Funcții de dispersie

Prof. Dr. Emeritus Adrian Atanasiu

Universitatea București

May 6, 2018

Definiție

O funcție de dispersie (hash) este o funcție de compresie al cărei principal scop este asigurarea integrității unui mesaj.

Cu ajutorul ei se construiește o "amprentă" a mesajului, care – prin verificare periodică – certifică dacă acesta a fost modificat sau nu.

Nu se pune problema confidențialității mesajului, ci doar a integrității și autenticității sale.

Definiție

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

- $\mathcal{X}$ *este mulțimea mesajelor posibile (criptate sau nu);*
- $\mathcal{Y}$ *este o mulțime finită de "amprente" (sau "rezumate");*
- $\mathcal{K}$ *este o mulțime finită de chei;*
- *Pentru fiecare cheie* $K \in \mathcal{K}$ *există o funcție de dispersie* $h_K \in \mathcal{H}$,

$$h_K : \mathcal{X} \longrightarrow \mathcal{Y}$$

Deși $\mathcal{X}$ poate fi o mulțime oricât de mare – $\mathcal{Y}$ este o mulțime finită. Vom avea totdeauna $card(\mathcal{X}) \geq card(\mathcal{Y})$.

Aproape toate aplicațiile vor solicita o condiție 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 $h_K(x) = y$.

Deși $\mathcal{X}$ poate fi o mulțime oricât de mare – $\mathcal{Y}$ este o mulțime finită. Vom avea totdeauna $card(\mathcal{X}) \geq card(\mathcal{Y})$.

Aproape toate aplicațiile vor solicita o condiție 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 $h_K(x) = y$.

Cazul cel mai simplu este când $card(\mathcal{K}) = 1$, deci există o singură cheie și o singură funcție de dispersie $h : \mathcal{X} \longrightarrow \mathcal{Y}$.

Funcție criptografică

Vom considera o singură funcție de dispersie

$$h : \mathcal{X} \longrightarrow \mathcal{Y}$$

Din punct de vedere al securității (criptografice), unica modalitate de a obține o pereche validă $(x, y) \in \mathcal{X} \times \mathcal{Y}$ este de a alege x și apoi de a calcula $y = h(x)$.
Trebuie ca următoarele probleme să fie dificile.

- **NIP** (Preimage): Fiind dat $y \in \mathcal{Y}$ este dificil de aflat $x \in \mathcal{X}$ astfel ca $y = h(x)$.

Funcție criptografică

Vom considera o singură funcție de dispersie

$$h : \mathcal{X} \longrightarrow \mathcal{Y}$$

Din punct de vedere al securității (criptografice), unica modalitate de a obține o pereche validă $(x, y) \in \mathcal{X} \times \mathcal{Y}$ este de a alege x și apoi de a calcula $y = h(x)$.
Trebuie ca următoarele probleme să fie dificile.

- **NIP** (Preimage): Fiind dat $y \in \mathcal{Y}$ este dificil de aflat $x \in \mathcal{X}$ astfel ca $y = h(x)$.
- **CSP** (2nd Preimage): Fiind dată o pereche validă (x, y), este dificil de aflat $x_1 \neq x$ cu $h(x_1) = h(x)$.

Funcție criptografică

Vom considera o singură funcție de dispersie

$$h : \mathcal{X} \longrightarrow \mathcal{Y}$$

Din punct de vedere al securității (criptografice), unica modalitate de a obține o pereche validă $(x, y) \in \mathcal{X} \times \mathcal{Y}$ este de a alege x și apoi de a calcula $y = h(x)$.

Trebuie ca următoarele probleme să fie dificile.

- **NIP** (Preimage): Fiind dat $y \in \mathcal{Y}$ este dificil de aflat $x \in \mathcal{X}$ astfel ca $y = h(x)$.
- **CSP** (2nd Preimage): Fiind dată o pereche validă (x, y), este dificil de aflat $x_1 \neq x$ cu $h(x_1) = h(x)$.
- **CP** (Collision): Este dificil de aflat valorile distincte $x, x_1 \in \mathcal{X}$ astfel ca $h(x) = h(x_1)$.

Problema **CP** nu conduce direct la o pereche validă.
Dar, dacă (x, y) este o pereche validă și (x, x_1) este o coliziune, atunci (x_1, y) este de asemenea o pereche validă.

Problema **CP** nu conduce direct la o pereche validă.
Dar, dacă (x, y) este o pereche validă și (x, x_1) este o coliziune, atunci (x_1, y) este de asemenea o pereche validă.

Definiție

O funcție de dispersie

$$h : \mathcal{X} \longrightarrow \mathcal{Y}$$

pentru care problema **CP** *este dificilă se numește "rezistentă la coliziuni"'.*

Evident, dacă o funcție este rezistentă la coliziuni, atunci ea va rezista și la 2nd Preimage (coliziuni slabe). Deci **CP** implică **CSP**.

Teoremă

Fie $h : \mathcal{X} \longrightarrow \mathcal{Y}$ *o funcție de dispersie, unde* $\mathcal{X}, \mathcal{Y}$ *sunt mulțimi finite,*

$$N = card(\mathcal{X}),\ M = card(\mathcal{Y}),\ N \geq 2M$$

Să presupunem că există un algoritm **A** *de inversare a lui h. Atunci va exista un algoritm probabilist LasVegas care găsește o coliziune pentru h cu probabilitate cel puțin* $\frac{1}{2}$.

Lemă

CP *implică* **NIP**.

Rezultă că este suficient ca o funcție de dispersie să fie cu coliziuni tari.

Definiție

O funcție de dispersie care verifică **CP** *se numește "funcție de dispersie criptografică".*

Paradoxul naşterilor

Sub o formă simplificată:

$\boxed{\text{Probabilitatea ca dintr-o mulțime aleatoare de 23 persoane să existe doi indivizi cu aceeași zi de naștere, este cel puțin } 1/2.}$

Din acesta rezultă – drept condiție de securitate pentru evitarea generării de coliziuni – o margine inferioară pentru $M = card(\mathcal{Y})$.

Atacul nașterilor

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

Atacul naşterilor

Este un atac prin forță brută care urmărește aflarea unor coliziuni. În general, *Oscar* generează aleator mesajele $x_1, x_2, \dots \in \mathcal{X}$. Pentru fiecare mesaj x_i calculează și stochează amprenta $y_i = h(x_i)$, comparând-o cu valorile stocate anterior. 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.

O amprentă de 40 biți este vulnerabilă la un atac al naşterilor folosind numai 2^{20} (aproximativ un milion) mesaje aleatoare. Se sugerează folosirea de amprente de minim 256 biți (aici atacul naşterilor va cere 2^{128} calcule).

Semnături digitale

Sistemele de semnătură digitală se aplică amprentelor $h(x)$ ale mesajelor x.

Să presupunem că *Oscar* și *Bob* semnează un contract.

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

Semnături digitale

Sistemele de semnătură digitală se aplică amprentelor $h(x)$ ale mesajelor x.

Să presupunem că *Oscar* și *Bob* semnează un contract.

1. *Oscar* pregătește două versiuni ale contractului: o versiune m_1 echitabilă, și 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 obține două mesaje m_1' (similar lui m_1) și m_2' (similar cu m_2) având $h(m_1') = h(m_2')$.

Semnături digitale

Sistemele de semnătură digitală se aplică amprentelor $h(x)$ ale mesajelor x.

Să presupunem că *Oscar* și *Bob* semnează un contract.

1. *Oscar* pregătește două versiuni ale contractului: o versiune m_1 echitabilă, și 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 obține două mesaje m_1' (similar lui m_1) și m_2' (similar cu m_2) având $h(m_1') = h(m_2')$.
3. Cei doi semnează $h(m_1')$.

Ulterior *Oscar* poate pretinde că s-a semnat contractul m_2' .

Exemplu

Parole One-time (S - Chei)

- Plecând de la o valoare inițială x_0, se generează o secvență de amprente:

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

Exemplu

Parole One-time (S - Chei)

- Plecând de la o valoare inițială x_0, se generează o secvență de amprente:

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

- Sistemul original stochează x_n, iar utilizatorul folosește x_{n-1} ca parolă.

Exemplu

Parole One-time (S - Chei)

- Plecând de la o valoare inițială x_0, se generează o secvență de amprente:

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

- Sistemul original stochează x_n, iar utilizatorul folosește x_{n-1} ca parolă.
- Sistemul verifică $h(x_{n-1}) \stackrel{?}{=} x_n$.
  Oscar nu poate afla x_{n-1} chiar dacă știe x_n, deoarece h este o funcție de dispersie criptografică.

Exemplu

Parole One-time (S - Chei)

- Plecând de la o valoare inițială x_0, se generează o secvență de amprente:

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

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

Merkle - Damgard

Este o modalitate de construcție a unei funcții de dispersie, bazată pe compresie.

Permite extensia funcției de dispersie pe un domeniu infinit, păstrând proprietățile de securitate; deci se pot amprenta mesaje de lungime arbitrară.

Merkle - Damgard

Este o modalitate de construcție a unei funcții de dispersie, bazată pe compresie.

Permite extensia funcției de dispersie pe un domeniu infinit, păstrând proprietățile de securitate; deci se pot amprenta mesaje de lungime arbitrară.

Construcția Merkle - Damgard se referă la mesaje peste un alfabet binar $\mathbb{Z}_2 = \{0, 1\}$.

Fie $h : \mathbb{Z}_2^m \longrightarrow \mathbb{Z}_2^t$ o funcție criptografică de dispersie, cu $m \geq t + 1$.

Ea compresează blocuri de m biți în blocuri de t biți.

Scopul construcției este realizarea unei funcții de dispersie $h^* : \mathcal{X} \longrightarrow \mathbb{Z}_2^t$, unde

$$\mathcal{X} = \bigcup_{i=m}^{\infty} \mathbb{Z}_2^i$$

Cazul $m \geq t + 2$

Fiecare element al lui $\mathcal{X}$ este reprezentat printr-un șir de biți.

Fie $|x| = n \geq m$; x se poate scrie

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

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

Se obține $k = \left\lceil \dfrac{n}{m - t - 1} \right\rceil$.

Definim funcția $h^*(x)$ 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 $m-t-1$ biți).

Definim funcția $h^*(x)$ 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 $m-t-1$ biți).
4. $g_1 \longleftarrow h(0^{t+1} \| y_1)$

Definim funcția $h^*(x)$ 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 $m-t-1$ biți).
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. $h^*(x) \longleftarrow g_{k+1}$

Teoremă

Fie $h : \mathbb{Z}_2^m \longrightarrow \mathbb{Z}_2^t$ $(m \geq t+2)$ *o funcție de dispersie criptografică. Funcția*

$$h^* : \bigcup_{i=m}^{\infty} \mathbb{Z}_2^i \longrightarrow \mathbb{Z}_2^t$$

definită prin algoritmul anterior este de asemenea o funcție criptografică.

