efficient encryptions (or signature verification). The NIST Special Publication on Computer Security (SP 800-78 Rev 1 of August



2007) does not allow public exponents  $e$  smaller than 65537, but does not state a reason for this restriction.

### 8.8.2 speed

RSA is much slower than DES and other symmetric cryptosystems. In practice, Bob typically encrypts a secret message with a symmetric algorithm, encrypts the (comparatively short) symmetric key with RSA, and transmits both the RSA-encrypted symmetric key and the symmetrically-encrypted message to Alice.

Symmetric-key algorithms are a class of algorithms for cryptography that use trivially related, often identical, cryptographic keys for both decryption and encryption.

The encryption key is trivially related to the decryption key, in that they may be identical or there is a simple transform to go between the two keys. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link.

Other terms for symmetric-key encryption are secret-key, single-key, one-key and eventually private-key encryption. Use of the latter term does conflict with the term *private key* in public key cryptography.

This procedure raises additional security issues. For instance, it is of utmost importance to use a strong random number generator for the symmetric key, because otherwise Eve (an eavesdropper wanting to see what was sent) could bypass RSA by guessing the symmetric key.

### 8.8.3 key distribution

As with all ciphers, how RSA public keys are distributed is important to security. Key distribution must be secured against a man-in-the-middle attack. Suppose Eve has some way to give Bob arbitrary keys and make him believe they belong to Alice. Suppose further that Eve can *intercept* transmissions between Alice and Bob. Eve sends Bob her own public key, which Bob believes to be Alice's. Eve can then intercept any ciphertext sent by Bob, decrypt it with her own secret key, keep a copy of the message, encrypt the message with Alice's public key, and send the new ciphertext to Alice. In principle, neither Alice nor Bob would be able to detect Eve's presence. Defenses against such attacks are often based on digital certificates or other components of a public key infrastructure.

In cryptography, a public key infrastructure (PKI) is an arrangement that binds public keys with respective user identities by means of a certificate authority (CA). The user identity must be unique for each CA. The binding is established through the registration and issuance process, which, depending on the level of assurance the binding has, may be carried out by software at a CA, or under human supervision. The PKI role that assures this binding is called the Registration Authority (RA). For each user, the user identity, the public key, their binding, validity conditions and other attributes are made unforgeable in public key certificates issued by the CA.

The term trusted third party (TTP) may also be used for certificate authority (CA). The term PKI is sometimes erroneously used to denote public key algorithms which, however, do not require the use of a CA.

## 8.9 security

The security of the RSA cryptosystem is based on two mathematical problems: the problem of factoring large numbers and the RSA problem. Full decryption of an RSA ciphertext is thought to be infeasible on the assumption that both of these problems are hard, i.e., no efficient algorithm exists for solving them. Providing security against *partial* decryption may require the addition of a secure padding scheme.

The RSA problem is defined as the task of taking  $e$ th roots modulo a composite  $n$ : recovering a value  $m$  such that  $c = m^e \pmod n$ , where  $(n, e)$  is an RSA public key and  $c$  is an RSA ciphertext. Currently the most promising approach to solving the RSA problem is to factor the modulus  $n$ . With the ability to recover prime factors, an attacker can compute the secret exponent  $d$  from a public key  $(n, e)$ , then decrypt  $c$  using the

## chapter 8

standard procedure. To accomplish this, an attacker factors  $n$  into  $p$  and  $q$ , and computes  $(p-1)(q-1)$  which allows the determination of  $d$  from  $e$ . No polynomial-time method for factoring large integers on a classical computer has yet been found, but it has not been proven that none exists. See integer factorization for a discussion of this problem.

As of 2005, the largest number factored by a general-purpose factoring algorithm was 663 bits long (see RSA-200), using a state-of-the-art distributed implementation. RSA keys are typically 1024–2048 bits long. Some experts believe that 1024-bit keys may become breakable in the near term (though this is disputed); few see any way that 4096-bit keys could be broken in the foreseeable future. Therefore, it is generally presumed that RSA is secure if  $n$  is sufficiently large. If  $n$  is 256 bits or shorter, it can be factored in a few hours on a personal computer, using software already freely available. Keys of 512 bits (or less) have been shown to be practically breakable in 1999 when RSA-155 was factored by using several hundred computers. A theoretical hardware device named TWIRL and described by Shamir and Tromer in 2003 called into question the security of 1024 bit keys. It is currently recommended that  $n$  be at least 2048 bits long.

In 1994, Peter Shor published Shor's algorithm, showing that a quantum computer could in principle perform the factorization in polynomial time. However, quantum computation is still in the early stages of development and may never prove to be practical.

### 8.9.1 adaptive chosen-ciphertext attack

An adaptive chosen-ciphertext attack (abbreviated as CCA2) is an interactive form of [chosen-ciphertext attack](#) in which an attacker sends a number of ciphertexts to be decrypted, then uses the results of these decryptions to select subsequent ciphertexts. It is to be distinguished from an indifferent chosen-ciphertext attack (CCA1).

The goal of this attack is to gradually reveal information about an encrypted message, or about the decryption key itself. For public-key systems, adaptive-chosen-ciphertexts are generally applicable only when they have the property of ciphertext malleability — that is, a ciphertext can be modified in specific ways that will have a predictable effect on the decryption of that message.

### 8.9.2 preventing the adaptive chosen-ciphertext attack

In order to prevent adaptive-chosen-ciphertext attacks, it is necessary to use an encryption or encoding scheme that limits ciphertext malleability. A number of encoding schemes have been proposed; the most common standard for RSA encryption is Optimal Asymmetric Encryption Padding (OAEP). Unlike ad-hoc schemes such as the padding used in PKCS #1 v1, OAEP has been proven secure under the random oracle model.

### 8.9.3 signing messages

Suppose Alice uses Bob's public key to send him an encrypted message. In the message, she can claim to be Alice but Bob has no way of verifying that the message was actually from Alice since anyone can use Bob's public key to send him encrypted messages. So, in order to verify the origin of a message, RSA can also be used to sign a message.

Suppose Alice wishes to send a signed message to Bob. She can use her own private key to do so. She produces a hash value of the message, raises it to the power of  $d \bmod n$  (as she does when decrypting a message), and attaches it as a "signature" to the message. When Bob receives the signed message, he uses the same hash algorithm in conjunction with Alice's public key. He raises the signature to the power of  $e \bmod n$  (as he does when encrypting a message), and compares the resulting hash value with the message's actual hash value. If the two agree, he knows that the author of the message was in possession of Alice's secret key, and that the message has not been tampered with since.

Note that secure padding schemes such as RSA-PSS are as essential for the security of message signing as they are for message encryption, and that the same key should never be used for both encryption and signing purposes

## 8.10 practical attacks

Adaptive-chosen-ciphertext attacks were largely considered to be a theoretical concern until 1998, when Daniel Bleichenbacher of Bell Laboratories demonstrated a practical attack against systems using RSA encryption in concert with the PKCS #1 v1 encoding function, including a version of the Secure Socket Layer (SSL) protocol used by thousands of web servers at the time.

The Bleichenbacher attacks took advantage of flaws within the PKCS #1 function to gradually reveal the content of an RSA encrypted message. Doing this requires sending several million test ciphertexts to the decryption device (eg, SSL-equipped web server.) In practical terms, this means that an SSL session key can be exposed in a reasonable amount of time, perhaps a day or less.

### 8.10.1 timing attacks

Kocher described a new attack on RSA in 1995: if the attacker *Eve* knows *Alice's* hardware in sufficient detail and is able to measure the decryption times for several known ciphertexts, she can deduce the decryption key  $d$  quickly. This attack can also be applied against the RSA signature scheme. In 2003, Boneh and Brumley demonstrated a more practical attack capable of recovering RSA factorizations over a network connection (e.g., from a Secure Socket Layer (SSL)-enabled webserver). This attack takes advantage of information leaked by the Chinese remainder theorem optimization used by many RSA implementations.

One way to thwart these attacks is to ensure that the decryption operation takes a constant amount of time for every ciphertext. However, this approach can significantly reduce performance. Instead, most RSA implementations use an alternate technique known as cryptographic blinding. RSA blinding makes use of the multiplicative property of RSA. Instead of computing  $c^d \bmod n$ , Alice first chooses a secret random value  $r$  and computes  $(r^e c)^d \bmod n$ . The result of this computation is  $r m \bmod n$  and so the effect of  $r$  can be removed by multiplying by its inverse. A new value of  $r$  is chosen for each ciphertext. With blinding applied, the decryption time is no longer correlated to the value of the input ciphertext and so the timing attack fails.

### 8.10.2 adaptive chosen ciphertext attacks

In 1998, Daniel Bleichenbacher described the first practical adaptive chosen ciphertext attack, against RSA-encrypted messages using the PKCS #1 v1 padding scheme (a padding scheme randomizes and adds structure to an RSA-encrypted message, so it is possible to determine whether a decrypted message is valid.) Due to flaws with the PKCS #1 scheme, Bleichenbacher was able to mount a practical attack against RSA implementations of the Secure Socket Layer protocol, and to recover session keys. As a result of this work, cryptographers now recommend the use of provably secure padding schemes such as Optimal Asymmetric Encryption Padding, and RSA Laboratories has released new versions of PKCS #1 that are not vulnerable to these attacks.

### 8.10.3 branch prediction analysis (BPA) attacks

Many processors use a branch predictor to determine whether a conditional branch in the instruction flow of a program is likely to be taken or not. Usually these processors also implement simultaneous multithreading (SMT). Branch prediction analysis attacks use a spy process to discover (statistically) the private key when processed with these processors.

## chapter 8

Simple Branch Prediction Analysis (SBPA) claims to improve BPA in a non-statistical way. In their paper, "On the Power of Simple Branch Prediction Analysis", the authors of SBPA (Onur Aciicmez and Cetin Kaya Koc) claim to have discovered 508 out of 512 bits of an RSA key in 10 iterations.

## chapter 9 elliptic curve cryptography

The Elliptic Curve Cryptography (ECC) is a public-key cryptography method based on the study of elliptic curves over finite fields.

### 9.1 history

The

### 9.2 finite fields

RSA involves a public key and a pr

### 9.3 elliptic curves over finite fields

RSA involves a public key and a pr

### 9.4 finite fields

RSA involves a public key and a pr

### 9.5 finite fields

RSA involves a public key and a pr

### 9.6 finite fields

RSA involves a public key and a pr

### 9.7 finite fields

RSA involves a public key and a pr

## chapter 10 standardul de semnătură digitală – DSS

### 10.1 ce este standardul de semnătură digitală?

DSS este un acronim pentru Digital Signature Standard, un standard al guvernului federal SUA, dezvoltat de către NSA (Național security Agency), standard care precizează modalitatea de generare a semnăturilor digitale pentru autentificarea documentelor electronice.

Specificația algoritmului de semnătură digitală (DSA) este conținută în FIPS (Federal Information Processing Standard) 186. Ultima versiune (pentru nov. 2019) a acestei specificații este FIPS 186-4 și a apărut în iulie 2013 - <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

### 10.2 DSA – algoritmul de semnătură digitală

O semnătură digitală DSA se calculează utilizând un set de parametri de domeniu, o cheie privată  $x$ , un număr secret  $k$  specific fiecărui mesaj, datele care urmează a fi semnate precum și o funcție de hash.

Semnătura digitală este verificată utilizând aceiași parametri de domeniu, cheia publică  $y$  asociată cheii private  $x$ , datele ce urmează a fi verificate precum și funcția hash utilizată la generare.

