Útoky na kryptograficky silné hashovacie funkcie

(bakalárska práca)

Michal Kevický

V Bratislave dňa: 13. júna 2008

Univerzita Komenského v Bratislava
Fakulta matematiky, fyziky a informatiky
Katedra informatiky

Útoky na kryptograficky silné hashovacie funkcie

(bakalárska práca)

Michal Kevický

Odbor: 9.2.1 Informatika
Vedúci bakalárskej práce
Mgr. Peter Košinár

V Bratislave dňa
13. júna 2008

Zadanie

Názov práce: Útoky na kryptograficky silné hashovacie funkcie

Cieľ práce: Vypracovať prehľad útokov na hashovacie funkcie (+ simulácie na oslabených / zjednodušených modeloch)

Zadávateľ: doc. RNDr. Daniel Olejár, PhD.

Školiteľ: Mgr. Peter Košinár

Autor: Michal Kevický

Čestné prehlásnie

Týmto čestne prehlasujem, že túto prácu som napísal sám s použitím uvedenej literatúry.

.....

Poďakovanie

Moja opätovná veľká vďaka za prejavenú ochotu, neuveriteľnú trpezlivosť a v neposlednom rade množstvo cenných rád patrí v prvom rade môjmu školitelovi a priateľovi *Mgr. Petrovi Košinárovi*. Za podporu a zhovievavosť rodine, blízkym, priateľom, kolektívu korešpondencného seminára z programovania, a všetkým pedagógom, ktorí sa s mojou maličkosťou počas doterajšieho pobytu na fakulte stretli a venovali mi svoj čas a vedomosti. Samozrejme aj všetkým ďalším, ktorých nie je v mojich silách ani rozsahových možnostiach tejto práce menovať.

Abstrakt

KEVICKÝ, Michal: Útoky na kryptograficky silné hashovacie funkcie (bakalárska práca)

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky
Katedra Informatiky

Študijný obor: 9.2.1 Informatika

Školiteľ: Mgr. Peter Košinár

V Bratislave dňa: 13. júna 2008
Rozsah: (40 strán)

Práca si kladie za cieľ poskytnúť problematiky neznalému čitateľovi ucelený prehľad používaných kryptograficky silných hashovacích funkcií súčasnosti. Zrozumiteľnou cestou ilustrovať ich anatómiu, poukázať na možnosti ich využitia a rovnako aj ich slabiny. Ďalej oboznámiť s históriou doteraz známych útokov na ne, aktuálnu úroveň bezpečnosti, ktorú poskytujú. Poprípade ilustrovať ďalšie možné smerovanie a trendy v oblasti prekonávania bezpečnosti existujúcich funkcií a tvorby funkcií nových.

Kľúčové slová: informačná bezpečnosť, kompresné funkcie, kryptograficky silné hashovacie funkcie, kolízie, útoky

Predhovor

Hashovacie funkcie sú ďalšou zo série technických vymožeností, ktoré nám viac či menej zjednodušujú život a stretávame sa s nimi v podstate každodenne. Pričom okom bežného človeka a v nemálo prípadoch aj okom informatikov zostanú nepovšimnuté. Dlho som neváhal, keď mi pán docent Olejár navrhol tému útokov na ne. Či už z hľadiska osobného alebo "vedeckého záujmu" som sa rozhodol túto tému spracovať. Nasledovali týždne štúdia a výsledok, verím že podaný v zrozumiteľnej forme, je Vám vo forme prehľadovej práce na danú tému práve k dispozícií.

Prajem Vám veľa trpezlivosti a príjemné čítanie.

Obsah

Kapitola 1

Úvod

Doba informačných technológií so sebou priniesla snahu automatizovať a digitalizovať všetky touto formou realizovateľné procesy spoločnosti a v súvislosti s digitalizáciou dát "stvorila" sadu v histórií dosiaľ neriešených problémov.

Chybovosť techniky priniesla potrebu funkcií slúžiacich na *overovanie integrity* prenášaných či uchovávaných dát v reálnom čase. Ideálne nenáročných na výpočtový výkon a ľahko i lacno implementovateľných aj na úrovni hardware-u.