Cazul $m = t + 1$

Algoritmul pe care s-a bazat demonstrația de sus este adevărat numai dacă $m \geq t + 2$.
Pentru $m = t + 1$ se va realiza o construcție diferită pentru h^*.

Cazul $m = t + 1$

Algoritmul pe care s-a bazat demonstrația de sus este adevărat numai dacă $m \geq t + 2$.
Pentru $m = t + 1$ se va realiza o construcție diferită pentru h^*.
Vom începe prin a codifica x folosind funcția f definită

$$f(0) = 0, \quad f(1) = 01.$$

Cazul $m = t + 1$

Algoritmul pe care s-a bazat demonstrația de sus este adevărat numai dacă $m \geq t + 2$.
Pentru $m = t + 1$ se va realiza o construcție diferită pentru h^*.
Vom începe prin a codifica x folosind funcția f definită

$$f(0) = 0, \quad f(1) = 01.$$

Construcția lui h^* se face pe baza următorului algoritm:

1. $y \longleftarrow y_1 \| \ldots \| y_k = 11 \| f(x_1) \| f(x_2) \| \ldots \| f(x_n) \quad (y_i \in \mathbb{Z}_2)$

Cazul $m = t + 1$

Algoritmul pe care s-a bazat demonstrația de sus este adevărat numai dacă $m \geq t + 2$.
Pentru $m = t + 1$ se va realiza o construcție diferită pentru h^*.
Vom începe prin a codifica x folosind funcția f definită

$$f(0) = 0, \quad f(1) = 01.$$

Construcția lui h^* se face pe baza următorului algoritm:

1. $y \longleftarrow y_1 \| \ldots \| y_k = 11 \| f(x_1) \| f(x_2) \| \ldots \| f(x_n) \quad (y_i \in \mathbb{Z}_2)$
2. $g_1 \longleftarrow h(0^t \| y_1)$

Cazul $m = t + 1$

Algoritmul pe care s-a bazat demonstrația de sus este adevărat numai dacă $m \geq t + 2$.
Pentru $m = t + 1$ se va realiza o construcție diferită pentru h^*.
Vom începe prin a codifica x folosind funcția f definită

$$f(0) = 0, \quad f(1) = 01.$$

Construcția lui h^* se face pe baza următorului algoritm:

1. $y \longleftarrow y_1\|\ldots\|y_k = 11\|f(x_1)\|f(x_2)\|\ldots\|f(x_n) \;\; (y_i \in \mathbb{Z}_2)$
2. $g_1 \longleftarrow h(0^t\|y_1)$
3. **for** $i = 1$ **to** $k - 1$ **do** $g_{i+1} \longleftarrow h(g_i\|y_{i+1})$
4. $h^*(x) \longleftarrow g_k$

Teoremă

Fie $h : \mathbb{Z}_2^{t+1} \longrightarrow \mathbb{Z}_2^t$ *o funcție de dispersie criptografică.*

Funcția $h^* : \bigcup_{i=t+1}^{\infty} \mathbb{Z}_2^i \longrightarrow \mathbb{Z}_2^t$ *este de asemenea o funcție criptografică.*

Teoremă

Fie $h : \mathbb{Z}_2^{t+1} \longrightarrow \mathbb{Z}_2^t$ *o funcție de dispersie criptografică.*
Funcția $h^* : \bigcup\limits_{i=t+1}^{\infty} \mathbb{Z}_2^i \longrightarrow \mathbb{Z}_2^t$ *este de asemenea o funcție criptografică.*

Teoremă

Fie $h : \mathbb{Z}_2^m \longrightarrow \mathbb{Z}_2^t, (m \geq t+1)$ *o funcție de dispersie criptografică.*
Există o funcție de dispersie criptografică

$$h^* : \bigcup_{i=m}^{\infty} \mathbb{Z}_2^i \longrightarrow \mathbb{Z}_2^t.$$

Dispersii bazate pe sisteme de criptare

Fie $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ este un sistem de criptare simetric, cu $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_2^n$.
Pentru a evita atacul naşterilor, $n \geq 128$.

Dispersii bazate pe sisteme de criptare

Fie $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ este un sistem de criptare simetric, cu $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_2^n$.
Pentru a evita atacul naşterilor, $n \geq 128$.
Fie şirul de biţi

$$x = x_1 \| x_2 \| \ldots \| x_k, \qquad x_i \in \mathbb{Z}_2^n \quad 1 \leq i \leq k.$$

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

Dispersii bazate pe sisteme de criptare

Fie $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ este un sistem de criptare simetric, cu $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_2^n$.
Pentru a evita atacul naşterilor, $n \geq 128$.
Fie şirul de biţi

$$x = x_1 \| x_2 \| \ldots \| x_k, \qquad x_i \in \mathbb{Z}_2^n \quad 1 \leq i \leq k.$$

Ideea construcţiei constă în fixarea unei *valori iniţiale* g_0 şi calcularea iterativă a lui $g_1, \ldots, g_k$ 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 funcții de dispersie, o parte din ele fiind sparte (independent de sistemul de criptare utilizat).

$g_i = e_{g_{i-1}}(x_i) \oplus x_i$ (Mathyas - Meyer - Oseas)

$g_i = e_{g_{i-1}}(x_i) \oplus g_{i-1}$ (Davies - Meyer)

$g_i = e_{g_{i-1}}(x_i) \oplus x_i \oplus g_{i-1}$ (Miyaguchi - Preneel)

$g_i = e_{g_{i-1}}(x_i \oplus g_{i-1}) \oplus x_i$

$g_i = e_{g_{i-1}}(x_i \oplus g_{i-1}) \oplus x_i \oplus g_{i-1}$

Funcția Matyias - Meyer - Oseas

Este inclusă în standardul ISO/IEC $10118 - 2$.

Funcția Matyias - Meyer - Oseas

Este inclusă în standardul ISO/IEC $10118-2$.

1. Blocul $z \in \mathcal{X}$ de lungime $2n$ este spart în două jumătăți: $z = x \| y$.

Funcția Matyias - Meyer - Oseas

Este inclusă în standardul ISO/IEC $10118 - 2$.

1. Blocul $z \in \mathcal{X}$ de lungime $2n$ este spart în două jumătăți: $z = x\|y$.
2. Se determină cheia K folosind o funcție $g : \mathbb{Z}_2^n \longrightarrow \mathbb{Z}_2^k$ (uzual $K = g(y)$ se obține din y cu ajutorul unei funcții booleene).

Funcția Matyias - Meyer - Oseas

Este inclusă în standardul ISO/IEC $10118-2$.

1. Blocul $z \in \mathcal{X}$ de lungime $2n$ este spart în două jumătăți: $z = x\|y$.
2. Se determină cheia K folosind o funcție $g : \mathbb{Z}_2^n \longrightarrow \mathbb{Z}_2^k$ (uzual $K = g(y)$ se obține din y cu ajutorul unei funcții booleene).
3. Amprenta lui z este

$$h(z) = e_K(x) \oplus x$$

Securitatea este în strânsă dependență cu securitatea oferită de sistemul de criptare folosit.

*MD*4

Funcțiile **MDi** (Message Digest nr. *i*) au fost construite de Ron Rivest.
În particular, *MD*4 a fost introdusă în 1990 pentru procesoare pe 32 biți.
Deși nu s-a dovedit suficient de sigură, principiile sale de construcție au fost ulterior utilizate pentru o clasă largă de funcții de dispersie, cum ar fi *MD*5, *SHA*, *SHA*1, *RIPEMD*160.

MD4

Funcțiile **MDi** (Message Digest nr. i) au fost construite de Ron Rivest.
În particular, *MD4* a fost introdusă în 1990 pentru procesoare pe 32 biți.
Deși nu s-a dovedit suficient de sigură, principiile sale de construcție au fost ulterior utilizate pentru o clasă largă de funcții de dispersie, cum ar fi *MD*5, *SHA*, *SHA*1, *RIPEMD*160.

Cu toate că securitatea lor nu a fost demonstrată, funcțiile de dispersie **MDi** au avantajul de a fi foarte rapide, deci practice pentru semnarea de mesaje lungi.

Descrierea funcției *MD*4

Fiind dat un șir $x \in \mathbb{Z}_2^*$, se definește tabloul $M = M_0 M_1 \ldots M_{n-1}$ cu fiecare "*cuvânt*" M_i de lungime 32 și $n \equiv 0 \ (mod\ 16)$.

Descrierea funcției *MD4*

Fiind dat un șir $x \in \mathbb{Z}_2^*$, se definește tabloul $M = M_0 M_1 \ldots M_{n-1}$ cu fiecare "*cuvânt*" M_i de lungime 32 și $n \equiv 0 \ (mod\ 16)$.
M este construit folosind algoritmul următor:

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

Descrierea funcției *MD4*

Fiind dat un șir $x \in \mathbb{Z}_2^*$, se definește tabloul $M = M_0 M_1 \ldots M_{n-1}$ cu fiecare "*cuvânt*" M_i de lungime 32 și $n \equiv 0 \ (mod\ 16)$.
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. $M = x \| 1 \| 0^d \| s$

$MD4$ construiește o amprentă a lui x pe 128 biți.

$MD4$ construiește o amprentă a lui x pe 128 biți.

1. $A \longleftarrow (67452301_{16},\ \ B \longleftarrow (efcdab89)_{16},$
   $C \longleftarrow (98badcfe)_{16},\ D \longleftarrow (10325476)_{16}.$

$MD4$ construiește o amprentă a lui x pe 128 biți.

1. $A \longleftarrow (67452301_{16},\ \ B \longleftarrow (efcdab89)_{16},$
   $C \longleftarrow (98badcfe)_{16},\ D \longleftarrow (10325476)_{16}.$
2. **for** $i = 0$ **to** $n/16 - 1$ **do**
   2.1. **for** $j = 0$ **to** 15 **do** $X_j \longleftarrow M_{16i+j}$

$MD4$ construiește o amprentă a lui x pe 128 biți.

1. $A \longleftarrow (67452301_{16}, \ \ B \longleftarrow (efcdab89)_{16},$
   $C \longleftarrow (98badcfe)_{16}, \ D \longleftarrow (10325476)_{16}.$
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. $AA \longleftarrow A, \ BB \longleftarrow B, \ CC \longleftarrow C, \ DD \longleftarrow D.$

$MD4$ construiește o amprentă a lui x pe 128 biți.

1. $A \longleftarrow (67452301_{16}, \; B \longleftarrow (efcdab89)_{16},$
   $C \longleftarrow (98badcfe)_{16}, \; D \longleftarrow (10325476)_{16}.$
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**

$MD4$ construiește o amprentă a lui x pe 128 biți.

