Capitolul 8

SHA-3

8.1 Lansarea proiectului $SHA-3$

8.1.1 Limitele funcţiilor de dispersie $SHA-1$ şi $SHA-2$

O problemă cu cele două funcţii de dispersie constă în faptul că amândouă au la bază acelaşi concept pentru procesare textelor: Merkle-Damgard. Deci orice atac reuşit asupra lui $SHA-1$ devine o falie de securitate pentru $SHA-2$.

Observaţia 8.1. *Un atac forţă brută asupra funcţiei de dispersie $SHA-1$ necesită cel puţin 280 cicluri pentru a găsi o coliziune pe o rundă completă $SHA-1$. Dar în february 2005 un colectiv condues de Xiaoyun Wang a dezvoltat un atac diferenţial care a spart o rundă în numai 269 cicluri. Acelaşi atac a fost reluat şi inbunătăţit de Martin Cochran în august 2008.*

În 2012, Mark Stevens a folosit mai multe servere cloud pentru a lansa un atac diferenţial asupra lui $SHA-1$, obţinând o aproape o coliziune după 258.5 *cicluri. Din estimarile sale, un atac puţin modificat poate genera coliziuni complete în doar* 261 *cicluri.*

$SHA-2$ este considertă încă o funcţie de dispersie sigură: singurele atacuri reuşite care se cunosc sunt asupra lui $SHA-2$ cu un număr redus de runde: 46-runde (varianta de 512 biţi) şi 41 runde (varianta de 256 biţi). Au fost necessare $2253,6$ cicluri pentru a sparge $SHA-2$ pe 256 biţi şi $2511,5$ cicluri pentru varianta de 512 biţi.

Deşi nu sunt anunţate atacuri reuşite asupra funcţiei $SHA-2$ cu număr complet de runde, este sigur că în mediul privat au fost dezvoltate mecanisme de atac care nu sunt făcute publice. Din acest motiv NIST a lansat competiţia $SHA-3$.

8.1.2 Restricţii NIST

În 2006 NIST a anunţat competiţia pentru o nouă fuincţie de dispersie standard, pe care a numit-o $SHA-3$. Scopul nu era înlocuirea lui $SHA-2$ (încă sigur) ci doar o alternativă la aceasta, generată de atacturile reuşite care au generat coliziunile lui $MD5$ şi $SHA-0$ precum şi atacurile teoretice făute publice asupra lui $SHA-1$.

Dintre cerinţele eliminatorii impuse de NIST (care au traiat o bună parte din candidaţi).

- Performanţele funcţiei de dispersie trebuie să aceleaşi indiferent de implementare,
- Resursa consumartă trebuie să fie minimă chiar dacă cantitatea de text de la intrare este mare.
- Performanţele de securitate treuie să nu fie inferioare funcţiilor actuale de dispersie. Trebuie să reziste la atacurile cunoscuite, păstraând un factor ridicat de securitate.
- Să scoată aceleaşi marimi de amprente ca $SHA-2$ $(224, 256, 384$ şi 512 biţi), dar să fie capabilă – la nevoie – de amprente de mărimi mai mari.
- Să reziste discuţiilor publice privind criptanaliza, atât codul sursă cât şi rezultatele analizelor trebuind să fie publice (eci orice entitate interesată să le poată accesa, analiza şi comenta). Orice falie de securitate găsită în timpul analizei trebuie abordată prin acoperirea ei sau reproiectare.
- Trebuie să fie scris în diverse coduri. Nu trebuie să se bazeze pe metoda Merkle-Damgard în generarea amprentei finale.

La competiţia $SHA-3$ s-a înscris pentru prima fază – încheiată la sfârşitul lui 2008 – 51 candidaţi. În iulie 2009 au fost calificate pentru al doilea rund 14 algoritmi, iar numai 5 au trecut în faza a treia (finală) începută in decembrie 2010. Este declarat câştigător în decembrie 2012 algoritmul Keccak.

8.2 Prezentare $SHA-3$

8.2.1 Noţiuni preliminare

Vom folosi notaţiile din standardul FIPS PUB 202.

1. **Pemutări** $KECCAK-p$:
   O permutare $KECCAK-p$ este definită prin doi parametri:

   (a) Lungimea (fixată) a şirurilor care se permută, numită "*adâncimea permutarii*".