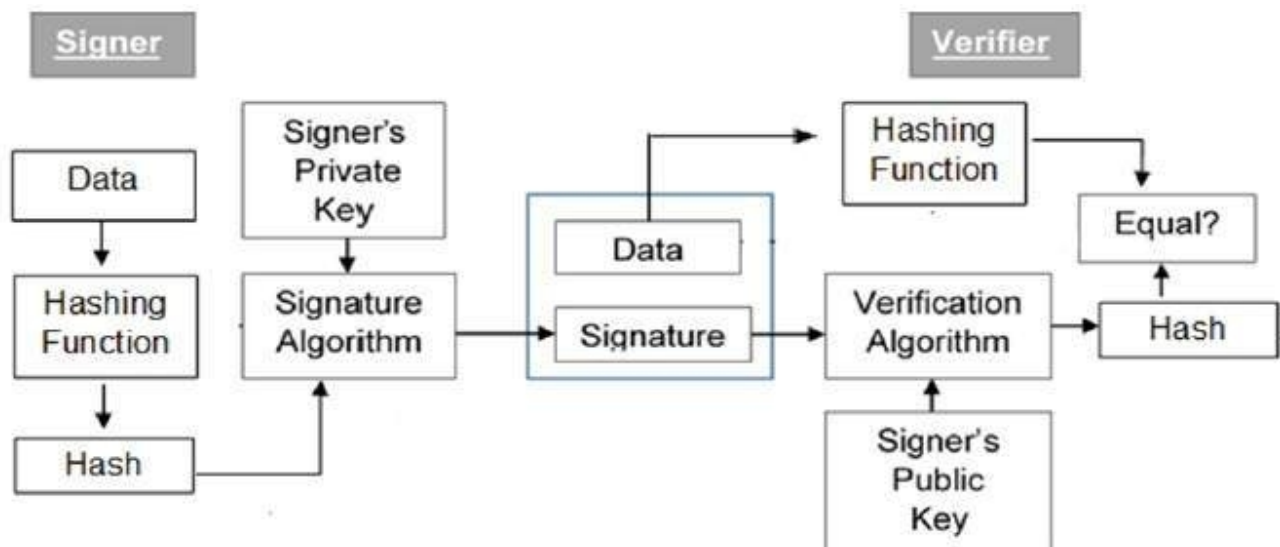


Fig. 10.1 Schema creării și verificării demnăturii digitale (poză de la tutorialspoint.com)

### 10.3 parametri DSA

DSA utilizează următorii parametri:

1.  $p$  = un număr prim, cu  $2^{L-1} < p < 2^L$ , unde  $1024 \leq L \leq 3072$ ,  $L$  e multiplu de 64
2.  $q$  = un divizor prim al lui  $p-1$ , cu  $2^{N-1} < q < 2^N$ , valorile lui  $N$  sunt 160, 224 sau 256

3.  $g = h(p-1)/q \bmod p$ , unde  $h$  este orice întreg cu  $1 < h < p - 1$  astfel încât  $h(p-1)/q \bmod p > 1$   
( $g$  are ordinul  $q \bmod p$ )
4.  $x =$  un întreg random (aleatoriu) sau pseudorandom (pseudo-aleatoriu) generat, cu  $0 < x < q$
5.  $y = gx \bmod p$
6.  $k =$  un întreg random (aleatoriu) sau pseudorandom (pseudo-aleatoriu) generat, cu  $0 < k < q$

Numerele  $p$ ,  $q$ , și pot fi publice și pot fi comune pentru un grup de utilizatori. Cheile private și publice ale unui utilizator sunt  $x$  și  $y$ , respectiv. În mod normal, sunt fixate pentru o perioadă de timp. Parametrii  $x$  și  $k$  sunt folosiți numai pentru generarea de semnături și trebuie păstrați în secret. Parametrul  $k$  trebuie regenerat pentru fiecare semnătură.

Parametrii  $p$  și  $q$  sunt generați după cum se specifică în anexa 2 sau folosind alte metode de securitate aprobate de FIPS. Parametrii  $x$  și  $k$  sunt generați în conformitate cu specificațiile din anexa 3 sau folosind alte metode de securitate aprobate de FIPS.

## 10.4 generarea semnăturii DSA

Semnătura unui mesaj  $M$  este perechea de numere  $r$  și  $s$  calculate după ecuațiile de mai jos:

$$r = (gk \bmod p) \bmod q$$

$$s = (k^{-1}(\text{SHA-1}(M) + xr)) \bmod q.$$

În cele de mai sus,  $k^{-1}$  este inversul multiplicativ al lui  $k$ , mod  $q$ ; adică,  $(k^{-1}k) \bmod q = 1$  și  $0 < k^{-1} < q$ . Valoarea  $\text{SHA-1}(M)$  este o ieșire de 160 biți din algoritmul Secure Hash specificat în FIPS 180-1.

Pentru utilizare în calcularea lui  $s$ , acest șir trebuie convertit într-un număr întreg. Regula de conversie este prezentată în anexa 2.2.

Ca opțiune, se poate verifica dacă  $r = 0$  sau  $s = 0$ . Dacă  $r = 0$  sau  $s = 0$ , ar trebui să se genereze o nouă valoare a lui  $k$ , iar semnătura să fie recalculată (este extrem de puțin probabil ca  $r = 0$  sau  $s = 0$  dacă semnăturile sunt generate corect).

Semnătura este transmisă împreună cu mesajul către verifcător.

## 10.5 verificarea semnăturii DSA

Înainte de a verifica semnătura într-un mesaj semnat,  $p$ ,  $q$  și  $g$  precum și cheia publică și identitatea expeditorului sunt puse la dispoziția verifcătorului într-o manieră autenticată.

Fie  $M'$ ,  $r'$  și  $s'$  versiunile primite ale lui  $M$ ,  $r$  și  $s$ , respectiv, și fie  $y$  cheia publică a semnatarului. Pentru a verifica semnătura, verifcătorul verifică mai întâi faptul că  $0 < r' < q$  și  $0 < s' < q$ ; în cazul în care oricare dintre condiții este încălcată, semnătura va fi respinsă. Dacă aceste două condiții sunt îndeplinite, verifcătorul calculează

$$w = (s')^{-1} \bmod q$$

$$u1 = ((\text{SHA-1}(M'))w) \bmod q$$

$$u2 = ((r')w) \bmod q$$

$$v = (((g)u1 (y)u2) \bmod p) \bmod q.$$

Dacă  $v = r'$ , atunci semnătura este verificată și verifcătorul poate avea încredere mare că mesajul primit a fost trimis de partea celui care deține cheia secretă  $x$  corespunzătoare lui  $y$ . Pentru o dovadă că  $v = r'$  când

## chapter 10

$M' = M$ ,  $r' = r$  și  $s' = s$ , a se vedea anexa 1.

Dacă  $v$  nu este egal cu  $r'$ , atunci este posibil ca mesajul să fi fost modificat, mesajul să fi fost semnat incorect de către semnatar sau este posibil să fi fost semnat de un impostor. Mesajul trebuie considerat nevalid.



## chapter 11 Infrastructura cheilor publice

### 11.1 ce este?

Infrastructura Cheilor Publice (PKI – Public Key infrastructure) este un sistem pentru crearea, distribuirea, identificarea și revocarea cheilor publice. [PKIS]

Sistemul constă dintr-un sistem de încredere care definește:

- roluri
- politici
- proceduri
- elemente hardware
- elemente software

### 11.2 componentele infrastructurii

O implementare PKI constă, în condiții normale, din următoarele 4 elemente:

- o autoritate de certificare (CA – Certificate Authority), care acționează ca rădăcină a arborelui de încredere. Aceste autorități eliberează certificate digitale care specifică proprietarul unei chei publice
- o autoritate de înregistrare (RA – Registration Authority) care validează înregistrarea unui certificat digital cu o cheie publică. Orice verificare a unui certificat digital ajunge la RA, care decide dacă să autentifice datele conținute în acesta
- o bază de date de certificate, care eliberează și revocă certificate digitale
- un depozit (store) în care sunt stocate certificatele cu cheile private asociate

### 11.3 chei publice

O cheie publică este un element care permite identificarea unei entități în spațiul electronic public.

- Protocol developed by Netscape for transmitting private documents via the Internet
- It uses a cryptographic system that uses 2 keys to encrypt data : a public key known to everyone and a private or secret key known only to the recipient of the message

### 11.4 certificat digital

Structura unui certificat digital este specificată în standardul X.509. În versiunea 3 a acestui standard, structura este următoarea:

- versiune – versiunea standardului X.509 utilizată pentru crearea certificatului
- serie – identificator generat de către autoritatea care a emis certificatul

## chapter 11

- identificator pentru algoritmul certificatului – identificator al algoritmului public utilizat de emitent pentru semnarea certificatului
- numele emitentului – numele autorității de certificare (CA) care a emis certificatul
- perioada de validitate – data de start și de expirare a certificatului
- numele proprietarului certificatului
- informații despre cheia publică a proprietarului certificatului
- identificatorul unic al emitentului
- extensii (opționale)
- semnătura digitală creată de emitent (CA), utilizând algoritmul specificat mai sus

Iată mai jos un exemplu – certificatul digital al uvt.ro

- versiune – V3
- serie – 03 2e f6 9c a0 65 75 89 d9 59 5b 9e da 22 07 c1 c8 d6
- identificator pentru algoritmul certificatului – sha256RSA
- numele emitentului – CN = Let's Encrypt Authority X3, O = Let's Encrypt, C = US
- perioada de validitate – start: Saturday, September 28, 2019 6:53:27 AM, end: Friday, December 27, 2019 6:53:27 AM
- numele proprietarului certificatului – uvt.ro
- informații despre cheia publică a proprietarului certificatului – 30 82 01 0a 02 82 01 01 00 db 7a 0d 3e 33 b0 64 0d aa 5c 93 27 bb c9 09 65 7b de 53 f2 c8 4f 53 be b2 c4 c1 ee 2a 85 94 aa 78 c4 03 fe ad 3c 71 1d 14 78 5e c7 62 7b d5 c4 1b d8 93 53 b1 86 c4 3c 3c 5d 1f 34 af 38 bd e0 af d6 dc 60 bf 31 35 27 f9 49 09 4f 89 bc 3f bd 3a a4 85 79 62 0d 3d 5e 98 76 28 19 6a 03 1b 36 0e 8b c8 2f 39 37 52 20 89 95 99 56 22 8e 6c dd 6a 4a df 1b d2 4a 00 53 ee 79 0a 87 f3 ce db a6 e7 a3 8f 6e 44 65 46 cf 67 db 48 3b 69 12 b2 fb d9 01 97 d6 d0 f6 74 af ba 67 78 cb 85 a1 28 cc f5 7c 62 5d fa 56 93 d0 4f 53 80 0d dd 2c 68 32 11 e0 cc 7b 24 b6 cd 2f 42 5a 79 ea 58 cc 30 36 68 8b 61 db 34 fa 18 51 b1 b6 45 a2 70 a1 2b 86 8e df 5b fe 83 48 33 97 ff 97 f6 df c1 04 25 f5 1a 7f 58 d6 5a 5f c7 c9 53 28 6b ed 7d 11 f1 46 41 d5 42 12 ab 39 5f 10 49 e4 8e 0a 0c 08 87 c3 02 03 01 00 01
- identificatorul unic al emitentului – 64 3c 2a 7f 0a 5d 97 c8 7e 5e 4d 94 eb af 5f f6 e0 82 d5 8a
- extensii (opționale)
- semnătura digitală creată de emitent (CA), utilizând algoritmul specificat mai sus – cd cc de 94 da 53 f4 02 bc 33 3a a4 f0 10 b1 7f b7 82 ac cc

### 11.5 autorități de certificare

## 11.6 autorități de înregistrare

## 11.7 infrastructura cheilor publice în România

În România, autoritățile de certificare (acreditate de guvern) sunt:

- SC Trans Sped SRL
- SC Digisign SA
- SC Certsign SA
- SC Alfatrust Certification SA
- SC Centrul de Calcul SA
- SRI – Institutul pentru tehnologii avansate (?)