1. $A \longleftarrow (67452301_{16}, \;\; B \longleftarrow (efcdab89)_{16},$
   $C \longleftarrow (98badcfe)_{16}, \;\; D \longleftarrow (10325476)_{16}.$
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 concatenarea celor patru cuvinte A, B, C, D.

$MD4$ construiește o amprentă a lui x pe 128 biți.

1. $A \longleftarrow (67452301_{16}, \ \ B \longleftarrow (efcdab89)_{16},$
   $C \longleftarrow (98badcfe)_{16}, \ D \longleftarrow (10325476)_{16}.$
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 concatenarea celor patru cuvinte A, B, C, D.
Tabelul M este împărțit în secvențe de câte 16 cuvinte, cărora li se aplică trei runde de transformări.

Construcția standard a lui *MD*4 folosește o arhitectură *little-endian*.

La o implementare *big-endian*, adunările se definesc astfel:

1. Se inter-schimbă

$$x_1 \leftrightarrow x_4, \ x_2 \leftrightarrow x_3, \ y_1 \leftrightarrow y_4, \ y_2 \leftrightarrow y_3$$

Construcția standard a lui $MD4$ folosește o arhitectură *little-endian*.

La o implementare *big-endian*, adunările se definesc 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 $Z = X + Y \; (mod \; 2^{32})$

Construcția standard a lui *MD*4 folosește o arhitectură *little-endian*.

La o implementare *big-endian*, adunările se definesc 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 $z_1 \leftrightarrow z_4,\; z_2 \leftrightarrow z_3$

Construcția standard a lui *MD*4 folosește o arhitectură *little-endian*.

La o implementare *big-endian*, adunările se definesc 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 construcție pentru *MD*4 folosesc trei funcții
$$f, g, h : \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \longrightarrow \mathbb{Z}_2^{32}$$
$f(X, Y, Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$
$g(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$
$s(X, Y, Z) = X \oplus Y \oplus Z$
și constantele $P = 5a827999,\ \ Q = 6ed9eba1$.

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. $D \longleftarrow (D + f(A, B, C) + X_5) \ll 7$
7. $C \longleftarrow (C + f(D, A, B) + X_6) \ll 11$
8. $B \longleftarrow (B + f(C, D, A) + X_7) \ll 19$
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. $D \longleftarrow (D + f(A, B, C) + X_{13}) \ll 7$
15. $C \longleftarrow (C + f(D, A, B) + X_{14}) \ll 11$
16. $B \longleftarrow (B + f(C, D, A) + X_{15}) \ll 19$

Etapa 2:

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

Etapa 3:

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

Criptanaliza schemei *MD4*

Etapele funcției *MD4* sunt caracterizate de permutări σ_i ale blocului $X = (X_0, \ldots X_{15})$ și deplasări ciclice spre stânga $\alpha_{i,j}$, $(i = 1, 2, 3,\ j = 1, 2, \ldots, 16)$.

Criptanaliza schemei *MD4*

Etapele funcției *MD4* sunt caracterizate de permutări σ_i ale blocului $X = (X_0, \ldots X_{15})$ și 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

$i/j \pmod 4$	0	1	2	3
1	3	7	11	19
2	3	5	9	13
3	3	9	11	15

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

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

Proprietăți ale funcțiile f și g:

- $f(\mathbf{1}, a, x) = f(\mathbf{0}, x, a) = f(x, a, a) = a, \ \forall a, x \in \mathbb{Z}_2^{32}$, $\mathbf{0} = 00\ldots0, \ \ \mathbf{1} = 11\ldots1$;
- $g(a, a, x) = g(a, x, a) = g(x, a, a) = a, \ \forall a, x \in \mathbb{Z}_2^{32}$;

În prima etapă σ_1 este permutarea identică, iar X_i sunt utilizate în ordinea inițială.

Deci, dacă $X_0, \ldots, X_{11}$ sunt fixate, se pot alege ușor valori pentru X_{12}, X_{13}, X_{14} astfel ca să se obțină 0 în regiștrii A, C și D.

În prima etapă σ_1 este permutarea identică, iar X_i sunt utilizate în ordinea inițială.

Deci, dacă $X_0, \ldots, X_{11}$ sunt fixate, se pot alege ușor valori pentru X_{12}, X_{13}, X_{14} astfel ca să se obțină 0 în regiștrii A, C și D.

Permutarea σ_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, regiștrii A, C, D rămân cu valoarea $\mathbf{0}$.

În prima etapă σ_1 este permutarea identică, iar X_i sunt utilizate în ordinea inițială.

Deci, dacă $X_0, \ldots, X_{11}$ sunt fixate, se pot alege ușor valori pentru X_{12}, X_{13}, X_{14} astfel ca să se obțină 0 în regiștrii A, C și D.

Permutarea σ_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, regiștrii A, C, D rămân cu valoarea $\mathbf{0}$.
Această proprietate are loc pentru orice valoare a lui X_{15}.

Atacul:

Atacul ignoră Etapa 3 și consideră ieșirea din Etapa 2 ca o funcție (aleatoare) de variabilă X_{15}, în care trei regiștri sunt permanent $\mathbf{0}$.

Deci dimensiunea aleatoare este redusă la 32 biți; rezultă că un atac bazat pe paradoxul nașterilor poate deduce o coliziune în aproximativ 2^{16} încercări.

Atac asupra lui *MD4* redus la primele două etape:

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**
   $(A, D, B, C) \longleftarrow (D, C, B, (A + f(B, C, D) + X_i) \ll \alpha_{1,i})$
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**
   $X'_i = X_i \quad (0 \le i \le 14)$
   $Output(X, X');$ $\quad$ *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 atacul pentru primele două etape, permutarea σ_3 asociază pe X_{15} (singura valoare modificată) doar registrului B.

După atacul pentru primele două etape, permutarea σ_3 asociază pe X_{15} (singura valoare modificată) doar registrului B.

Deci două blocuri de câte 128 biți X și X' ($X_i = X'_i$ pentru $i = 0, \ldots, 14$, $X_{15} \neq X'_{15}$) care dau o coliziune după primele două etape, au proprietatea că $h(X)$ și $h(X')$ diferă doar pe al doilea bloc de 32 biți.

După atacul pentru primele două etape, permutarea σ_3 asociază pe X_{15} (singura valoare modificată) doar registrului B.

Deci două blocuri de câte 128 biți X și X' ($X_i = X'_i$ pentru $i = 0, \ldots, 14$, $X_{15} \neq X'_{15}$) care dau o coliziune după primele două etape, au proprietatea că $h(X)$ și $h(X')$ diferă doar pe al doilea bloc de 32 biți.

Se pot obține ușor îmbunătățiri în care cele două amprente să difere doar într-un singur bit din zona respectivă.

Pe baza acestui atac Hans Dobbertin construiește un algoritm care generează toate coliziunile din $MD4$.

Funcția de dispersie *MD*5

Pentru a elimina slăbiciunile lui *MD*4, în 1991 Ron Rivest lansează varianta *MD*5, publicată în 1992 sub numele *RFC* 1321 *Internet Standard*.

Funcția de dispersie *MD*5

Pentru a elimina slăbiciunile lui *MD*4, în 1991 Ron Rivest lansează varianta *MD*5, publicată în 1992 sub numele *RFC* 1321 *Internet Standard*.

*MD*5 se bazează pe o funcție de "*criptare*" g propusă de Davies - Meyer: dacă aceasta este

$$g : \mathbb{Z}_2^{128} \times \mathbb{Z}_2^{512} \longrightarrow \mathbb{Z}_2^{128}$$

atunci

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

unde adunarea este realizată modulo 2^{32}.

Criptarea $g(H, X)$ constă din 4 runde și poate fi descrisă pe scurt:

1. Mesajul H de 128 biți este aranjat printr-o permutare (care depinde de rundă) într-o secvență de cuvinte A, B, C, D;

Criptarea $g(H, X)$ constă din 4 runde și poate fi descrisă pe scurt:

1. Mesajul H de 128 biți este aranjat printr-o permutare (care depinde de rundă) într-o secvență de cuvinte A, B, C, D;
2. Fiecare cuvânt trece printr-o transformare secvențială, bazată pe o schemă Feistel;

Criptarea $g(H, X)$ constă din 4 runde și poate fi descrisă pe scurt:

1. Mesajul H de 128 biți este aranjat printr-o permutare (care depinde de rundă) într-o secvență de cuvinte A, B, C, D;
2. Fiecare cuvânt trece printr-o transformare secvențială, bazată pe o schemă Feistel;
3. O transformare este definită de un S - box cu intrarea (A, K) (K – cheie de rundă derivată din X) și ieșirea B, C, D;

Criptarea $g(H, X)$ constă din 4 runde și poate fi descrisă pe scurt:

1. Mesajul H de 128 biți este aranjat printr-o permutare (care depinde de rundă) într-o secvență de cuvinte A, B, C, D;
2. Fiecare cuvânt trece printr-o transformare secvențială, bazată pe o schemă Feistel;
3. O transformare este definită de un S - box cu intrarea (A, K) (K – cheie de rundă derivată din X) și ieșirea B, C, D;
4. Ieșirea din S box este

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

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

$$f_1(B, C, D) = (B \wedge C) \vee ((\neg B) \wedge D), \quad f_3(B, C, D) = B \oplus C \oplus D,$$
$$f_2(B, C, D) = (D \wedge B) \vee ((\neg D) \wedge C), \quad f_4(B, C, D) = C \oplus (B \wedge (\neg D))$$

Și această funcție de dispersie – utilizată mai ales în aplicații Internet – a fost spartă destul de repede.

Mai mult, în 2006 V. Klima publică un algoritm care calculează – în mai puțin de un minut – coliziuni pentru $MD5$, folosind un PC standard.

Funcția de dispersie SHA-1

SHA-1 este o variantă a funcției de dispersie *SHA* – notată ca standard *FIPS* $180 - 1$ – care corectează o mică slăbiciune din *SHA*.

Funcția de dispersie SHA-1

SHA-1 este o variantă a funcției de dispersie *SHA* – notată ca standard *FIPS* $180-1$ – care corectează o mică slăbiciune din *SHA*.
Construcția sa este următoarea:
Fie x ($|x| \leq 2^{64}-1$) șirul binar care trebuie amprentat.

Funcția de dispersie SHA-1

SHA-1 este o variantă a funcției de dispersie *SHA* – notată ca standard *FIPS* $180-1$ – care corectează o mică slăbiciune din *SHA*.
Construcția sa este următoarea:
Fie x ($|x| \leq 2^{64}-1$) șirul 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.

Dacă $|s| < 64$, se adaugă zerouri la stânga.

