METODE CRIPTOGRAFICE DE PROTECȚIE A INFORMAȚIEI

Tema 5: Funcții hash criptografice

OBIECTIVELE TEMEI

În continuare, vom examina funcțiile hash criptografice și cele mai importante proprietăți ale acestora, precum și algoritmi de construcție a valorilor hash.

Obiectivele de bază ale capitolului sunt următoarele:

- Expunerea conceptelor de bază ale funcțiilor hash criptografice;
- Examinarea construcției Merkle-Damgård ce stă la baza funcțiilor hash iterative;
- Analiza diferențelor dintre clasa de funcții hash cu funcția de comprimare dedicată și clasa funcțiilor hash cu funcția de comprimare un cifru bloc cu cheie simetrică;
- Examinarea și analiza implementărilor algoritmilor hash dedicați din familiile MD, SHA, RIPEMD și HAVAL.

SUBIECTELE ABORDATE ÎN CADRUL TEMEI

1 Funcții hash criptografice

1.1 Aspecte teoretice ale funcțiilor hash

1.1.1 Conceptul funcțiilor hash

1.1.2 Clase de funcții hash criptografice

1.1.3 Funcții hash iterative

1.1.4 Extinderea funcțiilor de compresie la funcții hash (meta metoda Merkle)

1.2 Procedee de construcție a MDC-urilor

1.2.1 Funcții hash fără cheie bazate pe cifruri bloc

1.2.1.1 Algoritmi single-length MDC

1.2.1.2 Algoritmi double-length MDC

1.2.2 Funcții hash fără cheie customizate (dedicate)

SUBIECTELE ABORDATE ÎN CADRUL TEMEI

1.2.2.1 Familia de algoritmi hash MD

1.2.2.1.1 Algoritmul MD4

1.2.2.1.2 Algoritmul MD5

1.2.2.2 Familia de algoritmi SHA

1.2.2.2.1 Algoritmul SHA-1

1.2.2.2.2 Familia de algoritmi SHA-2

1.2.2.3 O comparație a funcțiilor hash criptografice

FUNCȚIILE HASH CRIPTOGRAFICE PROMOVEAZĂ INTEGRITATEA DATELOR

Anterior am studiat procedee criptografice ce promovează confidențialitatea datelor.

În continuare, vom examina mecanisme criptografice bazate pe criptografia cu cheie simetrică ce promovează un alt serviciu de securitate, și anume, integritatea datelor.

Protocoalele trebuie să fie definite astfel încât cel puțin să poată detecta atentatele de alterare și să identifice sursa tranzacțiilor în Internet.

Definiția 1.1.1. Integritatea datelor este o proprietate prin care se asigură că datele nu au fost modificate într-o manieră neautorizată din momentul creării acestora, pe parcursul transmiterii și stocării de către o sursă autorizată.

Operațiile care anulează integritatea datelor includ inserarea, eliminarea, reordonarea, substituția de biți și combinațiile acestora.

FUNCȚIILE HASH CRIPTOGRAFICE PROMOVEAZĂ INTEGRITATEA DATELOR

Funcția hash criptografică fără cheie realizează o compresie a mesajului la care este aplicată, astfel obținându-se un mesaj de lungime fixă.

Dacă se dorește asigurarea atât a confidențialității, cât și a integrității datelor, atunci se folosește următorul procedeu ce implică o funcție MDC h. Expeditorul mesajului x calculează valoarea hash $H = h(x)$ și o concatenează la mesaj, după care criptează șirul obținut în baza unui algoritm de criptare simetrică E și cheia k. Textul criptat generat $C := E_k(x \| h(x))$ este transmis destinatarului, care separă mesajul recuperat x' de hash-ul recuperat H'. Recipientul calculează valoarea hash $h(x')$ și o compară cu H'. Dacă acestea coincid, atunci mesajul recuperat este acceptat ca și autentic și integru. Această situație este descrisă în *Figura 1.1.2*.

FUNCȚIILE HASH CRIPTOGRAFICE PROMOVEAZĂ INTEGRITATEA DATELOR

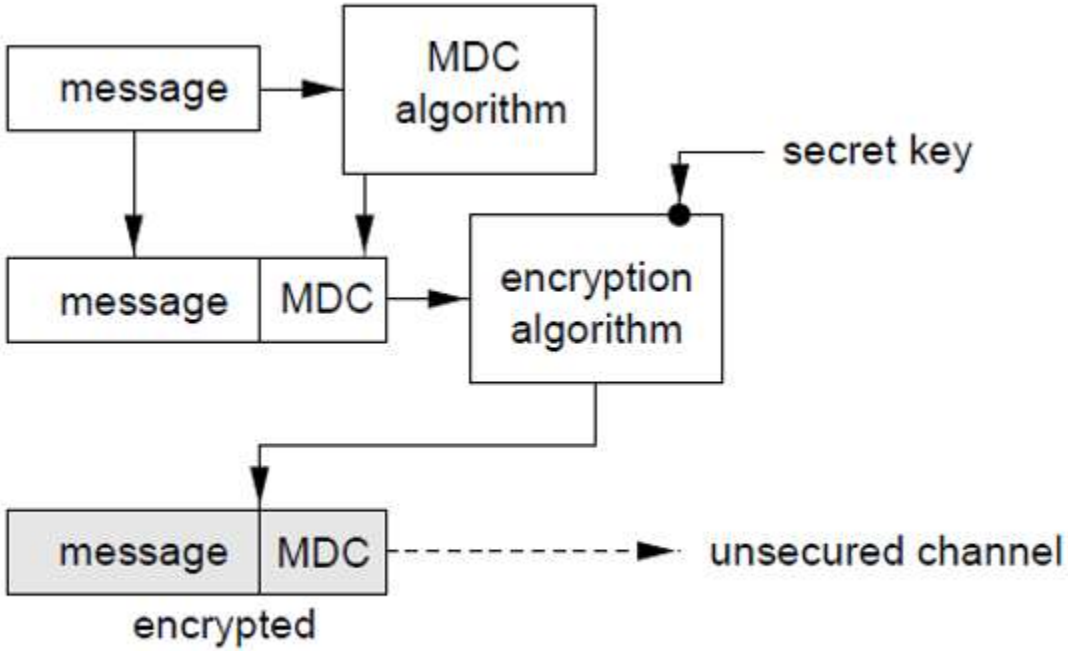


Figura 1.1.2. Asigurarea integrității datelor în baza schemei MDC și a criptării

FUNCȚIA HASH CRIPTOGRAFICĂ

Definiția 1.2.1. Funcția hash criptografică fără cheie (care mai este numită funcție de dispersie, de compresie sau amprentă digitală) este o funcție $h:\{0,1\}^* \to \{0,1\}^n$ ce satisface următoarele două proprietăți:

1. *Compresie* - h transformă un mesaj în reprezentare binară de lungime arbitrară finită într-un șir binar $h(x)$ de lungime fixă n.
2. *Eficiență de calcul* – fiind dată funcția h și un mesaj de intrare x, valoarea hash $h(x)$ se calculează eficient (în timp de calcul polinomial și în limita resurselor disponibile ale calculatorului).

FUNCȚIA HASH CRIPTOGRAFICĂ

Ideea de bază a unei funcții criptografice hash este că valoarea hash produsă servește ca și o amprentă compactă a mesajului inițial, care este utilizată ca și un identificator al mesajului.