   (b) Numărul n de iteraţii (*runde*) ale unei transformări interne.

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

   O rundă a unei permutări $KECCAK-p$ (notată *Rnd*) este formată din cinci transformări secvenţiale sau *paşi* ("step mappings" în original). Permutarea este specificată printr-o "*stare*".

2. **Stare**

   Starea unei permutări $KECCAK - p[b, nr]$ este un tabel de b biţi, la care se asociază alte două valori: $w = b/25$ şi $l = log_2(b/25)$. Cele trei variabile pot lua şapte valori, listate în tabelul de mai jos:

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

   Starile vor fi reprezentate prin tablouri de biţi de dimensiune $5 \times 5 \times w$.

   - Dacă S este o secvenţă care reprezintă starea, atunci biţii săi vor fi indexaţi sub forma
   $$S = S[0]||S[1]||\ldots||S[b-2]||S[b-1].$$
   - Dacă **A** este un tablou $5 \times 5 \times w$ de biţi care reprezintă starea, atunci indicii sunt triplete (x, y, z), $0 \le x < 5$, $0 \le y < 5$, $0 \le z < w$ de coordonate. Bitul de coordonate (x, y, z) va fi notat $\mathbf{A}[x, y, z]$.

   Conversii:

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

   Grafic, un tablou de stare este un paralelipiped de dimensiune $5 \times 5 \times w$. Cele 5 componente pe axele Ox, Oy sunt numerotate în ordine: $3 - 4 - 0 - 1 - 2$ (axa Ox, de la stânga spre dreapta, respectiv axa Oy de jos în sus).

   În acest fel, linia de w biţi corespunzătoare coordonatelor $(0, 0, z)$ este o axă centrală a paralelipipedului.

3. **Transformări**:

   Cele cinci transformări (paşi) care formează 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 aceleaşi dimensiuni.

Pasul 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:

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

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

  (c) $\forall\ (x, y, z) \in [0, 5) \times [0, 5) \times [0, w)$, fie
  $$A[x, y, z] = A[x, y, z] \otimes D[x, z].$$

  Efectul transformării θ este xor-area fiecărui bit de stare cu parităţile a două coloane din stare.

  În particular, pentru bitul $A[x_0, y_0, z_0]$, a x-coordonată a unei coloane este $(x_0 - 1)\ mod\ 5$ cu acceaşi 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:

  (a) $\forall z \in [0, w),\ A'[0, 0, z] = A[0, 0, z]$.

  (b) $(x, y) = (1, 0)$.

  (c) **for** $t = 0, 23$ **do**

    i. $\forall z \in [0, w),\quad A'[x, y, z] = A[x, y, (z(t + 1)(t + 2)/2)\ mod\ w]$;

    ii. $(x, y) = (y, (2x + 3y)\ mod\ 5)$.

  (d) **Return** A' .

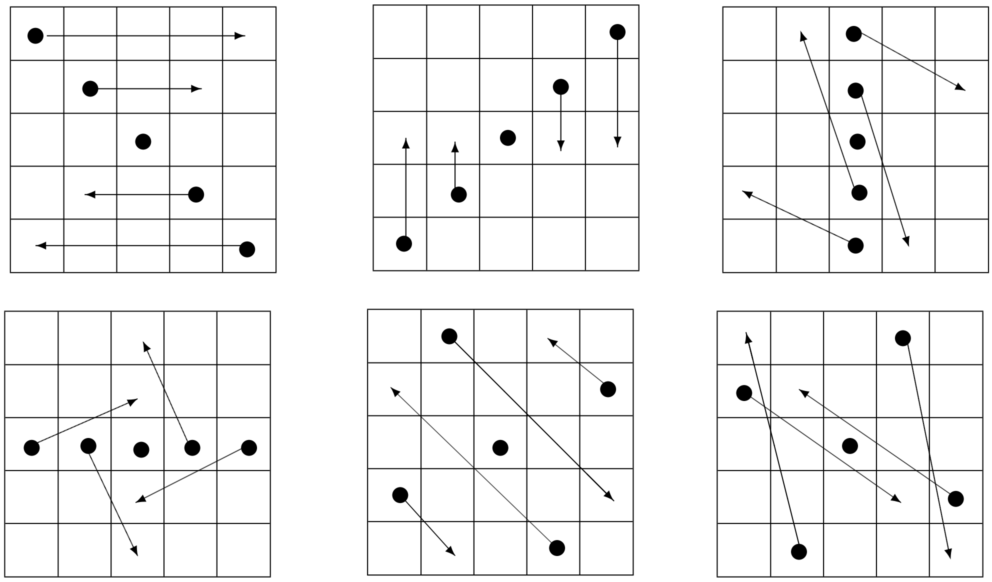
  Transformarea ρ roteşte biţii fiecărei linii[1] cu o mărime numită "*offset*", care depinde de coordonatele (x, y) ale liniei. Pentru fiecare bit al liniei, coordonata z este modificată adăugând offset-ul (modulo mărimea liniei).

