# Capitolul 1

# Funcţii de dispersie

## 1.1 Noţiuni preliminare

O funcţie de dispersie (funcţie *hash* în engleză) este – în esenţă o funcţie de compresie al cărei principal scop este asigurarea integrităţii unui mesaj. Cu ajutorul ei se construieşte o "amprentă" a mesajului, care – prin verificare periodică – certifică dacă acesta a fost modificat sau nu. De remarcat că nu se pune problema confidenţialităţii mesajului, ci doar a integrităţii şi autenticităţii sale.
Formal:

**Definiţia 1.1.** *O clasă de dispersie este un quadruplu* $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ *unde:*

- $\mathcal{X}$ *este mulţimea mesajelor posibile (criptate sau nu);*
- $\mathcal{Y}$ *este o mulţime finită de "amprente" (sau "rezumate");*
- $\mathcal{K}$ *este o mulţime finită de chei;*
- *Pentru fiecare cheie* $K \in \mathcal{K}$ *există o funcţie de dispersie* $h_K \in \mathcal{H}$, $h_K : \mathcal{X} \longrightarrow \mathcal{Y}$.

Ideea de "compresie" vine din faptul că – deşi $\mathcal{X}$ poate fi o mulţime oricât de mare, $\mathcal{Y}$ este o mulţime finită. Vom avea totdeauna $card(\mathcal{X}) \geq card(\mathcal{Y})$. Aproape toate aplicaţiile vor solicita o condiţie mai tare:

$$card(\mathcal{X}) \geq 2 \cdot card(\mathcal{Y}).$$

O pereche $(x, y) \in \mathcal{X} \times \mathcal{Y}$ este *validă pentru cheia* $K$ dacă $h_K(x) = y$.

**Observaţia 1.1.** *Deoarece principalul scop urmărit este cel de integritate, compresia nu este o regulă generală aplicată tuturor funcţiilor de dispersie. Există funcţii de dispersie în care* $card(\mathcal{X}) = card(\mathcal{Y})$. *Din acest punct de vedere, orice sistem de criptare poate fi privit ca o funcţie de dispersie. Totuşi, în acest capitol vom trata numai funcţii de dispersie care verifică şi condiţia de compresie.*

Vom considera următoarele notaţii:

$$card(\mathcal{X}) = N, \quad card(\mathcal{Y}) = M, \quad \mathcal{F}^{\mathcal{X},\mathcal{Y}} = \{f \mid f : \mathcal{X} \longrightarrow \mathcal{Y}\}$$

Evident $card(\mathcal{F}^{\mathcal{X},\mathcal{Y}}) = M^N$.
Orice clasă de dispersie $\mathcal{F} \subseteq \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ se numeşte $(N, M)$ - clasă de dispersie.

În general, o clasă de dispersie este caracterizată complet de perechea $(\mathcal{K}, h_K)$.

Cazul cel mai simplu este când $card(\mathcal{K}) = 1$, deci există o singură cheie şi o singură funcţie de dispersie $h : \mathcal{X} \longrightarrow \mathcal{Y}$. Atunci se poate identifica clasa de dispersie cu funcţia de dispersie $h$.[1]

## 1.2 Securitatea funcţiilor de dispersie

### 1.2.1 Funcţii criptografice de dispersie

Vom considera situaţia unei singure funcţii de dipersie $h : \mathcal{X} \longrightarrow \mathcal{Y}$. Din punct de vedere al securităţii (criptografice), unica modalitate de a obţine o pereche validă $(x, y) \in \mathcal{X} \times \mathcal{Y}$ este de a alege $x$ şi apoi de a calcula $y = h(x)$.
Acest lucru este implicat de condiţia ca următoarele 3 probleme să fie dificile.

- **NIP** (Non-inversabilă): Fiind dat $y \in \mathcal{Y}$ este dificil de aflat $x \in \mathcal{X}$ astfel ca $y = h(x)$.
- **CSP** (Coliziuni slabe): Fiind dată o pereche validă $(x, y)$, este dificil de aflat $x_1 \neq x$ cu $h(x_1) = h(x)$.
- **CP** (Coliziuni): Este dificil de aflat două valori distincte $x, x_1 \in \mathcal{X}$ astfel ca $h(x) = h(x_1)$.

Prin "*dificil*" se înţelege că pentru rezolvarea problemei respective nu se cunosc decât algoritmi de complexitate mare (eventual problema este $\mathcal{NP}$ - completă).

De remarcat că problema **CP** nu conduce direct la o pereche validă. Dar, dacă $(x, y)$ este o pereche validă şi $(x, x_1)$ este o coliziune, atunci $(x_1, y)$ este de asemenea o pereche validă.

**Definiţia 1.2.** *O funcţie de dispersie $h : \mathcal{X} \longrightarrow \mathcal{Y}$ pentru care problema* **CP** *este dificilă se numeşte "rezistentă la coliziuni"*

Deoarece $N > M$, orice funcţie de dispersie admite coliziuni. Totul este ca aceste coliziuni să nu poată fi găsite decât extrem de greu, neavând la îndemână decât algoritmi de complexitate mare.

Evident, dacă o funcţie este rezistentă la coliziuni, atunci ea va rezista şi la coliziuni slabe. Deci **CP** implică **CSP**.

[1] În literatură, acest caz se numeşte "unkeyed hash function".

**Teorema 1.1.** *Fie $h : \mathcal{X} \longrightarrow \mathcal{Y}$ o funcţie de dispersie, unde $\mathcal{X}, \mathcal{Y}$ sunt mulţimi finite, $N = card(\mathcal{X})$, $M = card(\mathcal{Y})$, $N \geq 2M$. Să presupunem că există un algoritm $\mathbf{A}$ de inversare a lui $h$. Atunci va exista un algoritm probabilist Las Vegas care găseşte o coliziune pentru $h$ cu probabilitate cel puţin $\frac{1}{2}$.*

*Demonstraţie.* Fie $\mathbf{A}$ algoritmul de inversiune pentru $h$, de forma unui oracol $\mathbf{A}$ care admite la intrare o amprentă $y \in \mathcal{Y}$ şi întoarce un element $\mathbf{A}(y) \in X$ astfel ca $h(\mathbf{A}(y)) = y$.

Să considerăm algoritmul următor (notat cu $\mathbf{B}$):

1. Fie $x \in \mathcal{X}$ ales aleator;
2. $y \longleftarrow h(x)$;
3. $x_1 \longleftarrow \mathbf{A}(y)$
4. **if** $x_1 \neq x$ **then** $x_1$ şi $x$ formează o coliziune pentru $h$ (succes)
   **else** sfârşit (eşec)

$\mathbf{B}$ este un algoritm probabilist de tip *Las Vegas*, deoarece întoarce o coliziune sau nici un răspuns. Deci, este suficient să calculăm probabilitatea sa de succes.

Pentru orice $x, x_1 \in \mathcal{X}$ se defineşte $x \sim x_1$ dacă $h(x) = h(x_1)$.

Este uşor de demonstrat că $\sim$ este relaţie de echivalenţă. Definim

$$[x] = \{x_1 \mid x_1 \in \mathcal{X}, \ \ x \sim x_1\}$$

Fiecare clasă de echivalenţă $[x]$ este formată din mesaje care produc aceeaşi amprentă ca şi $x$. Numărul de clase de echivalenţă este deci cel mult $M$.
Fie $\mathcal{C} = \mathcal{X}/\sim$ mulţimea claselor de echivalenţă.

Presupunem că $x$ este ales aleator din $\mathcal{X}$ la pasul 1. Pentru acest mesaj pot apare (la pasul 3) $n = card([x])$ valori $x_1$ posibile. $n - 1$ din ele sunt diferite de $x$ şi duc la reuşita algoritmului în pasul 4.
Pentru o alegere particulară a lui $x$, probabilitatea de succes este deci $\dfrac{n-1}{n}$.

Probabilitatea de succes a algoritmului este calculată ca medie peste toate alegerile posibile ale lui $x$:

$$Pr_{succes} = \frac{1}{N}\sum_{x\in\mathcal{X}}\frac{n-1}{n} = \frac{1}{N}\sum_{c\in\mathcal{C}}\sum_{x\in c}\frac{card(c)-1}{card(c)} = \frac{1}{N}\sum_{c\in\mathcal{C}}(card(c)-1)$$
$$= \frac{1}{N}\left(\sum_{c\in\mathcal{C}} card(c) - \sum_{c\in\mathcal{C}} 1\right) \geq \frac{N-M}{N} \geq \frac{N-N/2}{N} = \frac{1}{2}$$

Probabilitatea de succes este deci cel puţin $\frac{1}{2}$. □

Din această teoremă se deduce:

**Lema 1.1. CP** *implică* **NIP**.

Rezultă că este suficient ca o funcţie de dispersie să fie cu coliziuni tari, aceasta implicând celelalte două proprietăţi de securitate.
De aceea, orice funcţie de dispersie trebuie să satisfacă problema **CP**.

**Definiţia 1.3.** *O funcţie de dispersie care verifică* **CP** *se numeşte "funcţie criptografică de dispersie".*

### 1.2.2 Atacul naşterilor

O altă condiţie de securitate – care impune o margine inferioară pentru $M = card(\mathcal{Y})$ – rezultă dintr-o metodă simplă de obţinere a coliziunilor, numită *atacul naşterilor*; numele provine de la *paradoxul naşterilor*. Sub o formă simplificată, acesta este:

| Probabilitatea ca dintr-o mulţime aleatoare de 23 persoane să existe doi indivizi cu aceeaşi zi de naştere, este cel puţin 1/2. |
|---|

*Demonstraţie.* Fie $\mathcal{X}, \mathcal{Y}$ două mulţimi finite, $card(\mathcal{X}) = N,\ card(\mathcal{Y}) = M; N \geq 2M$ şi fie $h : \mathcal{X} \longrightarrow \mathcal{Y}$ o funcţie de dispersie.
Este uşor de arătat că există cel puţin $N$ coliziuni, dar problema este de a le pune în evidenţă.
O metodă simplă constă în a extrage aleator $k$ mesaje distincte $x_1, \ldots, x_k \in \mathcal{X}$, de a calcula $y_i = h(x_i),\ 1 \leq i \leq k$ şi de a căuta dacă există o coliziune printre ele (făcând, de exemlu, o triere printre valorile $y_i$).