M are o lungime divizibilă cu 512; se sparge în blocuri de 512 biți:

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

Dacă $|s| < 64$, se adaugă zerouri la stânga.
M are o lungime divizibilă cu 512; se sparge în blocuri de 512 biți:

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

Se definesc funcțiile

$$f_i : \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \longrightarrow \mathbb{Z}_2^{32} \ (0 \le i \le 79)$$

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

Dacă $|s| < 64$, se adaugă zerouri la stânga.
M are o lungime divizibilă cu 512; se sparge în blocuri de 512 biți:

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

Se definesc funcțiile

$$f_i : \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \longrightarrow \mathbb{Z}_2^{32} \; (0 \le i \le 79)$$

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

și constantele $K_0, K_1, \ldots, K_{79}$:

$$K_t = \begin{cases} (5ab27999)_{16} & 0 \le t \le 19 \\ (6ed9eba1)_{16} & 20 \le t \le 39 \\ (8f1bbcdc)_{16} & 40 \le t \le 59 \\ (ca62c1d6)_{16} & 60 \le t \le 79 \end{cases}$$

Algoritmul SHA-1

1. $H_0 \longleftarrow (67452301)_{16},\ H_1 \longleftarrow (efcdab89)_{16},\ H_2 \longleftarrow (98badcfe)_{16},$
$H_3 \longleftarrow (10325476)_{16},\ H_4 \longleftarrow (c3d2e1f0)_{16}.$

Algoritmul SHA-1

1. $H_0 \longleftarrow (67452301)_{16},\ H_1 \longleftarrow (efcdab89)_{16},\ H_2 \longleftarrow (98badcfe)_{16},$
   $H_3 \longleftarrow (10325476)_{16},\ H_4 \longleftarrow (c3d2e1f0)_{16}.$
2. **for** $i \longleftarrow 1$ **to** n **do**
   1. Fie $M_i = W_0 \| W_1 \| \ldots \| W_{15}, \quad W_i \in \mathbb{Z}_2^{32}.$

Algoritmul SHA-1

1. $H_0 \longleftarrow (67452301)_{16},\ H_1 \longleftarrow (efcdab89)_{16},\ H_2 \longleftarrow (98badcfe)_{16},$ $H_3 \longleftarrow (10325476)_{16},\ H_4 \longleftarrow (c3d2e1f0)_{16}.$
2. **for** $i \longleftarrow 1$ **to** n **do**
    1. Fie $M_i = W_0 \| W_1 \| \ldots \| W_{15}, \quad W_i \in \mathbb{Z}_2^{32}$.
    2. **for** $t \longleftarrow 16$ **to** 79 **do**
       $W_t \longleftarrow (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \lll 1)$

Algoritmul SHA-1

1. $H_0 \longleftarrow (67452301)_{16},\ H_1 \longleftarrow (efcdab89)_{16},\ H_2 \longleftarrow (98badcfe)_{16},$
   $H_3 \longleftarrow (10325476)_{16},\ H_4 \longleftarrow (c3d2e1f0)_{16}.$
2. **for** $i \longleftarrow 1$ **to** n **do**
   1. Fie $M_i = W_0 \| W_1 \| \ldots \| W_{15},\quad W_i \in \mathbb{Z}_2^{32}$.
   2. **for** $t \longleftarrow 16$ **to** 79 **do**
      $W_t \longleftarrow (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \ll 1)$
   3. $A \longleftarrow H_0,\ B \longleftarrow H_1,\ C \longleftarrow H_2,\ D \longleftarrow H_3,\ E \longleftarrow H_4.$

Algoritmul SHA-1

1. $H_0 \longleftarrow (67452301)_{16},\ H_1 \longleftarrow (efcdab89)_{16},\ H_2 \longleftarrow (98badcfe)_{16},$
   $H_3 \longleftarrow (10325476)_{16},\ H_4 \longleftarrow (c3d2e1f0)_{16}.$
2. **for** $i \longleftarrow 1$ **to** n **do**
   1. Fie $M_i = W_0 \| W_1 \| \ldots \| W_{15}, \quad W_i \in \mathbb{Z}_2^{32}$.
   2. **for** $t \longleftarrow 16$ **to** 79 **do**
      $W_t \longleftarrow (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \ll 1)$
   3. $A \longleftarrow H_0,\ B \longleftarrow H_1,\ C \longleftarrow H_2,\ D \longleftarrow H_3,\ E \longleftarrow H_4.$
   4. **for** $t \longleftarrow 0$ **to** 79 **do**
      1. $temp \longleftarrow (A \ll 5) + f_t(B, C, D) + E + W_t + K_t$

Algoritmul SHA-1

1. $H_0 \longleftarrow (67452301)_{16},\ H_1 \longleftarrow (efcdab89)_{16},\ H_2 \longleftarrow (98badcfe)_{16},$
   $H_3 \longleftarrow (10325476)_{16},\ H_4 \longleftarrow (c3d2e1f0)_{16}.$
2. **for** $i \longleftarrow 1$ **to** n **do**
   1. Fie $M_i = W_0 \| W_1 \| \dots \| W_{15}, \quad W_i \in \mathbb{Z}_2^{32}.$
   2. **for** $t \longleftarrow 16$ **to** 79 **do**
      $W_t \longleftarrow (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \ll 1)$
   3. $A \longleftarrow H_0,\ B \longleftarrow H_1,\ C \longleftarrow H_2,\ D \longleftarrow H_3,\ E \longleftarrow H_4.$
   4. **for** $t \longleftarrow 0$ **to** 79 **do**
      1. $temp \longleftarrow (A \ll 5) + f_t(B, C, D) + E + W_t + K_t$
      2. $E \longleftarrow D,\ D \longleftarrow C, C \longleftarrow (B \ll 30),\ B \longleftarrow A$

Algoritmul SHA-1

1. $H_0 \longleftarrow (67452301)_{16},\ H_1 \longleftarrow (efcdab89)_{16},\ H_2 \longleftarrow (98badcfe)_{16},$
   $H_3 \longleftarrow (10325476)_{16},\ H_4 \longleftarrow (c3d2e1f0)_{16}.$
2. **for** $i \longleftarrow 1$ **to** n **do**
   1. Fie $M_i = W_0 \| W_1 \| \dots \| W_{15}, \quad W_i \in \mathbb{Z}_2^{32}.$
   2. **for** $t \longleftarrow 16$ **to** 79 **do**
      $W_t \longleftarrow (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \lll 1)$
   3. $A \longleftarrow H_0,\ B \longleftarrow H_1,\ C \longleftarrow H_2,\ D \longleftarrow H_3,\ E \longleftarrow H_4.$
   4. **for** $t \longleftarrow 0$ **to** 79 **do**
      1. $temp \longleftarrow (A \lll 5) + f_t(B, C, D) + E + W_t + K_t$
      2. $E \longleftarrow D,\ D \longleftarrow C, C \longleftarrow (B \lll 30),\ B \longleftarrow A$
      3. $A \longleftarrow temp.$

Algoritmul SHA-1

1. $H_0 \longleftarrow (67452301)_{16},\ H_1 \longleftarrow (efcdab89)_{16},\ H_2 \longleftarrow (98badcfe)_{16},$
   $H_3 \longleftarrow (10325476)_{16},\ H_4 \longleftarrow (c3d2e1f0)_{16}.$
2. **for** $i \longleftarrow 1$ **to** n **do**
   1. Fie $M_i = W_0 \| W_1 \| \ldots \| W_{15}, \quad W_i \in \mathbb{Z}_2^{32}.$
   2. **for** $t \longleftarrow 16$ **to** 79 **do**
      $W_t \longleftarrow (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \ll 1)$
   3. $A \longleftarrow H_0,\ B \longleftarrow H_1,\ C \longleftarrow H_2,\ D \longleftarrow H_3,\ E \longleftarrow H_4.$
   4. **for** $t \longleftarrow 0$ **to** 79 **do**
      1. $temp \longleftarrow (A \ll 5) + f_t(B, C, D) + E + W_t + K_t$
      2. $E \longleftarrow D,\ D \longleftarrow C, C \longleftarrow (B \ll 30),\ B \longleftarrow A$
      3. $A \longleftarrow temp.$
   5. $H_0 \longleftarrow H_0 + A,\ H_1 \longleftarrow H_1 + B,\ H_2 \longleftarrow H_2 + C,$
      $H_3 \longleftarrow H_3 + D,\ H_4 \longleftarrow H_4 + E.$

Algoritmul SHA-1

1. $H_0 \longleftarrow (67452301)_{16},\ H_1 \longleftarrow (efcdab89)_{16},\ H_2 \longleftarrow (98badcfe)_{16},$
   $H_3 \longleftarrow (10325476)_{16},\ H_4 \longleftarrow (c3d2e1f0)_{16}.$
2. **for** $i \longleftarrow 1$ **to** n **do**
   1. Fie $M_i = W_0 \| W_1 \| \ldots \| W_{15}, \quad W_i \in \mathbb{Z}_2^{32}.$
   2. **for** $t \longleftarrow 16$ **to** 79 **do**
      $W_t \longleftarrow (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} \ll 1)$
   3. $A \longleftarrow H_0,\ B \longleftarrow H_1,\ C \longleftarrow H_2,\ D \longleftarrow H_3,\ E \longleftarrow H_4.$
   4. **for** $t \longleftarrow 0$ **to** 79 **do**
      1. $temp \longleftarrow (A \ll 5) + f_t(B, C, D) + E + W_t + K_t$
      2. $E \longleftarrow D,\ D \longleftarrow C, C \longleftarrow (B \ll 30),\ B \longleftarrow A$
      3. $A \longleftarrow temp.$
   5. $H_0 \longleftarrow H_0 + A,\ H_1 \longleftarrow H_1 + B,\ H_2 \longleftarrow H_2 + C,$
      $H_3 \longleftarrow H_3 + D,\ H_4 \longleftarrow H_4 + E.$
3. **Output**: $y = H_0 \| H_1 \| H_2 \| H_3 \| H_4.$

Diferența dintre SHA și SHA-1: în SHA transformarea de la pasul (2.b) se realizează fără nici o rotație (ceea ce permitea la SHA aflarea coliziunilor în aproximativ 2^{61} pași).

Diferența dintre SHA și SHA-1: în SHA transformarea de la pasul $(2.b)$ se realizează fără nici o rotație (ceea ce permitea la SHA aflarea coliziunilor în aproximativ 2^{61} pași).
Versiunea SHA-1 este mai eficientă, atacul nașterilor conducând la aflarea coliziunilor după 2^{80} pași.

Diferența dintre SHA și SHA-1: în SHA transformarea de la pasul $(2.b)$ se realizează fără nici o rotație (ceea ce permitea la SHA aflarea coliziunilor în aproximativ 2^{61} pași).
Versiunea SHA-1 este mai eficientă, atacul nașterilor conducând la aflarea coliziunilor după 2^{80} pași.