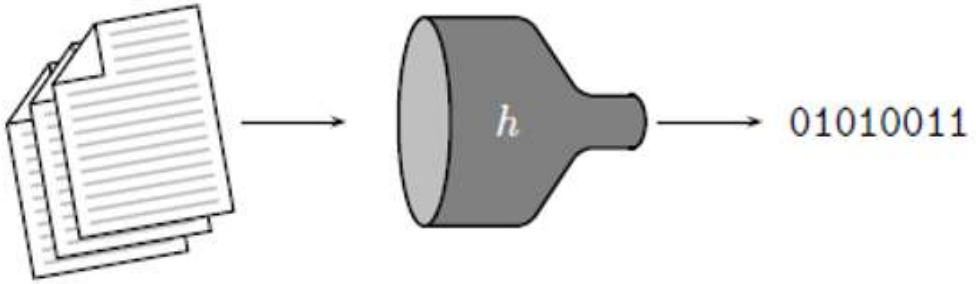
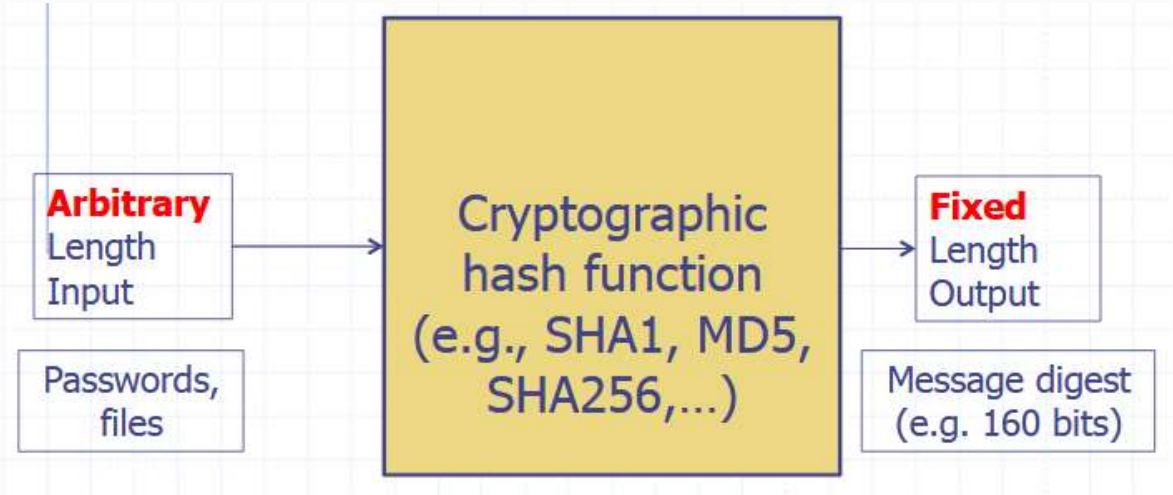


Figura 1.2.1. Funcția hash aplică un mesaj de lungime arbitrară într-un mesaj de lungime fixă



FUNCȚIA HASH CRIPTOGRAFICĂ

Pentru funcția hash h vom formula încă trei proprietăți de securitate pe lângă cele din definiție (compresie și eficiență de calcul). Vom considera că x și x' sunt două mesaje de intrare, iar y și y' - două valori hash.

- *Rezistența imaginii* (unidirecțională) – pentru o valoare hash dată y este computațional dificil de găsit un mesaj x', a cărui valoare hash să fie valoarea y, adică astfel încât $h(x') = y$.
- *Rezistența secundară a imaginii* (rezistență la coliziuni slabe) - fiind dat un mesaj x, este computațional dificil să se determine un mesaj $x' \neq x$ care să aibă aceeași valoare hash ca și x, adică astfel încât $h(x) = h(x')$.
- *Rezistența la coliziune* (rezistență la coliziuni tari) – este computațional dificil de găsit două mesaje distincte x și x' care să genereze aceeași valoare hash, adică astfel încât $h(x) = h(x')$.

ARGUMENTAREA PROPRIETĂȚILOR FUNCȚIEI HASH

Funcțiile hash sunt utilizate în cadrul schemelor de semnătură digitală cu scopul de a amplifica eficiența și securitatea schemelor. Mesajul de semnat este supus unei compresii, astfel obținându-se un mesaj de lungime fixă, iar în loc să se semneze mesajul x, se semnează valoarea hash h(x) a acestuia.

Este importantă satisfacerea proprietății de rezistență a imaginii pentru funcția hash, deoarece în caz contrar, fiind cunoscut y, adversarul C va fi capabil să determine mesajul x' astfel încât $h(x') = y$.

Vom cere ca funcția hash h să fie cu rezistență la coliziuni slabe, deoarece în caz contrar, un adversar C poate să obțină semnătura lui $h(x)$ a unei entități A, după care să determine un x' astfel încât $h(x) = h(x')$ și să afirme că entitatea A a semnat x'.

Dacă adversarul C poate să aleagă mesajul x pe care A îl semnează, atunci C trebuie doar să determine perechea coliziune (x, x') încât $h(x) = h(x')$. În acest caz, la fel este necesară rezistența la coliziuni tari a funcției hash h.

RELAȚII DINTRE PROPRIETĂȚILE FUNCȚIEI HASH

Funcţia hash cu coliziuni tari este și cu coliziuni slabe

Funcţia hash cu coliziuni tari poate să nu fie funcție hash cu rezistență a imaginii

Funcţia hash cu rezistență a imaginii poate să nu fie funcție hash cu coliziuni slabe

CLASE DE FUNCȚII HASH CRIPTOGRAFICE

Funcțiile hash se divizează în două clase: funcții hash fără cheie (acestea sunt definite în baza mesajului) și funcții hash cu cheie (definite în baza mesajului și a cheii secrete). Vom examina, în special, două tipuri de funcții hash:

1. Coduri de integritate a mesajelor, numite și coduri de detectare a modificării (MDC=Modification Detection Code), care sunt o subclasă a funcțiilor hash fără cheie și care, la rândul lor, se divizează în două categorii:

Funcții hash unidirecționale, care sunt funcții hash ce posedă suplimentar proprietățile de rezistență a imaginii și de rezistență secundară a imaginii.

Funcții hash rezistente la coliziune, care sunt funcții hash ce posedă suplimentar proprietățile de rezistență secundară a imaginii și de rezistență la coliziune.

2. Coduri de autentificare a mesajelor (MAC=Message Authentication Codes), care sunt o subclasă a funcțiilor hash cu cheie și pe care le vom examina în cadrul unei alte lecții.

FUNCȚII HASH ITERATIVE

În această secțiune este examinat un model general de construcție a funcției hash. Majoritatea funcțiilor hash fără cheie sunt concepute ca și procese iterative în cadrul cărora se prelucrează succesiv blocuri de lungime fixă construite în baza mesajului inițial (a se vedea Figura 1.2.3). Iterațiile implică o funcție de compresie (numită funcție hash iterativă) ce procesează blocurile mesajului „completat".

Astfel, mesajul inițial x este supus procedurii de *preprocesare*, care include 3 etape: completarea mesajului (procedură numită *padding*), divizarea mesajului completat în blocuri de aceeași lungime și setarea valorii hash inițiale.