Cum $x_i$ sunt extrase aleator, se poate considera că şi $y_i$ sunt elemente aleatoare (nu neapărat distincte) din $\mathcal{Y}$. Probabilitatea ca din $k$ extrageri, toate elementele $y_i$ să fie distincte, se determină astfel:

Se face o ordonare a numerelelor $y_i$; fie $y_1, \ldots, y_k$ această ordine.
Prima extragere $y_1$ este total aleatoare; probabilitatea ca $y_2 \neq y_1$ este $1 - 1/N$, probabilitatea ca $y_3$ să fie distinct de $y_1$ şi $y_2$ este $1 - 2/N$ etc.

Probabilitatea să nu avem $k$ coliziuni va fi deci:

$$\left(1 - \frac{1}{N}\right)\left(1 - \frac{2}{N}\right)\ldots\left(1 - \frac{k-1}{N}\right) = \prod_{i=1}^{k-1}\left(1 - \frac{i}{N}\right)$$

Dacă $x$ este un număr real suficient de mic, atunci se poate folosi aproximarea (rezultată din dezvoltarea în serie *Mac Laurin*) $1 - x \approx e^{-x}$. Vom obţine atunci:

$$\prod_{i=1}^{k-1}\left(1 - \frac{i}{N}\right) \approx \prod_{i=1}^{k-1} e^{\frac{-i}{N}} = e^{\frac{-k(k-1)}{2N}}.$$

Probabilitatea să existe cel puţin o coliziune este atunci $1 - e^{\frac{-k(k-1)}{2N}}$.
Notând această probabilitate cu $\epsilon$, se obţine ecuaţia de necunoscută $k$ : $e^{\frac{-k(k-1)}{2N}} \approx 1-\epsilon$, care conduce la $k^2 - k \approx 2N \cdot ln\frac{1}{1-\epsilon}$.

Dacă se ignoră termenul $-k$, se obţine $k \approx \sqrt{2N \cdot ln\frac{1}{1-\epsilon}}$.

Luând acum $\epsilon = 0.5$, se ajunge la $k \approx 1.17\sqrt{N}$.

De remarcat că alte valori pentru $\epsilon$ conduc la valori diferite pentru $k$, dar multiplicate totdeauna cu un factor constant $\sqrt{N}$.

Dacă $\mathcal{X}$ este mulţimea fiinţelor umane, $Y$ este mulţimea celor 365 zile dintr-un an (nebisect) iar $h(x)$ reprezintă data de naştere a persoanei $x$, se obţine paradoxul naşterilor. În aproximarea de sus, cu $N = 365$ avem $k \approx 22.5$.
Deci, printre 23 persoane alese întâmplător, probabilitatea ca două să aibă aceeaşi zi de naştere este $1 - 1/2$. □

**Atacul naşterilor** este un atac prin forţă brută care urmăreşte aflarea unor coliziuni. În general, *Oscar* generează aleator mesajele $x_1, x_2, \ldots \in \mathcal{X}$. Pentru fiecare mesaj $x_i$ el calculează şi stochează amprenta $y_i = h(x_i)$, comparând-o cu valorile stocate anterior.[2] Dacă $h(x_i)$ coincide cu o valoare $h(x_j)$ deja stocată, *Oscar* a găsit o coliziune $(x_i, x_j)$. Conform paradoxului naşterilor, aceasta se va întâmpla după aproximativ $2^{N/2}$ mesaje.

Acest tip de atac impune – ca măsură de securitate – o margine inferioară asupra lungimii amprentelor.
O amprentă de 40 biţi este vulnerabilă la un atac al naşterilor folosind numai $2^{20}$ (aproximativ un milion) mesaje aleatoare.
Se sugerează de aceea folosirea de amprente de cel puţin 256 biţi (aici atacul naşterilor va cere $2^{128}$ calcule).

**Exemplul 1.1.** *Unele sisteme de semnătură digitală (care vor fi studiate în capitolul următor) se aplică unei amprente $h(x)$ a mesajului $x$. Să presupunem că Oscar şi Bob semnează un contract.*

1. *Oscar pregăteşte două versiuni ale contractului: o versiune $m_1$ echitabilă, şi o versiune $m_2$ avantajoasă lui Oscar – ambele de dimensiune $n$.*

2. *Apoi generează $\mathcal{O}(2^{n/2})$ variante minore ale celor două versiuni, până obţine două mesaje $m_1'$ (similar lui $m_1$) şi $m_2'$ (similar cu $m_2$) având $h(m_1') = h(m_2')$.*
   *De exemplu, dacă $m_1$ conţine o figură (bitmap sau jpeg), modificările sunt insesizabile.*

3. *Cei doi semnează $h(m_1')$.*

*Ulterior Oscar poate pretinde că semnătura se referă la contractul $m_2'$.*

[2] În practică, mesajele sunt generate folosind un generator de numere pseudo-aleatoare. Deci mesajele pot fi recalculate oricând, nefiind necesară decât stocarea amprentelor.

### 1.2.3 Aplicaţii ale funcţiilor de dispersie

Funcţiile de dispersie sunt utilizate într-o paletă largă de aplicaţii criptografice. Listăm o mică parte din ele:

- *Detectare de viruşi*:

  Scopul este detectarea modificării unui fişier.
  - Pentru un fişier $F$ se calculează $c = h(F)$, unde $h$ este o funcţie de dispersie criptografică (este suficient să satisfacă **CSP**).
  - Valoarea $c$ este trimisă printr-un canal sigur (de exemplu o dischetă sau un CD) utilizatorului care suspectează coruperea fişierului primit.
  - Acesta compară $c$ cu $h(F')$, unde $F'$ este fişierul primit.

- *Distribuţia de Software/Chei Publice*:

  Scopul este descărcarea de către utilizator a copiei "adevărate" de soft sau cheie publică.
  Ideea este similară cu cea anterioară.
  Utilizatorul obţine – via un canal securizat – amprenta produsului soft (sau a cheii publice) şi o compară cu amprenta produsă de el pentru acelaşi produs descărcat de pe Internet.

- *Stocări în baze de date nesigure*

  - Utilizatorul construieşte o tabelă cu amprente ale paginilor/documentelor, pe care le stochează într-o bază de date nesecurizată.
  - Apoi, când utilizează aceste pagini/documente, recalculează şi verifică amprentele lor la fiecare descărcare pe calculatorul propriu.

- *Securizarea referinţelor şi pointerilor*:

  - Orice referinţă la o pagină web este amprentată şi verificată atunci când se face apel la adresa respectivă.
  - Acelaşi procedeu se aplică şi $URL$-urilor.

- *Datare*:

  Toate fişierele sau documentele sunt amprentate, după care li se asociază o "ştampilă de timp" de către o unitate autorizată[3].

- *Parole One-time(S - Chei)*:

  - Plecând de la o valoare iniţială $x_0$, se generează o secvenţă de amprente:

$$x_0 \xrightarrow{h(x_0)} x_1 \xrightarrow{h(x_1)} x_2 \cdots \xrightarrow{h(x_{n-1})} x_n$$

[3]Detalii despre *Time Stamping* sunt prezentate în Capitolul 3.

- Sistemul original stochează $x_n$, iar utilizatorul foloseşte $x_{n-1}$ ca parolă.
- Sistemul verifică $h(x_{n-1}) \stackrel{?}{=} x_n$. *Oscar* nu poate afla $x_{n-1}$ chiar dacă ştie $x_n$, deoarece $h$ este o funcţie de dispersie criptografică.
- În continuare, sistemul stochează $x_{n-1}$, iar utilizaotrul foloseşte ca parolă $x_{n-2}$.

- *Generatori de numere pseudoaleatoare*:

  Unele funcţii de dispersie asigură amprentelor o componentă aleatoare semnificativă, ceea ce le face utile – în anumite cazuri – pentru generarea de secvenţe de numere pseudoaleatoare.

- *Coduri de autentificare a mesajelor (MAC)*:

  Vor fi prezentaţi detaliat în Capitolul 3.

## 1.3 Funcţia de dispersie Chaum-van Heijst-Pfitzmann

Funcţia de dispersie prezentată în această secţiune a fost definită în [1]. Ea nu este suficient de rapidă pentru aplicaţii practice dar constituie un exemplu foarte simplu de funcţie de dispersie care admite o probă de securitate pe o ipoteză rezonabilă.

Fie $p$ un număr prim mare astfel ca $q = (p-1)/2$ să fie de asemenea prim.

Fie $\alpha, \beta \in Z_p$ două elemente primitive.

Valoarea $log_\alpha \beta$ nu este publică şi se presupune că este dificil de obţinut.

Funcţia de dispersie *Chaum-van Heijst-Pfitzmann* $h : Z_q \times Z_q \longleftarrow Z_p^*$ este definită prin:

$$h(x_1, x_2) = \alpha^{x_1}\beta^{x_2} \ (mod\ p)$$

**Teorema 1.2.** *Fiind dată o coliziune pentru funcţia de dispersie Chaum-van Heijst-Pfitzmann, calculul lui $log_\alpha \beta$ este uşor.*

*Demonstraţie.* Să presupunem că a apărut o coliziune

$$h(x_1, x_2) = h(x_3, x_4) \text{ cu } (x_1, x_2) \neq (x_3, x_4)$$

Avem deci $\alpha^{x_1}\beta^{x_2} \equiv \alpha^{x_3}\beta^{x_4} \ (mod\ p)$, sau $\alpha^{x_1 - x_3} \equiv \beta^{x_4 - x_2} \ (mod\ p)$.