Sofistikovanejšie dátové štruktúry miestami vyžadujú *pseudonáhodnú funkciu*, ktorá má často nemalý vplyv na ich výkon, prípadne rovnomernosť jeho rozloženia.

Na riešenie týchto problémov dnes využívame aj tzv. **hashovacie funkcie**[1], ktoré sa javia ako efektívna metóda na konštrukciu malých "otlačkov" z relatívne veľkých objemov dát.

Neskorší dopyt po funkciách podobných vlastností použiteľných v oblastiach citlivých na zmenu dát, pri overovaní "pravosti správ" a pod. bol podnetom pre vznik triedy tzv. **kryptograficky silných hashovacích funkcií *(KSHF)***.

Definično-terminologický úvod do problematiky poskytne *Kapitola 2*. *Kapitola 3* oboznámi s bežne používanými konštrukciami a s väčšinou populárnych kryptograficky silných hashovacích funkcií dnešnej doby.

[1] Napriek faktu, že v našich končinách je zvykom používať označenie *hašovacie funkcie*, v ďalšom texte túto jazykovú mutáciu nenájdete.

Kapitola 4 uvádza prehľad typov útokov na uvedené hashovacie funkcie spolu s ich "historickým kontextom". Keďže široko používaný koncept hashovacích funkcií tvorených z blokových šifier pomaly dosluhuje a prelomenie zostávajúcich existujúcich funkcií tohoto typu sa javí byť otázkou nie príliš dlhej doby, objavuje sa potreba novej koncepcie ich tvorby. Aspoň na krátky čas. Čo nám prichystá budúcnosť zľahka poodhaľuje *kapitola 5*.

Kapitola 2

Definície, využitie KSHF, motivácia útokov

2.1 Definície

2.1.1 Hashovacia funkcia, KSHF

Pod pojmom *hashovacia funkcia* rozumieme takú funkciu

$$f : X \to Y, \quad |X|, |Y| \in \mathbb{N}$$

kde X je konečná množina, Y môže byť aj nekonečná
Pod pojmom *efektívne nájsť* rozumieme nájsť výsledok dostupnými prostriedkami v dosiahnuteľnom čase.

2.1.2 Jednosmernosť, odolnosť voči kolíziám

Ak chceme hovoriť o *kryptograficky silných hashovacích funkciách*, potom k podmienke pre hashovaciu funkciu pridávame ešte podmienky na

- *jednosmernosť* (alebo tiež *odolnosť vzoru*, *preimage resistance*)
  tzn. k známemu obrazu $y \in Y$ nevieme efektívne nájsť jeho vzor $x \in X$ taký, že $f(x) = y$
- *slabú odolnosť voči kolíziám* (*second preimage resistance*)
  k vzoru $x \in X$ nevieme efektívne nájsť nejaký ďalší $x' \in X \setminus x$ s takým istým obrazom ($f(x) = f(x')$)

- *silnú odolnosť voči kolíziám*
  nevieme efektívne nájsť ľubovoľné dva $x_1, x_2 \in X, x_1 \neq x_2$ vzory s rovnakým obrazom ($f(x_1) = f(x_2)$)

2.1.3 Padding (vypchávka)

Pri blokových hashovacích funkciách dosť často nastáva situácia, kedy vstup nemá požadovanú dĺžku a je žiadané doplnenie niekoľkých bitov vhodným spôsobom (napríklad bitmi 0) na získanie bloku požadovanej dĺžky. Tento proces nazývame *padding*.

2.1.4 Avalanche effect

Vítaným javom je aj tzv. *lavínový efekt* (*avalanche effect*), kedy aj malá zmena vstupu rapídne ovplyvní celý výstup (spôsobí "posun"výslednej hodnoty niekde úplne inde).

2.2 Využitie KSHF

Tam, kde je potrebné

- generovať otlačky dokumentov pre spätné overovanie ich pravosti resp. nepozmenenie ich obsahu (e-maily, bezpečnostné certifikáty, digitálne podpisy, kontrolné sumy, hlavičky súborov v p2p sieťach, ...)
- bezpečne uchovávať uložené "kópie" hesiel užívateľov slúžiace na autentifikáciu
- ...