Pentru o listă la zi la nivelul Uniunii Europene, utilizați link-ul:- <https://webgate.ec.europa.eu/tl-browser/>

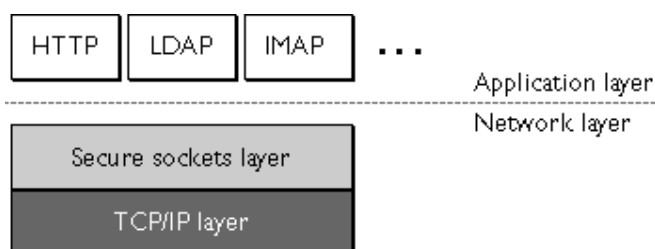
## 11.8 exemple din restul lumii

## chapter 12 secure socket layer – SSL, TLS

### 12.1 SSL ( Secure Sockets Layer)

- Protocol dezvoltat de Netscape pentru transmiterea documentelor private prin internet
- Utilizează un sistem criptografic care folosește 2 chei pentru criptarea datelor: o cheie publică cunoscută de toată lumea și o cheie privată sau secretă cunoscută numai destinatarului mesajului

### 12.2 protocolul SSL



Protocolul SSL rulează peste TCP / IP și sub protocoale de nivel superior, cum ar fi HTTP sau IMAP. Utilizează TCP / IP în numele protocoalelor de nivel superior și, în acest proces, permite unui server capabil de SSL să se autentifice la un client compatibil SSL, permite clientului să se autentifice pe server și permite ambelor mașini să stabilească o conexiune criptată.

- **Autentificarea serverului SSL** permite utilizatorului să confirme identitatea serverului
- **Autentificarea clientului SSL** permite unui server să confirme identitatea utilizatorului
- O **conexiune SSL criptată** necesită ca toate informațiile trimise între un client și un server să fie criptate de software-ul care trimite și decriptate de software-ul primitor, oferind astfel un grad ridicat de confidențialitate

### 12.3 strângerea de mână (handshake) SSL

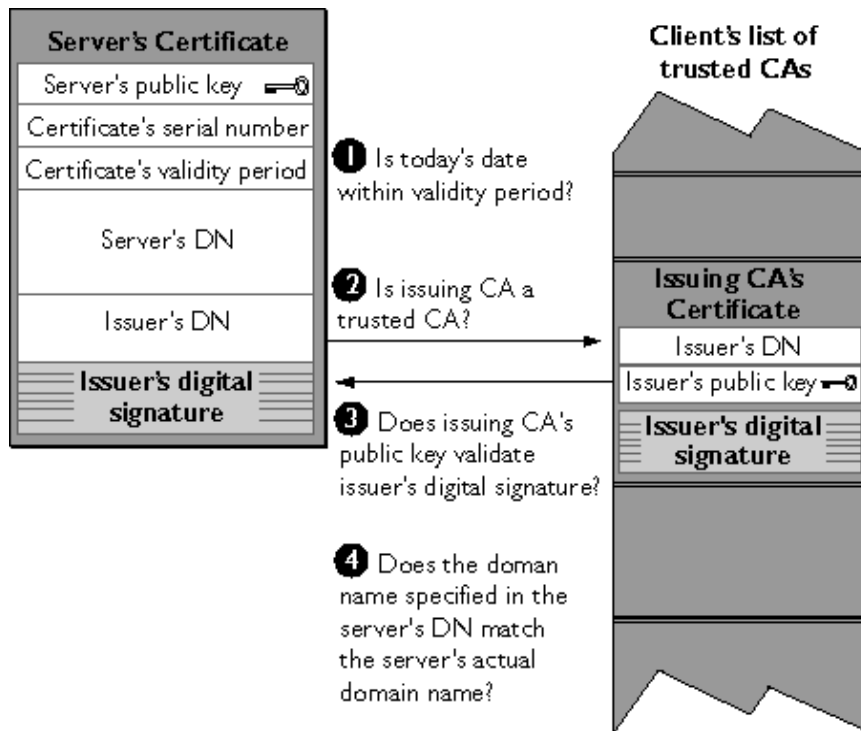
1. Clientul trimite serverului numărul de versiune SSL al clientului, setările de cifrare, datele aleatorii generate și alte informații de care serverul are nevoie ca să comunice cu clientul folosind SSL.
2. Serverul trimite clientului numărul de versiune SSL al serverului, setările de cifrare, datele aleatorii generate și alte informații de care clientul are nevoie pentru a comunica cu serverul prin SSL. Serverul trimite, de asemenea, propriul său certificat și, dacă clientul solicită o resursă server care necesită autentificarea clientului, solicită certificatul clientului.
3. Clientul folosește o parte din informațiile trimise de server pentru a autentifica serverul (consultați autentificarea serverului pentru detalii). Dacă serverul nu poate fi autentificat, utilizatorul este avertizat despre problemă și este informat că nu poate fi stabilită o conexiune criptată și autentificată. Dacă serverul poate fi autentificat cu succes, clientul trece la Pasul 4.

4. Folosind toate datele generate în strângerea de mână până acum, clientul (cu cooperarea serverului, în funcție de cifrul folosit) creează secretul premaster pentru sesiune, îl criptează cu cheia publică a serverului (obținut din certificatul serverului, trimis la pasul 2) și trimite secretul premaster criptat către server.
5. Dacă serverul a solicitat autentificarea clientului (un pas opțional în strângerea de mână), clientul semnează, de asemenea, un pachet de date care este unic pentru această strângere de mână și cunoscută atât de client cât și de server. În acest caz, clientul trimite atât datele semnate, cât și propriul certificat al clientului către server împreună cu secretul premaster criptat.
6. Dacă serverul a solicitat autentificarea clientului, serverul încearcă să autentifice clientul (consultați Autentificarea client pentru detalii). Dacă clientul nu poate fi autentificat, sesiunea este încheiată. Dacă clientul poate fi autentificat cu succes, serverul folosește cheia sa privată pentru a decripta secretul premaster, apoi efectuează o serie de pași (pe care clientul îi realizează pornind de la același secret premaster) pentru a genera secretul master.
7. Atât clientul cât și serverul folosesc secretul master pentru a genera cheile de sesiune, care sunt chei simetrice utilizate pentru criptarea și decriptarea informațiilor schimbate în timpul sesiunii SSL și pentru a verifica integritatea acestora - adică pentru a detecta orice modificare a datelor între momentul în care a fost trimis și momentul în care este primit prin conexiunea SSL.
8. Clientul trimite un mesaj către server prin care îl informează că mesajele viitoare de la client vor fi criptate cu cheia de sesiune. Apoi trimite un mesaj separat (criptat) care indică faptul că porțiunea client a strângerii de mână a fost terminată.
9. Serverul trimite un mesaj către client informându-l că mesajele viitoare de la server vor fi criptate cu cheia de sesiune. Apoi trimite un mesaj separat (criptat) care indică faptul că porțiunea de server a strângerii de mână este terminată.
10. Strângerea de mână SSL este acum completă și sesiunea SSL a început. Clientul și serverul folosesc cheile de sesiune pentru a cripta și decripta datele pe care le transmit reciproc și pentru a-i valida integritatea.

## 12.4 autentificarea serverului

În cazul autentificării serverului, clientul criptează secretul premaster cu cheia publică a serverului. Doar cheia privată corespunzătoare poate decripta corect secretul, astfel încât clientul are o anumită certitudine că identitatea asociată cu cheia publică este de fapt serverul cu care este conectat clientul. În caz contrar, serverul nu poate decripta secretul premaster și nu poate genera cheile simetrice necesare sesiunii, iar sesiunea va fi încheiată.

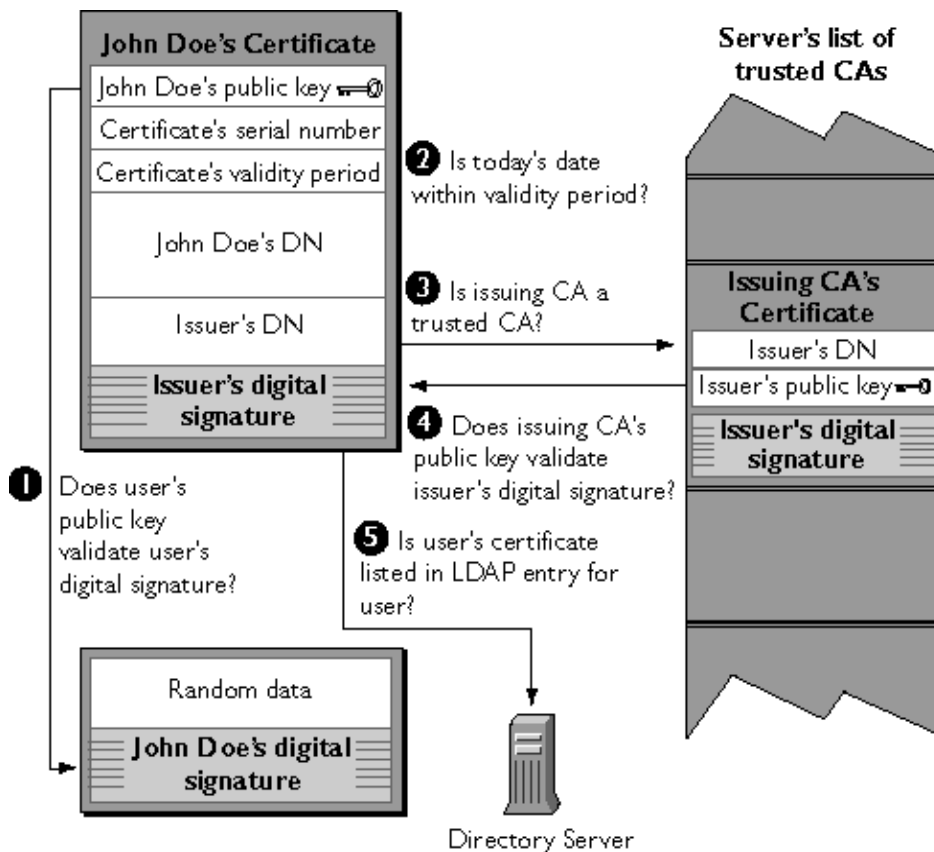
Exemplu: Cum autentifică un client Netscape un certificat de server



## 12.5 autentificarea clientului

În cazul autentificării clientului, clientul criptează unele date aleatorii cu cheia privată a clientului - adică creează o semnătură digitală. Cheia publică din certificatul clientului poate valida corect semnătura digitală numai dacă a fost utilizată cheia privată corespunzătoare. În caz contrar, serverul nu poate valida semnătura digitală și sesiunea este încheiată.

Exemplu: Cum autentifică un client Netscape un certificat de server



## 12.6 algoritmi de criptare (cifruri) utilizate cu SSL

- **DES.** Data Encryption Standard, un algoritm de criptare utilizat de guvernul S.U.A.
- **AES.** Advanced Encryption Standard – algoritmul folosit curent pentru încipatre simetrică
- **DSA.** Algoritmul de semnătură digitală, parte a standardului de autentificare digitală utilizat de guvernul S.U.A.
- **KEA.** Algoritmul de schimb de chei, un algoritm utilizat pentru schimbul de chei de către guvernul Statelor Unite.
- **MD5.** Algoritmul Message Digest dezvoltat de Rivest.
- **RC2 și RC4.** Algoritmul lui Rivest de criptare dezvoltat pentru RSA Data Security.
- **RSA.** Un algoritm cu cheie publică atât pentru criptare cât și pentru autentificare. Dezvoltat de Rivest, Shamir și Adleman.
- Schimb de chei **RSA.** Un algoritm de schimb de chei pentru SSL bazat pe algoritmul RSA.
- **SHA-1.** Algoritmul Secure Hash 1, o funcție de hash folosită de guvernul SUA.
- **SKIPJACK.** Un algoritm secretizat cu cheie simetrică implementat într-un hardware compatibil FORTEZZA utilizat de guvernul SUA.
- **TRIPLE DES.** DES aplicat de trei ori

## chapter 13 secure shell

### 13.1 istoric și dezvoltare

#### 13.1.1 versiunea 1.0

În 1995, Tatu Ylönen, un cercetător la Universitatea de Tehnologie din Helsinki, Finlanda, a proiectat prima versiune a protocolului (acum numită SSH-1), determinată de un atac de furt de parole din rețeaua sa universitară. Scopul SSH a fost înlocuirea protocoalelor anterioare rlogin, TELNET și rsh, care nu ofereau o autentificare puternică sau nu garantau confidențialitatea. Ylönen a lansat implementarea ca freeware în iulie 1995, iar instrumentul a câștigat rapid în popularitate. Spre sfârșitul anului 1995, baza de utilizatori SSH a crescut la 20.000 de utilizatori în cincizeci de țări.