Vom nota $d = (x_4 - x_2, p-1)$.

Cum $p - 1 = 2q$ cu $q$ număr prim, rezultă $d \in \{1, 2, q, p-1\}$.

Să trecem în revistă fiecare din aceste patru posibilităţi.

- $\mathbf{d = 1}$ : Notând $y = (x_4 - x_2)^{-1} \ (mod\ p-1)$, vom avea

  $$\beta \equiv \beta^{(x_4 - x_2)y} \ (mod\ p) \equiv \alpha^{(x_1 - x_3)y} \ (mod\ p)$$

  şi se poate calcula logaritmul discret $log_\alpha \beta = (x_1 - x_3)(x_4 - x_2)^{-1} \ (mod\ p-1)$.

- $\mathbf{d = 2}$: Deoarece $p - 1 = 2q,\ q$ prim, deducem $(x_4 - x_2, q) = 1$.

  Dacă notăm $y = (x_4 - x_2)^{-1}\ (mod\ q)$, atunci $(x_4 - x_2)y = kq + 1$ pentru un număr întreg $k$.
  Calculând, se obţine

  $$\beta^{(x_4-x_2)y} \equiv \beta^{kq+1}\ (mod\ p) \equiv (-1)^k\beta\ (mod\ p) \equiv \pm\beta\ (mod\ p)$$

  deoarece $\beta^q \equiv -1\ (mod\ p)$.

  Deci, $\alpha^{(x_1-x_3)y} \equiv \beta^{(x_4-x_2)y}\ (mod\ p) \equiv \pm\beta\ (mod\ p)$.

  De aici se deduce $log_\alpha\beta = (x_1 - x_3)(x_4 - x_2)^{-1}\ (mod\ p - 1)$, sau

  $$log_\alpha\beta = (x_1 - x_3)(x_4 - x_2)^{-1} + q\ (mod\ p - 1).$$

  Se poate verifica uşor care din aceste două rezultate este cel corect.

- $\mathbf{d = q}$: Din $0 \le x_2 \le q - 1,\ 0 \le x_4 \le q - 1$ rezultă $-(q - 1) \le x_4 - x_2 \le q - 1$.
  Este deci imposibil să avem $(x_4 - x_2, p - 1) = q$.

- $\mathbf{d = p - 1}$: Această variantă este posibilă numai dacă $x_4 = x_2$. Avem atunci $\alpha^{x_1}\beta^{x_2} \equiv \alpha^{x_3}\beta^{x_2}\ (mod\ p),\ $ deci $\alpha^{x_1} \equiv \alpha^{x_3}\ (mod\ p)$, de unde rezultă $x_1 = x_3$.
  S-a ajuns la $(x_1, x_2) = (x_3, x_4)$, ceea ce contrazice ipoteza.

În concluzie, $h$ este cu coliziuni tari, depinzând de calculul lui $log_\alpha\beta$ în $Z_p$. □

**Exemplul 1.2.** *Să luăm* $p = 12347$, *deci* $q = 6173,\ \alpha = 2,\ \beta = 8461$.
*Presupunem că s-a găsit coliziunea*

$$\alpha^{5692}\beta^{144} \equiv \alpha^{212}\beta^{4214}\ (mod\ 12347).$$

*Deci* $x_1 = 5692,\ x_2 = 144,\ x_3 = 212,\ x_4 = 4214$. *Avem* $(x_4 - x_2, p - 1) = 2$
*şi se începe calculul:*

$$y = (x_4 - x_2)^{-1}\ (mod\ q) = (4214 - 144)^{-1}\ (mod\ 6173) = 4312$$

*Calculăm apoi*

$$y' = (x_1 - x_3)y\ (mod\ p - 1) = (5692 - 212)4312\ (mod\ 12346) = 11862.$$

*Se obţine* $log_\alpha\beta \in \{y', y' + q\ (mod\ p - 1)\}$.
*Într-adevăr,*

$$\alpha^{y'}\ (mod\ p) = 2^{11862}\ (mod\ 12346) = 9998,$$

*deci*

$$log_\alpha\beta = y' + q\ (mod\ p - 1) = 11862 + 6173\ (mod\ 12346) = 5689.$$

*Se verifică imediat că* $2^{5689} \equiv 8461\ (mod\ 12347)$.

## 1.4 Construcţia Merkle - Damgard a unei funcţii de dispersie

Vom prezenta o modalitate de construcţie a unei funcţii de dispersie, bazată pe compresie. În particular, ea permite extensia funcţiei de dispersie pe un domeniu infinit, păstrând proprietăţile de securitate; avantajul unei asemenea situaţii este acela că se pot amprenta mesaje de lungime arbitrară.

Construcţia Merkle - Damgard se referă la mesaje peste un alfabet binar $Z_2 = \{0, 1\}$.

Fie $h : Z_2^m \longrightarrow Z_2^t$ o funcţie criptografică de dispersie, cu $m \geq t + 1$. Ea face de fapt o compresie a blocurilor de $m$ biţi în blocuri de $t$ biţi.

Scopul construcţiei este realizarea unei funcţii de dispersie $h^* : \mathcal{X} \longrightarrow Z_2^t$, unde

$$\mathcal{X} = \bigcup_{i=m}^{\infty} Z_2^i$$

Să considerăm la început cazul $m \geq t + 2$.

Fiecare element al lui $\mathcal{X}$ este reprezentat printr-un şir de biţi; vom nota lungimea şirului $x$ cu $|x|$, iar concatenarea a două şiruri $x, y \in \mathcal{X}$ cu $x\|y$.

Fie $|x| = n \geq m$; atunci $x$ se poate scrie sub forma:

$$x = x_1\|x_2\| \ldots \|x_k$$

unde $\quad |x_1| = |x_2| = \ldots = |x_{k-1}| = m - t - 1, \quad |x_k| = m - t - 1 - d,\ 0 \leq d \leq m - t - 2.$

În acest fel se obţine $k = \left\lceil \dfrac{n}{m - t - 1} \right\rceil$. Definim funcţia $h^*(x)$ cu algoritmul următor:

1. **for** $i := 1$ **to** $k - 1$ **do** $y_i \longleftarrow x_i$
2. $y_k \longleftarrow x_k\|0^d$
3. $y_{k+1} \longleftarrow [d]_2$ (reprezentarea binară a lui $d$ pe $m - t - 1$ biţi).
4. $g_1 \longleftarrow h(0^{t+1}\|y_1)$
5. **for** $i = 1$ **to** $k$ **do** $g_{i+1} \longleftarrow h(g_i\|1\|y_{i+1})$
6. $h^*(x) \longleftarrow g_{k+1}$

Notăm

$$y(x) = y_1\|y_2\| \ldots \|y_{k+1}$$

$y_k$ se obţine completând $x_k$ la dreapta cu $d$ zerouri, astfel ca toate blocurile $y_i$ $(1 \leq i \leq k)$ să fie de lungime $m - t - 1$.

De asemenea, la pasul 3, $y_{k+1}$ este completat la stânga cu zerouri pentru a obţine lungimea $m-t-1$.

Pentru dispersia lui $x$ se construieşte $y(x)$ apoi se tratează iterativ blocurile $y_1, y_2, \ldots, y_{k+1}$. Din construcţia lui $y_{k+1}$ se asigură injectivitatea aplicaţiei $y$.

**Teorema 1.3.** *Fie $h : Z_2^m \longrightarrow Z_2^t$ $(m \geq t+2)$ o funcţie criptografică de dispersie. Funcţia $h^* : \bigcup_{i=m}^{\infty} Z_2^i \longrightarrow Z_2^t$ definită prin algoritmul de mai sus, este de asemenea o funcţie criptografică.*

*Demonstraţie.* Trebuie demonstrat că funcţia $h^*$ verifică problema **CP**.
Să presupunem prin absurd că există $x \neq x'$ astfel ca $h^*(x) = h^*(x')$. Vom arăta cum, plecând de la această coliziune se poate construi în timp polinomial o coliziune pentru $h$, lucru imposibil deoarece $h$ este o funcţie criptografică – deci cu coliziuni tari.

Fie $y(x) = y_1 \| y_2 \| \ldots \| y_{k+1}$ şi $y(x') = y'_1 \| y'_2 \| \ldots \| y'_{s+1}$, unde $x$ şi $x'$ sunt completate la pasul 2 cu $d$ respectiv $d'$ zerouri. S-a notat cu $g_1, \ldots, g_{k+1}$ respectiv $g'_1, \ldots, g'_{s+1}$ valorile calculate de paşii 4 şi 5.

Apar două cazuri:

1. $|x| \not\equiv |x'| \ (mod\ m-t-1)$.

   Atunci $d \neq d',\ y_{k+1} \neq y'_{s+1}$. Avem:

   $$h(g_k \| 1 \| y_{k+1}) = g_{k+1} = h^*(x) = h^*(x') = g'_{s+1} = h(g'_s \| 1 \| y'_{s+1}),$$

   care este o coliziune pentru $h$ deoarece $y_{k+1} \neq y'_{s+1}$.

2. $|x| \equiv |x'| \ (mod\ m-t-1)$.