Dar și pentru această funcție de dispersie s-au publicat coliziuni începând cu 2005, când o echipă de la Universitatea Shandong afirmă că a găsit coliziuni pentru SHA-1 în 2^{69} operații (iar pentru o variantă SHA-1 de 58 runde, în numai 2^{33} operații) .

Diferența dintre SHA și SHA-1: în SHA transformarea de la pasul (2.b) se realizează fără nici o rotație (ceea ce permitea la SHA aflarea coliziunilor în aproximativ 2^{61} pași).
Versiunea SHA-1 este mai eficientă, atacul nașterilor conducând la aflarea coliziunilor după 2^{80} pași.

Dar și pentru această funcție de dispersie s-au publicat coliziuni începând cu 2005, când o echipă de la Universitatea Shandong afirmă că a găsit coliziuni pentru SHA-1 în 2^{69} operații (iar pentru o variantă SHA-1 de 58 runde, în numai 2^{33} operații) .

La 30 mai 2001 *NIST* propune versiunea SHA-2; aceasta include SHA-1 combinat cu alte funcții de dispersie: SHA-256, SHA-384, SHA-512 (indicii se referă la mărimea amprentei).

Limitele funcțiilor de dispersie SHA-1 și SHA-2

Ambele au la bază același concept pentru procesarea textelor: Merkle-Damgard.
Deci orice atac reușit asupra lui SHA-1 devine o falie de securitate pentru SHA-2.

Limitele funcțiilor de dispersie SHA-1 și SHA-2

Ambele au la bază același concept pentru procesarea textelor: Merkle-Damgard.
Deci orice atac reușit asupra lui SHA-1 devine o falie de securitate pentru SHA-2.

SHA-2 este considerată încă o funcție de dispersie criptografică sigură: singurele atacuri reușite care se cunosc sunt asupra lui SHA-2 cu un număr redus de runde: 46-runde (varianta de 512 biți) și 41 runde (varianta de 256 biți).

Cerințe NIST

În 2006 NIST a anunțat competiția pentru standardul SHA-3.

Cerințe NIST

În 2006 NIST a anunțat competiția pentru standardul SHA-3.
Dintre cerințele eliminatorii impuse de NIST:

- Performanțele să aceleași indiferent de implementare.

Cerințe NIST

În 2006 NIST a anunțat competiția pentru standardul SHA-3.
Dintre cerințele eliminatorii impuse de NIST:

- Performanțele să aceleași indiferent de implementare.
- Resursa consumată trebuie să fie minimă chiar pentru o cantitate mare de text la intrare.

Cerințe NIST

În 2006 NIST a anunțat competiția pentru standardul SHA-3.
Dintre cerințele eliminatorii impuse de NIST:

- Performanțele să aceleași indiferent de implementare.
- Resursa consumată trebuie să fie minimă chiar pentru o cantitate mare de text la intrare.
- Performanțele de securitate trebuie să nu fie inferioare funcțiilor actuale de dispersie.

Cerințe NIST

În 2006 NIST a anunțat competiția pentru standardul SHA-3.
Dintre cerințele eliminatorii impuse de NIST:

- Performanțele să aceleași indiferent de implementare.
- Resursa consumată trebuie să fie minimă chiar pentru o cantitate mare de text la intrare.
- Performanțele de securitate trebuie să nu fie inferioare funcțiilor actuale de dispersie.
- Să scoată aceleași marimi de amprente ca SHA-2, dar să fie capabilă – la nevoie – de amprente de mărimi mai mari.

Cerințe NIST

În 2006 NIST a anunțat competiția pentru standardul SHA-3.
Dintre cerințele eliminatorii impuse de NIST:

- Performanțele să aceleași indiferent de implementare.
- Resursa consumată trebuie să fie minimă chiar pentru o cantitate mare de text la intrare.
- Performanțele de securitate trebuie să nu fie inferioare funcțiilor actuale de dispersie.
- Să scoată aceleași marimi de amprente ca SHA-2, dar să fie capabilă – la nevoie – de amprente de mărimi mai mari.
- Să reziste discuțiilor privind criptanaliza, atât codul sursă cât și rezultatele analizelor trebuind să fie publice.

Cerințe NIST

În 2006 NIST a anunțat competiția pentru standardul SHA-3.
Dintre cerințele eliminatorii impuse de NIST:

- Performanțele să aceleași indiferent de implementare.
- Resursa consumată trebuie să fie minimă chiar pentru o cantitate mare de text la intrare.
- Performanțele de securitate trebuie să nu fie inferioare funcțiilor actuale de dispersie.
- Să scoată aceleași marimi de amprente ca SHA-2, dar să fie capabilă – la nevoie – de amprente de mărimi mai mari.
- Să reziste discuțiilor privind criptanaliza, atât codul sursă cât și rezultatele analizelor trebuind să fie publice.
- Nu trebuie să se bazeze pe metoda Merkle-Damgard în generarea amprentei finale.

- La competiția SHA-3 s-a înscris pentru prima fază – încheiată la sfârșitul lui 2008 – 51 candidați.

- La competiția SHA-3 s-a înscris pentru prima fază – încheiată la sfârșitul lui 2008 – 51 candidați.
- În iulie 2009 s-au calificat pentru al doilea rund 14 algoritmi: BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grostl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, Skein.

- La competiția SHA-3 s-a înscris pentru prima fază – încheiată la sfârșitul lui 2008 – 51 candidați.
- În iulie 2009 s-au calificat pentru al doilea rund 14 algoritmi: BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grostl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, Skein.
- În decembrie 2010 trec în faza a treia (finală) 5 algoritmi: BLAKE (Jean-Phillipe Aumasson), Grostl (Lars R. Knudsen), JH (Hongjun Wu), Keccak și Skein (Bruce Schneier).

- La competiția SHA-3 s-a înscris pentru prima fază – încheiată la sfârșitul lui 2008 – 51 candidați.
- În iulie 2009 s-au calificat pentru al doilea rund 14 algoritmi: BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grostl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, Skein.
- În decembrie 2010 trec în faza a treia (finală) 5 algoritmi: BLAKE (Jean-Phillipe Aumasson), Grostl (Lars R. Knudsen), JH (Hongjun Wu), Keccak și Skein (Bruce Schneier).
- În decembrie 2012 este declarat câștigător algoritmul Keccak.

Pemutări KECCAK-p

O permutare KECCAK-p este definită prin doi parametri:

1. Lungimea (fixată) a șirurilor care se permută, numită "adâncimea permutarii".
2. Numărul n de iterații (*runde*) ale unei transformări interne.

Pemutări KECCAK-p

O permutare KECCAK-p este definită prin doi parametri:

1. Lungimea (fixată) a șirurilor care se permută, numită "adâncimea permutarii".
2. Numărul n de iterații (*runde*) ale unei transformări interne.

Se notează cu $KECCAK - p[b, nr]$ o permutare KECCAK-p cu nr runde și adâncime b; ea este definită pentru orice întreg pozitiv nr și $b \in \{25, 50, 100, 200, 400, 800, 1600\}$.

Stare

O rundă *Rnd* a unei permutări KECCAK-p este formată din cinci transformări secvențiale sau pași (step mappings). Permutarea este specificată printr-o stare.

Stare

O rundă *Rnd* a unei permutări KECCAK-p este formată din cinci transformări secvențiale sau pași (step mappings). Permutarea este specificată printr-o stare.

Starea unei permutări $KECCAK-p[b,nr]$ este un tabel de b biți, la care se asociază alte două valori: $w = b/25$ și $l = log_2(b/25)$.

Stare

O rundă *Rnd* a unei permutări KECCAK-p este formată din cinci transformări secvențiale sau pași (step mappings). Permutarea este specificată printr-o stare.

Starea unei permutări $KECCAK - p[b, nr]$ este un tabel de b biți, la care se asociază alte două valori: $w = b/25$ și $l = log_2(b/25)$.

Cele trei variabile pot lua șapte valori din tabelul:

b	25	50	100	200	400	800	1600
w	1	2	4	8	16	32	64
l	0	1	2	3	4	5	6

Reprezentări

Starile sunt tablouri de biți de dimensiune $5 \times 5 \times w$.

Reprezentări

Starile sunt tablouri de biți de dimensiune $5 \times 5 \times w$.
Componentele pe axele Ox, Oy sunt numerotate 3-4-0-1-2.
În acest fel, linia de w biți corespunzătoare coordonatelor $(0, 0, z)$ este o axă centrală a paralelipipedului.

Reprezentări

Starile sunt tablouri de biți de dimensiune $5 \times 5 \times w$.
Componentele pe axele Ox, Oy sunt numerotate 3-4-0-1-2.
În acest fel, linia de w biți corespunzătoare coordonatelor $(0, 0, z)$ este o axă centrală a paralelipipedului.

- Dacă S este o secvență care reprezintă starea, atunci biții săi vor fi indexați sub forma
$$S = S[0]||S[1]|| \dots ||S[b-2]||S[b-1].$$

Reprezentări

Starile sunt tablouri de biți de dimensiune $5 \times 5 \times w$.
Componentele pe axele Ox, Oy sunt numerotate 3-4-0-1-2.
În acest fel, linia de w biți corespunzătoare coordonatelor $(0, 0, z)$ este o axă centrală a paralelipipedului.

- Dacă S este o secvență care reprezintă starea, atunci biții săi vor fi indexați sub forma
$$S = S[0]||S[1]||\dots||S[b-2]||S[b-1].$$
- Dacă $\mathbf{A}$ este un tablou $5 \times 5 \times w$ de biți care reprezintă starea, atunci indicii sunt triplete (x, y, z), $0 \leq x < 5$, $0 \leq y < 5$, $0 \leq z < w$.
  Bitul de coordonate (x, y, z) va fi notat $\mathbf{A}[x, y, z]$.