FUNCȚII HASH ITERATIVE

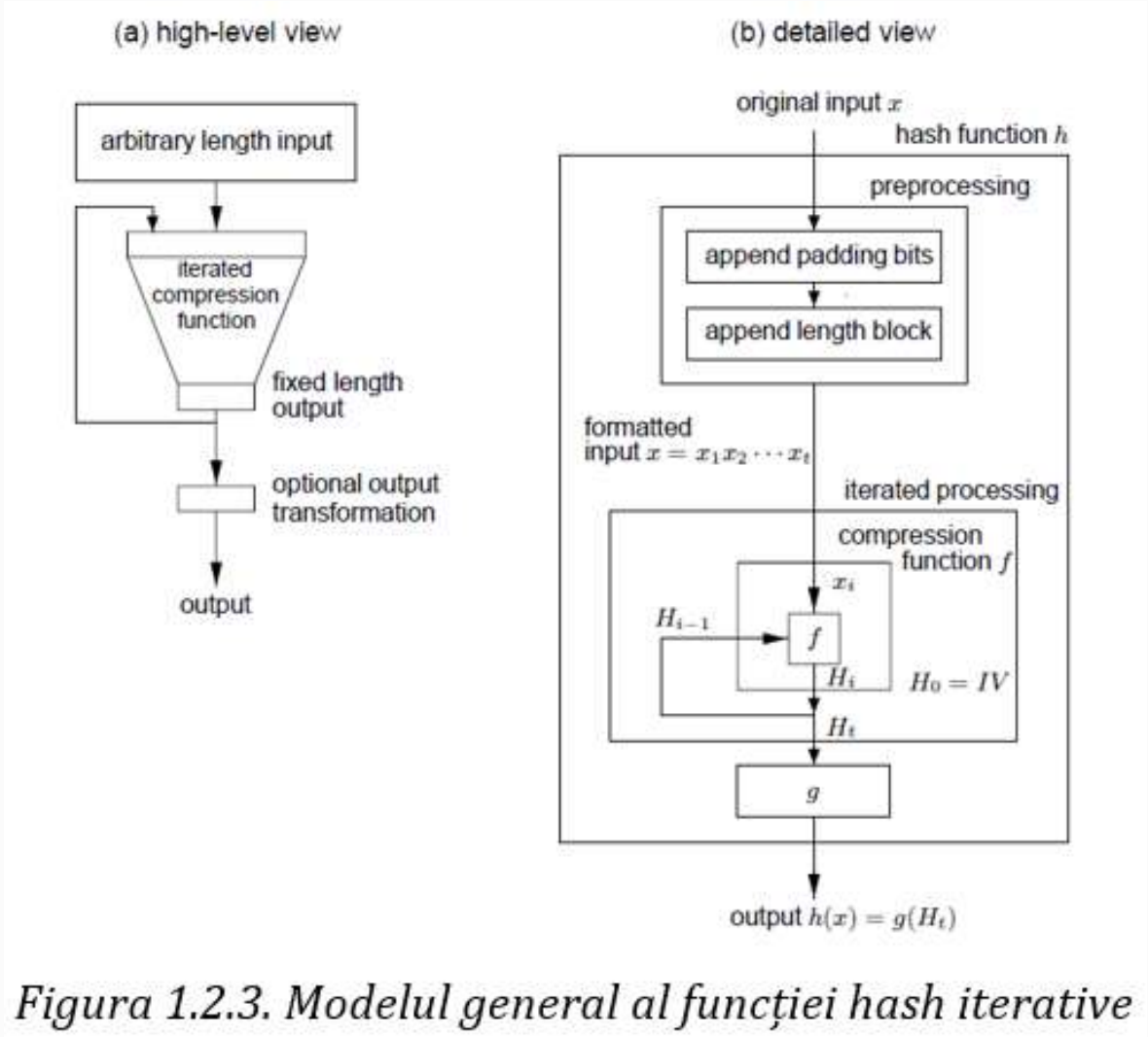


Figura 1.2.3. Modelul general al funcției hash iterative

FUNCȚII HASH ITERATIVE

Pentru a diviza mesajul inițial x (scris în cod binar) în blocuri x_i de aceeași lungime $n > 0$, acesta este completat cu biți suplimentari (procedura de padding) până lungimea șirului devine un multiplu al lui n. De regulă, completarea cu biți se face astfel: se concatenează bitul 1, după care urmează biți de zero până lungimea ultimului bloc devine egală cu n. În continuare, la mesajul completat prin procedura de padding, se anexează blocul lungime, care este format din reprezentarea binară a lungimii în biți a mesajului inițial, extinsă la stânga cu zerouri până se atinge numărul de biți într-un bloc. Procedura de concatenare a blocului lungime la mesajul completat este numită MD-fortificare (MD-strengthening), unde MD este un acronim pentru Merkle-Damgård.

O altă variantă de completare a mesajului constă în concatenarea (dacă este necesar) de biți de 0 la sfârșitul mesajului până lungimea acestuia devine un multiplu al lungimii de bloc.

FUNCȚII HASH ITERATIVE

Fie $M = M_1 ... M_t$ este mesajul obținut după aplicarea procedurilor de completare și de MD-fortificare. Fiecare bloc de mesaj M_i, $i = \overline{1,t}$, are lungimea n.

La etapa finală a preprocesării algoritmul precizează o valoare hash inițială $H_0 := IV$.

În continuare este aplicat procedeul iterativ ce implică funcția de compresie f (a se vedea Figura 1.2.3). Pornind de la valoarea inițială H_0, funcția f prelucrează, pe rând, fiecare bloc M_i de lungime n al mesajului obținut după etapa de preprocesare.

FUNCȚII HASH ITERATIVE

Fie H_i semnifică rezultatul parțial obținut după efectuarea iterației i. Atunci procesul iterativ construit pe baza funcției de compresie f, care prelucrează blocurile șirului $M = M_1 M_2 \ldots M_t$, este descris în modul următor:

$$H_0 = IV,\ H_i = f\left(H_{i-1}, M_i\right),\ i = \overline{1,t},\ h(x) = g\left(H_t\right).$$

O transformare de ieșire opțională g este utilizată la etapa finală pentru a transforma variabila H_t pe n biți într-un rezultat $g\left(H_t\right)$ pe m biți; deseori g este transformarea identică $g\left(H_t\right) = H_t$.

Funcțiile hash particulare se definesc în baza funcției de compresie selectate și a transformării de ieșire.

EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH

Amintim că o funcție $f : X \to Y$ este unidirecțională dacă pentru orice $x \in X$ se poate calcula eficient $f(x)$, în schimb, pentru orice $y \in Y$ dat este computațional dificil de aflat $x \in X$ astfel încât $y = f(x)$. O funcție este rezistentă la coliziune dacă este computațional dificil de aflat două valori $x_1, x_2 \in X$, $x_1 \neq x_2$, astfel încât $f(x_1) = f(x_2)$.

EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH

Funcția hash preia la intrare un mesaj de lungime arbitrară finită și întoarce un mesaj de careva lungime fixată. Funcția de compresie preia la intrare două mesaje de lungimi fixate și întoarce un mesaj de lungime fixată. De regulă, lungimea mesajului întors de către funcția de compresie este mai mică decât suma lungimilor celor două mesaje de la intrare. Prin aplicare repetată funcțiile de compresie unidirecționale permit construcția de funcții hash. Această construcție este bazată pe următorul rezultat:

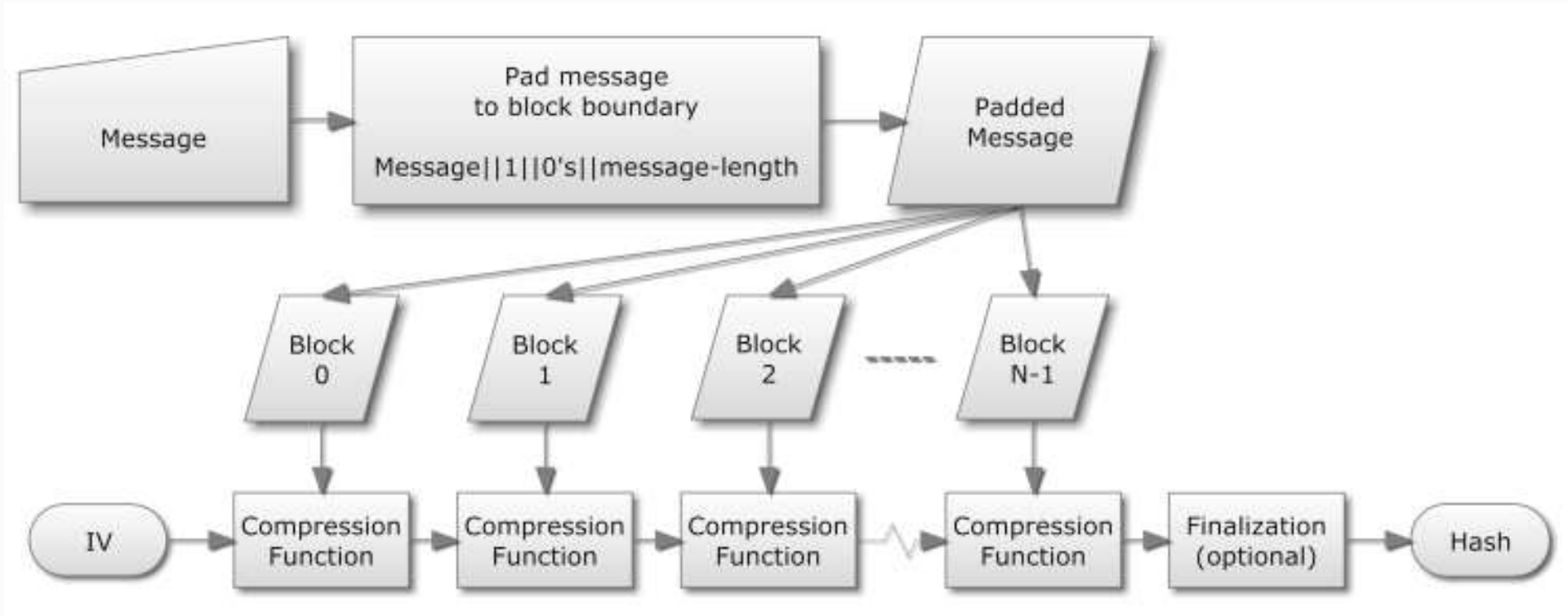
Propoziția 1.2.4. Orice funcție de compresie unidirecțională f rezistentă la coliziune poate fi extinsă la o funcție hash h rezistentă la coliziune.

EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH

Procedura de extindere menționată poate fi realizată eficient în baza meta-metodei Merkle (care mai poartă denumirea de construcție Merkle-Damgård) [2]. Această construcție este utilizată în proiectarea mai multor funcții hash, precum MD5, SHA1 și SHA2.

Deoarece funcția de compresie nu poate prelucra mesaje de lungime arbitrară, mesajul inițial este divizat pe blocuri de aceeași lungime (fixată în prealabil, de exemplu, de 512 sau 1024 biți), folosind procedura de completare și de MD-fortificare. În continuare, fiecare bloc al mesajului este prelucrat de către funcția de compresie unidirecțională. Mai exact, la fiecare pas se combină un bloc al mesajului cu ieșirea generată la pasul precedent.

EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH



EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH

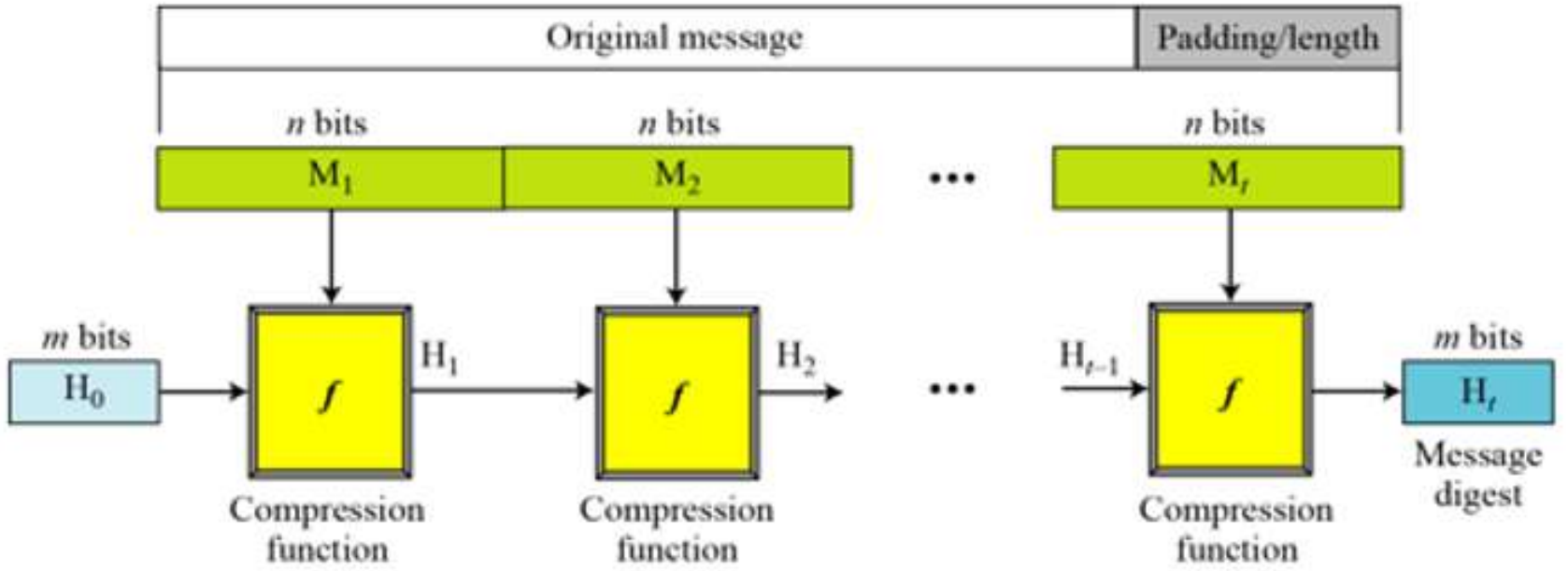


Figura 1.2.4. Construcția Merkle-Damgård

EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH

În practică sunt utilizate două variante de completare (padding) a mesajului inițial:

Algoritmul de padding (varianta 1)

Date de intrare:
- x – *șirul binar ce reprezintă mesajul inițial*
- n – *lungimea în biți a blocului de mesaj*

Date de ieșire:
- x' – șirul obținut în urma aplicării procedurii de padding, de lungime un multiplu al lui n

Pașii algoritmului:

1. Dacă este necesar, la șirul x se concatenează biți de 0 astfel încât să se obțină un șir x' a cărui lungime în biți este un multiplu al lui n.

EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH

Algoritmul de padding (varianta 2)

Date de intrare:

x – *șirul binar ce reprezintă mesajul inițial*

n – *lungimea în biți a blocului de mesaj*

Date de ieșire:

x' – șirul obținut în urma aplicării procedurii de padding, de lungime un multiplu al lui n

Pașii algoritmului:

1. Se concatenează la x un singur bit de 1.
2. Dacă este necesar, la șirul obținut la pasul 1 se concatenează biți de 0 astfel încât să se obțină un șir x' a cărui lungime în biți este un multiplu al lui n.

EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH

Prima variantă a algoritmului de padding are dezavantajul că cei mai din dreapta biți de 0 ai mesajului inițial nu pot fi delimitați de biții de 0 concatenați. În acest caz, este necesar să se cunoască lungimea mesajului inițial. Dezavantajul variantei a doua a algoritmului de padding constă în aceea că chiar dacă lungimea șirului inițial x este un multiplu al lui n, oricum șirul x' va conține un bloc suplimentar în raport cu x. În practică, completarea ultimului bloc conform algoritmului al doilea de padding este combinată cu procedura de MD-fortificare, astfel încât ultimul bloc de mesaj să includă atât padding-ul, cât și reprezentarea binară a lungimii de bloc.

EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH

Vom expune separat algoritmul de MD-fortificare, întrucât, în continuare, ne vom referi de mai multe ori la acesta.

Algoritmul de MD-fortificare (MD-strengthening)
Date de intrare: x - *mesajul inițial* n - *lungimea de bloc al mesajului* *Date de ieșire:* $M = M_1 \ldots M_t$ - șirul obținut după completarea și MD-fortificarea mesajului x
Pașii algoritmului: 1. La sfârșitul mesajului obținut după completare (procedura de padding) este concatenat un șir de biți ce reprezintă lungimea mesajului inițial x. Reprezentarea binară pentru lungimea mesajului inițial este completată la stânga cu biți de 0, astfel încât lungimea mesajului format (după completare și MD-fortificare) să fie un multiplu al lungimii de bloc n.

EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH

Astfel, metoda de extindere Merkle-Damgård reduce problema de aflare a funcției hash rezistente la coliziune la o problemă de aflare a funcției de compresie rezistente la coliziune. Funcția de compresie poate fi una dedicată (proiectată special pentru acest scop) sau este construită în baza unui cifru bloc.