   Aici trebuie studiate două subcazuri:

   **(2a).** $|x| = |x'|$. Atunci $k = s,\ y_{k+1} = y'_{s+1}$. Similar primului caz,

   $$h(g_k \| 1 \| y_{k+1}) = g_{k+1} = h^*(x) = h^*(x') = g'_{k+1} = h(g'_k \| 1 \| y'_{k+1})$$

   Dacă $g_k \neq g'_k$, s-a aflat o coliziune pentru $h$.
   Să presupunem deci că $g_k = g'_k$. Vom avea:

   $$h(g_{k-1} \| 1 \| y_k) = g_k = g'_k = h(g'_{k-1} \| 1 \| y'_k)$$

   Fie s-a găsit o coliziune pentru $h$, fie $g_{k-1} = g'_{k-1},\ y_k = y'_k$.
   Dacă nu a fost coliziune, se poate continua până la

   $$h(0^{t+1} \| y_1) = g_1 = g'_1 = h(0^{t+1} \| y'_1)$$

   Cum $y_1 \neq y'_1$ înseamnă coliziune pentru $h$, să presupunem $y_1 = y'_1$. Vom avea $y_i = y'_i$ $(1 \leq i \leq k+1)$ deci $y(x) = y(x')$.

Cum aplicaţia $y$ este injectivă, rezultă $x = x'$, ceea ce contrazice ipoteza.

**(2b).** $|x| \neq |x'|$.

Se poate presupune, fără a micşora generalitatea, că $|x| < |x'|$, deci $s > k$.
Vom proceda similar cu **(2a)**.
Dacă nu s-au găsit coliziuni pentru $h$, se va obţine şirul de egalităţi

$$h(0^{t+1}\|y_1) = g_1 = g'_{s-k+1} = h(g'_{s-k}\|1\|y'_{s-k+1})$$

Primii $t+1$ biţi din $0^{t+1}\|y_1$ conţin numai zerouri în timp ce primii $t+1$ biţi din $g'_{s-k}\|1\|y'_{s-k+1}$ conţin un 1. Deci s-a găsit o coliziune pentru $h$.

Rezultă că pentru toate cazurile studiate, s-a ajuns la coliziuni.

□

Algoritmul pe care s-a construit toată demonstraţia de sus este adevărat numai dacă $m \geq t+2$. Rămâne cazul $m = t+1$; aici se va realiza o construcţie diferită pentru $h^*$.

Ca anterior, fie $|x| = n \geq m$.

Vom începe prin a codifica $x$ folosind funcţia $f$ definită

$$f(0) = 0, f(1) = 01$$

Construcţia lui $h^*$ se face pe baza următorului algoritm:

1. $y \longleftarrow y_1\|y_2\|\ldots\|y_k = 11\|f(x_1)\|f(x_2)\|\ldots\|f(x_n)$ $(y_i \in Z_2)$
2. $g_1 \longleftarrow h(0^t\|y_1)$
3. $d$ **for** $i = 1$ **to** $k-1$ **do** $g_{i+1} \longleftarrow h(g_i\|y_{i+1})$
4. $h^*(x) \longleftarrow g_k$

Codificarea $x \mapsto y = y(x)$ din pasul 1 verifică câteva proprietăţi importante:

- $y$ este injectivă
- $y$ este liberă de prefix: nu există $x, x', z$ astfel ca $y(x) = z\|y(x')$

**Teorema 1.4.** *Fie $h : Z_2^{t+1} \longrightarrow Z_2^t$ o funcţie criptografică de dispersie.*
*Funcţia $h^* : \bigcup_{i=t+1}^{\infty} Z_2^i \longrightarrow Z_2^t$ este de asemenea o funcţie criptografică.*

*Demonstraţie.* Să presupunem prin absurd că există $x \neq x'$ cu $h^*(x) = h^*(x')$.
Vom avea

$$y(x) = y_1 y_2 \dots y_k, \quad y(x') = y'_1 y'_2 \dots y'_s.$$

Apar două cazuri:

1. $k = s$.
   Similar cu demonstraţia teoremei anterioare, se găseşte fie o coliziune pentru $h$, fie $y(x) = y(x')$, ceea ce duce la $x = x'$, contradicţie.

2. $k \neq s$. Vom considera situaţia $s > k$. Modul de abordare a acestui caz este similar cu cele de mai sus. Presupunând că nu s-au găsit coliziuni pentru $h$, se obţine şirul de identităţi
   $$y_k = y'_s,\ y_{k-1} = y'_{s-1},\ \dots y_1 = y'_{s-k+1},$$
   ceea ce contrazice faptul că codificarea este liberă de prefix.

Deci $h$ este cu coliziuni tari. □

Să rezumăm într-o singură teoremă rezultatele obţinute:

**Teorema 1.5.** *Fie $h : Z_2^m \longrightarrow Z_2^t$, $(m \geq t+1)$ o funcţie criptografică de dispersie. Există atunci o funcţie criptografică de dispersie*

$$h^* : \bigcup_{i=m}^{\infty} Z_2^i \longrightarrow Z_2^t.$$

*Numărul de calcule al lui h pentru efectuarea unei aplicări a lui $h^*$ este cel mult*

$$\begin{cases} 1 + \left\lceil \dfrac{n}{m-t-1} \right\rceil & \text{dacă } m \geq t+2, \\ 2n+2 & \text{dacă } m = t+1, \end{cases}$$

*unde $n = |x|$.*

## 1.5 Funcţii de dispersie bazate pe sisteme de criptare

Deoarece multe metode de construcţie ale funcţiilor de dispersie (inclusiv cele de sus) sunt prea lente pentru o utilizare practică, sunt propuse şi alte variante. Una din ele constă în folosirea unui sistem de criptare pentru a genera o funcţie criptografică de dispersie.

Să presupunem că $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ este un sistem de criptare simetric, cu $\mathcal{P} = \mathcal{C} = \mathcal{K} = Z_2^n$. Pentru a evita atacul naşterilor se va lua $n \geq 128$ (deci nu se poate folosi $DES$).

Fie şirul de biţi

$$x = x_1 \| x_2 \| \dots \| x_k, \qquad x_i \in Z_2^n \quad 1 \leq i \leq k.$$

Dacă numărul de biţi nu este multiplu de $n$, $x$ se va completa ca în paragraful anterior; pentru simplificare, nu vom trata aici această chestiune.

Ideea construcţiei constă în fixarea unei *valori iniţiale* $g_0$ şi calcularea iterativ a lui $g_1, g_2, \ldots, g_k$ astfel ca $g_i = f(x_i, g_{i-1})$, unde $f$ este o funcţie care utilizează regulile de criptare. Rezultatul final al dispersiei este amprenta $h(x) = g_k$.

Au fost propuse mai multe funcţii de dispersie, unele din ele fiind sparte (independent de sistemul de criptare utilizat). O anumită securitate mai mare a fost probată pentru următoarele variante:

$$g_i = e_{g_{i-1}}(x_i) \oplus x_i \qquad \text{(Mathyas - Meyer - Oseas)}$$
$$g_i = e_{g_{i-1}}(x_i) \oplus g_{i-1} \qquad \text{(Davies - Meyer)}$$
$$g_i = e_{g_{i-1}}(x_i) \oplus x_i \oplus g_{i-1} \qquad \text{(Miyaguchi - Preneel)}$$
$$g_i = e_{g_{i-1}}(x_i \oplus g_{i-1}) \oplus x_i$$
$$g_i = e_{g_{i-1}}(x_i \oplus g_{i-1}) \oplus x_i \oplus g_{i-1}$$

Pentru un sistem de criptare mai general, cu $\mathcal{P} = \mathcal{C} = 2^n$ ($n \geq 128$), $\mathcal{K} = 2^k$ sunt cunoscute alte două modalităţi de construcţie de funcţii de dispersie (prezentate în [3] drept funcţii de compresie), implementate frecvent sub o formă combinată.

- *Funcţii de dispersie MDC (Modification Detection Code)*:
  Textul clar se reîmparte în blocuri de lungime $n + r$. Funcţia de dispersie

  $$h : Z_2^{n+r} \longrightarrow Z_2^n$$

  este definită astfel: Dacă $z = x \| y$, $|x| = n$, $|y| = k$, atunci $h(z) = e_y(x)$.

- *Funcţia Matyias - Meyer - Oseas* (inclusă în standardul $ISO/IEC$ $10118 - 2$):

  1. Blocul $z \in \mathcal{X}$ de lungime $2n$ este spart în două jumătăţi: $z = x \| y$.
  2. Se determină cheia $K$ folosind o funcţie $g : Z_2^n \longrightarrow Z_2^k$
     (uzual $K = g(y)$ se obţine din $y$ cu ajutorul unei funcţii booleane).
  3. Amprenta lui $z$ este $h(z) = e_K(x) \oplus x$.

Securitatea unor astfel de funcţii de dispersie este în strânsă dependenţă cu securitatea oferită de sistemul de criptare folosit.

## 1.6 Funcţii de dispersie bazate pe aritmetica modulară

Plecând de la ideea că există echipamente soft şi chiar hard deosebit de performante pentru realizarea de operaţii în aritmetica modulară, au fost propuse diverse funcţii de

dispersie bazate pe astfel de calcule. Singurul lor dezavantaj este viteza relativ mică de obţinere a amprentelor.

Cele mai cunoscute sunt cele din clasa $MASH$ (Modular Arithmetic Secure Hash). $MASH1$ a fost construit pentru a fi inclus în standardul $ISO/IEC$. El foloseşte un modul $n$ de tip $RSA$ (deci lungimea sa este esenţială pentru asigurarea securităţii). Mai mult, mărimea acestui modul (1024 biţi conform cerinţelor actuale) determină mărimile blocurilor care sunt prelucrate iterativ, precum şi mărimea amprentei.