Conversii

- *Conversia șirurilor în tablouri de stare:*

$$A[x,y,z] = S[w(5y+x)+z].$$

Conversii

- *Conversia șirurilor în tablouri de stare:*
$$A[x, y, z] = S[w(5y + x) + z].$$
- *Conversia tablourilor de stare în șiruri:*
Pentru fiecare pereche $(i, j) \in [0, 5) \times [0, 5)$, definim $Lane(i, j)$ prin

$$Lane(i, j) = A[i, j, 0]||A[i, j, 1]||A[i, j, 2]||\ldots||A[i, j, w-2]||A[i, j, w-1].$$

Conversii

- *Conversia șirurilor în tablouri de stare:*
$$A[x,y,z] = S[w(5y+x)+z].$$
- *Conversia tablourilor de stare în șiruri:*
Pentru fiecare pereche $(i,j) \in [0,5) \times [0,5)$, definim $Lane(i,j)$ prin

$$Lane(i,j) = A[i,j,0]||A[i,j,1]||A[i,j,2]||\ldots||A[i,j,w-2]||A[i,j,w-1].$$

Pentru fiecare întreg $j \in [0,5)$ definim $Plane(j)$ prin

$$Plane(j) = Lane(0,j)||Lane(1,j)||Lane(2,j)||Lane(3,j)||Lane(4,j).$$

Conversii

- *Conversia șirurilor în tablouri de stare:*
$$A[x, y, z] = S[w(5y + x) + z].$$
- *Conversia tablourilor de stare în șiruri:*
Pentru fiecare pereche $(i, j) \in [0, 5) \times [0, 5)$, definim $Lane(i, j)$ prin
$$Lane(i, j) = A[i, j, 0]||A[i, j, 1]||A[i, j, 2]||\dots||A[i, j, w-2]||A[i, j, w-1].$$
Pentru fiecare întreg $j \in [0, 5)$ definim $Plane(j)$ prin
$$Plane(j) = Lane(0, j)||Lane(1, j)||Lane(2, j)||Lane(3, j)||Lane(4, j).$$
Atunci
$$S = Plane(0)||Plane(1)||Plane(2)||Plane(3)||Plane(4).$$

Transformări

Cele cinci transformări care compun o rundă din KECCAK-p[b, nr] sunt notate $\theta, \rho, \pi, \chi, \iota$.
Fiecare transformare modifică un tablou de stare A într-un tablou A' de aceleași dimensiuni.

Transformări

Cele cinci transformări care compun o rundă din KECCAK-p[b, nr] sunt notate $\theta, \rho, \pi, \chi, \iota$.
Fiecare transformare modifică un tablou de stare A într-un tablou A' de aceleași dimensiuni.

ι are un parametru suplimentar de intrare – *indexul de rundă* i_r.

Celelalte transformări nu depind de indexul de rundă.

Transformarea θ:

$\theta(A) = A'$ unde:

1. $\forall\ (x, z) \in [0, 5) \times [0, w)$
   $C[x, z] = A[x, 0, z] \otimes A[x, 1, z] \otimes A[x, 2, z] \otimes A[x, 3, z] \otimes A[x, 4, z].$

Transformarea θ:

$\theta(A) = A'$ unde:

1. $\forall\ (x, z) \in [0, 5) \times [0, w)$
   $C[x, z] = A[x, 0, z] \otimes A[x, 1, z] \otimes A[x, 2, z] \otimes A[x, 3, z] \otimes A[x, 4, z].$
2. $\forall\ (x, z) \in [0, 5) \times [0, w)$
   $D[x, z] = C[((x - 1))\ mod\ 5, z] \otimes C[(x + 1)\ mod\ 5,\ (z - 1)\ mod\ w].$

Transformarea θ:

$\theta(A) = A'$ unde:

1. $\forall\ (x,z) \in [0,5) \times [0,w)$
   $C[x,z] = A[x,0,z] \otimes A[x,1,z] \otimes A[x,2,z] \otimes A[x,3,z] \otimes A[x,4,z].$
2. $\forall\ (x,z) \in [0,5) \times [0,w)$
   $D[x,z] = C[((x-1))\ mod\ 5, z] \otimes C[(x+1)\ mod\ 5,\ (z-1)\ mod\ w].$
3. $\forall\ (x,y,z) \in [0,5) \times [0,5) \times [0,w)$
   $$A[x,y,z] = A[x,y,z] \otimes D[x,z].$$

Transformarea θ:

$\theta(A) = A'$ unde:

1. $\forall\ (x, z) \in [0, 5) \times [0, w)$
$C[x, z] = A[x, 0, z] \otimes A[x, 1, z] \otimes A[x, 2, z] \otimes A[x, 3, z] \otimes A[x, 4, z].$
2. $\forall\ (x, z) \in [0, 5) \times [0, w)$
$D[x, z] = C[((x - 1))\ mod\ 5, z] \otimes C[(x + 1)\ mod\ 5,\ (z - 1)\ mod\ w].$
3. $\forall\ (x, y, z) \in [0, 5) \times [0, 5) \times [0, w)$
$$A[x, y, z] = A[x, y, z] \otimes D[x, z].$$

θ xor-ează fiecare bit de stare cu paritățile a două coloane din stare.

Transformarea θ:

$\theta(A) = A'$ unde:

1. $\forall\ (x,z) \in [0,5) \times [0,w)$
   $C[x,z] = A[x,0,z] \otimes A[x,1,z] \otimes A[x,2,z] \otimes A[x,3,z] \otimes A[x,4,z].$
2. $\forall\ (x,z) \in [0,5) \times [0,w)$
   $D[x,z] = C[((x-1))\ mod\ 5, z] \otimes C[(x+1)\ mod\ 5,\ (z-1)\ mod\ w].$
3. $\forall\ (x,y,z) \in [0,5) \times [0,5) \times [0,w)$
   $$A[x,y,z] = A[x,y,z] \otimes D[x,z].$$

θ xor-ează fiecare bit de stare cu paritățile a două coloane din stare.

EX: Pentru bitul $A[x_0,y_0,z_0]$, a x-coordonată a unei coloane este $(x_0-1)\ mod\ 5$ cu aceeași z-coordonată z_0, în timp ce x-coordonata celeilalte coloane este $(x_0+1)\ mod\ 5$, cu z-coordonata $(z_0-1)\ mod\ w$.

Transformarea ρ:

$\rho(A) = A'$ unde:

1. $\forall z \in [0, w),\ A'[0, 0, z] = A[0, 0, z]$.
2. $(x, y) = (1, 0)$.

Transformarea ρ:

$\rho(A) = A'$ unde:

1. $\forall z \in [0, w),\ A'[0,0,z] = A[0,0,z]$.
2. $(x, y) = (1, 0)$.
3. **for** $t = 0, 23$ **do**
    1. $\forall z \in [0, w),\ \ A'[x,y,z] = A[x,y,(z(t+1)(t+2)/2)\ mod\ w]$;

Transformarea ρ:

$\rho(A) = A'$ unde:

1. $\forall z \in [0, w),\ A'[0, 0, z] = A[0, 0, z]$.
2. $(x, y) = (1, 0)$.
3. **for** $t = 0, 23$ **do**
    1. $\forall z \in [0, w),\ A'[x, y, z] = A[x, y, (z(t+1)(t+2)/2)\ mod\ w]$;
    2. $(x, y) = (y, (2x + 3y)\ mod\ 5)$.

Transformarea ρ:

$\rho(A) = A'$ unde:

1. $\forall z \in [0, w),\ A'[0, 0, z] = A[0, 0, z]$.
2. $(x, y) = (1, 0)$.
3. **for** $t = 0, 23$ **do**
   1. $\forall z \in [0, w),\ \ A'[x, y, z] = A[x, y, (z(t+1)(t+2)/2)\ mod\ w]$;
   2. $(x, y) = (y, (2x + 3y)\ mod\ 5)$.
4. **Return** A' .

Transformarea ρ:

$\rho(A) = A'$ unde:

1. $\forall z \in [0, w),\ A'[0, 0, z] = A[0, 0, z]$.
2. $(x, y) = (1, 0)$.
3. **for** $t = 0, 23$ **do**
    1. $\forall z \in [0, w),\ A'[x, y, z] = A[x, y, (z(t+1)(t+2)/2)\ mod\ w]$;
    2. $(x, y) = (y, (2x + 3y)\ mod\ 5)$.
4. **Return** A' .

ρ rotește biții fiecărei linii cu o mărime numită "offset", care depinde de coordonatele (x, y) ale liniei.

Transformarea ρ:

$\rho(A) = A'$ unde:

1. $\forall z \in [0, w),\ A'[0, 0, z] = A[0, 0, z]$.
2. $(x, y) = (1, 0)$.
3. **for** $t = 0, 23$ **do**
    1. $\forall z \in [0, w),\ \ A'[x, y, z] = A[x, y, (z(t+1)(t+2)/2)\ mod\ w]$;
    2. $(x, y) = (y, (2x + 3y)\ mod\ 5)$.
4. **Return** A' .

ρ rotește biții fiecărei linii cu o mărime numită "offset", care depinde de coordonatele (x, y) ale liniei.
Pentru fiecare bit al liniei, coordonata z este modificată adăugând offset-ul (modulo mărimea liniei).

Transformarea π:

$\pi(A) = A'$ este definită

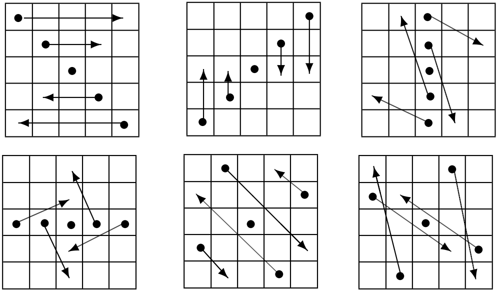
$$A'[x, y, z] = A[(x + 3y) \; mod \; 5, x, z], \;\; \forall (x, y, z) \in [0, 5) \times [0, 5) \times [0, w).$$

Transformarea π:

$\pi(A) = A'$ este definită

$$A'[x, y, z] = A[(x + 3y) \; mod \; 5, x, z], \;\; \forall (x, y, z) \in [0, 5) \times [0, 5) \times [0, w).$$

Transformarea π rearanjează pozițiile liniilor.



Transformarea χ:

$\chi(A) = A'$ este definită

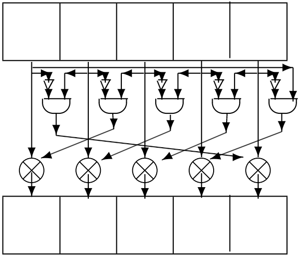
$A'[x, y, z] =$

$A[x, y, z] \otimes ((A[(x + 1) \ mod \ 5, y, z] \otimes 1) \cdot A[(x + 2) \ mod \ 5, y, z])$

Transformarea χ:

$\chi(A) = A'$ este definită
$A'[x, y, z] =$
$A[x, y, z] \otimes ((A[(x + 1) \ mod \ 5, y, z] \otimes 1) \cdot A[(x + 2) \ mod \ 5, y, z])$

Transformarea χ face un *XOR* pentru fiecare bit cu o funcție neliniară de alți doi biți aflați pe aceeași linie.



Transformarea ι:

Este parametrizată de indexul de rundă i_r.