2.3 Motivácie útokov na KSHF

Vzhľadom na ich široké nasadenie, aj motivácie útokov sú rôzne, zväčša však

- čisto vedecká, teda motivácia poznania
- vypísaná finančná odmena za zlomenie danej hashovacej funkcie
- možnosť nepozorovanej modifikácie "zabezpečených" dát

Kapitola 3

Prehľad KSHF, ich konštrukcie a nasadenia

3.1 Deklarovaná odolnosť:

Vzhľadom na fakt, že sa jedná o kryptograficky silné hashovacie funkcie očakávame od nich isté vhodné správanie, tzn. (nech N je dĺžka výstupu v bitoch)

- ak je daný a jeho obraz, potom nájdenie iného vzoru s rovnakým obrazom nás stojí minimálne - 2^N volaní hashovacej funkcie
- nájdenie ľubovoľných 2 rôznych vzorov s rovnakým obrazom nás stojí minimálne $2^{N/2}$ volaní hashovacej funkcie
- nájdenie správy s daným n-bitovým obrazom nás stojí aspoň 2^N volaní hashovacej funkcie

3.2 Stručný prehľad konštrukcií

3.2.1 Konštrukcie z blokových šifier

Nech m je pôvodná správa, potom m rozdelíme na bloky $(m_1, m_2, \ldots, m_n)$ rovnakej dĺžky (dĺžka bloku spomínanej funkcie + *padding* [2.1.3] v prípade potreby), nech E je šifrovacia funkcia, potom:

$H_0 = inicializačný\ vektor$
$H_n \rightarrow$ výsledok hashovacej funkcie

Matyas, Meyer, Oseas

$H_i = E_{H_{i-1}}(m_i) \oplus m_i$, pre $i = 1, 2, \ldots, n$

Davies, Meyer

$H_i = E_{m_i}(H_{i-1}) \oplus H_{i-1}$, pre $i = 1, 2, \ldots, n$

Miyaguchi, Preneel

$H_i = E_{H_{i-1}}(m_i) \oplus H_{i-1} \oplus m_i$

3.2.2 Iterované hashovacie funkcie

Rovnako ako v predchádzajúcom prípade m rozdelená na bloky rovnakej dĺžky podľa funkcie. V prípade, že dĺžka správy nie je násobkom veľkosti bloku, nastupuje tzv. *padding* (2.1.3). Následné spracovanie pomocou kompresnej funkcie f

$H_0 =$ *inicializačný vektor*
$H_i = f(m_i, H_{i-1})$, pre $i = 1, 2, \ldots, n$
$H_n \rightarrow$ výsledok hashovacej funkcie

Merkle-Damgård

Konštrukcia rozširujúca funkciu $f : \{0,1\}^{n+r+1} \rightarrow \{0,1\}^n, r \geq 1$ odolnú voči kolíziam na $h : \{0,1\}^* \rightarrow \{0,1\}^n$ taktiež odolnú voči kolíziam.

1. vstup x (dĺžky l) rozdelíme na bloky dĺžky r-bitov $(x_1, x_2, \ldots, x_t$, v prípade potreby doplníme posledný blok nulami)
2. na záver doplníme toľko blokov, koľko bude nevyhnutné na uloženie binárneho zápisu l
3. $$\begin{aligned} h(x) &= H_{t+1} \\ H_1 &= f(0^{n+1} \parallel x_1) \\ H_i &= f(H_{i-1} \parallel 1 \parallel x_i),\ \textit{pre}\ i = 2, 3, \ldots t+1 \end{aligned}$$

Výsledná funkcia h je rovnako odolná voči kolíziám, ako jej kompresná funkcia.

3.3 HAVAL

Autori: *Yuliang Zheng, Josef Pieprzyk, Jennifer Seberry*
Rok uvedenia: 1992
Veľkosť bloku: 1024-bit
Počet prechodov blokom: 3, 4, 5
Veľkosť výstupu: 128-bit, 160-bit, 192-bit, 224-bit, 256-bit
Špecifikácia: *http://labs.calyptix.com/files/haval-paper.pdf*
Licencia: BSD licence

Oficiálna implementácia HAVAL-u je dostupná na webovej stránke (posledná revízia: P. Barreto, apr 1997). Podľa experimentov autorov, pri 3 prechodoch je HAVAL o 60% rýchlejší, pri 4 prechodoch o 15% rýchlejší a pri 5 prechodoch rovnako rýchly ako MD5.