Fie deci $p, q$ două numere prime mari şi $n = pq$ un număr scris în binar pe $M$ biţi. Definim mărimea $N$ (numărul de biţi) a amprentei prin

$$N = 16 \cdot \left\lceil \frac{M}{16} \right\rceil$$

Mesajul de intrare este $x \in Z_2^b \quad (0 \leq b \leq 2^{N/2})$. Funcţia de dispersie $MASH1$ este definită astfel:

1. Se definesc $H_0 = \mathbf{0}$ şi $A = f0 \ldots f0$, ambele scrise pe $N$ biţi.
2. Se extinde $x$ cu zerouri până la cel mai apropiat multiplu de $N/2$ şi se scrie $x = x_1 \ldots x_t, \quad |x_i| = N/2$.
3. $x_{t+1} \longleftarrow b$, unde $b$ este reprezentat pe $N/2$ biţi.
4. Pentru $i = 1, 2, \ldots, t$, blocul $x_i$ se extinde la $n$ biţi astfel: $x_i$ se împarte în subblocuri de câte 4 biţi, după care se inserează biţii 1111 la începutul fiecărui subbloc.
   Pentru $x_{t+1}$ se procedează analog, singura diferenţă fiind biţii inseraţi: 1010 în loc de 1111.
   Fie $y_i$ extensia lui $x_i$ $(1 \leq i \leq t+1)$.
5. **for** $i = 1$ **to** $t + 1$ **do**
   $$H_i \longleftarrow ((((H_{i-1} \oplus y_i) \vee A)^2 \ (mod\ n)) \gg N) \oplus H_{i-1}$$
   unde $\gg$ reprezintă deplasarea ciclică spre dreapta.
6. $h(x) \longleftarrow H_{t+1}$

$MASH2$ are ca unică diferenţă înlocuirea exponentului $e = 2$ de la pasul 4. cu exponentul $e = 2^8 + 1$.

## 1.7 Funcţia de dispersie MD4

Funcţiile **MDi** (Message Digest nr. $i$) au fost construite de Ron Rivest. În particular, **MD4** a fost introdusă în 1990 ([12]) pentru procesoare pe 32 biţi. Deşi nu s-a dovedit suficient de sigură, principiile sale de construcţie au fost ulterior utilizate pentru o clasă largă de funcţii de dispersie, cum ar fi **MD5**, **SHA**, **SHA1**, **RIPEMD160**. Cu toate că securitatea lor nu a fost demonstrată (şi atacurile apărute de-a lungul timpului au demonstrat diverse slăbiciuni de construcţie), funcţiile de dispersie **MDi** au avantajul de a fi foarte rapide, deci practice pentru semnarea de mesaje lungi.

### 1.7.1 Descrierea funcţiei MD4

Fiind dat un şir $x \in Z_2^*$, se defineşte tabloul $M = M_0 M_1 \dots M_{n-1}$ cu fiecare $M_i$ (numit *cuvânt*) de lungime 32 şi $n \equiv 0 \ (mod\ 16)$.

$M$ este construit folosind algoritmul următor:

1. $d \longleftarrow (447 - |x|) \ (mod\ 512)$
2. $s \longleftarrow$ reprezentarea binară a lui $|x| \ (mod\ 2^{64})$, $|s| = 64$
3. $M = x \| 1 \| 0^d \| s$

**MD4** construieşte o amprentă a lui $x$ pe 128 biţi. Algoritmul de generare este:

1. $A \longleftarrow 67452301\ (hex)$ $B \longleftarrow efcdab89\ (hex)$
   $C \longleftarrow 98badcfe\ (hex)$ $D \longleftarrow 10325476\ (hex)$
2. **for** $i = 0$ **to** $n/16 - 1$ **do**
   - 2.1. **for** $j = 0$ **to** $15$ **do** $X_j \longleftarrow M_{16i+j}$
     - 2.1.1. $AA \longleftarrow A$ $BB \longleftarrow B$ $CC \longleftarrow C$ $DD \longleftarrow D$
     - 2.1.2. **Etapa 1**
     - 2.1.3. **Etapa 2**
     - 2.1.4. **Etapa 3**
     - 2.1.5. $A \longleftarrow A + AA$ $B \longleftarrow B + BB$
       $C \longleftarrow C + CC$ $D \longleftarrow D + DD$

Amprenta $h(x)$ este – în final – concatenarea celor patru cuvinte $A, B, C, D$ (numite *registre*). Algoritmul împarte tabelul $M$ în şiruri de câte 16 cuvinte consecutive, cărora le aplică trei etape de transformări. Adunările de la pasul 2.1.5 sunt efectuate modulo $2^{32}$.

Fie $\alpha, \beta \in Z_2^*$, $|\alpha| = |\beta|$. Cele 3 etape se bazează pe următoarele operaţii:

- $\alpha \wedge \beta$ *şi* logic bit cu bit ($AND$)

- $\alpha \vee \beta$ *sau* logic bit cu bit ($OR$)
- $\alpha \oplus \beta$ *sau exclusiv* bit cu bit ($XOR$)
- $\neg\alpha$ complementară bit cu bit ($NOT$)
- $\alpha + \beta$ suma modulo $2^{32}$
- $\alpha \ll s$ rotirea circulară cu $s$ biţi spre stânga ($1 \leq s \leq 31$)

Implementarea acestor operaţii se realizează direct pe hard şi ţine cont de arhitectura calculatorului. Să detaliem puţin acest lucru.

Fie $a_1a_2a_3a_4$ un cuvânt de patru octeţi, unde $a_i$ este un număr întreg din intervalul $[0, 255]$.

- Într-o arhitectură *big-endian* (o staţie $SPARK$ de exemplu), un cuvânt reprezintă întregul $a_1 2^{24} + a_2 2^{16} + a_3 2^8 + a_4$
- Într-o arhitectură *little-endian* (cum este familia $Intel\ 80xxx$), un cuvânt reprezintă întregul $a_4 2^{24} + a_3 2^{16} + a_2 2^8 + a_1$

Construcţia standard a lui **MD4** s-a bazat pe o arhitectură *little-endian*. Cum amprenta rezultată în urma procesului de dispersie trebuie să fie independentă de arhitectura calculatorului folosit, la o implementare a lui **MD4** pe un calculator *big-endian*, adunările se vor defini astfel:

1. Se inter-schimbă $x_1 \leftrightarrow x_4,\ x_2 \leftrightarrow x_3,\ y_1 \leftrightarrow y_4,\ y_2 \leftrightarrow y_3$
2. Se calculează $Z = X + Y\ (mod\ 2^{32})$
3. Se inter-schimbă $z_1 \leftrightarrow z_4,\ z_2 \leftrightarrow z_3$

Cele trei etape de construcţie ale funcţiei de dispersie **MD4** folosesc trei funcţii $f, g, h : Z_2^{32} \times Z_2^{32} \times Z_2^{32} \longrightarrow Z_2^{32}$ definite:

$$f(X, Y, Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$$
$$g(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$$
$$s(X, Y, Z) = X \oplus Y \oplus Z$$

Descrierile celor trei etape sunt:

**Etapa 1**:

**1.** $A \longleftarrow (A + f(B, C, D) + X_0) \ll 3$
**2.** $D \longleftarrow (D + f(A, B, C) + X_1) \ll 7$
**3.** $C \longleftarrow (C + f(D, A, B) + X_2) \ll 11$
**4.** $B \longleftarrow (B + f(C, D, A) + X_3) \ll 19$
**5.** $A \longleftarrow (A + f(B, C, D) + X_4) \ll 3$
**6.** $D \longleftarrow (D + f(A, B, C) + X_5) \ll 7$
**9.** $A \longleftarrow (A + f(B, C, D) + X_8) \ll 3$
**10.** $D \longleftarrow (D + f(A, B, C) + X_9) \ll 7$
**11.** $C \longleftarrow (C + f(D, A, B) + X_{10}) \ll 11$
**12.** $B \longleftarrow (B + f(C, D, A) + X_{11}) \ll 19$
**13.** $A \longleftarrow (A + f(B, C, D) + X_{12}) \ll 3$
**14.** $D \longleftarrow (D + f(A, B, C) + X_{13}) \ll 7$

**7.** $C \longleftarrow (C + f(D,A,B) + X_6) \lll 11$ **15.** $C \longleftarrow (C + f(D,A,B) + X_{14}) \lll 11$
**8.** $B \longleftarrow (B + f(C,D,A) + X_7) \lll 19$ **16.** $B \longleftarrow (B + f(C,D,A) + X_{15}) \lll 19$

**Etapa 2**:
**1.** $A \longleftarrow (A + g(B,C,D) + X_0 + P) \lll 3$ **9.** $A \longleftarrow (A + g(B,C,D) + X_2 + P) \lll 3$
**2.** $D \longleftarrow (D + g(A,B,C) + X_4 + P) \lll 5$ **10.** $D \longleftarrow (D + g(A,B,C) + X_6 + P) \lll 5$
**3.** $C \longleftarrow (C + g(D,A,B) + X_8 + P) \lll 9$ **11.** $C \longleftarrow (C + g(D,A,B) + X_{10} + P) \lll 9$
**4.** $B \longleftarrow (B + g(C,D,A) + X_{12} + P) \lll 13$ **12.** $B \longleftarrow (B + g(C,D,A) + X_{14} + P) \lll 13$
**5.** $A \longleftarrow (A + g(B,C,D) + X_1 + P) \lll 3$ **13.** $A \longleftarrow (A + g(B,C,D) + X_3 + P) \lll 3$
**6.** $D \longleftarrow (D + g(A,B,C) + X_5 + P) \lll 5$ **14.** $D \longleftarrow (D + g(A,B,C) + X_7 + P) \lll 5$
**7.** $C \longleftarrow (C + g(D,A,B) + X_9 + P) \lll 9$ **15.** $C \longleftarrow (C + g(D,A,B) + X_{11} + P) \lll 9$
**8.** $B \longleftarrow (B + g(C,D,A) + X_{13} + P) \lll 13$ **16.** $B \longleftarrow (B + g(C,D,A) + X_{15} + P) \lll 13$
unde s-a folosit constanta $P = 5a827999$.