Meta-metoda Merkle

Date de intrare:
- x – *mesajul inițial*
- f – funcția de compresie rezistentă la coliziune

Date de ieșire:
- h – funcția hash fără cheie rezistentă la coliziune

Pașii algoritmului:

1. Să admitem că f aplică mesaje de intrare pe $m+n$ biți în mesaje pe m biți (de exemplu, $m=128, n=512$). În baza lui f se construiește funcția hash h și valoarea hash corespunzătoare acesteia pe m biți (a se vedea Figura 1.2.4).

EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH

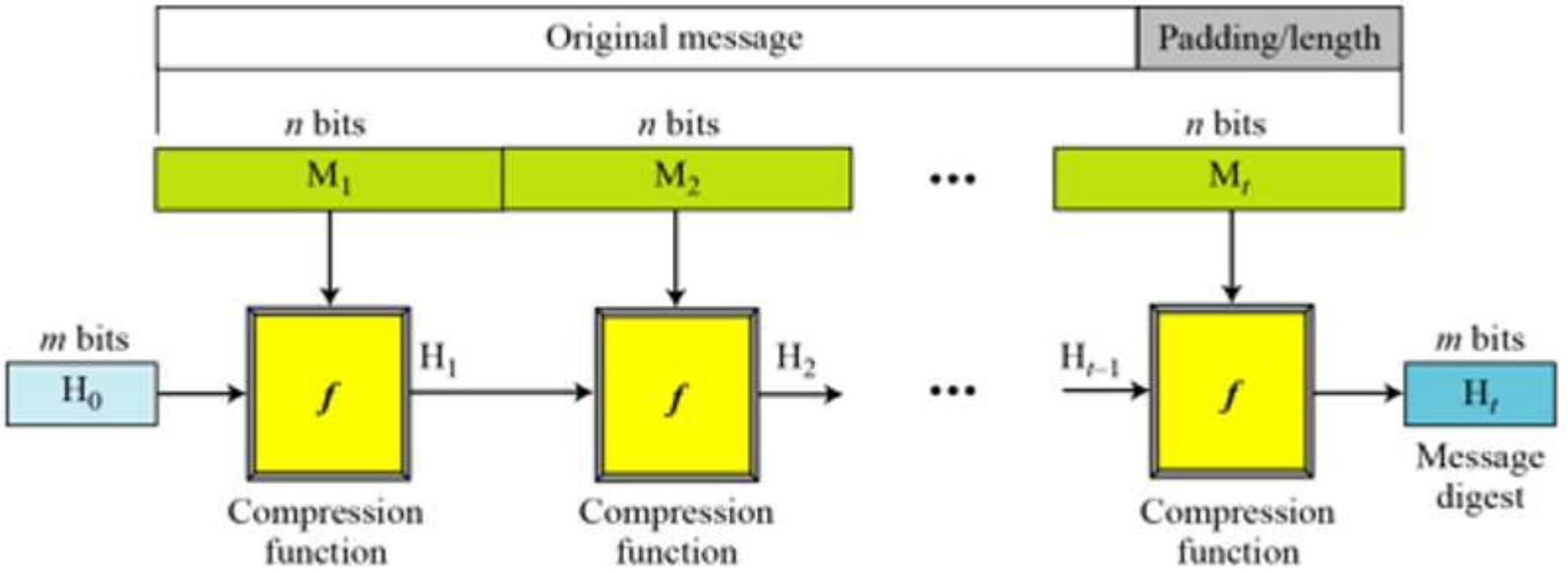


Figura 1.2.4. Construcția Merkle-Damgård

EXTINDEREA FUNCȚIILOR DE COMPRESIE LA FUNCȚII HASH

2. La mesajul inițial x se aplică procedurile de completare și de MD-fortificare, rezultând mesajul $M = M_1 ... M_t$ cu blocurile $M_i, i = \overline{1,t}$, de lungime n fiecare

3. Se definește valoarea hash pe m biți a lui x prin relația $h(x) = H_t$, în care H_t se definește prin formula de recurență:

$$H_0 = \{0\}^m, H_i = f(H_{i-1} \| M_i), i = \overline{1,t}.$$

Am notat prin $\{0\}^m$ șirul ce conține m biți de 0.

PROCEDEE DE CONSTRUCȚIE A MDC-URILOR

Vom analiza codurile MDC ca și o subclasă a funcțiilor hash fără cheie. Din punct de vedere structural, codurile MDC pot fi divizate în baza operațiilor esențiale utilizate în cadrul funcțiilor de compresie interioare. Astfel, din punctul de vedere menționat, trei categorii de funcții hash iterative sunt utilizate:

- funcții hash bazate pe cifruri bloc,
- funcții hash dedicate (customizate),
- funcții hash bazate pe aritmetica modulară.

Funcțiile hash dedicate sunt proiectate special pentru procedura de calcul al valorii hash, astfel ca să fie eficiente și independente de alte componente ale sistemului (operații adaptate posibilităților de calcul ale sistemului).

PROCEDEE DE CONSTRUCȚIE A MDC-URILOR

Primele construcţii ale funcţiilor hash fără cheie au fost bazate pe cifrurile bloc (precum DES). Performanța de calcul a acestor funcții hash nu era una prea bună (acestea erau de 2-4 ori mai lente ca cifrurile bloc corespunzătoare). Există funcții hash bazate pe aritmetica modulară, care, la fel, sunt lente și pentru ele există suspiciuni concrete cu privire la securitatea lor.

Cele mai populare funcții hash utilizate în aplicații sunt funcțiile hash customizate proiectate pe baza familiei MD4.

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE (DEDICATE)

Vom examina cele mai cunoscute funcții hash customizate, adică algoritmii care sunt special proiectați pentru procedura de calcul a amprentei digitale, astfel încât să ofere performanță de calcul fără a apela nemijlocit la alte programe software precum cifrurile bloc sau operații de aritmetică modulară.

Familia de algoritmi hash MD

R. Rivest a proiectat o serie de funcții hash denumite MD, acronim ce provine de la expresia „Message Digest", urmat de un număr, cele mai cunoscute fiind MD4, MD5 și MD6. Cei mai utilizați în practică algoritmi hash, precum MD5, SHA1, SHA2, RIPEMD și HAVAL reprezintă, de fapt, variante ale algoritmului MD4.

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

MD4 este o funcție hash iterativă ce operează la nivel de cuvinte pe 32 biți și generează o valoare hash pe 128 biți. Funcția de rundă (funcția de comprimare) preia la intrare o variabilă de legătură formată din 4 cuvinte și un bloc de mesaj din 16 cuvinte și generează în baza acestora o nouă variabilă de legătură. Toate operațiile sunt efectuate asupra cuvintelor pe 32 biți. Transformarea MD4 constă din 3 runde a câte 16 pași fiecare. La fiecare pas primul cuvânt al variabilei de legătură este combinat prin adunare cu un cuvânt al blocului de mesaj și un cuvânt generat cu o funcție neliniară ce depinde de celelalte 3 cuvinte ale variabilei de legătură. Rezultatul este supus unei operații de rotație ciclică cu un număr variabil de poziții. Transformarea descrisă este reversibilă. La fiecare rundă este utilizat o singură dată fiecare cuvânt al blocului de mesaj, dar ordinea de utilizare diferă de la o rundă la alta. După efectuarea celor 3 runde, este adăugată variabila de legătură precedentă la variabila de legătură nouă pentru a face funcția ireversibilă.