  Exemplul 8.1. *Tabelul următor listează offseturile fiecărei linii, calculate la Pasul* (c)*i.*

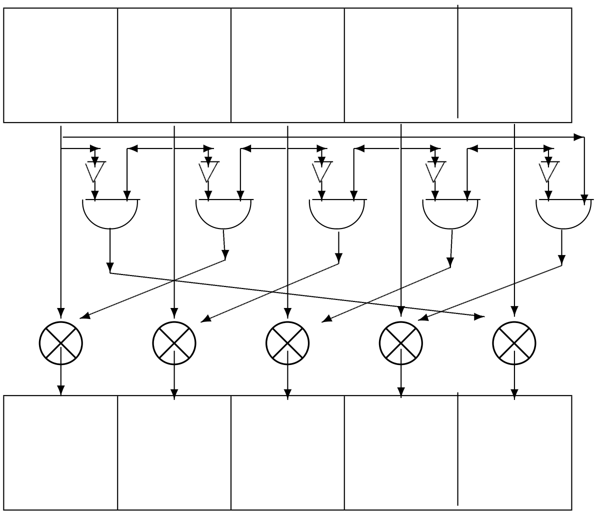
	$x = 3$	$x = 4$	$x = 0$	$x = 1$	$x = 2$
$y = 2$	153	231	3	10	171
$y = 1$	55	276	36	300	6
$y = 0$	28	91	0	1	190
$y = 4$	120	78	210	66	253
$y = 3$	21	136	105	45	15

[1] O linie în starea A este un şir de biţi $(x_0, y_0, z),\ z = 0, 1, \ldots, w$, paralel cu axa Oz

- **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 poziţiile liniilor. Mai jos este reprezentată transformarea π pentru biţii unei "felii" a paralelipipedului.



- **Transformarea χ:** $\chi(A) = A'$ este definită
$A'[x,y,z] = A[x,y,z] \otimes ((A[(x+1) \; mod \; 5, y, z] \otimes 1) \cdot A[(x+2) \; mod \; 5, y, z])$,
$\forall (x,y,z) \in [0,5) \times [0,5) \times [0,w)$.
De remarcat că multiplicarea $\cdot$ este de fapt operatorul logic AND, iar operaţia XOR cu 1 este complementarea.
Transformarea χ face un XOR pentru fiecare bit cu o funcţie neliniară de alţi doi biţi aflaţi pe aceeaşi linie, ca în figură:



- **Transformarea ι:**
  Este parametrizată de indexul de rundă i_r dat de Pasul 2 al Algoritmului pentru $KECCAK - p[b, nr]$ (secţiunea următoare)
  Algoritmul de calcul pentru ι foloseşte un LFSR entru a calcula o funcţie $rc(t)$
  Function $rc(t)$:
  (a) **if** $t\ mod\ 255 = 0$ **then return** 1.
  (b) $R = 10000000$.
  (c) **for** $i = 1, t\ mod\ 255$ **do**
    i. $R = 0||R$;
    ii. $R[0] = R[0] \otimes R[8]$;
    iii. $R[4] = R[4] \otimes R[8]$;
    iv. $R[5] = R[5] \otimes R[8]$;
    v. $R[6] = R[6] \otimes R[8]$;
    vi. $R = Trunc_8[R]$.
  (d) **Return** $R[0]$.
  Algoritm $\iota(A, i_r) = A'$
  (a) $A'[x, y, z] = A[x, y, z], \quad \forall(x, y, z) \in [0, 5) \times [0, 5) \times [0, w)$.
  (b) $RC = 0^w$.
  (c) **for** $j = 0, l$ **do** $RC[2^j 1] = rc(j + 7i_r)$.
  (d) $A'[0, 0, z] = A'[0, 0, z] \otimes RC[z], \quad \forall z \in [0, w)$.
  Folosind parametrul de rundă i_r şi funcţia rc dată de LFSR, algoritmul determină $l + 1$ biţi ale unei constante de rundă, notată RC. Această constantă modifică prin XOR-are valorile liniei centrale din stare.
  Toate celelalte 24 linii nu sunt modificate de transformarea ι.