**Etapa 3**:
**1.** $A \longleftarrow (A + s(B,C,D) + X_0 + P) \lll 3$ **9.** $A \longleftarrow (A + s(B,C,D) + X_1 + P) \lll 3$
**2.** $D \longleftarrow (D + s(A,B,C) + X_8 + P) \lll 9$ **10.** $D \longleftarrow (D + s(A,B,C) + X_9 + P) \lll 9$
**3.** $C \longleftarrow (C + s(D,A,B) + X_4 + P) \lll 11$ **11.** $C \longleftarrow (C + s(D,A,B) + X_5 + P) \lll 11$
**4.** $B \longleftarrow (B + s(C,D,A) + X_{12} + P) \lll 15$ **12.** $B \longleftarrow (B + s(C,D,A) + X_{13} + P) \lll 15$
**5.** $A \longleftarrow (A + s(B,C,D) + X_2 + P) \lll 3$ **13.** $A \longleftarrow (A + s(B,C,D) + X_3 + P) \lll 3$
**6.** $D \longleftarrow (D + s(A,B,C) + X_{10} + P) \lll 9$ **14.** $D \longleftarrow (D + s(A,B,C) + X_{11} + P) \lll 9$
**7.** $C \longleftarrow (C + s(D,A,B) + X_6 + P) \lll 11$ **15.** $C \longleftarrow (C + s(D,A,B) + X_7 + P) \lll 11$
**8.** $B \longleftarrow (B + s(C,D,A) + X_{14} + P) \lll 15$ **16.** $B \longleftarrow (B + s(C,D,A) + X_{15} + P) \lll 15$
unde $P = 6ed9eba1$.

**Observaţia 1.2.** *Funcţia $f$ mai poate fi definită sub forma*

$$f(X,Y,Z) = \textbf{if } X \textbf{ then } Y \textbf{ else } Z$$

*Funcţia $g(X,Y,Z)$ ia (pe biţi) valoarea majoritară dintre valorile (pe biţi) ale lui $X, Y, Z$.*

**Exemplul 1.3.** *([9]): Câteva amprente obţinute cu ajutorul funcţiei* **MD4***:*

| *Secvenţă ASCII* | *Amprentă (scrisă în hexazecimal)* |
|---|---|
| "" | $31d6cfe0d16ae931b73c59d7e0c089c0$ |
| "$a$" | $bde52cb31de33e46245e05fbdbd6fb24$ |
| "$abc$" | $a448017aaf21d8525fc10ae87aa6729d$ |
| "'$abcdefghijklmnopqrstuvwxyz$" | $d79e1c308aa5bbcdeea8ed63df412da9$ |

## 1.7.2 Criptanaliza schemei MD4

Etapele funcţiei **MD4** sunt caracterizate de permutări $\sigma_i$ ale blocului $X = (X_0, \ldots X_{15})$ şi deplasări ciclice spre stânga $\alpha_{i,j}$, $i = 1,2,3$, $j = 1,2,\ldots,16$.
Cele trei permutări – corespunzătoare celor 3 etape – sunt

$$\sigma_1 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$
$$\sigma_2 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 4 & 8 & 12 & 1 & 5 & 9 & 13 & 2 & 6 & 10 & 14 & 3 & 7 & 11 & 15 \end{pmatrix}$$
$$\sigma_3 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 8 & 4 & 12 & 2 & 10 & 6 & 14 & 1 & 9 & 5 & 13 & 3 & 11 & 7 & 15 \end{pmatrix}$$

Deplasările ciclice spre stânga $\alpha_{i,j}$ sunt

| $i/j \ (mod\ 4)$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 1 | 3 | 7 | 11 | 19 |
| 2 | 3 | 5 | 9 | 13 |
| 3 | 3 | 9 | 11 | 15 |

De exemplu, $\alpha_{3,6} = 11$.

Funcţiile $f$ şi $g$ au următoarele proprietăţi;

- $f(\mathbf{1}, a, x) = f(\mathbf{0}, x, a) = f(x, a, a) = a, \quad \forall a, x \in Z_2^{32}, \quad \mathbf{0} = 00\ldots0, \ \ \mathbf{1} = 11\ldots1;$
- $g(a, a, x) = g(a, x, a) = g(x, a, a) = a, \quad \forall a, x \in Z_2^{32};$

În prima etapă $\sigma_1$ este permutarea identică, iar $X_i$ sunt utilizate în ordinea iniţială. Deci, dacă $X_0, \ldots, X_{11}$ sunt fixate, se pot alege uşor valori pentru $X_{12}, X_{13}, X_{14}$ astfel ca să se obţină 0 în regiştrii $A, C$ şi $D$.
Permutarea $\sigma_2$ din Etapa 2 asociază pe rând registrului $B$ valorile din $X_{12}, X_{13}, X_{14}, X_{15}$. Deci, dacă celelalte valori sunt $-P$, iar $X_{12}, X_{13}, X_{14}$ sunt alese astfel ca să fie îndeplinită proprietatea anterioară, regiştrii $A, C, D$ rămân cu valoarea $\mathbf{0}$.
Această proprietate are loc pentru orice valoare a lui $X_{15}$.

Atacul ignoră Etapa 3 şi consideră ieşirea din Etapa 2 ca o funcţie (aleatoare) de variabilă $X_{15}$, în care trei regiştri sunt permanent $\mathbf{0}$. Deci dimensiunea aleatoare este redusă la 32 biţi; rezultă că un atac bazat pe paradoxul naşterilor poate deduce o coliziune (două blocuri de 32 biţi care dau aceiaşi ieşire după Etapa 2) în aproximativ $2^{16}$ încercări.

Deci un atac asupra funcţiei de dispersie **MD4** redusă la primele două etape este:

**Intrare**: $A, B, C, D$;

1. **for** $i = 0, 1, 2, 4, 5, 6, 8, 9, 10, 12, 13$ **do** $X_{\sigma_2^{-1}(i)} \longleftarrow -P$
2. $Random(X_{11})$
3. **for** $i = 0$ **to** $11$ **do** $(A, D, B, C) \longleftarrow (D, C, B, (A + f(B, C, D) + X_i) \lll \alpha_{1,i})$
4. **for** $i = 12$ **to** $14$ **do**
   4.1. $X_i \longleftarrow -(A + f(B, C, D))$
   4.2. $(A, D, C, B) \longleftarrow (D, C, B, \mathbf{0})$
5. $B_0 \longleftarrow A$
6. **if** $u(X_{15}) = u(X_{15}{}')$ **then**
   $X_i{}' = X_i \quad (0 \leq i \leq 14)$
   $Output(X, X'); \quad Exit$

**Function** $u(X_{15})$:

1. $A \longleftarrow \mathbf{0}, \quad C \longleftarrow \mathbf{0}, \quad D \longleftarrow \mathbf{0}$;
2. $B \longleftarrow ((B_0 + X_{15} + P) \lll \alpha_{1,15})$;
3. **for** $i = 0$ **to** $15$ **do**
   $(A, D, C, B) \longleftarrow (D, C, B, (A + g(B, C, D) + X_{\sigma_2(i)} + P) \lll \alpha_{2,i})$
4. $Return(B)$.

După ce se aplică acest atac pentru primele două etape, permutarea $\sigma_3$ din Etapa 3 asociază pe $X_{15}$ (singura valoare care se modifică) doar registrului $B$; deci două blocuri de câte 128 biţi $X$ şi $X'$ (cu $X_i = X_i{}'$ pentru $i = 0, \ldots, 14$, $X_{15} \neq X_{15}{}'$) care duc la o coliziune după primele două etape, au proprietatea că $h(X)$ şi $h(X')$ diferă doar pe al doilea bloc de 32 biţi.
Se pot obţine uşor îmbunătăţiri în care cele două amprente să difere doar într-un singur bit din zona respectivă.

Pe baza acestui atac Hans Dobbertin construieşte un algoritm care generează toate coliziunile din **MD4**. Pentru detalii a se vedea [4] şi [16].

### 1.7.3 Funcţia de dispersie MD5

Pentru a elimina slăbiciunile sistemului **MD4**, în 1991 Ron Rivest propune o nouă variantă – **MD5**, publicată anul următor sub numele *RFC* 1321 *Internet Standard*.

Schematic, **MD5** se bazează pe o "funcţie de criptare" $g$ propusă de Davies - Meyer: dacă aceasta este

$$g : Z_2^{128} \times Z_2^{512} \longrightarrow Z_2^{128}$$

atunci

$$h(H, X) = H + g(H, X)$$

unde adunarea este realizată modulo $2^{32}$.

Criptarea $g(H, X)$ constă din 4 runde şi – în linii mari – poate fi descrisă astfel:

- Mesajul $H$ de 128 biţi este aranjat printr-o permutare (care depinde de rundă) într-o secvenţă de cuvinte $A, B, C, D$;
- Fiecare din aceste cuvinte trece printr-o transformare secvenţială, bazată pe o schemă de tip Feistel;
- O transformare este definită de un $S$ - box cu intrarea $(A, K)$ ($K$ – cheie de rundă derivată din $X$) şi ieşirea $B, C, D$;
- Ieşirea din $S$ box este

$$(A + f_i(B, C, D) + K + k_{i,j} + B) \ll \alpha_{i,j}$$

unde $\alpha_{i,j}$ şi $k_{i,j}$ sunt definite printr-o tabelă, iar $f_i$ este o funcţie booleană de rundă, definită

$$\begin{aligned} f_1(B, C, D) &= (B \wedge C) \vee ((\neg B) \wedge D) \\ f_2(B, C, D) &= (D \wedge B) \vee ((\neg D) \wedge C) \\ f_3(B, C, D) &= B \oplus C \oplus D \\ f_4(B, C, D) &= C \oplus (B \wedge (\neg D)) \end{aligned}$$

**Exemplul 1.4.** *([9]): Câteva amprente obţinute cu ajutorul funcţiei* **MD5***:*

| *Secvenţă ASCII* | *Amprentă (scrisă în hexazecimal)* |
|---|---|
| "" | $d41d8cd98f00b204e9800998ecf8427e$ |
| "$a$" | $0cc175b9c0f1b6a831c399e269772661$ |
| "$abc$" | $900150983cd24fb0d6963f7d28e17f72$ |
| "'$abcdefghijklmnopqrstuvwxyz$" | $c3fcd3d76192e4007dfb496cca67e13b$ |

Mai multe detalii referitoare la sistemul **MD5** pot fi găsite în [13] sau [16].