În decembrie 1995, Ylönen a fondat SSH Communications Security pentru a comercializa și dezvolta SSH. Versiunea originală a software-ului SSH a folosit diverse părți de software gratuite, cum ar fi GNU libgmp, dar versiunile ulterioare lansate de SSH Secure Communications au evoluat spre software tot mai proprietar.

#### 13.1.2 versiunea 2.0

„Secsh” a fost numele oficial al grupului de lucru IETF (Internet Engineering Task Force) responsabil pentru versiunea 2 a protocolului SSH. În 1996, o versiune revizuită a protocolului, SSH-2, a fost adoptată ca standard. Această versiune este incompatibilă cu SSH-1. SSH-2 prezintă atât îmbunătățiri de securitate, cât și ale funcționalității față de SSH-1. O mai bună securitate, de exemplu, vine prin schimbul de chei Diffie-Hellman și verificarea puternică a integrității prin intermediul codurilor de autentificare a mesajelor. Noile caracteristici ale SSH-2 includ posibilitatea de a rula orice număr de sesiuni shell pe o singură conexiune SSH.

### 13.2 OpenSSH

În 1999, dezvoltatorii care doreau să aibă disponibilă o versiune de software gratuită au revenit la versiunea anterioară 1.2.12 a programului SSH inițial, care a fost ultima lansată sub licență open source. OSSH-ul lui Björn Grönvall a fost ulterior dezvoltat din această bază de cod. La scurt timp după aceea, dezvoltatorii OpenBSD au creat o nouă ramură a codului lui Grönvall și au lucrat pe larg la acesta, creând OpenSSH, care a fost livrat cu versiunea 2.6 a OpenBSD. Din această versiune, s-a format o ramură de „portabilitate” pentru a porta OpenSSH către alte sisteme de operare.

Se estimează că, începând cu anul 2000, existau 2.000.000 de utilizatori de SSH.

Începând cu 2005, OpenSSH este cea mai populară implementare SSH, venită implicit într-un număr mare de sisteme de operare. Între timp, OSSH a devenit învechită.



## 13.3 standardul de internet SSH-2

### 13.3.1 publicare originală

În 2006, protocolul SSH-2 menționat mai sus a devenit un standard de internet propus, cu publicarea de către grupul de lucru RFC de la IETF „secsh”. A fost publicat pentru prima dată în ianuarie 2006.

- [RFC 4250](#), The Secure Shell (SSH) Protocol Assigned Numbers
- [RFC 4251](#), The Secure Shell (SSH) Protocol Architecture
- [RFC 4252](#), The Secure Shell (SSH) Authentication Protocol
- [RFC 4253](#), The Secure Shell (SSH) Transport Layer Protocol
- [RFC 4254](#), The Secure Shell (SSH) Connection Protocol
- [RFC 4255](#), Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints
- [RFC 4256](#), Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)
- [RFC 4335](#), The Secure Shell (SSH) Session Channel Break Extension
- [RFC 4344](#), The Secure Shell (SSH) Transport Layer Encryption Modes
- [RFC 4345](#), Improved Arcfour Modes for the Secure Shell (SSH) Transport Layer Protocol

### 13.3.2 modificări ulterioare

Ulterior a fost modificat și extins de următoarele publicații.

- [RFC 4419](#), Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol (March 2006)
- [RFC 4432](#), RSA Key Exchange for the Secure Shell (SSH) Transport Layer Protocol (March 2006)
- [RFC 4716](#), The Secure Shell (SSH) Public Key File Format (Nov 2006)

## 13.4 utilizări

Exemplu de tunelare a unei aplicații X11 peste SSH: utilizatorul „josh” utilizează SSH de la mașina locală „foofighter” la mașina de la distanță „tengwar” pentru a rula xeyes.

Conectarea la OpenWrt prin SSH folosind PuTTY care rulează pe Windows.

SSH este un protocol care poate fi utilizat pentru multe aplicații pe mai multe platforme, inclusiv UNIX, Microsoft Windows, Apple Mac și Linux. Unele dintre aplicațiile de mai jos pot necesita funcții care sunt disponibile sau compatibile doar cu clienți sau servere SSH specifice. De exemplu, utilizarea protocolului SSH pentru a implementa un VPN este posibilă, dar în prezent doar cu serverul OpenSSH și implementarea clientului.

- Pentru conectare la un shell pe o gazdă de la distanță (înlocuirea Telnet și rlogin)
- Pentru executarea unei singure comenzi pe o gazdă la distanță (înlocuirea rsh)
- Pentru copierea fișierelor de pe un server local pe o gazdă la distanță. Consultați SCP, ca o alternativă pentru rcp

## chapter 13

- În combinație cu SFTP, ca o alternativă sigură la transferul de fișiere FTP
- În combinație cu rsync to backup, copiați fișierele în mod eficient și sigur
- Pentru redirectionarea portului sau tunelarea unui port (nu trebuie confundat cu o VPN care rutează pachetele între diferite rețele sau poduri două domenii de difuzare într-unul singur).
- Pentru utilizarea ca VPN criptat cu drepturi depline. Rețineți că numai serverul și clientul OpenSSH acceptă această caracteristică.
- Pentru redirectionarea X11 prin mai multe gazde
- Pentru navigarea pe web printr-o conexiune proxy criptată cu clienții SSH care acceptă protocolul SOCKS.
- Pentru montarea sigură a unui director pe un server la distanță ca sistem de fișiere pe un computer local folosind [SSHFS](#).
- Pentru monitorizarea și gestionarea automată de la distanță a serverelor prin unul sau mai multe dintre mecanismele descrise mai sus.
- Pentru colaborarea sigură a mai multor utilizatori de canale SSH, unde este posibilă transferul, schimbul, schimbul și recuperarea sesiunilor deconectate.

## 13.5 arhitectură

Diagrama pachetului binar SSH-2.

Protocolul SSH-2 are o arhitectură internă (definită în RFC 4251) cu straturi bine separate. Acestea sunt:

- **Stratul de transport (RFC 4253).** Acest strat gestionează schimbul inițial de chei și autentificarea serverului și stabilește criptarea, compresia și verificarea integrității. Acesta expune stratului superior o interfață pentru trimiterea și primirea pachetelor sub formă de text de până la 32.768 de octeți fiecare (mai multe pot fi permise de implementare). Stratul de transport prevede, de asemenea, schimbul de chei, de obicei după ce 1 GB de date au fost transferate sau după ce a trecut 1 oră, oricare dintre acestea este mai devreme.
- **Stratul de autentificare a utilizatorului (RFC 4252).** Acest strat gestionează autentificarea clientului și oferă o serie de metode de autentificare. Autentificarea este bazată pe client: atunci când se solicită o parolă, poate fi solicitat clientul SSH, nu serverul. Serverul nu răspunde decât la solicitările de autentificare ale clientului. Metodele de autentificare utilizate pe scară largă includ următoarele:
  - **parolă:** o metodă de autentificare simplă a parolei, inclusiv o facilitare care permite schimbarea unei parole. Această metodă nu este implementată de toate programele.
  - **publickey:** o metodă pentru autentificarea bazată pe cheie publică, care acceptă, de regulă, cel puțin pachetele de chei DSA sau RSA, alte implementări acceptând și certificatele X.509.
  - **Tastatură interactivă (RFC 4256):** o metodă versatilă în care serverul trimite una sau mai multe solicitări de introducere a informațiilor, iar clientul le afișează și trimite răspunsuri introduse de utilizator. Folosit pentru a furniza o singură dată autentificare cu parolă, cum ar fi S / Key sau SecurID. Folosit de unele configurații OpenSSH atunci când PAM este furnizorul de autentificare gazdă de bază pentru a furniza în mod eficient autentificarea cu parolă, uneori ducând la imposibilitatea de a vă autentifica cu un client care acceptă doar metoda de autentificare simplă a parolei.
  - **Metodele de autentificare GSSAPI** care oferă o schemă extensibilă pentru a efectua autentificarea SSH folosind mecanisme externe precum Kerberos 5 sau NTLM, oferind un singur semn pe capacitatea sesiunilor SSH. Aceste metode sunt de obicei implementate prin implementări comerciale SSH pentru utilizare în organizații, deși OpenSSH are o implementare GSSAPI funcțională.
- **Stratul de conexiune (RFC 4254).** Acest nivel definește conceptul de canale, solicitări de canale și solicitări globale folosind serviciile SSH care sunt furnizate. O singură conexiune SSH poate găzdui

mai multe canale simultan, fiecare transferând date în ambele direcții. Solicitățile de canal sunt utilizate pentru a transmite date specifice canalului în afara bandei, cum ar fi dimensiunea modificată a unei ferestre a terminalului sau codul de ieșire al unui proces din partea serverului. Clientul SSH solicită să fie redirecționat un port din partea serverului folosind o solicitare globală. Tipurile de canale standard includ:

- shell pentru shell-uri terminale, solicitări SFTP și exec (inclusiv transferuri SCP)
- direct-tcpip pentru conexiunile transmise client-server
- redirecționat-tcpip pentru conexiunile transmise de la server la client
- Înregistrarea SSHFP DNS (RFC 4255) oferă amprentele cheie ale gazdelor publice pentru a ajuta la verificarea autenticității gazdei.

Această arhitectură deschisă oferă o flexibilitate considerabilă, permițând SSH-ului să fie utilizat într-o varietate de scopuri dincolo de shell-ul securizat. Funcționalitatea singurului strat de transport este comparabilă cu TLS; stratul de autentificare a utilizatorului este extrem de extensibil cu metode de autentificare personalizate; iar stratul de conexiune oferă posibilitatea de a multiplexa multe sesiuni secundare într-o singură conexiune SSH, o caracteristică comparabilă cu BEEP și care nu este disponibilă în TLS.

## 13.6 probleme de securitate

Deoarece SSH-1 are defecte de proiectare inerente care îl fac vulnerabil (de exemplu, atacuri om-în-mijloc), acum este în general învechit și ar trebui evitat prin dezactivarea explicită a recesiunii la SSH-1. În timp ce majoritatea serverelor și clienților moderni acceptă SSH-2, unele organizații utilizează în continuare software fără suport pentru SSH-2 și astfel SSH-1 nu poate fi întotdeauna evitat.

În toate versiunile SSH, este important să verificați cheile publice necunoscute înainte de a le accepta ca valide. Acceptarea cheii publice a unui atacator ca cheie publică validă are ca efect dezvăluirea parolei transmise și permite atacuri de tipul omului în mijloc.

## chapter 14 securitatea datelor

Securitatea datelor este modalitatea de a asigura protejarea datelor împotriva modificărilor nedorite și că accesul la acestea este controlat în mod adecvat. Astfel, securitatea datelor ajută la asigurarea confidențialității. De asemenea, ajută la protejarea datelor cu caracter personal.