Vzhľadom na rozsah špecifikácie HAVAL-u neuvádzam ani stručný popis konštrukcie, v prípade záujmu odporúčam *špecifikáciu*.

3.4 Rodina MD

Názov: *Message Digest*
Autor: *Prof. Ronald Rivest* , MIT.

3.4.1 MD2

Rok uvedenia: 1989
Veľkosť bloku: 128-bit
Počet prechodov blokom: 18
Veľkosť výstupu: 128-bit
Špecifikácia: RFC 1319 (*http://tools.ietf.org/html/rfc1319*)
Využitie: kontrolná suma v e-mailoch
Iné: Optimalizované pre 8-bitové počítače

- padding na dĺžku $n \times 128$ bit, $n \in \mathbb{N}$ pridaním i bajtov hodnoty i za koniec vstupu
- pridanie 128-bit checksum-u, tvoreného z “náhodnej"permutácie cifier čísla, následne XOR-ovanými so všetkými blokmi vstupu
- inicializácia “MD bufferu” - vynulovanie 48 bajtového bufferu X na počítanie hash sumy
- N - dĺžka vstupu po paddingu (v bajtoch)

```
/* prejdi každým 512-bitovým blokom.*/
For i = 0 to N/16-1 do

   /* skopíruj blok do buffera X. */
   For j = 0 to 15 do
      Set X[16+j] to M[i*16+j].
      Set X[32+j] to (X[16+j] xor X[j]).
   end /* slučky na j */

   Set t to 0.

   /* 18 prechodov, slučka na j */
   For j = 0 to 17 do

      For k = 0 to 47 do
         Set t and X[k] to (X[k] xor S[t]).
      end /* slučky na k */

      Set t to (t+j) modulo 256.
   end /* slučky na j */

end /* slučky na i */
```

- $MD2(M) = X_1 X_2 \dots X_{16}$

3.4.2 MD4

Rok uvedenia: 1990
Veľkosť bloku: 512-bit
Počet prechodov blokom: 3
Veľkosť výstupu: 128-bit
Špecifikácia: RFC 1320 (*http://tools.ietf.org/html/rfc1320*)
Využitie: ed2k
Iné: Optimalizované pre 32-bitové počítače. Vyvinuté pre potreby RSA Data Security.

1. padding na dĺžku $(n * 512bit = 448bit) \equiv MOD\ 512, n \in \mathbb{N}$ pridaním jedného bitu 1 a zvyšok doplnený bitmi 0

2. pridanie 64-bit čísla (dĺžku pred paddingom) za koniec vstupu. Ak si reprezentácia pôvodnej dĺžky vyžaduje viac než 64-bitové číslo, pridáme na záver iba posledných (spodných) 64-bitov z čísla reprezentujúceho danú dĺžku.
3. inicializácia "MD bufferu" - 4 32-bit slová (A, B, C, D) nasledovne (spodné bajty zľava, hexadecimálne)
   (A,B,C,D) = (01 23 45 67, 89 ab cd ef, fe dc ba 98, 76 54 32 10)
4. počítanie sumy. 3 pomocné funkcie ($\oplus$ značí XOR, $\vee$ značí OR)

$$\begin{aligned} F(X,Y,Z) &= XY \vee \bar{X}Z \\ G(X,Y,Z) &= XY \vee XZ \vee YZ \\ H(X,Y,Z) &= X \oplus Y \oplus Z \end{aligned}$$