Transformarea ι:

Este parametrizată de indexul de rundă i_r.
Algoritmul folosește un LFSR pentru o funcție $rc(t)$
Function $rc(t)$:

1. **if** $t\ mod\ 255 = 0$ **then return** 1.
2. $R = 10000000$.
3. **for** $i = 1, t\ mod\ 255$ **do**
    1. $R = 0||R$;
    2. $R[0] = R[0] \otimes R[8]$;
    3. $R[4] = R[4] \otimes R[8]$;
    4. $R[5] = R[5] \otimes R[8]$;
    5. $R[6] = R[6] \otimes R[8]$;
    6. $R = Trunc_8[R]$.
4. **Return** $R[0]$.

Algoritm $\iota(A, i_r) = A'$

1. $A'[x, y, z] = A[x, y, z], \ \ \forall (x, y, z) \in [0, 5) \times [0, 5) \times [0, w).$

Algoritm $\iota(A, i_r) = A'$

1. $A'[x, y, z] = A[x, y, z], \ \ \forall (x, y, z) \in [0, 5) \times [0, 5) \times [0, w)$.
2. $RC = 0^w$.

Algoritm $\iota(A, i_r) = A'$

1. $A'[x, y, z] = A[x, y, z], \;\; \forall (x, y, z) \in [0, 5) \times [0, 5) \times [0, w).$
2. $RC = 0^w.$
3. **for** $j = 0, l$ **do** $RC[2^j 1] = rc(j + 7i_r).$

Algoritm $\iota(A, i_r) = A'$

1. $A'[x, y, z] = A[x, y, z], \ \ \forall (x, y, z) \in [0, 5) \times [0, 5) \times [0, w)$.
2. $RC = 0^w$.
3. **for** $j = 0, l$ **do** $RC[2^j 1] = rc(j + 7i_r)$.
4. $A'[0, 0, z] = A'[0, 0, z] \otimes RC[z], \ \ \forall z \in [0, w)$.

Algoritm $\iota(A, i_r) = A'$

1. $A'[x, y, z] = A[x, y, z], \;\; \forall (x, y, z) \in [0, 5) \times [0, 5) \times [0, w)$.
2. $RC = 0^w$.
3. **for** $j = 0, l$ **do** $RC[2^j 1] = rc(j + 7i_r)$.
4. $A'[0, 0, z] = A'[0, 0, z] \otimes RC[z], \;\; \forall z \in [0, w)$.

Folosind parametrul de rundă i_r și funcția rc dată de LFSR, algoritmul determină $l + 1$ biți ale unei constante de rundă RC.

Algoritm $\iota(A, i_r) = A'$

1. $A'[x, y, z] = A[x, y, z], \ \forall (x, y, z) \in [0, 5) \times [0, 5) \times [0, w)$.
2. $RC = 0^w$.
3. **for** $j = 0, l$ **do** $RC[2^j 1] = rc(j + 7i_r)$.
4. $A'[0, 0, z] = A'[0, 0, z] \otimes RC[z], \ \forall z \in [0, w)$.

Folosind parametrul de rundă i_r și funcția rc dată de LFSR, algoritmul determină $l + 1$ biți ale unei constante de rundă RC.

Această constantă modifică prin XOR-are valorile liniei centrale din stare.
Toate celelalte 24 linii nu sunt modificate de transformarea ι.

Funcția $KECCAK - p[b, n_r]$

Fiind date un tablou de stare A și un index de rundă i_r, funcția de rundă Rnd este compunerea celor cinci transformări de mai sus:

$$Rnd(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$$

Funcția $KECCAK - p[b, n_r]$

Fiind date un tablou de stare A și un index de rundă i_r, funcția de rundă Rnd este compunerea celor cinci transformări de mai sus:

$$Rnd(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$$

Permutarea $KECCAK - p[b, n_r]$ constă din n_r iterații ale funcției Rnd aplicată unei secvențe de b biți.

Funcția $KECCAK - p[b, n_r]$

Fiind date un tablou de stare A și un index de rundă i_r, funcția de rundă Rnd este compunerea celor cinci transformări de mai sus:

$$Rnd(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$$

Permutarea $KECCAK - p[b, n_r]$ constă din n_r iterații ale funcției Rnd aplicată unei secvențe de b biți.

$\mathbf{KECCAK - p[b, n_r](S) = S'}$

1. Transformă S într-un tablou de stare A;

Funcția $KECCAK - p[b, n_r]$

Fiind date un tablou de stare A și un index de rundă i_r, funcția de rundă Rnd este compunerea celor cinci transformări de mai sus:

$$Rnd(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$$

Permutarea $KECCAK - p[b, n_r]$ constă din n_r iterații ale funcției Rnd aplicată unei secvențe de b biți.

$\mathbf{KECCAK - p[b, n_r](S) = S'}$

1. Transformă S într-un tablou de stare A;
2. **for** $i_r = 12 + 2l - n_r, 12 + 2l - 1$ **do** $A = Rnd(A, i_r)$.

Funcția $KECCAK - p[b, n_r]$

Fiind date un tablou de stare A și un index de rundă i_r, funcția de rundă Rnd este compunerea celor cinci transformări de mai sus:

$$Rnd(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$$

Permutarea $KECCAK - p[b, n_r]$ constă din n_r iterații ale funcției Rnd aplicată unei secvențe de b biți.

$\mathbf{KECCAK - p[b, n_r](S) = S'}$

1. Transformă S într-un tablou de stare A;
2. **for** $i_r = 12 + 2l - n_r, 12 + 2l - 1$ **do** $A = Rnd(A, i_r)$.
3. Transformă A într-un șir de biți S

Funcția $KECCAK - p[b, n_r]$

Fiind date un tablou de stare A și un index de rundă i_r, funcția de rundă Rnd este compunerea celor cinci transformări de mai sus:

$$Rnd(A, i_r) = \iota(\chi(\pi(\rho(\theta(A)))), i_r)$$

Permutarea $KECCAK - p[b, n_r]$ constă din n_r iterații ale funcției Rnd aplicată unei secvențe de b biți.

$\mathbf{KECCAK - p[b, n_r](S) = S'}$

1. Transformă S într-un tablou de stare A;
2. **for** $i_r = 12 + 2l - n_r, 12 + 2l - 1$ **do** $A = Rnd(A, i_r)$.
3. Transformă A într-un șir de biți S
4. **Return** S .

Construcția Sponge

Este varianta folosită de SHA-3 similară construcției Merkle-Damgard pentru *MD*5 și SHA-1.

Construcția Sponge

Este varianta folosită de SHA-3 similară construcției Merkle-Damgard pentru $MD5$ și SHA-1.
Construcția este formată din trei componente:

1. O funcție de bază f pentru șiruri de lungime fixată;

Construcția Sponge

Este varianta folosită de SHA-3 similară construcției Merkle-Damgard pentru $MD5$ și SHA-1.
Construcția este formată din trei componente:

1. O funcție de bază f pentru șiruri de lungime fixată;
2. Un parametru r numit "*rată*";

Construcția Sponge

Este varianta folosită de SHA-3 similară construcției Merkle-Damgard pentru $MD5$ și SHA-1.
Construcția este formată din trei componente:

1. O funcție de bază f pentru șiruri de lungime fixată;
2. Un parametru r numit "*rată*";
3. O regulă de completare numită "*pad*".

Construcția Sponge

Este varianta folosită de SHA-3 similară construcției Merkle-Damgard pentru $MD5$ și SHA-1.
Construcția este formată din trei componente:

1. O funcție de bază f pentru șiruri de lungime fixată;
2. Un parametru r numit "*rată*";
3. O regulă de completare numită "*pad*".

Funcția rezultată din aceste componente se numește *funcție sponge* $SPONGE[f, pad, r]$.

Construcția Sponge

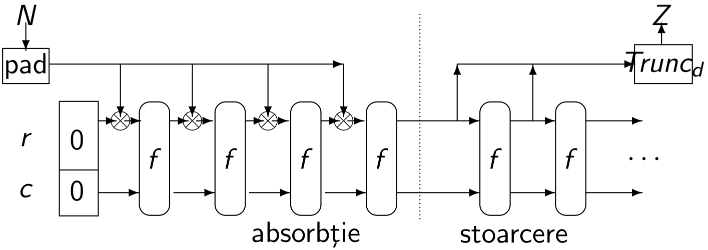
Este varianta folosită de SHA-3 similară construcției Merkle-Damgard pentru $MD5$ și SHA-1.
Construcția este formată din trei componente:

1. O funcție de bază f pentru șiruri de lungime fixată;
2. Un parametru r numit "*rată*";
3. O regulă de completare numită "*pad*".

Funcția rezultată din aceste componente se numește *funcție sponge* $SPONGE[f, pad, r]$.
Are la intrare doi parametri: un șir de biți N (de lungime arbitrară) și lungimea d a secvenței rezultate: $SPONGE[f, pad, r](N, d)$.

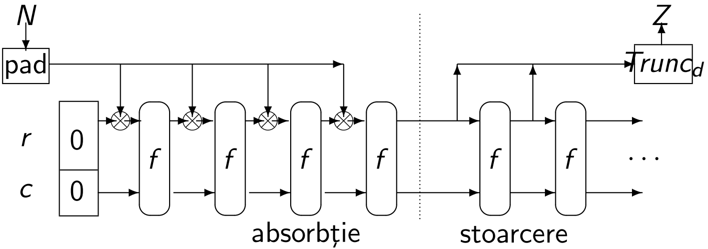
Reprezentarea funcției SPONGE

Funcția $Z = SPONGE[f, pad, r](N, d)$ este ilustrată de schema:



Reprezentarea funcției SPONGE

Funcția $Z = SPONGE[f, pad, r](N, d)$ este ilustrată de schema:



Funcția f transformă un șir de b biți în altă secvență de aceeași lungime.

Funcția SHA-3 este bazată pe instanțieri ale construcției *SPONGE* în care funcția f este o permutare (deci inversabilă).

Funcția SHA-3 este bazată pe instanțieri ale construcției *SPONGE* în care funcția f este o permutare (deci inversabilă).
Rata r ($r < b$) este un întreg pozitiv. "*Capacitatea*" – notată cu c este definită prin $c = b - r$..

Funcția SHA-3 este bazată pe instanțieri ale construcției *SPONGE* în care funcția f este o permutare (deci inversabilă).
Rata r ($r < b$) este un întreg pozitiv. "*Capacitatea*" – notată cu c este definită prin $c = b - r$..

Regula de completare *pad* este o funcție definită de condiția:
$\forall x > 0, m \geq 0,\ pad(x, m)$ este un șir binar cu proprietatea
$m + |pad(x, m)|$ se divide cu x.

Funcția SHA-3 este bazată pe instanțieri ale construcției *SPONGE* în care funcția f este o permutare (deci inversabilă).
Rata r ($r < b$) este un întreg pozitiv. "*Capacitatea*" – notată cu c este definită prin $c = b - r$..