În Marea Britanie, Legea privind protecția datelor este utilizată pentru a se asigura că datele personale sunt accesibile celor pe care-l privește și oferă despăgubiri persoanelor fizice în cazul în care există inexactități. Acest lucru este deosebit de important pentru a se asigura că persoanele sunt tratate în mod echitabil, de exemplu în scopul verificării creditului. Legea privind protecția datelor prevede că numai persoanele fizice și companiile cu motive legitime și legale pot prelucra informații personale și că acestea nu pot fi partajate.

Standardul internațional ISO / IEC 17799 acoperă securitatea datelor sub subiectul securității informațiilor, iar unul dintre principiile sale cardinale este că toate informațiile stocate, adică datele, ar trebui să fie deținute astfel încât să fie clar a cui este responsabilitatea de a proteja și controla accesul la aceste date. [DATS]

### 14.1 coruperea datelor

Coruperea (modificările nedorite) datelor se referă la erorile din datele computerului care apar în timpul transmiterii sau recuperării, introducând modificări neintenționate la datele originale. Sistemele de stocare și transmisie a computerelor utilizează o serie de măsuri pentru a asigura integritatea datelor, lipsa erorilor.

Coruperea datelor în timpul transmisiei are o varietate de cauze. Întreruperea transmiterii datelor provoacă pierderi de informații. Condițiile de mediu pot interfera cu transmisia de date, în special atunci când avem de-a face cu metode de transmisie wireless. Norii puternici pot bloca transmisiile prin satelit. Rețelele wireless sunt susceptibile la interferențe de la dispozitive precum cuptoare cu microunde.

Pierderea datelor în timpul stocării are două mari cauze: eșecul hardware și software. Defecțiunea hard-disk-ului se încadrează în prima categorie, în timp ce eșecul software apare de obicei din cauza erorilor din cod.

Când coruperea datelor se comportă ca un proces Poisson, în care fiecare bit de date are o probabilitate independentă de modificare, coruperea datelor poate fi în general detectată prin utilizarea sumelor de verificare și poate fi adesea corectată prin utilizarea codurilor de corectare a erorilor.

Dacă este detectată o corupere necorectată a datelor, se pot aplica proceduri precum retransmisie automată sau restaurare din copii de rezervă. Matricele de discuri RAID, stochează și evaluează biți de paritate pentru datele dintr-un set de hard disk-uri și pot reconstrui datele corupte în urma eșecului unui singur disc.

Dacă sunt folosite mecanisme adecvate pentru detectarea și remedierea corupției datelor, integritatea datelor poate fi menținută. Acest lucru este deosebit de important în sectorul bancar, în cazul în care o eroare nedetectată poate afecta drastic un sold al contului, cât și în utilizarea datelor criptate sau comprimate, în cazul în care o mică eroare poate face un set de date extinse inutilizabil.

### 14.2 confidențialitatea datelor

Confidențialitatea datelor este relația dintre colectarea și diseminarea datelor, tehnologiile implicate, așteptările publicului în ce privește ocrotirea vieții private și problemele legale din domeniu.

Problemele de confidențialitate există oriunde sunt colectate și stocate informații de identificare personală - în formă digitală sau în alt mod. Controlul necorespunzător sau inexistent al divulgării poate fi

cauza principală a problemelor de confidențialitate. Probleme de confidențialitate a datelor pot apărea ca răspuns la informații dintr-o gamă largă de surse, cum ar fi:

- registre medicale
- investigații și proceduri de justiție penală
- instituții și tranzacții financiare
- trăsături biologice, cum ar fi materialul genetic
- înregistrări privind rezidența și locația
- etnie

Provocarea confidențialității datelor constă în partajarea datelor, protejând în același timp informațiile de identificare personală. Domeniul securității datelor și al securității informațiilor are rolul de a proiecta și utiliza software, hardware și resurse umane pentru a rezolva această problemă.

### 14.3 remanența datelor

Remanența (persistența) datelor este reprezentarea reziduală a datelor care au fost, într-un fel, șterse sau eliminate în mod nominal. Acest reziduu se poate datora datelor rămase intacte printr-o operație de ștergere nominală sau prin proprietățile fizice ale mediului de stocare. Remanența datelor poate face posibilă dezvăluirea accidentală a informațiilor sensibile, în cazul în care suportul de stocare este eliberat într-un mediu necontrolat (de exemplu, aruncat în coșul de gunoi sau dat unui terț).

De-a lungul timpului, s-au dezvoltat diverse tehnici pentru combaterea remanenței datelor. În funcție de eficacitatea și intenția, acestea sunt adesea clasificate ca fiind fie de compensare sau de purjare / sanitizare. Metodele specifice includ suprascrierea, demagnetizarea, criptarea și distrugerea fizică.

### 14.4 scurgerea de date

Scurgerile de date pot include incidente, cum ar fi furtul sau pierderea de mass-media digitale, cum ar fi hard disk-uri sau laptop-uri care conțin astfel de suporturi pe care aceste informații sunt stocate necriptat, postarea de astfel de informații pe web la nivel mondial sau de pe un calculator accesibil la Internet fără precauții adecvate de securitate a informațiilor, transferul acestor informații către un sistem care nu este complet deschis, dar nu este acreditat în mod corespunzător sau formal pentru securitate la nivelul aprobat, cum ar fi e-mailul necriptat sau transferul acestor informații către sistemele de informații ale unei agenții eventual ostilă, cum ar fi o corporație concurentă sau o națiune străină, unde poate fi expusă la tehnici de decriptare mai intense.

### 14.5 furtul datelor

Furtul de date este o problemă în creștere comisă în primul rând de angajații din birouri cu acces la tehnologie, cum ar fi computere desktop și dispozitive portabile capabile să stocheze informații digitale, precum unități flash, iPod-uri și chiar camere digitale. Întrucât angajații petrec adesea o cantitate considerabilă de timp dezvoltând contacte și informații confidențiale și protejate de drepturi de autor pentru compania pentru care lucrează, ei simt adesea că au un anumit drept la informații și sunt înclinați să copieze și / sau să ștergă o parte din aceasta atunci când părăsesc compania, sau să le folosească greșit în timp ce sunt încă la muncă.

## chapter 14

În timp ce majoritatea organizațiilor au implementat firewall-uri și sisteme de detectare a intruziunilor, foarte puțini iau în considerare amenințarea venită din partea angajatului mediu care copiază datele proprii pentru câștig sau utilizare personală de către o altă companie. Un scenariu obișnuit este acela în care o persoană de vânzări face o copie a bazei de date de contact pentru a fi utilizată în următoarea lor slujbă. De obicei, aceasta este o încălcare clară a condițiilor lor de angajare.

Daunele cauzate de furtul de date pot fi considerabile, având capacitatea de a transmite fișiere foarte mari prin e-mail, pagini web, dispozitive USB, stocare DVD și alte dispozitive portabile. Dispozitivele media amovibile sunt din ce în ce mai mici, cu o capacitate crescută de stocare, iar activități precum podslurping sunt din ce în ce mai frecvente. Acum este posibil să stocați 1 TB de date pe un dispozitiv care să încapă în buzunarul unui angajat, date care ar putea contribui la căderea unei afaceri.

### 14.6 separarea protecției și securității

În informatică, separarea protecției și securității este o alegere de proiectare. Se poate defini **protecția** ca mecanism și **securitatea** ca politică, făcând astfel distincția protecție-securitate ca un caz particular al principiului distincției mecanism-politică.

Adoptarea acestei distincții într-o arhitectură de calculator, de obicei, înseamnă că protecția este asigurată ca un mecanism de toleranță la erori de hardware / firmware și kernel, susținerea sistemului de operare și aplicațiile care rulează pe partea de sus în punerea în aplicare a politicilor lor de securitate. În acest proiect, politicile de securitate se bazează, prin urmare, pe mecanismele de protecție și pe tehnici suplimentare de criptografie.

Cele două abordări hardware majore pentru securitate și / sau protecție sunt domeniile de protecție ierarhice (arhitecturi cu inel cu „modul supraveghetor” și „mod utilizator”) și adresare bazată pe capacitate.

Prima abordare adoptă o politică deja la nivelurile de arhitectură inferioare (hw / firmware / kernel), restricționând restul sistemului să se bazeze pe ea; prin urmare, alegerea de a distinge între protecție și securitate în proiectarea arhitecturii generale duce la respingerea demersului ierarhic în favoarea abordării bazate pe capacitate.

Modelul Bell-LaPadula este un exemplu de model în care protecția și securitatea nu sunt separate. În Landwehr 1981 există un tabel care arată care modele pentru securitatea computerului separă mecanismul de protecție și politica de securitate. Cele cu separare sunt: matricea de acces, UCLA Data Secure Unix, preluarea și filtrarea; iar dintre cele fără separare menționăm: high-water mark, Bell și LaPadula (original și revizuit), fluxul de informații, dependență puternică și constrângeri.

## chapter 15 controlul accesului

### 15.1 access control lists

### 15.2 file system ACLs

For file systems, the access control lists are operating system specific data structures which specify individual or group rights to certain objects, like files, directories or processes.

#### 15.2.1 the NTFS example

NTFS – New Technology File System is the standard file system for MS based operating systems, starting with Windows NT. Besides support for ACLs, NTFS offers file system journaling, as well.

### 15.3 network ACLs

### 15.4 passwords

### 15.5 password efficiency

### 15.6 password storing

chapter 15

15.7 passwords over the network

15.8 password breaking



## chapter 16 securitatea rețelelor

## chapter 17 amenințări specifice internetului

### 17.1 generalități

Virusii, atacurile hackerilor și alte amenințări cibernetice sunt acum o parte din viața de zi cu zi. Malware este răspândit pe tot internetul, hackerii fura date confidențiale și casuțe de email invadate de spam sunt prețul pe care oamenii îl plătesc pentru comoditatea de calcul. Orice calculator sau rețea neprotejată sunt vulnerabile.

Utilizatorii de acasă pot pierde date personale valoroase cu un singur click pe un site greșit. Jocurile pentru copii schimbă virusii fără ca cineva să știe. Primești un mail care solicita o actualizare a detaliilor de plată și un hacker obține accesul la contul vostru bancar. Pe calculator se instalează o pagină ascunsă și computerul dumneavoastră devine un zombi.

Internetul a devenit o resursă critică pe care oamenii se bazează pe el pentru a-și face munca sau pentru divertisment. Ei folosesc internetul pentru a efectua cercetări și pentru a aduna informații. Oamenii folosesc email-ul sau rețelele de socializare populare care să-i ajute să păstreze legătura cu colegii și clienții. Încărcarea, descărcarea, partajarea fișierelor și alte produse de lucru sunt acum activități de zi cu zi.

Din păcate, atunci când oamenii îndeplinesc aceste sarcini zilnic, expun companiile pentru care lucrează la riscuri grave de securitate. Angajații trebuie să fie preocupați mai mult decât să prevină pur și simplu oamenii care fac lucruri la serviciu pe care aceștia nu ar trebui să le facă – navigarea pe site-uri restricționate sau necorespunzătoare, de exemplu. Acum oamenii sunt expuși la amenințări nocive, distructive în timp ce își fac pur și simplu meseria. Companiile ar trebui să examineze măsurile IT de securitate a acestora și să determine dacă sunt suficiente pentru protecția împotriva acestor amenințări transmise prin web.

Firewall-urile Gateway și antivirusul nu pot proteja împotriva complexului și multitudinea de programe daunatoare care amenința infrastructurile IT. Firewall-urile pot detecta traficul web, dar majoritatea nu au nici un mijloc de monitorizare a informațiilor specifice care sunt transferate. Soluțiile antivirusului sunt reactive, nu preventive; sunt eficiente numai împotriva amenințărilor foarte specifice și oferă o protecție limitată numai după ce un atac a avut deja loc. Organizațiile trebuie să își completeze sistemele de securitate existente cu o soluție care să completeze aceste măsuri de protecție la nivel de conținut,

### 17.2 expunerea la amenințări

Amenințările la care oameni sunt expuși zilnic, variază în funcție de ceea ce fac aceștia pe internet.