N - dĺžka vstupu po paddingu (v bajtoch)

```
/* prejdeme všetky 512-bitové bloky */
For i = 0 to N/16-1 do

  /* skopírujeme i-ty blok do X */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* slučky na j */

  /* uloženie A do AA, B do BB, C do CC, D do DD. */
  AA = A    BB = B    CC = C    DD = D

  /* 1. prechod */
  /* [abcd k s] značí operáciu
       a = (a + F(b,c,d) + X[k]) <<< s. */
  /* vykonať nasledujúcich 16 operácií */
  [ABCD  0  3]  [DABC  1  7]  [CDAB  2 11]  [BCDA  3 19]
  [ABCD  4  3]  [DABC  5  7]  [CDAB  6 11]  [BCDA  7 19]
  [ABCD  8  3]  [DABC  9  7]  [CDAB 10 11]  [BCDA 11 19]
  [ABCD 12  3]  [DABC 13  7]  [CDAB 14 11]  [BCDA 15 19]

  /* 2. prechod */
  /* [abcd k s] značí operáciu
       a = (a + G(b,c,d) + X[k] + 5A827999) <<< s. */
  /* vykonať nasledujúcich 16 operácií */
  [ABCD  0  3]  [DABC  4  5]  [CDAB  8  9]  [BCDA 12 13]
```

```
    [ABCD  1  3]  [DABC  5  5]  [CDAB  9  9]  [BCDA 13 13]
    [ABCD  2  3]  [DABC  6  5]  [CDAB 10  9]  [BCDA 14 13]
    [ABCD  3  3]  [DABC  7  5]  [CDAB 11  9]  [BCDA 15 13]

    /* 3. prechod */
    /* [abcd k s] značí operáciu
         a = (a + H(b,c,d) + X[k] + 6ED9EBA1) <<< s. */
    /* vykonať nasledujúcich 16 operácií */
    [ABCD  0  3]  [DABC  8  9]  [CDAB  4 11]  [BCDA 12 15]
    [ABCD  2  3]  [DABC 10  9]  [CDAB  6 11]  [BCDA 14 15]
    [ABCD  1  3]  [DABC  9  9]  [CDAB  5 11]  [BCDA 13 15]
    [ABCD  3  3]  [DABC 11  9]  [CDAB  7 11]  [BCDA 15 15]

    /* následne prirátať ku každému slovu hodnotu,
       ktorú mal pred 3 prechodmi */
    A = A + AA    B = B + BB    C = C + CC    D = D + DD

end /* slučky na i */
```

5. $MD4(M) = ABCD$

3.4.3 MD5

Rok uvedenia: 1992
Veľkosť bloku: 512-bit
Počet prechodov blokom: 4
Veľkosť výstupu: 128-bit
Špecifikácia: RFC 1321 (*http://tools.ietf.org/html/rfc1321*)
Využitie: checksum (hlavne na unixoch, md5sum), ukladanie hashov hesiel
Iné: Optimalizované pre 32-bitové počítače. Vznikla ako bezpečnejšia (zato však pomalšia) alternatíva k MD4. Vyvinuté pre potreby RSA Data Security.

1. padding na dĺžku $(n * 512bit = 448bit) \equiv MOD\ 512, n \in \mathbb{N}$ pridaním jedného bitu 1 a zvyšok doplnený bitmi 0
2. pridanie 64-bit čísla (dĺžku pred paddingom) na záver vstupu. Ak si reprezentácia pôvodnej dĺžky vyžaduje viac než 64-bitové číslo, pridáme na záver len posledných (spodných) 64-bitov z čísla reprezentujúceho danú dĺžku.

3. inicializácia "MD bufferu" - 4 32-bit slová (A, B, C, D) nasledovne (spodné bajty zľava, hexadecimálne)
   (A,B,C,D) = (01 23 45 67, 89 ab cd ef, fe dc ba 98, 76 54 32 10)
4. počítanie sumy. 4 pomocné funkcie ($\oplus$ značí XOR, $\vee$ značí OR)

$$\begin{aligned} F(X,Y,Z) &= XY \vee \bar{X}Z \\ G(X,Y,Z) &= XZ \vee Y\bar{Z} \\ H(X,Y,Z) &= X \oplus Y \oplus Z \\ I(X,Y,Z) &= Y \oplus (X \vee \bar{Z} \end{aligned}$$

Využíva 64 prvkovú tabuľku $T[1 \ldots 64]$ zostrojenú nasledovne:
$T[i]$ - i-ty prvok tabulky, celočíselná časť z $4294967296 \times |sin(i)|$
(vstup pre sínus je v radiánoch)[1]