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

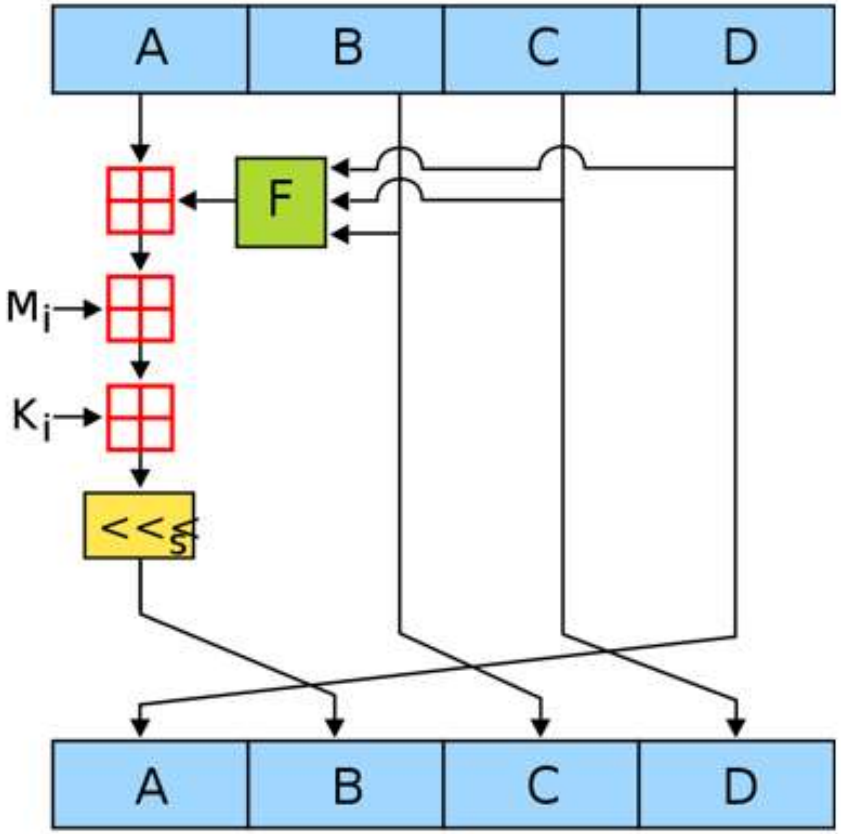


Figura 1.3.5. O operație MD4

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

Notație	*Semnificație*
u, v, w	cuvinte pe 32 biți
$0x67452301$	număr întreg pe 32 biți în format hexazecimal (cel mai puțin semnificativ octet este 01)
$+$	adunarea modulo 2^{32}
$not(u)$	complementare (negație) pe biți
$u \; and \; v$	conjuncție logică pe biți (și logic)
$u \; or \; v$	disjuncție logică pe biți (sau logic)
$u \oplus v$	disjuncție exclusivă pe biți (sau exclusiv)
$rol(u,s)$	rotație ciclică la stânga a lui u cu s biți; $rol(u,s) := (u << s) \, or \left(u >> card\left((u)_2\right) - s\right)$

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

Algoritmul MD4 utilizează următoarele funcții logice ce acceptă la intrare câte 3 cuvinte pe 32 biți și întorc un cuvânt pe 32 biți:

$f(u,v,w) = (u \ and \ v) \ or \ (not(u) \ and \ w)$ (multiplex);

$g(u,v,w) = (u \ and \ v) \ or \ (u \ and \ w) \ or \ (v \ and \ w)$ (majority);

$h(u,v,w) = u \oplus v \oplus w$ (exor).

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

Remarca 1.3.1. (little endian vs big endian) Ținând cont că implementările realizate pe diferite procesoare implică conversii între cuvintele pe 32 biți și grupurile de 4 octeți, vom specifica o convenție exactă de utilizare a acestora. Fie $B_1B_2B_3B_4$ un șir de 4 octeți care se va interpreta ca și un cuvânt pe 32 biți cu valoarea numerică W. În arhitecturile little endian (procesoare din familia 80x86) octetul cu adresa de memorie cea mai mică (B_1) este cel mai puțin semnificativ octet: $B_4 2^{24} + B_3 2^{16} + B_2 2^8 + B_1$, iar în arhitecturile big endian (procesoare 680x0 și RISC) B_1 este cel mai semnificativ octet: $B_1 2^{24} + B_2 2^{16} + B_3 2^8 + B_4$.

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

Algoritmul MD4

Date de intrare:

x - *șirul de biți de lungime arbitrară* $l \geq 0$, *pentru care se calculează valoarea hash*

Date de ieșire:

$h(x)$ - valoarea hash pe 128 biți a lui x

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

Pașii algoritmului:

1. *Definirea constantelor*

Se definesc 4 cuvinte pe 32 biți, $h_i, i=\overline{1,4}$, în reprezentare hexazecimală, care definesc variabila de legătură inițială IV:

$$h_1 = 0x67452301,\ h_2 = 0xefcdab89,\ h_3 = 0x98badcfe,\ h_4 = 0x10325476\ .$$

Se definesc constantele aditive pe 32 biți (în reprezentare hexazecimală):

$$y_j = 0x00000000,\ j=\overline{0,15}$$

$$y_j = 0x5a827999,\ j=\overline{16,31}\ .$$

$$y_j = 0x6ed9eba1,\ j=\overline{32,47}$$

Constanta $0x5a827999$ reprezintă primii 32 biți ai părții fracționare pentru $\sqrt{2}$, iar $0x6ed9eba1$ – primii 32 biți ai părții fracționare pentru $\sqrt{3}$.

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

Se definește ordinea de accesare a cuvintelor blocului de mesaj (fiecare listă conține valori de la 0 la 15):

$$z_j = j,\ j = \overline{0,15}$$

$$z_j = \mathrm{mod}\left(4\left(j-16\right),15\right),\ j = \overline{16,30},\ z_{31} = 15$$

$$z\left(32...47\right) = \left[0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15\right]$$

Pentru operația *rol* de rotație ciclică la stânga cu un număr de biți se definește numărul de poziții de biți pentru rotație:

$$s\left(0...15\right) = \left[3,7,11,19,3,7,11,19,3,7,11,19,3,7,11,19\right]$$

$$s\left(16...31\right) = \left[3,5,9,13,3,5,9,13,3,5,9,13,3,5,9,13\right] \quad .$$

$$s\left(32...47\right) = \left[3,9,11,15,3,9,11,15,3,9,11,15,3,9,11,15\right]$$

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

2. *Preprocesare*

Șirul de biți x este completat astfel încât lungimea acestuia să devină un multiplu al lui 512, unde 512 este lungimea blocului în biți. În acest sens, se concatenează un bit de 1, după care încă $r-1\,(\geq 0)$ biți de zero, unde r este cel mai mic întreg pentru care lungimea totală a șirului completat este cu 64 mai mică ca un multiplu al lui 512, adică $\mathrm{mod}(l+r,512)=\mathrm{mod}(448,512)$.

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

După procedura de padding urmează MD-fortificarea, conform căreia la șirul rezultat se concatenează reprezentarea pe 64 biți a numărului întreg $\mathrm{mod}(l, 2^{64})$, scrisă sub formă de două cuvinte pe 32 biți, cuvântul cel mai puțin semnificativ fiind scris primul (conversia între șirul de octeți și cuvântul pe 32 biți este little-endian). Fie m numărul de blocuri pe 512 biți ale șirului rezultat ($l + r + 64 = 512m = 32 \cdot 16m$), adică funcția iterativă se va executa de m ori. Ținând cont că se lucrează la nivel de cuvinte pe 32 biți, vom considera că șirul de intrare constă din $16m$ cuvinte pe 32 biți: $x_0 x_1 \ldots x_{16m-1}$.

La fel, se inițializează: $H_1 := h_1, H_2 := h_2, H_3 := h_3, H_4 := h_4$.

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

3. Procesarea

Pentru $i=\overline{0,m-1}$ se copiază blocul i de 16 cuvinte pe 32 biți în locația temporară $X_j := x_{16i+j}$, $j=\overline{0,15}$, după care aceasta se procesează în trei runde a câte 16 pași (pasul procesează un cuvânt al blocului de mesaj) înainte de a reînnoi variabila de legătură:

Se inițializează variabila de legătură: $A := H_1, B := H_2, C := H_3, D := H_4$.

Runda 1.

Pentru $j=\overline{0,15}$ execută

$t := A + f(B,C,D) + X_{z_j} + y_j;\ A := D;\ D := C;\ C := B;\ B := rol(t, s(j))$.

Runda 2.