În paragrafele următoare prezentăm unele dintre aceste amenințări pe baza activității principale în care sunt angajați oamenii.

#### 17.2.1 accesul la internet

În timp ce navighează pe web, oamenii pot vizita în mod inconștient site-uri web dăunătoare – site-uri web care au fost deja atacate sau proiectate special pentru a distribui un software dăunător. Când un utilizator vizitează unul dintre aceste site-uri, hackerii pot exercita control asupra calculatorului, fișierelor descărcate, instalarea keylogger-urilor sau a altor coduri dăunătoare.

## 17.2.2 partajarea fișierelor

Atunci când oamenii partajează fișierele folosind rețelele peer-to-peer, acestia descarcă adesea spyware și coduri de mobil dăunătoare împreună cu produsul de lucru prevăzut. Spyware aduna informații despre utilizator – adesea înregistrarea tastelor apasate, obiceiurile de navigare web, parole, adrese de email și transferă informațiile respective înapoi în site-ul sursa prin portul 80 de comunicații back-channel.

Codul dăunător poate fi transmis prin web – virusuri, cai troieni, viermi sau cod de internet malițios.

Codurile de mobil dăunătoare achiziționate se distribuie folosind pagini web sau cod HTML, incluzând ActiveX sau cod Javascript și este încorporat în paginile web.

## 17.2.3 mesagerie instant

Folosind aplicațiile de mesagerie, oamenii pot „vorbi” și partaja fișiere fără nici un efort. Aplicațiile de mesagerie pot ajuta la promovarea comunicării între membrii echipei și de a reduce numărul de întâlniri față în față necesare. Acesta poate fi, de asemenea, un instrument de e-commerce de neprețuit, cu reprezentanți de servicii pentru clienți care sprijină clienții noi prin răspunderea la întrebările despre produs, ajutând la finalizarea tranzacțiilor etc.

Din păcate, poate fi și utilizat pentru a transmite informații proprietarilor companiei în format necriptat și pentru a transfera fișierele atașate care ocolesc complet infrastructura de securitate existentă.

În plus, multe descărcări instant sunt infestate de viruși, cai troieni și viermi. De fapt, mai mulți viermi au vizat clienți IM specifici, trimitând utilizatorilor email-uri de înșelătorie și folosind liste de prieteni IM pentru răspândire.

## 17.2.4 e-mail

Chiar și trimiterea și răspunderea la email-uri poate fi o activitate riscantă. Email-ul oferă hackerilor o modalitate ușoară de distribuire a conținutului dăunător. Mesajele prin email pot include atașamente de fișiere infectate cu viruși, viermi, cai troieni sau alte programe care daunează. Hackerii trimit fișierele infectate și speră ca destinatarul să le deschidă. Se folosesc alte email-uri cu intenții rele care acționează asupra vulnerabilității browserului pentru a se răspândi. Un exemplu este viermele Nimda, care a rulat automat pe calculatoare cu o versiune vulnerabilă

## 17.3 phishing

Activitatea care constă în obținerea datelor confidențiale(phishing) este o altă amenințare care valorifică popularitatea email-ului ca instrument de comunicare. În multe comploturi, hoții trimit email-uri cu aspect oficial, dar fals pentru a-i păcăli pe destinatari să dezvăluie un cont confidențial sau informații despre utilizator. Destinatarii sunt încurajați să facă click pe linkurile din email-uri, ducându-i la ceea ce par a fi pagini de servicii pentru clienți, completate cu link-uri, logo-uri, limba și toate aspectele familiare a site-ului web autentic. De fapt, unele site-uri web frauduloase sunt atât de convingătoare încât bara de adrese a utilizatorului arată ca sunt conectați la un site bancar sau site e-commerce.

Hoții sunt considerați hackeri, deoarece folosesc ingineria socială pentru a-și păcăli și înșela țintele. De exemplu, un hoț poate trimite un email folosind fatadele unui bănci majore, a unui card de credit sau a unui serviciu E-money ca PayPal. Email-ul nu va arata doar oficial, ci va avea și un nume de domeniu de rețea cu aspect oficial și adresa de retur. Conținutul va avea un mesaj inofensiv, cum ar fi: „Informațiile contului tău necesită actualizare”.

Presupunerea hoților este ca oamenii vor deschide emailul, îl vor citi și vor crede conținutul. Ei speră ca

## chapter 17

cititorul va face click pe linkul furnizat, deoarece pare oficial și va fi direcționat către un site care arată exact ca cel real (PayPal etc.). În realitate, utilizatorul a fost direcționat către un site fals și urmează să scrie informații confidențiale ale contului care vor fi înregistrate și trimise înapoi către atacator. Impactul înșelătoriei este asupra întreprinderilor cât și asupra consumatorilor. Sunt bine cunoscute băncile de încredere și alți furnizori de servicii online care sunt îngrijorate de faptul că temerile de a identifica furtul și spargerea conturilor vor opri consumatorii să facă cumpărături și prelucrare online a altor tranzacții financiare. Visa Internațional a intrat în prima agregare la nivel mondial, serviciul în efortul de a combate înșelătoria.

Phishing-ul poate viza și informații confidențiale ale companiei. Prin persoanele țintă (trimiterea unui email tuturor persoanelor de la o anumită companie presupusă de la departamentul IT, de exemplu), hoții au obținut cu succes accesul la nume de utilizatori și parole corporative. Folosind aceste informații, hackerii ar putea să se infiltreze și să acceseze rețeaua corporativă și, la rândul lor, informații confidențiale despre clienți sau utilizatori, care poate prezenta nu numai probleme de răspundere juridică, ci și probleme de conformitate cu reglementările.

### 17.4 pharming

Programul automat dăunător care se află în așteptare până când un utilizator se conectează la un site web țintă (în principal bănci, alte instituții financiare și site-uri de e-commerce) utilizează o nouă schemă numită „pharming”. Ca phishing-ul, acești hoți își propun să fure informații confidențiale despre cont. Spre deosebire de phishing, această metodă nu se bazează pe email-uri false pentru a ademini victimele nesatisfăcute, de fapt, este aproape nedetectabil. Pharming folosește viruși de tip cal troian care modifică comportamentul browserelor web. Utilizatorul încearcă să acceseze un site bancar online sau unul dintre celelalte site-uri țintă care declanșează de fapt browserul către un site fraudulos. Odată ce o mașină este infectată, utilizatorul poate tasta adresa URL corectă și poate ajunge în continuare tot pe site-ul fraudulos.

### 17.5 site-uri web atacate

Hackerii pot transforma un site web într-unul malițios. Când site-urile web sunt atacate, site-urile, în sine, devin vectori de atac și sunt folosiți pentru distribuirea codului malițios. Când serverul web a unei companii este compromis, clienții (sau potențiali clienți) sunt infectați în mod involuntar cu codul malițios atunci când pur și simplu vizitează site-ul, aceste infecții apar fără ca clientul să fie nevoit să ruleze programe sau să deschidă vreun atașament.

### 17.6 site-uri web defrișate (spoofed)

Infraactorii cibernetici valorifică încrederea consumatorilor în anumite produse și mărci și folosesc această încredere pentru a păcălii utilizatorii în divulgarea informațiilor confidențiale despre cont. Un scenariu tipic implică trimiterea utilizatorilor a unui site înșelător, rugându-i să facă click pe un link pentru a-și actualiza informațiile despre cont. HTML-ul din email-uri arată convingător și familiar. Mulți utilizatori respectă cu ușurință „cererea băncii lor” oferind informațiile contului de pe site-urile concetate – site-uri care par valide, dar sunt, de fapt, frauduloase.

Indiferent dacă utilizatorii cad sau nu victime ale acestor planuri, aceștia devin precauți și suspectează comunicațiile de pe site-uri de e-commerce sau site-uri bancare, și acum sunt mai puțin susceptibile să se angajeze în tranzacții online. Aceste temeri – deși justificate pot avea un impact asupra comerțului global.

## chapter 18 viruşı

## chapter 19 troieni

### 19.1 definiție

Un cal troian (uneori prescurtat ca troian), este un malware care nu se auto-reproduce, care pare să îndeplinească o funcție de dorit pentru utilizator, dar în schimb facilitează accesul neautorizat la sistemul informatic al utilizatorului. Termenul este derivat din povestea Calului troian în mitologia greacă.

Un program aparent inocent conceput pentru a eluda caracteristicile de securitate ale unui sistem. Metoda obișnuită de introducere a unui cal troian este prin donarea unui program sau a unei părți dintr-un program unui utilizator al sistemului a căru securitate trebuie încălcată. Codul donat va îndeplini în mod evident o funcție utilă; destinatarul nu va ști că codul are alte efecte, cum ar fi scrierea unei copii a numelui său de utilizator și a parolei într-un fișier a căru existență este cunoscută doar donatorului și din care donatorul va colecta ulterior orice date au fost scrise

### 19.2 purpose and operation

Trojan horses are designed to allow a hacker remote access to a target computer system. Once a Trojan horse has been installed on a target computer system, it is possible for a hacker to access it remotely and perform various operations. The operations that a hacker can perform are limited by user privileges on the target computer system and the design of the Trojan horse.

Operations that could be performed by a hacker on a target computer system include:

- Use of the machine as part of a botnet (i.e. to perform spamming or to - perform Distributed Denial-of-service (DDoS) attacks)
- Data theft (e.g. passwords, credit card information, etc.)
- Installation of software (including other malware)
- Downloading or uploading of files
- Modification or deletion of files
- Keystroke logging
- Viewing the user's screen
- Wasting computer storage space

Trojan horses require interaction with a hacker to fulfill their purpose, though the hacker need not be the individual responsible for distributing the Trojan horse. In fact, it is possible for hackers to scan computers on a network using a port scanner in the hope of finding one with a Trojan horse installed, that the hacker can then use to control the target computer.

### 19.3 installation and distribution

Trojan horses can be installed through the following methods:

1. **Software downloads** (i.e., a Trojan horse included as part of a software application downloaded from a file sharing network)

2. **Websites containing executable content** (i.e., a Trojan horse in the form of an ActiveX control)
3. **Email attachments**
4. **Application exploits** (i.e., flaws in a web browser, media player, messaging client, or other software that can be exploited to allow installation of a Trojan horse). Also, there have been reports of compilers that are themselves Trojan horses. While compiling code to executable form, they include code that causes the output executable to become a Trojan horse.

Users can be tricked into installing Trojan horses by being enticed or frightened. For example, a Trojan horse might arrive in email described as a computer game. When the user receives the mail, they may be enticed by the description of the game to install it. Although it may in fact be a game, it may also be taking other action that is not readily apparent to the user, such as deleting files or mailing sensitive information to the attacker. As another example, an intruder may forge an advisory from a security organization, such as the CERT Coordination Center, that instructs system administrators to obtain and install a patch.

Other forms of "*social engineering*" can be used to trick users into installing or running Trojan horses. For example, an intruder might telephone a system administrator and pose as a legitimate user of the system who needs assistance of some kind. The system administrator might then be tricked into running a program of the intruder's design.

Software distribution sites can be compromised by intruders who replace legitimate versions of software with Trojan horse versions. If the distribution site is a central distribution site whose contents are mirrored by other distribution sites, the Trojan horse may be downloaded by many sites and spread quickly throughout the Internet community.

Because the Domain Name System (DNS) does not provide strong authentication, users may be tricked into connecting to sites different than the ones they intend to connect to. This could be exploited by an intruder to cause users to download a Trojan horse, or to cause users to expose confidential information.