Şi această funcţie de dispersie – utilizată mai ales în aplicaţii Internet – este spartă destul de repede. Mai mult, în 2006 V. Klima ([7]) publică un algoritm care calculează – în mai puţin de un minut – coliziuni pentru **MD5**, folosind un PC standard.

## 1.8 Funcţia de dispersie $SHA1$

**SHA1** este o variantă a funcţiei de dispersie $SHA$ – notată ca standard $FIPS\ 180-1$ – care corectează o mică slăbiciune din **SHA**. Construcţia sa este următoarea:

Fie $x$ $(|x| \leq 2^{64}-1)$ şirul binar care trebuie amprentat.
Prima parte a algoritmului este identică cu cea de la **MD4**:

1. $d \longleftarrow (447-|x|)\ (mod\ 512)$
2. $s \longleftarrow$ reprezentarea binară a lui $|x|\ (mod\ 2^{64}), |s| = 64$
3. $M = x\|1\|0^d\|s$

Dacă $|s| < 64$, se adaugă zerouri la stânga până se ajunge la egalitate.
Blocul final $M$ (care intră în algoritmul de dispersie) are o lungime divizibilă cu 512; îl vom scrie ca o concatenare de $n$ blocuri, fiecare de 512 biţi:

$$M = M_1\|M_2\|\ldots\|M_n$$

Fie funcţiile $f_i : Z_2^{32} \times Z_2^{32} \times Z_2^{32} \longrightarrow Z_2^{32} \quad (0 \leq i \leq 79)$, definite astfel:

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & 0 \leq t \leq 19 \\ B \oplus C \oplus D & 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & 40 \leq t \leq 59 \\ B \oplus C \oplus D & 60 \leq t \leq 79 \end{cases}$$

Deci fiecare funcţie $f_t$ ia trei cuvinte la intrare şi scoate un cuvânt la ieşire.

Se mai definesc constantele $K_0, K_1, \ldots, K_{79}$ astfel:

$$K_t = \begin{cases} 5A827999 & 0 \leq t \leq 19 \\ 6ED9EBA1 & 20 \leq t \leq 39 \\ 8F1BBCDC & 40 \leq t \leq 59 \\ CA62C1D6 & 60 \leq t \leq 79 \end{cases}$$

Algoritmul de compresie **SHA1** este:

1. $H_0 \longleftarrow 67452301, \quad H_1 \longleftarrow EFCDAB89, \quad H_2 \longleftarrow 98BADCFE,$
$H_3 \longleftarrow 10325476, \quad H_4 \longleftarrow C3D2E1F0.$

2. **for** $i \longleftarrow 1$ **to** $n$ **do**

**2.1.** Fie $M_i = W_0 \| W_1 \| \ldots \| W_{15}, \quad W_i \in Z_2^{32}$.

**2.2.** **for** $t \longleftarrow 16$ **to** $79$ **do**

$$W_t \longleftarrow (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \ll 1)$$

**2.3.** $A \longleftarrow H_0, \quad B \longleftarrow H_1, \quad C \longleftarrow H_2, \quad D \longleftarrow H_3, \quad E \longleftarrow H_4$

**2.4.** **for** $t \longleftarrow 0$ **to** $79$ **do**

**2.4.1.** $temp \longleftarrow (A \ll 5) + f_t(B, C, D) + E + W_t + K_t$

**2.4.2.** $E \longleftarrow D,\ D \longleftarrow C,\ C \longleftarrow (B \ll 30),\ B \longleftarrow A$

**2.4.3.** $A \longleftarrow temp$

**2.5** $H_0 \longleftarrow H_0 + A, \quad H_1 \longleftarrow H_1 + B, \quad H_2 \longleftarrow H_2 + C,$
$H_3 \longleftarrow H_3 + D, \quad H_4 \longleftarrow H_4 + E$

3. **Output**: $y = H_0 \| H_1 \| H_2 \| H_3 \| H_4$.

Funcţia de compresie **SHA1** formează pentru fiecare bloc de 512 biţi un bloc de 160 biţi.

**Exemplul 1.5.** *([9]): Câteva amprente obţinute cu ajutorul funcţiei* **SHA1***:*

| *Secvenţă ASCII* | *Amprentă (scrisă în hexazecimal)* |
|---|---|
| "" | $da39a3ee5e6b4b0d3255bfef95601890afd80809$ |
| "$a$" | $86f7e437faa5a7fce15d1ddcb9eaeaea377667b8$ |
| "$abc$" | $a9993e364706816aba3e25717850c26c9cd0d89d$ |
| "'$abcdefghijklmnopqrstuvwxyz$" | $32d10c7b8cf96570ca04ce37f2a19d84240d3a89$ |

Algoritmul original **SHA** avea o slăbiciune[4] care permite aflarea coliziunilor în aproximativ $2^{61}$ paşi. Versiunea **SHA1** este mai eficientă, atacul naşterilor conducând la aflarea coliziunilor abia după $2^{80}$ paşi. Dar şi pentru această funcţie de dispersie au început să fie publicate coliziuni începând cu 2005, când o echipă de la Universitatea Shandong, China afirmă că a găsit coliziuni pentru **SHA1** în $2^{69}$ operaţii (iar pentru o variantă **SHA1** de 58 runde, în numai $2^{33}$ operaţii) .

[4]Diferenţa dintre **SHA** şi **SHA1** este minoră: în **SHA** transformarea de la pasul 2.2 se efectuează fără nici o rotaţie !

La 30 mai 2001 $NIST$ propune o nouă versiune **SHA2**; aceasta include **SHA1** precum şi alte trei funcţii de dispersie $\mathbf{SHA_{256}}$, $\mathbf{SHA_{384}}, \mathbf{SHA_{512}}$ (cifrele se referă la mărimea amprentei).

În momentul scrierii acestei cărţi este lansat un concurs pentru un nou standard de dispersie care va fi numit **SHA3**. Detalii asupra ultimelor rezultate se pot găsi la

$$http://ehash.iaik.tugraz.at/wiki/The\_SHA-3\_Zoo$$

## 1.9 Funcţii de dispersie aleatoare

Deşi satisface **CP**, o funcţie de dispersie criptografică $h$ poate dezvălui unele informaţii despre mesajele amprentatate.
De exemplu, într-un sistem multi-user, parolele utilizatorilor sunt amprentate cu o anumită funcţie de dispersie, iar aceste amprente sunt stocate într-un fişier.
Cât de sigur este acesta ?

*Oscar* poate accesa – într-o modalitate oarecare – fişierul cu parole şi găseşte amprentele tuturor parolelor.

- Multe parole sunt cuvinte sau propoziţii din limbajul natural. *Oscar* poate crea (anterior atacului) o tabelă cu amprente ale tuturor cuvintelor din dicţionar, după care compară amprentele din fişier cu această tabelă. Este aşa numitul *dictionary attack*.

- Dacă două persoane au aceiaşi parolă (lucru frecvent posibil), *Oscar* va dispune de această informaţie (deoarece $h$ este o funcţie deterministă).

O soluţie la această problemă constă în utilizarea de factori aleatori în construirea funcţiilor de dispersie. Anume, odată cu crearea unei parole este generat şi un număr aleator $r$; acesta este amprentat odată cu parola. Numărul $r$ este păstrat petru verificare.
De exemplu

$$h(x) = (r, MD5(r, x)) \quad \text{sau} \quad h(x) = (r, r^{MD5(x)})$$

În acest fel se elimină multe din problemele generate de utilizarea frecventă a unei singure funcţii de dispersie.

## 1.10 Clase de dispersie tari universale

Aceste clase de funcţii de dispersie sunt utilizate pe scară largă în diverse arii criptografice. Să începem cu o definiţie:

**Definiţia 1.4.** *Fie $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ o $(N, M)$ - clasă de dispersie. Ea este "tare universală" dacă*

$$(\forall x, x' \in \mathcal{X})(\forall y, y' \in \mathcal{Y})(x \neq x') \Longrightarrow card\left(\{K \in \mathcal{K} \mid h_K(x) = y,\ h_K(x') = y'\}\right) = \frac{card(\mathcal{K})}{M^2}$$

**Exemplul 1.6.** *Fie*

$$\mathcal{X} = \mathcal{Y} = Z_3, \quad \mathcal{K} = Z_3 \times Z_3$$

*Pentru fiecare* $K = (a, b) \in \mathcal{K}$ *şi* $x \in \mathcal{X}$ *definim*

$$h_{a,b}(x) = ax + b \pmod 3$$

*Fie*

$$\mathcal{H} = \{h_{a,b} \mid (a, b) \in \mathcal{K}\}.$$

*Se verifică uşor că* $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ *formează o clasă de dispersie tare universală.*

**Lema 1.2.** *Fie* $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ *o* $(N, M)$ *- clasă de dispersie tare universală. Atunci*

$$(\forall x \in \mathcal{X})(\forall y \in \mathcal{Y}) \left( card\left(\{K \in \mathcal{K} \mid h_K(x) = y\}\right) = \frac{card(\mathcal{K})}{M} \right).$$

*Demonstraţie.* Fie $x, x' \in \mathcal{X},\ x \neq x'$ şi $y \in \mathcal{Y}$. Atunci
$card\left(\{K \in \mathcal{K} \mid h_K(x) = y\}\right) = \sum_{y' \in \mathcal{Y}} card(\{K \in \mathcal{K} \mid h_K(x) = y,\ h_K(x') = y'\}) =$
$= \sum_{y' \in \mathcal{Y}} \frac{card(\mathcal{K})}{M^2} = \frac{card(\mathcal{K})}{M}.$ □

Pe baza acestei leme se pot defini diverse construcţii de clase de dispersie tari universale. Astfel:

**Teorema 1.6.** *Fie* $q$ *un număr prim. Pentru* $a, b \in Z_q$ *definim funcţia* $f_{a,b} : Z_q \longrightarrow Z_q$ *prin*

$$f_{a,b}(x) = ax + b \pmod q$$

*Atunci* $(Z_q, Z_q, Z_q \times Z_q, \{f_{a,b} \mid a, b \in Z_q\})$ *este o clasă de dispersie tare universală.*

*Demonstraţie.* De remarcat că această teoremă generalizează Exemplul 1.6.
Fie $x, x', y, y' \in Z_q$ cu $x \neq x'$. Trebuie arătat că există o cheie unică $(a, b) \in Z_q \times Z_q$ cu $ax + b = y,\ ax' + b = y'$ (calculele sunt efectuate modulo $q$). Aceasta este soluţia (unică) a sistemului de două ecuaţii liniare, având ca necunoscute $a, b \in Z_q$. Mai exact

$$a = \frac{y' - y}{x' - x} \pmod q, \qquad b = y - x \cdot \frac{y' - y}{x' - x} \pmod q$$

□

**Teorema 1.7.** *Fie* $p$ *un număr întreg pozitiv şi* $q$ *un număr prim.*
*Definim* $\mathcal{X} = Z_2^p \setminus \{0, 0, \ldots, 0)\}$.