Pentru $j=\overline{16,31}$ execută

$t := A + g(B,C,D) + X_{z_j} + y_j;\ A := D;\ D := C;\ C := B;\ B := rol(t, s(j))$.

Runda 3.

Pentru $j=\overline{32,47}$ execută

$t:=A+h(B,C,D)+X_{z_j}+y_j;\ A:=D;\ D:=C;\ C:=B;\ B:=rol(t,s(j))$.

Se reînnoiesc variabilele de legătură:

$$H_1:=H_1+A, H_2:=H_2+B, H_3:=H_3+C, H_4:=H_4+D.$$

4. Completarea

Valoarea hash $h(x)$ a mesajului inițial x este $H_1\|H_2\|H_3\|H_4$.

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

Dacă pentru funcțiile f, g, h și locația temporară X_j la etapa i folosim notațiile:

$$f(i,u,v,w) := f(u,v,w),\ i = \overline{0,15},$$

$$f(i,u,v,w) := g(u,v,w),\ i = \overline{16,31},$$

$$f(i,u,v,w) := h(u,v,w),\ i = \overline{32,47},$$

$$X_{ij},\ i = \overline{0,m-1},\ j = \overline{0,15},$$

atunci etapa de procesare a algoritmului MD4 se poate scrie sub forma:

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

$A := H_1, B := H_2, C := H_3, D := H_4$

Pentru $i = \overline{0, m-1}$ execută

{

Pentru $j = \overline{0, 47}$ execută

{

$t := A + f(j, B, C, D) + X_{i,z_j} + y_j;\ A := D;\ D := C;\ C := B;\ B := rol(t, s(j))$

}

$H_1 := H_1 + A, H_2 := H_2 + B, H_3 := H_3 + C, H_4 := H_4 + D$

}

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD4

Tabel cu valori hash pentru testarea algoritmului:

"The quick brown fox jumps over the lazy dog"	1bee69a46ba811185c194762abaeae90
„The quick brown fox jumps over the lazy cog"	b86e130ce7028da59e672d56ad0113df
„abc"	a448017aaf21d8525fc10ae87aa6729d
„ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"	043f8582f241db351ce627e153e7f0e4
„" (mesajul vid)	31d6cfe0d16ae931b73c59d7e0c089c0

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD5

În anul 1991 R. Rivest propune o versiune mai securizată a lui MD4, numită MD5 [6], și care a cunoscut o utilizare la scară largă în practică. Modificările ce au fost efectuate în algoritmul MD4 pentru a construi algoritmul MD5 sunt următoarele:

1. MD5 conține o rundă adițională, adică are 4 runde de câte 16 pași;
2. MD5 folosește o funcție multiplex în rundele 1 și 2 și o nouă funcție $k(u,v,w)$ în runda 4;
3. Este modificată ordinea în care sunt utilizate cuvintele blocului de mesaj în rundele 2 și 3 și este definită ordinea pentru runda 4;
4. Pentru a amplifica efectul de avalanșă au fost introduse 16 valori diferite pentru realizarea operației de rotație ciclică;
5. Fiecare din cei 4×16 pași conține o constantă aditivă unică, bazată pe partea întreagă a lui $2^{32}\sin(j)$ la pasul j (care necesită circa 256 de octeți de memorie adițională);
6. A fost modificat nucleul algoritmului. Se adaugă rezultatul pasului precedent la fiecare din cei 64 pași, mai exact, în rundele 1,2,3 (și 4) în loc de $B := rol(t, s(j))$ se utilizează $B := B + rol(t, s(j))$.

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD5

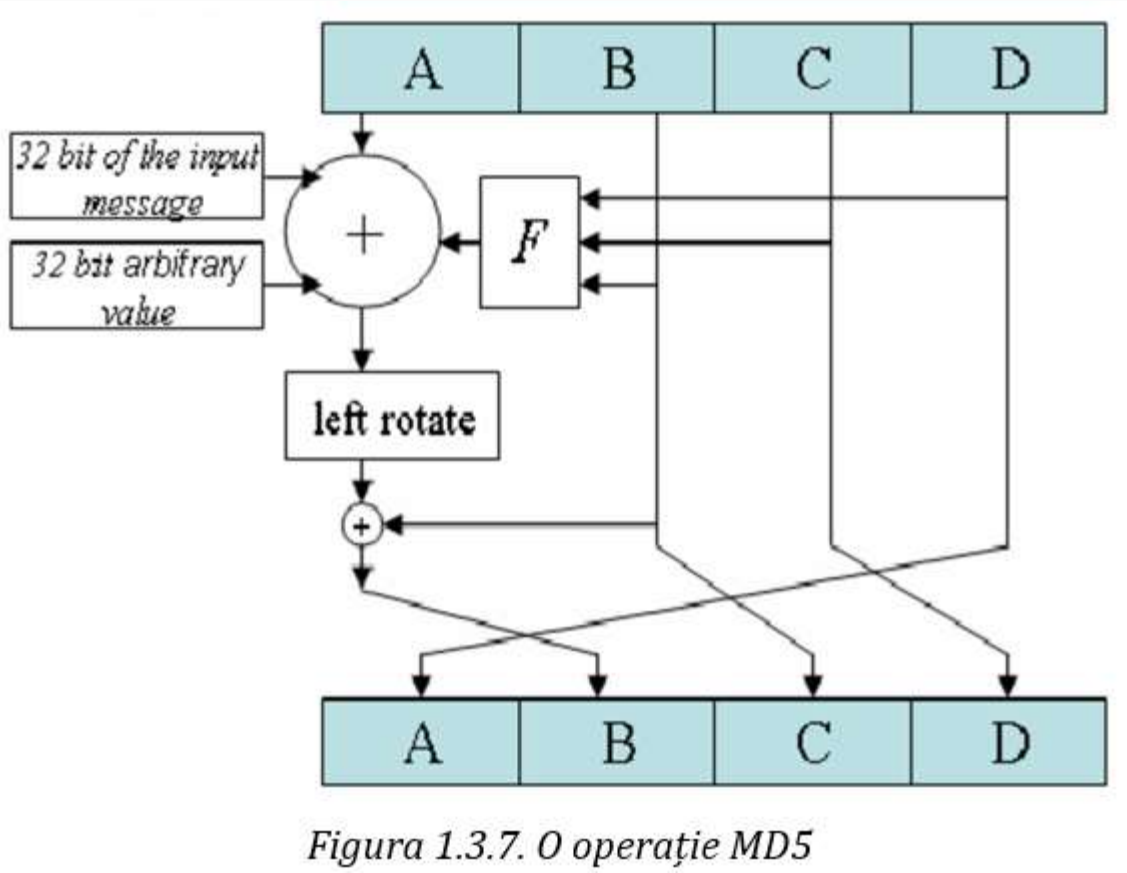
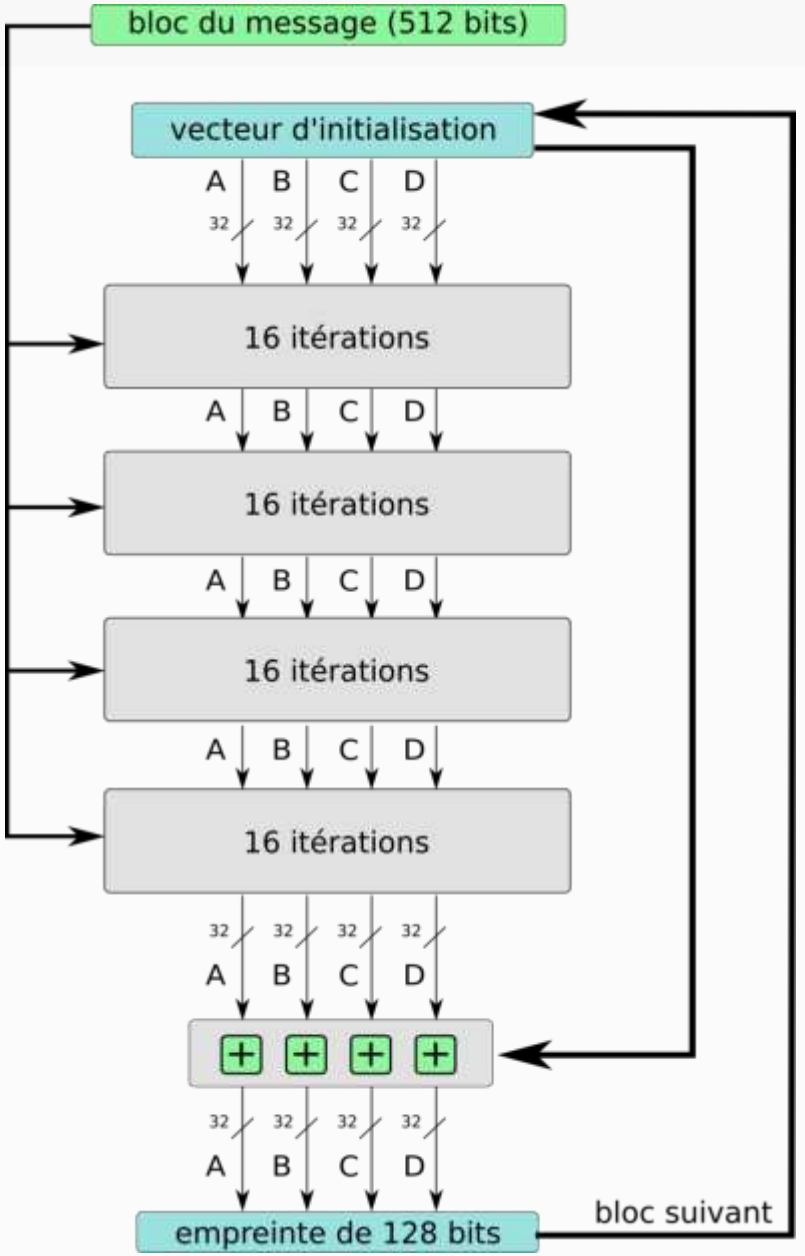