N - dĺžka vstupu po paddingu (v bajtoch)

```
/* pre každý 512-bitový blok */
For i = 0 to N/16-1 do

  /* skopíruj i-ty blok do X */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* slučky na j */

/* uloženie A do AA, B do BB, C do CC, D do DD. */
  AA = A    BB = B    CC = C    DD = D

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

  /* 2. prechod */
```

[1] záujemcom o konkrétne hodnoty prvkov T odporúčam pozrieť RFC 1321. Drobné upozornenie: v čase písania bola v publikovanom RFC drobná chyba v pseudokóde 3. a 4. prechodu - [*abcd k s t*] má byť nahradené [*abcd k s i*]

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

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

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

/* následne prirátať ku každému slovu hodnotu,
   ktorú mal pred 3 prechodmi */
A = A + AA    B = B + BB    C = C + CC    D = D + DD

end /* slučky na i */
```

5. $MD5(M) = ABCD$

Rozdiely oproti MD4

- pridaný 4. prechod
- každý krok prirátava vlastnú unikátnu konštantu ($T[i]$)
- funkcia g v 2. prechode bola zmenená z $(XY \vee XZ \vee YZ)$ na $(XZ \vee Y\bar{Z})$ čo ju má spraviť menej symetrickou.

- každý krok využíva výsledok z predchádzajúceho kroku, s cieľom urýchlovať avalanche effect.
- zmena poradia prístupu k jednotlivým slovám vstupu v 2. a 3. prechode (čo zaručuje menšiu podobnosť vzorov[2]).
- počet bitových rotácií v každom prechode bol optimalizovaný pre rýchlejšie dosiahnutie avalanche effect-u.

3.5 PANAMA

Autori: Joan Daemen, Craig Clapp
Rok uvedenia: 1998
Veľkosť bloku: 256-bit
Počet prechodov blokom: 4
Veľkosť výstupu: 256-bit
Špecifikácia:
http://homes.esat.kuleuven.be/ jlano/stream/papers/panamadc.pdf

PANAMA je kryptografický model využiteľný jednak ako hashovacia funkcia, no rovnako ako prúdová šifra. Jadro sa skladá z 17 (32-bit) stavových slov ($a_0, a_1, \ldots, a_{16} \rightarrow 544$ bitov) a 32 dielneho buffera (s 8 32-bitovými slovami na jeden diel $\rightarrow$ 8192-bit). Pracuje v 3 módoch - *reset* (buffer aj stav nastavené na 0), *push* (berie 256-bitové vstupy, žiadny výstup), *pull* (žiadny vstup, 256-bitový výstup). Diely označujeme b^j ($0 \leq j \leq 31$) a následne jednotlivé slová v nich b_i^j ($0 \leq i \leq 7$).

Stavový stroj PANAMA:
Update bufferu označovaný ako λ ($d = \lambda(b)$)

$$\begin{array}{lcll} d^j & = & b^{j-1} & \text{pre } j \notin \{0, 25\} \\ d^0 & = & b^{31} \oplus q & \\ d_i^{25} & = & b_i^{24} \oplus b_{i+2 \ mod \ 8}^{31} & \text{pre } 0 \leq j < 8 \end{array}$$

V *push* móde, q je vstupný blok p, v *pull* móde je časť stavu a (tzn. všetkých 8 slov: $q_i = a_{i+1}$ pre $0 \leq i < 8$).
Stav aktualizujúca transformácia označovaná

$$\rho = \sigma \circ \theta \circ \pi \circ \gamma$$

(kde $\circ$ symbolizuje asociatívnu kompozíciu transformácií, najpravejšia je vykonávaná najprv)

[2] to make these patterns less like each other

θ je invertovateľná lineárna transformácia