*Pentru fiecare* $\mathbf{r} \in Z_q^p$ *definim*

$$f_{\mathbf{r}}(\mathbf{x}) = \mathbf{r} \cdot \mathbf{x} \pmod q$$

*unde* $\mathbf{r} \cdot \mathbf{x}$ *este produsul scalar al vectorilor* $\mathbf{r}$ *şi* $\mathbf{x}$*, calculat modulo* $q$.

*Atunci* $(\mathcal{X}, Z_q, Z_q^p, \{f_{\mathcal{R}} \mid \mathcal{R} \in Z_q^p\})$ *este o clasă de de dispersie tare universală.*

*Demonstraţie.* Fie $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, $\mathbf{x} \neq \mathbf{x}'$ şi $y, y' \in Z_q$. Trebuie arătat că numărul vectorilor $\mathbf{r} \in Z_q^p$ astfel ca $\mathbf{r} \cdot \mathbf{x} = y \ (mod\ q)$, $\mathbf{r} \cdot \mathbf{x}' = y' \ (mod\ q)$ este constant. Vectorii $\mathbf{r}$ căutaţi sunt soluţia sistemului de două ecuaţii liniare cu $p$ necunoscute peste $Z_q$. Cele două ecuaţii sunt liniar independente; deci numărul de soluţii ale sistemului este constant: $q^{p-2}$. □

## 1.11 Exerciţii

**1.1.** *Fie $h : \mathcal{X} \longrightarrow \mathcal{Y}$ o $(N, M)$ - funcţie de dispersie. Pentru orice $y \in Y$ notăm $h^{-1}(y) = \{x \mid h(x) = y\}$ şi $s_y = |h^{-1}(y)|$. Fie*

$$S = card(\{\{x_1, x_2\} \mid x_1 \neq x_2,\ h(x_1) = h(x_2)\}).$$

*$S$ este numărul perechilor din $\mathcal{X}$ care formează o coliziune pentru $h$.*

- *Arătaţi relaţia $\sum_{y \in \mathcal{Y}} s_y = N$.*
  *Astfel, se poate defini media elementelor $s_y$ prin $\mathbf{s} = \frac{N}{M}$.*
- *Arătaţi relaţia $S = \sum_{y \in \mathcal{Y}} C_{s_y}^2 = \frac{1}{2} \sum_{y \in \mathcal{Y}} s_y^2 - \frac{N}{2}$.*
- *Arătaţi relaţia $\sum_{y \in \mathcal{Y}} (s_y - \mathbf{s})^2 = 2S + N - \frac{N^2}{M}$.*
- *Deduceţi inegalitatea $S \leq \frac{1}{2}\left(\frac{N^2}{M} - N\right)$.*

  *Arătaţi ca avem egalitate dacă şi numai dacă $s_y = \frac{N}{M}$ pentru orice $y \in \mathcal{Y}$.*

**1.2.** *Fie $p = 15083, \alpha = 154, \beta = 2307$ parametrii pentru funcţia de dispersie Chaum - van Heijst - Pfitzmann. Fiind dată coliziunea $\alpha^{7431}\beta^{5564} \equiv \alpha^{1459}\beta^{954} \ (mod\ p)$, calculaţi $log_\alpha \beta$.*

**1.3.** *([6]) Fie $n = pq$ unde $p, q$ sunt numere prime distincte (secrete) astfel încât $p = 2p_1 + 1$, $q = 2q_1 + 1$, $p, q$ numere prime. Fie $\alpha \in Z_n^*$ un element de ordin $2p_1q_1$ (acesta este ordinul maximal în $Z_n$). Se defineşte funcţia de dispersie $h : \{1, \ldots, n^2\} \longrightarrow Z_n^*$ prin*

$$h(x) = \alpha^x \ (mod\ n).$$

*Să presupunem $n = 603241$, $\alpha = 11$ şi că s-a găsit tripla coliziune*

$$h(1294755) = h(80115359) = h(52738737).$$

*Utilizaţi această informaţie pentru a factoriza $n$.*

**1.4.** *Fie $h_1 : Z_2^{2m} \longrightarrow Z_2^m$ o funcţie de dispersie cu coliziuni tari.*

- *Fie* $h_2 : Z_2^{4m} \longrightarrow Z_2^m$ *definită prin regulile:*
  - *scrie* $x \in Z_2^{4m}$ *sub forma* $x = x_1 \| x_2,\ x_1, x_2 \in Z_2^{2m}$*;*
  - *defineşte* $h_2(x) = h_1(h_1(x_1) \| h_1(x_2))$.

  *Arătaţi că* $h_2$ *este cu coliziuni tari.*

- *Pentru orice număr întreg* $i \geq 2$ *se defineşte funcţia de dispersie* $h_i : Z_2^{2^i m} \longrightarrow Z_2^m$ *definită recursiv prin regulile:*
  - *scrie* $x \in Z_2^{2^i m}$ *sub forma* $x = x_1 \| x_2,\ x_1, x_2 \in Z_2^{2^{i-1} m}$*;*
  - *defineşte* $h_i(x) = h_1(h_{i-1}(x_1) \| h_{i-1}(x_2))$.

  *Arătaţi că* $h_i$ *este cu coliziuni tari.*

**1.5.** *Fie* $f : Z_2^m \longrightarrow Z_2^m$ *o funcţie pentru care problema* **CSP** *este satisfăcută. Definim funcţia de dispersie* $h : Z_2^{2m} \longrightarrow Z_2^m$ *astfel: dacă* $x_1, x_2 \in Z_2^m$ *fie* $x = x_1 \| x_2$ *şi*

$$h(x) = f(x_1 \oplus x_2)$$

*Arătaţi că* $h$ *nu satisface problema* **CSP**.

**1.6.** *Fie* $q$ *un număr prim impar. Pentru* $a, b \in Z_q$ *definim* $f_{a,b} : Z_q \longrightarrow Z_q$ *prin*

$$f_{a,b}(x) = (x + a)^2 + b \pmod q$$

*Demonstraţi că* $(Z_q, Z_q, Z_q \times Z_q, \{f_{a,b} \mid a, b \in Z_q\})$ *este o* $(q, q)$ *- clasă de dispersie tare universală.*

# Bibliografie


[1] D. Chaum, E. van Heijst, B. Pfitzmann - *Cryptographically strong undeniable signatures, unconditionally secure for the signer*, Lecture Notes in Computer Science, 576 (1992), 470-484.

[2] I.B. Damgard -*A design principle for hash functions*, Lecture Notes in Computer Science, 435 (1990), 516-427.

[3] H. Delfs, H. Knebl - *Introduction to Cryptography, Second edition*, Springer Verlag, 2007

[4] H. Dobbertin - *Cryptanalysis of MD4*, Journal of Cryptology, 11 (1998), pp. 253-271.

[5] T. ElGamal - *A public key cryptosystem and a signature scheme based on discrete algorithms*, IEEE Transactions on Information Theory, 31 (1985), 469-472

[6] J. Gibson - *Discrete logarithm hash function that is collision free and one way*, IEEE Proceedings-E, 138 (1991), 407-410.

[7] V. Klima - *Tunnels in Hash Functions: MD5 Collisions within a Minute*, Cryptology ePrint Archive, http://eprint.iacr.org, Report 105 (2006)

[8] R.C. Merkle - *A fast software one-way functions and DES*, Lecture Notes in Computer Science, 435 (1990), 428-446

[9] A. Menezes, P. van Oorschot, S. Vanstome - *handbook of Applied Cryptography*, CRC Press Inc (1997)

[10] *Secure hash Standard*, National Bureau of Standards, FIPS Publications 180, 1993

[11] B. Preneel, R. Govaerts, J. Vandewalle - *Hash functions based on block ciphers: a syntetic approach*, LNCS, 773 (1994), pp. 368-378

[12] R.L. Rivest - *The MD4 message digest algorithm*, LNCS, 537, (1991), 303-311

[13] R. L. Rivest - *The MD5 Message Digest Algorithm*, RFC 1321 (1992)

[14] D. Stinton - *Cryptographie, theorie et pratique*, International Thompson Publishing France, 1995

[15] A. Salomaa - *Criptografie cu chei publice*, Ed. Militara, 1994

[16] S. Vaudenay - *A Classical Introduction to Cryptography*, Springer Verlag 2006

[17] H.C.Williams - *Some public-key criptofunctions as intractable as factorisation*, Cryptologia, 9 (1985), 224-237.