Figura 1.3.7. O operație MD5



FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD5

Funcțiile logice utilizate în cadrul algoritmului sunt următoarele:

$f(u,v,w) = (u \ and \ v) \ or \ (not(u) \ and \ w)$ (multiplex);

$g(u,v,w) = (u \ and \ w) \ or \ (v \ and \ not(w))$ (multiplex);

$h(u,v,w) = u \oplus v \oplus w$ (exor);

$k(u,v,w) = v \oplus (u \ or \ not(w))$ (-).

Algoritmul MD5

Date de intrare:

x - *șirul de biți de lungime arbitrară* $l \geq 0$, *pentru care se calculează valoarea hash*

Date de ieșire:

$h(x)$ - valoarea hash pe 128 biți a lui x

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD5

Pașii algoritmului:

1. *Definirea constantelor*

Se definesc 4 cuvinte pe 32 biți, $h_i, i=\overline{1,4}$, în reprezentare hexazecimală, care definesc variabila de legătură inițială IV:

$$h_1 = 0x67452301,\ h_2 = 0xefcdab89,\ h_3 = 0x98badcfe,\ h_4 = 0x10325476\ .$$

Se definesc constantele aditive pe 32 biți (în reprezentare hexazecimală):

Valorile $y_j, j=\overline{0,63}$, sunt obținute din primii 32 de biți ai lui $abs\left(\sin\left(j+1\right)\right), j=\overline{0,63}$ (adică $floor\left(2^{32}\cdot abs\left(\sin\left(j+1\right)\right)\right)$), unde abs este valoarea absolută.

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD5

Se definește ordinea de accesare a cuvintelor blocului de mesaj:

$$z_j = j,\ j = \overline{0,15},$$
$$z_j = \mathrm{mod}\left(1+5(j-16),16\right),\ j = \overline{16,31},$$
$$z_j = \mathrm{mod}\left(5+3(j-32),16\right),\ j = \overline{32,47},$$
$$z_j = \mathrm{mod}\left(7(j-48),16\right),\ j = \overline{48,63}.$$

Pentru operația *rol* de rotație ciclică la stânga cu un număr de biți se definește numărul de poziții de biți pentru rotație:

$$s(0...15) = [7,12,17,22,7,12,17,22,7,12,17,22,7,12,17,22],$$
$$s(16...31) = [5,9,14,20,5,9,14,20,5,9,14,20,5,9,14,20],$$
$$s(32...47) = [4,11,16,23,4,11,16,23,4,11,16,23,4,11,16,23],$$
$$s(48...63) = [6,10,15,21,6,10,15,21,6,10,15,21,6,10,15,21].$$

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD5

2. *Preprocesare*

Ca și în algoritmul MD4.

3. *Procesarea*

Pentru $i=\overline{0,m-1}$ se copiază blocul i de 16 cuvinte pe 32 biți în locația temporară $X_j := x_{16i+j}$, $j=\overline{0,15}$, după care aceasta se procesează în patru runde a câte 16 pași înainte de a reînnoi variabila de legătură:

Se inițializează variabila de legătură: $A := H_1, B := H_2, C := H_3, D := H_4$.

Runda 1.

Pentru $j=\overline{0,15}$ execută

$t := A + f(B,C,D) + X_{z_j} + y_j;\ A := D;\ D := C;\ C := B;\ B := B + rol(t, s(j))$.

Runda 2.

Pentru $j=\overline{16,31}$ execută

$t:=A+g(B,C,D)+X_{z_j}+y_j;\ A:=D;\ D:=C;\ C:=B;\ B:=B+rol(t,s(j))$.

Runda 3.

Pentru $j=\overline{32,47}$ execută

$t:=A+h(B,C,D)+X_{z_j}+y_j;\ A:=D;\ D:=C;\ C:=B;\ B:=B+rol(t,s(j))$.

Runda 4.

Pentru $j=\overline{48,63}$ execută

$t:=A+k(B,C,D)+X_{z_j}+y_j;\ A:=D;\ D:=C;\ C:=B;\ B:=B+rol(t,s(j))$.

Se reînnoiesc variabilele de legătură:

$$H_1:=H_1+A, H_2:=H_2+B, H_3:=H_3+C, H_4:=H_4+D.$$

4. Completarea

Valoarea hash $h(x)$ a mesajului inițial x este $H_1\|H_2\|H_3\|H_4$.

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD5

Dacă pentru funcțiile f, g, h, k și locația temporară X_j la etapa i folosim notațiile:

$$f(i,u,v,w) := f(u,v,w),\ i = \overline{0,15},$$
$$f(i,u,v,w) := g(u,v,w),\ i = \overline{16,31},$$
$$f(i,u,v,w) := h(u,v,w),\ i = \overline{32,47},$$
$$f(i,u,v,w) := k(u,v,w),\ i = \overline{48,63},$$

$$X_{ij},\ i = \overline{0,m-1},\ j = \overline{0,15},$$

atunci etapa de procesare a algoritmului MD5 se poate scrie sub forma:

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD5

$A := H_1, B := H_2, C := H_3, D := H_4$

Pentru $i = \overline{0, m-1}$ execută

{

Pentru $j = \overline{0, 63}$ execută

{

$t := A + f(j, B, C, D) + X_{i,z_j} + y_j;\ A := D;\ D := C;\ C := B;\ B := B + rol(t, s(j))$

}

$H_1 := H_1 + A, H_2 := H_2 + B, H_3 := H_3 + C, H_4 := H_4 + D$

}

FUNCȚII HASH FĂRĂ CHEIE CUSTOMIZATE. ALGORITMUL MD5

Tabel cu valori hash pentru testare:

"The quick brown fox jumps over the lazy dog"	9e107d9d372bb6826bd81d3542a419d6
„The quick brown fox jumps over the lazy cog"	1055d3e698d289f2af8663725127bd4b
„abc"	900150983cd24fb0d6963f7d28e17f72
„ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"	d174ab98d277d9f5a5611c2c9f419d9f
„" (mesajul vid)	d41d8cd98f00b204e9800998ecf8427e