Intruders may install Trojan horse versions of system utilities after they have compromised a system. Often, collections of Trojan horses are distributed in toolkits that an intruder can use to compromise a system and conceal their activity after the compromise, e.g., a toolkit might include a Trojan horse version of ls which does not list files owned by the intruder. Once an intruder has gained administrative access to your systems, it is very difficult to establish trust in it again without rebuilding the system from known-good software. A Trojan horse may be inserted into a program by a compiler that is itself a Trojan horse.

Finally, a Trojan horse may simply be placed on a web site to which the intruder entices victims. The Trojan horse may be in the form of a Java applet, JavaScript, ActiveX control, or other form of executable content.

## 19.4 removal

Antivirus software is designed to detect and delete Trojan horses, as well as preventing them from ever being installed. Although it is possible to remove a Trojan horse manually, it requires a full understanding of how that particular Trojan horse operates. In addition, if a Trojan horse has possibly been used by a hacker to access a computer system, it will be difficult to know what damage has been done and what other problems have been introduced. In situations where the security of the computer system is critical, it is advisable to simply erase all data from the hard disk and reinstall the operating system and required software.

## 19.5 current use

Due to the growing popularity of botnets among hackers, Trojan horses are becoming more common. According to a survey conducted by BitDefender from January to June 2009, "Trojan-type malware is on the rise, accounting for 83-percent of the global malware detected in the wild".

## chapter 19

Trojan horses can be particularly effective when offered to systems staff who can run code in highly privileged modes. Two remedies are effective: no code should be run unless its provenance is absolutely certain; no code should be run with a higher level of privilege than is absolutely essential.

### 19.6 solutions

The best advice with respect to Trojan horses is to **avoid them** in the first place. System administrators (including the users of single-user systems) should take care to verify that every piece of software that is installed is from a trusted source and has not been modified in transit. When digital signatures are provided, users are encouraged to validate the signature (as well as validating the public key of the signer). When digital signatures are not available, you may wish to acquire software on tangible media such as CDs, which bear the manufacturer's logo. Of course, this is not foolproof either. Without a way to authenticate software, you may not be able to tell if a given piece of software is legitimate, regardless of the distribution media. Software developers and software distributors are strongly encouraged to use cryptographically strong validation for all software they produce or distribute. Any popular technique based on algorithms that are widely believed to be strong will provide users a strong tool to defeat Trojan horses. Anyone who invests trust in digital signatures must also take care to validate any public keys that may be associated with the signature. It is not enough for code merely to be signed -- it must be signed by a trusted source. Do not execute anything sent to you via unsolicited electronic mail. Use caution when executing content such as Java applets, JavaScript, or Active X controls from web pages. You may wish to configure your browser to disable the automatic execution of web page content. Apply the principle of least privilege in daily activity: do not retain or employ privileges that are not needed to accomplish a given task. For example, do not run with enhanced privilege, such as "root" or "administrator," ordinary tasks such as reading email. Install and configure a tool such as Tripwire® that will allow you to detect changes to system files in a cryptographically strong way.

**Educate your users regarding the danger of Trojan horses.** Use firewalls and virus products that are aware of popular Trojan horses. Although it is impossible to detect all possible Trojan horses using a firewall or virus product (because a Trojan horse can be arbitrary code), they may aid you in preventing many popular Trojan horses from affecting your systems.

Review the source code to any open source products you choose to install. Open source software has an advantage compared to proprietary software because the source code can be widely reviewed and any obvious Trojan horses will probably be discovered very quickly. However, open source software also tends to be developed by a wide variety of people with little or no central control. This makes it difficult to establish trust in a single entity. Keep in mind that reviewing source code may be impractical at best, and that some Trojan horses may not be evident.

**Adopt the use of cryptographically strong mutual authentication systems**, such as ssh, for terminal emulation, X.509 public key certificates in web servers, S/MIME or PGP for electronic mail, and kerberos for a variety of services. Avoid the use of systems that trust the domain name system for authentication, such as telnet, ordinary http (as opposed to https), ftp, or smtp, unless your network is specifically designed to support that trust. Do not rely on timestamps, file sizes, or other file attributes when trying to determine if a file contains a Trojan horse.

**Exercise caution** when downloading unauthenticated software. If you choose to install software that has not been signed by a trusted source, you may wish to wait for a period of time before installing it in order to see if a Trojan horse is discovered.



## chapter 20 exploituri software

### 20.1 definiție

„Un **exploit** este o componentă de software, o bucată de date sau o secvență de comenzi care profită de un bug, glitch sau vulnerabilitate pentru a provoca un comportament neintenționat sau neanticipat pe software, hardware sau ceva electronic (de obicei computerizat)“.

Aceasta include lucruri precum obținerea forțată a controlului asupra unui sistem informatic sau care permite escaladarea privilegiilor sau un atac DoS.

### 20.2 clasificare

Există mai multe metode de clasificare a exploiturilor.

1. Cel mai obișnuit este modul în care exploitul contactează software vulnerabil.

- Un **exploit de la distanță** funcționează printr-o rețea și exploatează vulnerabilitatea securității fără a avea acces prealabil la sistemul vulnerabil.
- Un **exploit local** necesită un acces prealabil la sistemul vulnerabil și, de obicei, crește privilegiile persoanei care execută exploitul față de cele acordate de administratorul de sistem.

2. O altă clasificare este prin acțiunea împotriva sistemului vulnerabil

- acces la date neautorizate
- executarea codului arbitrar
- DoS

Multe exploituri sunt concepute pentru a oferi acces la nivel de super-user la un sistem de calcul. Cu toate acestea, este posibil să folosiți mai multe exploituri, mai întâi pentru a obține acces la nivelul de jos, apoi pentru a escalada privilegiile în mod repetat până când ajunge la acces total.

În mod normal, un singur exploit poate profita doar de o vulnerabilitate specifică a software-ului. Adesea, când este publicat un exploit, vulnerabilitatea este rezolvată printr-un patch și exploitul devine învechit pentru versiunile mai noi ale software-ului. Acesta este motivul pentru care unii hackeri de nu își publică exploiturile, ci le păstrează private pentru a le folosi ei sau alți hackeri.

### 20.3 tipuri

Exploiturile sunt clasificate și numite după aceste criterii:

1. Tipul de vulnerabilitate folosită
2. Indiferent dacă trebuie să fie rulate pe aceeași mașină cu programul care are vulnerabilitatea (locală) sau pot fi rulate pe o mașină pentru a ataca un program care rulează pe o altă mașină (de la distanță).

## chapter 20

### 3. Efectele folosirii exploitorilor

Exploitorii de software sunt aproape mereu făcute pentru a face pagube, dar pot fi și inofensive.

Exemple de pagube:

1. Ștergerea sau suprascrierea datelor
2. Reinstalarea după ce este șters
3. Criptarea fișierelor pentru a șantaja
4. Coruperea fișierelor într-un mod subtil
5. Încărcarea sau descărcarea de fișiere
6. Copierea de linkuri false, care duc la site-uri false, chat-uri, sau alte site-uri web cu conturi, aratarea de conturi false
7. Falsificarea istoricului de descărcări, de filme și orice altceva avea computerul care nu au fost accesate de proprietar
8. Posibilitatea de a accesa dispozitivul de la distanță. Se numește RAT (Troian pentru acces la distanță)
9. Descărcarea de alți virusi și malware
10. Configurarea rețelelor de calculatoare zombie pentru a lansa atacuri DDoS sau a trimite spam.
11. Spionarea utilizatorului unui computer și raportarea ascunsă a datelor precum obiceiurile de navigare către alte persoane (a se vedea articolul despre spyware)
12. Screenshot-uri
13. Înregistrarea apăsărilor de taste pentru a fura informații precum parole și numere de carduri de credit
14. Phishing pentru domeniul bancar sau alte detalii ale contului, care pot fi utilizate pentru activități infracționale
15. Instalarea unui „backdoor” pe un sistem informatic
16. Deschiderea și închiderea CD-ROM
17. Redarea de sunete, videoclipuri sau afișarea de imagini
18. Apelarea folosind modemul la numere costisitoare, provocând astfel facturi uriase de telefon
19. Recoltarea adreselor de e-mail și utilizarea lor pentru spam
20. Reporniți computerul ori de câte ori este pornit programul infectat
21. Dezactivarea sau interferarea cu programele antivirus și firewall
22. Dezactivarea sau interferarea cu alte forme concurente de malware
23. Oprirea aleatorie a computerului
24. Instalarea de virusi
25. Încetinirea computerului

## 20.4 atacarea dependențelor software

Aplicațiile se bazează foarte mult pe mediul lor pentru a funcționa corect. Depind de sistemul de operare

pentru a oferi resurse precum memoria și spațiul pe disc; se bazează pe sistemul de fișiere pentru a citi și scrie date; utilizează structuri precum Registrul Windows pentru a stoca și recupera informații; iar lista continuă. Toate aceste resurse oferă o intrare în software, nu la fel de mult ca și utilizatorul uman. Ca orice intrare, dacă software-ul primește o valoare în afara intervalului preconizat, acesta poate eșua. Inducerea scenariilor de eșec ne poate permite să urmărim o aplicație în mediul său neintenționat și să expunem vulnerabilitățile critice.

#### 20.4.1 blocarea accesului către librăriile de date

Software-ul depinde de bibliotecile din sistemul de operare, furnizorii și componentele incluse în aplicație. Tipurile de bibliotecă țintite depind de aplicație. Uneori, DLL-urile au nume obscure care oferă puține indicii cu privire la ceea ce fac. Altele vă pot oferi indicii cu privire la serviciile pe care le prestează pentru aplicație. Acest atac este conceput pentru a se asigura că aplicația testată nu se comportă nesigur dacă bibliotecile de software nu se pot încărca.

Atunci când apar defecțiuni, gestionarea de erori a aplicației este executată. Totuși, uneori, aceste situații nu sunt luate în considerare în timpul proiectării aplicației, iar rezultatul este temuta excepție neîndoelnică. Chiar dacă există cod pentru gestionarea acestor tipuri de erori, acest cod este numai bun pentru erori, deoarece este probabil că a fost supus unor teste mult mai puțin decât restul aplicației.

#### 20.4.2 manipularea valorilor registrului aplicației

Registrul Windows conține informații cruciale pentru funcționarea normală a sistemului de operare și a aplicațiilor instalate. Pentru sistemul de operare, registrul ține evidența informațiilor, cum ar fi locațiile fișierelor cheie, structura directorului, căile de execuție și numerele de versiune ale bibliotecii. Aplicațiile se bazează pe această informație și pe alte informații stocate în registru pentru a funcționa corect. Cu toate acestea, nu toate informațiile stocate în registru sunt securizate nici de utilizatori, nici de alte aplicații instalate. Acest atac testează că aplicațiile nu stochează informații sensibile în registru sau au încredere în registru pentru a se comporta întotdeauna în modul cum a fost prevăzut.

#### 20.4.3 forțarea aplicației pentru a folosi fișiere corupte

Software-ul poate face un număr limitat de acțiuni înainte de a avea nevoie să stocheze sau să citească date permanente. Datele sunt combustibilul care conduce o aplicație, deci mai devreme sau mai târziu toate aplicațiile vor trebui să interacționeze cu sistemul de fișiere. Fișierele corupte sunt ca și cum ai pune zahăr în rezervorul de benzină al mașinii tale; dacă nu relizezi înainte de a porni mașina, dauna poate fi inevitabilă. Acest atac determină dacă aplicațiile pot gestiona date corupte cu brio, fără a expune informații sensibile sau a permite un comportament nesigur.