8.2.2 Funcţia $KECCAK - p[b, n_r]$

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

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

Permutarea $KECCAK - p[b, n_r]$ constă din n_r iteraţii ale funcţiei Rnd aplicată unei secvenţe de b biţi;

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

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 şir de biţi S

4. **Return** S .

Observaţia 8.2. *Familia de permutări KECCAK-f definită în documentul original este un caz particular al familiei KECCAK-p:*

KECCAK-f [b] = KECCAK-p[b, 12+2l].

Deci permutarea KECCAK-p[1600, 24] care stă la baza celor şase funcţii $SHA-3$ *este echivalentă cu KECCAK-f [1600].*

8.3 Construcţia Sponge

Construcţia Sponge este varianta folosită de $SHA-3$ similar cosntrucţiei Merkle-Damgard în $MD5$ şi $SHA-1$. Termenul "*sponge*" (burete) sugerează ideea de a "absoarbe" date şi apoi a le "stoarce" într-o formă nouă, de lungime fixată.

Construcţia este formată din trei componente:

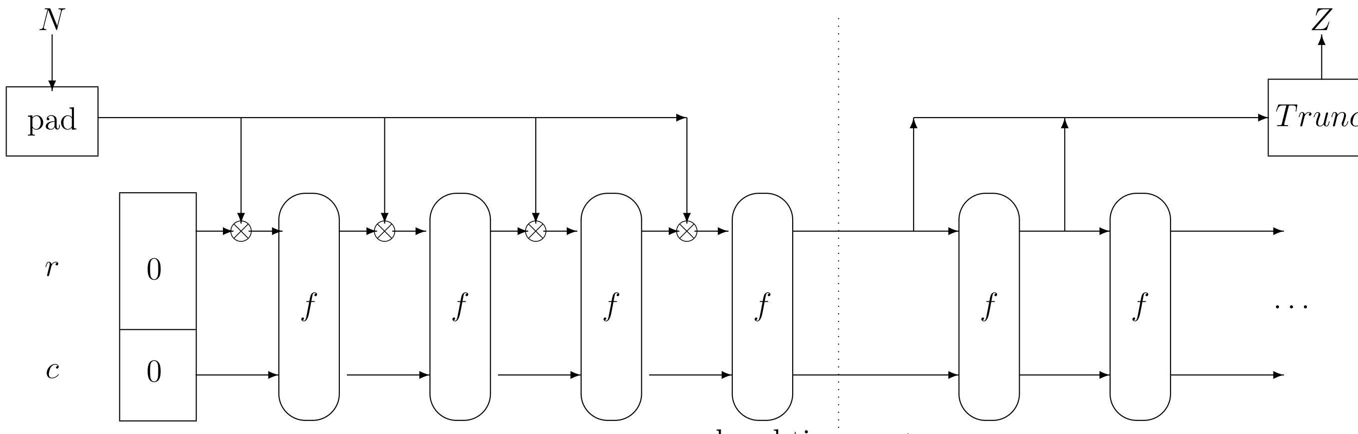
1. O funcţie de bază f pentru şiruri de lungime fixată;

2. Un parametru r numit "*rată*";

3. O regulă de completare numită "*pad*".

Funcţia rezultată din aceste componente se numeşte *funcţie sponge* $SPONGE[f, pad, r]$. Ea are la intrare doi parametri: un şir de biţi N de lungime arbitrară , şi lungimea d a secvenţei rezultate: $SPONGE[f, pad, r](N, d)$.

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



Funcţia f transformă un şir de b biţi în altă secvenţă de aceeaşi lungime; similar notaţiei de la stare, w este "*adâncimea*" lui f. Funcţia $SHA-3$ definită mai jos este bazată pe instanţieri ale constructiei $SPONGE$ în care funcţia f este o permutare (deci inversabilă).

Rata r este un întreg pozitiv cu restricţia $r < b$. "*Capacitatea*" – notată cu c este definită prin $c = b - r$..