Regula de completare *pad* este o funcție definită de condiția:
$\forall x > 0, m \geq 0,\ pad(x, m)$ este un șir binar cu proprietatea $m + |pad(x, m)|$ se divide cu x.

În cadrul construcției *SPONGE* $x = r$ și $m = |N|$, deci șirul de intrare completat prin *pad* poate fi separat în secvențe de câte r biți.

Funcția SHA-3 este bazată pe instanțieri ale construcției *SPONGE* în care funcția f este o permutare (deci inversabilă).
Rata r ($r < b$) este un întreg pozitiv. "*Capacitatea*" – notată cu c este definită prin $c = b - r$..

Regula de completare *pad* este o funcție definită de condiția:
$\forall x > 0, m \geq 0,\ pad(x, m)$ este un șir binar cu proprietatea $m + |pad(x, m)|$ se divide cu x.

În cadrul construcției *SPONGE* $x = r$ și $m = |N|$, deci șirul de intrare completat prin *pad* poate fi separat în secvențe de câte r biți.
Acest lucru se realizează cu algoritmul $SPONGE[f, pad, r](N, d)$, care primește la intrare șirul N și o valoare întreagă nenegativă d.
Ieșirea este un șir Z de lungime d.

$SPONGE[f, pad, r](N, d)$:

1. $P = N || pad(r, |N|)$;
2. $n = |P|/r$;
3. $c = b - r$;

$SPONGE[f, pad, r](N, d)$:

1. $P = N||pad(r, |N|)$;
2. $n = |P|/r$;
3. $c = b - r$;
4. Se construiește secvența (unică) $P_0, P_1, \ldots, P_{n-1}$ de șiruri de lungime r astfel ca $P = P_0|| \ldots ||P_{n-1}$;

$SPONGE[f, pad, r](N, d)$:

1. $P = N||pad(r, |N|)$;
2. $n = |P|/r$;
3. $c = b - r$;
4. Se construiește secvența (unică) $P_0, P_1, \ldots, P_{n-1}$ de șiruri de lungime r astfel ca $P = P_0|| \ldots ||P_{n-1}$;
5. $S = 0^b$;
6. **for** $i = 0, n-1$ **do** $S = f(S \otimes (P_i||0^c))$;

$SPONGE[f, pad, r](N, d)$:

1. $P = N||pad(r, |N|)$;
2. $n = |P|/r$;
3. $c = b - r$;
4. Se construiește secvența (unică) $P_0, P_1, \ldots, P_{n-1}$ de șiruri de lungime r astfel ca $P = P_0|| \ldots ||P_{n-1}$;
5. $S = 0^b$;
6. **for** $i = 0, n-1$ **do** $S = f(S \otimes (P_i||0^c))$;
7. $Z = \lambda$ (șirul vid);
8. $Z = Z||Trunc_r(S)$;
9. **if** $d \leq |Z|$ **then return** $Trunc_d(Z)$;

$SPONGE[f, pad, r](N, d)$:

1. $P = N || pad(r, |N|)$;
2. $n = |P|/r$;
3. $c = b - r$;
4. Se construiește secvența (unică) $P_0, P_1, \ldots, P_{n-1}$ de șiruri de lungime r astfel ca $P = P_0 || \ldots || P_{n-1}$;
5. $S = 0^b$;
6. **for** $i = 0, n-1$ **do** $S = f(S \otimes (P_i || 0^c))$;
7. $Z = \lambda$ (șirul vid);
8. $Z = Z || Trunc_r(S)$;
9. **if** $d \leq |Z|$ **then return** $Trunc_d(Z)$;
10. $S = f(S)$; **goto** 8.

Discuție parametri

- Intrarea d determină numărul de biți pe care îi scoate Algoritmul *SPONGE*, dar nu afectează valorile lor.

Discuție parametri

- Intrarea d determină numărul de biți pe care îi scoate Algoritmul *SPONGE*, dar nu afectează valorile lor.
  Teoretic, ieșirea poate fi privită ca o secvență infinită de biți al cărei calcul este oprit în mmentul în care se obține numărul de biți solicitat.

Discuție parametri

- Intrarea d determină numărul de biți pe care îi scoate Algoritmul *SPONGE*, dar nu afectează valorile lor.
  Teoretic, ieșirea poate fi privită ca o secvență infinită de biți al cărei calcul este oprit în mmentul în care se obține numărul de biți solicitat.
- Funcția de completare *pad* scoate un șir $P = 10^*1$, astfel ca $m + |P|$ se divide cu x:

  Algorithm $pad(x, m)$**:**

  1. $j = (-m - 2) \ mod \ x$;
  2. **Return** $P = 1||0^j||1$.

KECCAK

KECCAK este familia de funcții SPONGE cu permutarea $KECCAK - p[b, 12 + 2l]$ ca funcție de bază și cu *pad* ca regulă de completare.

KECCAK

KECCAK este familia de funcții SPONGE cu permutarea $KECCAK - p[b, 12 + 2l]$ ca funcție de bază și cu *pad* ca regulă de completare.

Ea este parameterizată de orice alegere a valorilor r (rata) și c (capacitatea) astfel ca $r + c \in \{25, 50, 100, 200, 400, 800, 1600\}$ (valorile solicitate pentru b).

KECCAK

KECCAK este familia de funcții SPONGE cu permutarea $KECCAK - p[b, 12 + 2l]$ ca funcție de bază și cu *pad* ca regulă de completare.

Ea este parameterizată de orice alegere a valorilor r (rata) și c (capacitatea) astfel ca $r + c \in \{25, 50, 100, 200, 400, 800, 1600\}$ (valorile solicitate pentru b).

Când se adaugă restricția $b = 1600$, familia *KECCAK* se notează $KECCAK[c]$ și valoarea lui r se deduce direct din c.

KECCAK

KECCAK este familia de funcții SPONGE cu permutarea $KECCAK - p[b, 12 + 2l]$ ca funcție de bază și cu *pad* ca regulă de completare.

Ea este parameterizată de orice alegere a valorilor r (rata) și c (capacitatea) astfel ca $r + c \in \{25, 50, 100, 200, 400, 800, 1600\}$ (valorile solicitate pentru b).

Când se adaugă restricția $b = 1600$, familia *KECCAK* se notează $KECCAK[c]$ și valoarea lui r se deduce direct din c.

În particular,

$$KECCAK[c] = SPONGE[KECCAK - p[1600, 24], pad, 1600 - c].$$

KECCAK

KECCAK este familia de funcții SPONGE cu permutarea $KECCAK - p[b, 12 + 2l]$ ca funcție de bază și cu *pad* ca regulă de completare.

Ea este parameterizată de orice alegere a valorilor r (rata) și c (capacitatea) astfel ca $r + c \in \{25, 50, 100, 200, 400, 800, 1600\}$ (valorile solicitate pentru b).

Când se adaugă restricția $b = 1600$, familia *KECCAK* se notează $KECCAK[c]$ și valoarea lui r se deduce direct din c.

În particular,

$$KECCAK[c] = SPONGE[KECCAK - p[1600, 24], pad, 1600 - c].$$

Deci, fiind dat un șir de biți N și o lungime de ieșire d,

$$KECCAK[c](N, d) = SPONGE[KECCAK - p[1600, 24], pad, 1600 - c](N, d).$$

Specificațiile funcției de dispersie SHA-3

Pentru un mesaj M se definesc patru funcții de dispersie SHA-3:

$$SHA3-224(M) = KECCAK[448](M||01,224);$$
$$SHA3-256(M) = KECCAK[512](M||01,256);$$
$$SHA3-384(M) = KECCAK[768](M||01,384);$$
$$SHA3-512(M) = KECCAK[1024](M||01,512).$$

Specificațiile funcției de dispersie SHA-3

Pentru un mesaj M se definesc patru funcții de dispersie SHA-3:

$SHA3-224(M) = KECCAK[448](M||01, 224);$
$SHA3-256(M) = KECCAK[512](M||01, 256);$
$SHA3-384(M) = KECCAK[768](M||01, 384);$
$SHA3-512(M) = KECCAK[1024](M||01, 512).$
În fiecare caz lungimea amprentei este dublată (adică $c = 2d$), iar șirul de intrare în $KECCAK[c]$ este $N = M||01$.

Specificațiile funcției de dispersie SHA-3

Pentru un mesaj M se definesc patru funcții de dispersie SHA-3:

$SHA3-224(M) = KECCAK[448](M||01, 224);$
$SHA3-256(M) = KECCAK[512](M||01, 256);$
$SHA3-384(M) = KECCAK[768](M||01, 384);$
$SHA3-512(M) = KECCAK[1024](M||01, 512).$
În fiecare caz lungimea amprentei este dublată (adică $c = 2d$), iar șirul de intrare în $KECCAK[c]$ este $N = M||01$.
Sufixul 01 separă diverse domenii de utilizare ale funcției de dispersie (actuale sau viitoare); de exemplu funcțiile $SHAKE$:
$SHAKE128(M, d) = KECCAK[256](M||1111, d),$
$SHAKE256(M, d) = KECCAK[512](M||1111, d).$

Securitate

Funcție	Mărime Ieșire	Coliziune	Preimage	2nd Preimage
SHA-1	160	< 80	160	$160 - L(M)$
SHA-224	224	112	224	$min(224, 256 - L(M))$
SHA-256	256	128	256	$256 - L(M)$
SHA-384	384	192	384	384
SHA-512	512	256	512	$512 - \lvert M \rvert$
SHA3-224	224	112	224	224
SHA3-256	256	128	256	256
SHA3-384	384	192	384	384
SHA3-512	512	256	512	512
SHAKE 128	d	$min(d/2, 128)$	$\geq min(d, 128)$	$min(d, 128)$
SHAKE 256	d	$min(d/2, 256)$	$\geq min(d, 256)$	$min(d, 256)$

Securitate

Funcție	Mărime Ieșire	Coliziune	Preimage	2nd Preimage
SHA-1	160	< 80	160	$160 - L(M)$
SHA-224	224	112	224	$min(224, 256 - L(M))$
SHA-256	256	128	256	$256 - L(M)$
SHA-384	384	192	384	384
SHA-512	512	256	512	$512 - \|M\|$
SHA3-224	224	112	224	224
SHA3-256	256	128	256	256
SHA3-384	384	192	384	384
SHA3-512	512	256	512	512
SHAKE 128	d	$min(d/2, 128)$	$\geq min(d, 128)$	$min(d, 128)$
SHAKE 256	d	$min(d/2, 256)$	$\geq min(d, 256)$	$min(d, 256)$

$L(M) = \lceil log_2(|M|/B) \rceil$, unde B este lungimea blocului de lucru în biți ($B = 512$ pentru SHA-1, SHA-224, SHA-256 și $B = 1024$ pentru SHA-512).

The END