O aplicație mare poate citi și scrie în sute de fișiere în procesul de îndeplinire a sarcinilor prescrise. Fiecare fișier pe care îl citește o aplicație furnizează date de intrare; oricare dintre aceste fișiere poate fi un potențial punct de eșec și deci o bună ocazie pentru un atac. Fișierele deosebit de interesante de verificat sunt cele care sunt utilizate exclusiv de aplicație și care nu sunt destinate utilizatorului să le citească sau să le modifice; sunt fișiere în care este cel mai probabil să fie implementate verificări adecvate privind integritatea datelor.

#### 20.4.4 manipularea și înlocuirea fișierelor accesate de aplicație

Acest atac implică, de asemenea, dependențe de sistem de fișiere. În atacurile anterioare, încercam să facem ca aplicația să proceseze date corupte. În aceasta, manipulăm date, executabile sau biblioteci în moduri care obligă aplicația să se comporte nesigur. Acest atac poate fi aplicat oricând o aplicație citește sau scrie în sistemul de fișiere, lansează un alt executabil sau accesează funcționalitatea dintr-o bibliotecă. Scopul acestui atac este de a testa dacă aplicația ne permite să facem ceva ce nu ar trebui să putem face.

#### 20.4.5 forțarea aplicației să funcționeze cu resurse diminuate

O aplicație este un set de instrucțiuni pentru executarea hardware-ului computerului. În primul rând, computerul va încărca aplicația în memorie și apoi va oferi aplicației memorie suplimentară în care să stocheze și să manipuleze datele sale interne. Memoria este numai temporară; pentru a fi cu adevărat utilă, o aplicație trebuie să stocheze date permanente. Acolo vine sistemul de fișiere și, odată cu acesta, nevoia de spațiu pe disc. Fără suficient spațiu de memorie, majoritatea aplicațiilor nu vor putea să își îndeplinească funcția prevăzută.

Obiectivul acestui atac este să privezi aplicația de oricare dintre aceste resurse, astfel încât testerii să înțeleagă cât de robustă și sigură este aplicarea lor sub stres. Decizia cu privire la ce scenarii de eșec să încerce (și când) poate fi determinată numai de la caz la caz. O regulă generală este de a bloca o resursă atunci când o aplicație pare cea mai mare nevoie de ea.

### 20.5 atacul asupra interfeței utilizatorului

Interfața utilizatorului este de obicei cel mai confortabil „loc de vânătoare” pentru buguri, pentru testarea securității. Este modul în care suntem obișnuiți să interacționăm cu aplicațiile noastre și modul în care se așteaptă dezvoltorii de aplicații să o facem. Atacurile discutate aici se concentrează asupra intrărilor aplicate software-ului prin interfața sa de utilizator. Majoritatea erorilor de securitate rezultă dintr-un comportament suplimentar, neintenționat și nedocumentat al utilizatorului. Din interfața de utilizator, acest lucru se referă la gestionarea intrărilor neașteptate de la utilizator într-un mod care compromite aplicația, sistemul pe care este rulată sau datele sale. Rezultatul ar putea fi creșterea privilegiilor (un utilizator normal care dobândește drepturi administrative) sau care permite vizualizarea informațiilor secrete de către un utilizator neautorizat.

#### 20.5.1 anomalii de tip „buffer overflow” ( rescrierea unor fișiere adiacente)

Acestea sunt de departe cele mai notorii probleme de securitate din software. Ele apar atunci când aplicațiile nu reușesc să restricționeze corect durata de intrare. Unele dintre acestea nu prezintă o mare amenințare la adresa securității. Altele, însă, pot permite hackerilor să preia controlul asupra unui sistem trimițând un cod bine elaborat către aplicație. Acest al doilea tip este denumit de industrie drept „exploatabil”, deoarece părți din șir pot fi executate dacă sunt interpretate ca cod. Ceea ce se întâmplă uneori este că o cantitate fixă de memorie este alocată pentru a deține intrarea utilizatorului. Dacă atunci dezvoltorii nu reușesc să restricționeze lungimea șirurilor de intrare introduse de utilizator, datele pot suprascrie instrucțiunile de aplicare, permițând utilizatorului să execute un cod arbitrar și să aibă acces la mașină.

#### 20.5.2 examinarea comutatoarelor și opțiunile comune

Unele aplicații sunt tolerante la variația intrării utilizatorului în cadrul unei configurații implicite. Majoritatea

configurațiilor implicite sunt alese de dezvoltatorii de aplicații, iar majoritatea testelor sunt executate în aceste condiții, mai ales dacă opțiunile sunt obscure sau sunt introduse folosind comutatoarele ( - sau /) din linia de comandă. Când aceste configurații sunt schimbate, software-ul este adesea obligat să utilizeze căi de cod care pot fi grav sub-testate și astfel rezultatele pot fi imprevizibile. Evident, testarea unei game largi de intrări sub fiecare set posibil de configurații este imposibil pentru aplicații mari. Acest atac se concentrează asupra unor configurații mai obscure, cum ar fi cele în care comutatoarele sunt setate prin linia de comandă la pornire.

### **20.5.3 exploatarea caracterelor de „evadare”, a seturilor de caractere și comenzilor**

Unele aplicații pot trata anumite caractere ca fiind echivalente atunci când fac parte dintr-un șir. În cele mai multe cazuri, un șir cu litera „a” într-o anumită poziție este probabil să nu fie procesat diferit de un șir similar cu litera „z” în aceeași poziție. În acest sens, urmează în mod natural întrebarea „Ce caractere sau combinații de caractere sunt tratate diferit?”. Aceasta este întrebarea din spatele acestui atac. Forțând aplicația să proceseze caractere și comenzi speciale, uneori o putem forța să se comporte în moduri în care designerii săi nu au intenționat. Factorii care afectează care caracterele și comenzi pot fi interpretate diferit includ limbajul de programare în care a fost scrisă aplicația, bibliotecile prin care datele utilizatorului sunt transmise, dar și cuvinte și șiruri specifice care sunt rezervate de sistemul de operare de bază.

## chapter 21 carduri inteligente

### 21.1 definition

ing

### 21.2 classification

There are sever

## chapter 22 biometrics

### 22.1 definition

“An exploit is a piece of software, a chunk of data, or sequence of commands that take advantage of a bug, glitch or vulnerability in order to cause unintended or unanticipated behavior to occur on computer software, hardware, or something electronic (usually computerized)”.

This frequently includes things like violently gaining control of a computer system or allowing privilege escalation or a denial of service attack.

### 22.2 classification

There are several methods of classifying exploits.

1. The most common one is by how the exploit contacts the vulnerable software.
  - A '**remote exploit**' works over a network and exploits the security vulnerability without any prior access to the vulnerable system.
  - A '**local exploit**' requires prior access to the vulnerable system and usually increases the privileges

## chapter 23 crypto currencies

### 23.1 definition

“An exploit is a piece of software, a chunk of data, or sequence of commands that take advantage of a bug, glitch or vulnerability in order to cause unintended or unanticipated behavior to occur on computer software, hardware, or something electronic (usually computerized)”.

This frequently includes things like violently gaining control of a computer system or allowing privilege escalation or a denial of service attack.

### 23.2 classification

There are several methods of classifying exploits.



## chapter 24 comerțul electronic

### 24.1 file system

For file systems, the access control lists are operating system specific data structures which specify individual or group rights to certain objects, like files, directories or processes.

## chapter 25    protocoale de securitate a rețelelor

În acest capitol acoprim câteva protocoale de securitate a rețelelor care sunt mai puțin vizibile utilizatorilor obișnuiți. Aceste protocoale sunt:

- IPsec
- DNSSEC

### 25.1 IPsec

Data corruption refers to errors in [computer data](#) that occur during transmission or retrieval, introducing unintended changes to the original data. Computer storage and transmission systems use a number of measures to provide [data integrity](#), the lack of errors.

#### 25.1.1 ce este?

IPsec este un

#### 25.1.2 ce este?

IPsec este un

#### 25.1.3 ce este?

IPsec este un

#### 25.1.4 ce este?

IPsec este un

#### 25.1.5 ce este?

IPsec este un

-

## 25.2 DNSSEC

DNSSEC este un acronim pentru DNS (Domain Name System) Security Extensions.

DNS este un element fundamental în arhitectura internetului și care (printre altele) asigură translatarea numelor de domenii în adrese de IP. Transmiterea oricărei cereri adresată unui serviciu specificat (parțial) prin nume de domeniu este precedată de o interogare a unui server DNS care returnează adresa IP asociată domeniului (/serviciului), adresă necesară elementelor de rețea, precum routere, switch-uri, pentru trimiterea acestei cerei la destinație.

### 25.2.1 cum funcționează DNS-ul?

Exemplificăm modul de operare al DNS-ului cu un exemplu simplu. Atunci când tastăm o adresă de domeniu în câmpul de adresă al unui browser, acesta utilizează un **stub resolver**, care este parte a sistemului de operare, pentru inițierea procesului de aflare a adresei IP a serviciului cerut.

Stub resolver-ul este un client DNS simplu care trimite mai departe cererea către un client DNS mai complex, denumit **recursive solver**. Aceștia sunt, în general, administrați de către operatorii de rețea (ISP-uri, etc.). Un recursive solver trimite cererea către unul sau mai mulți **authoritative name servers**.

Datele DNS pentru orice domeniu sunt stocate pe un astfel de server, undeva, pe internet. Datele DNS pentru un anumit domeniu constituie o **zonă**. Unele organizații operează propriile servere de numire (name servers) pentru a-și publica zonele, dar de obicei, acest task este delegat către furnizori (third parties) specializați. Există diferite tipuri de organizații care fac acest lucru, precum registras, registries, companii de găzduire web (web hostings), network servers providers, etc. [DNSI]

### 25.2.2 context

Sistemul DNS nu este sigur. Fiind proiectat în anii 1980, atunci când internetul era mult mai puțin extins, securitatea sa nu era o prioritate. Din acest motiv, atunci când un recursive solver trimite o interogare către un authoritative name server, resolver-ul nu are ici o modalitate de verificare a autenticității răspunsului. Resolver-ul doar verifică doar IP-ul sursei răspunsului, adresă care poate fi cu ușurință falsificată (sau spoofed). Astfel, un atacator poate redirecționa utilizatorul către un site malițios, fără ca utilizatorul să realizeze acest lucru. [DNSI]

### 25.2.3 ce este?

IPsec este un

### 25.2.4 ce este?

IPsec este un

### 25.2.5 ce este?

IPsec este un

### 25.2.6 ce este?

IPsec este un

- 

## 25.3 elemente de securitate în IPv6

Data corruption refers to errors in [computer data](#) that occur during transmission or retrieval, introducing unintended changes to the original data. Computer storage and transmission systems use a number of measures to provide [data integrity](#), the lack of errors.

### 25.3.1 ce este?

IPsec este un

### 25.3.2 ce este?

IPsec este un

### 25.3.3 ce este?

IPsec este un

### 25.3.4 ce este?

IPsec este un

### 25.3.5 ce este?

IPsec este un



## Bibliography

[DATS] – Data security Wikipedia - [https://en.wikipedia.org/wiki/Data\\_security](https://en.wikipedia.org/wiki/Data_security)

[DESW] – DES Wikipedia - [https://en.wikipedia.org/wiki/Data\\_Encryption\\_Standard](https://en.wikipedia.org/wiki/Data_Encryption_Standard)

[DNSI] – DNSSEC – what is it and why is it important -

<https://www.icann.org/resources/pages/dnssec-what-is-it-why-important-2019-03-05-en>

[KSF] – the keccak sponge family function [http://keccak.noekeon.org/specs\\_summary.html](http://keccak.noekeon.org/specs_summary.html)

[SHA1C] - <https://phys.org/news/2017-02-cwi-google-collision-industry-standard.html>

[PKIS] – Total despre PKI - <https://www.ssl2buy.com/wiki/public-key-infrastructure>