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

În cadrul construcţiei *SPONGE* $x = r$ şi $m = |N|$, deci şirul de intrare completat prin *pad* poate fi separat în secvenţe de câte r biţi. Avest lucru se realizează cu algoritmul $SPONGE[f, pad, r](N, d)$, care primeşte la intrare şirul N şi o valoare întreagă nenegativă d. Ieşirea este un şir Z de lungime d.

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

1. $P = N||pad(r, |N|)$;
2. $n = |P|/r$;
3. $c = b - r$;
4. Se construieşte secvenţa (unică) $P_0, P_1, \ldots, P_{n-1}$ de şiruri 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. $Z = \lambda$ (şirul vid);
8. $Z = Z||Trunc_r(S)$;
9. **if** $d \leq |Z|$ **then return** $Trunc_d(Z)$;
10. $S = f(S)$; **goto** 8.

Observaţia 8.3. *Intrarea d determină numărul de biţi pe care îi scoate Algoritmul SPONGE , dar nu afectează valorile lor. Teoretic, ieşirea poate fi privită ca o secvenţă infinită de biţi al cărei calcul este oprit în mmentul în care se obţine numărul de biţi solicitat.*

Funcţia de completare *pad* – care scoate un şir $P = 10^*1$, astfel ca $m + |P|$ se divide cu x – se defineşte astfel:

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

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

8.4 KECCAK

$KECCAK$ este familia de funcţii SPONGE (definite prima oară în [1]) cu permutarea $KECCAK - p[b, 12 + 2l]$ ca funcţie de bază şi cu pad ca regulă de completare. Ea este parameterizată de orice alegere a valorilor r (rata) şi c (capacitatea) astfel ca $r + c \in \{25, 50, 100, 200, 400, 800, 1600\}$ (valorile solicitate pentru b).

Când se adaugă restricţia $b = 1600$, familia $KECCAK$ se notează $KECCAK[c]$ şi vaoarea lui r se deduce direct din c. În particular,

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

Deci, fiind dat un şir de biţi N şi o lungime de ieşire d,

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

8.5 Specificaţiile funcţiei de dispersie $SHA - 3$

Fiind dat un mesaj M se definesc patru funcţii de dispersie $SHA - 3$:

$$\begin{aligned}
SHA3 - 224(M) &= KECCAK[448](M||01, 224);\\
SHA3 - 256(M) &= KECCAK[512](M||01, 256);\\
SHA3 - 384(M) &= KECCAK[768](M||01, 384);\\
SHA3 - 512(M) &= KECCAK[1024](M||01, 512).
\end{aligned}$$

În fiecare caz lungimea amprentei este dublată (adică $c = 2d$), iar şirul de intrare în $KECCAK[c]$ este $N = M||01$.

Sufixul 01 este introdus pentru a separa diversele domenii de utilizare ale funcţiei de dispersie (actuale sau viitoare), cum ar fi de exemplu funcţiile $SHAKE$:

$$\begin{aligned}
SHAKE128(M, d) &= KECCAK[256](M||1111, d),\\
SHAKE256(M, d) &= KECCAK[512](M||1111, d).
\end{aligned}$$

8.6 Securitate

După publicarea standardului $SHA-3$ estimările de securitate stabilite până acum (2015) sunt:

Funcţie	**Mărime Ieşire**	**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	$min(d/2, 256)$	$\geq min(d, 256)$	$min(d, 256)$

La atacurile contra a doua primagine pentru un mesaj M s-a folosit funcţia $L(M) = \lceil log_2(|M|/B)\rceil$, unde B este lungimea blocului de lucru în biţi ($B = 512$ pentru SHA-1, SHA-224, SHA-256 şi $B = 1024$ pentru SHA-512).

8.7 Concluzie

Per total, Keccak este o alegere bună pentru atandardul $SHA - 3$. Este rapid, face o distribuţie uniformă a biţilor, rezistă la atacurile obişnuite de coliziune. Dar suntem încă la început.

Bibliografie

[1] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche – *The KECCAK reference*, Version 3.0, January 2011, http://keccak.noekeon.org/Keccak-reference-3.0.pdf.

[2] SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. FIPS PUB 202

[3] http://www.drdobbs.com/security/keccak-the-new-sha-3-encryption-standard/240154037

[4] http://keccak.noekeon.org/specs_summary.html

[5] http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=919061