$$c = \theta(a) \iff c_i = a_i \oplus a_{i+1} \oplus a_{i+4} \quad \text{pre } 0 \leq i < 17 \qquad \text{(s indexami } MOD \text{ 17)}$$

γ je invertovateľná nelineárna transformácia

$$c = \gamma(a) \iff c_i = a_i \oplus (a_{i+1} OR \oplus a_{i+2}^{-}) \quad \text{pre } 0 \leq i < 17 \qquad \text{(s indexami } MOD \text{ 17)}$$

Permutácia π kombinuje cyklické rotácie slov a permutácie pozície slov. Ak definujeme τ_k ako rotáciu cez k pozícii od najmenej po najviac významný bajt, dostávame

$$c = \pi(a) \iff c_i = \tau_k(a_j) \text{ s } j = 7i \; MOD \; 17 \text{ a } k = i(i+1)/2 \; MOD \; 32.$$

σ je transformácia symbolizujúca bitový súčet buffer-a a vstupného slova, definovaná (nech $c = \sigma(a)$

$$\begin{array}{llll} c_0 = a_0 \oplus 00000001_{hex} & & \\ c_{i+1} = a_{i+1} \oplus l_i & & \text{pre } 0 \leq i < 8 \\ c_{i+9} = a_{i+9} \oplus b_i^{16} & & \text{pre } 0 \leq i < 8 \end{array}$$

Push mód: l značí vstup p
Pull mód: $l = b^4$, výstup z pozostáva z 8 slov:

$$z_i = a_{i+9} \quad \text{pre } 0 \leq i < 8$$

Samotné hashovanie:

1. padding pridaním jedného bitu 1 a vhodným počtom 0 bitov tak, aby dĺžka výsledku bola násobkom 256 bitov.
2. nech m' je správa po aplikovaní paddingu, potom ju rozdelíme na V rovnakých blokov dĺžky 256 bitov ($m' = p^1 p^2 \dots p^V$)

krok	**mód**	**vstup**	**výstup**
0	reset	-	-
$1, \dots, V$	push	p^t	-
$V+1, \dots, V+32$	pull	-	-
$V+33$	pull	-	h

3. $PANAMA(m) = h$

Rozdiely oproti MD5, SHA, RIPEMD:

- Paralelný charakter transformácie v iterovacej funkcií. Funkcie s návrhom odvodeným od MD4 vykonajú vrámci jedného cyklu viac jednoduchých prechodov za sebou, PANAMA jeden komplikovaný.
- Rozsah stavovej premennej. Funkcie s návrhom inšpirované MD4 majú stavovú premennú veľkosťou podobnú veľkosti výstupu, čo PANAMA so svojími 8 kB viacnásobne prevyšuje.
- Výsledok. Funkcie s návrhom odvodeným od MD4 majú po spracovaní vstupu výslednú hodnotu uloženú v stavovej premennej, PANAMA vyžaduje na jej dorátanie ešte 32 opeácií v móde pull naprázdno.

3.6 Rodina RIPEMD

Názov: RACE Integrity Primitives Evaluation Message Digest
Autori: *Hans Dobbertin, Antoon Bosselaers, Bart Preneel*

Pôvodný RIPEMD bol nahradený RIPEMD-128 kvôli podozreniam zo raniteľnosti. Verzie RIPEMD-128 a RIPEMD-256, rovnako ako verzia RIPEMD-160 a RIPEMD-320, sú dvojice algoritmov líšiacich sa konštrukčne iba veľkosťou výstupu, kvôli prevencií náhodných kolízií. Nezaručuje to však vyššiu úroveň bezpečnosti oproti "menej bitovým" verziám. Boli navrhnuté na akademickej pôde a nie sú viazané žiadnymi patentami. V porovnaní s používanými KSHF, sú RIPEMD a RIPEMD-128 pomalšie ako MD4 a MD5, no rýchlejšie ako SHA-1. Čo sa však nedá povedať o RIPEMD-160, ktorá je pomalšia aj ako SHA-1. Na druhej strane "má toho zatiaľ na rováši najmenej". Hashovacie funkcie rodiny RIPEMD sú optimalizované pre 32-bit procesory.

3.6.1 RIPEMD-128/256

Iné: RIPEMD-128 - súčasť štandardu ISO/IEC 10118-3
Konštrukcia je popísaná v nižšie uvedenej špecifikácií.

3.6.2 RIPEMD-160/320

Rok uvedenia: 1996
Veľkosť bloku: 512-bit
Počet prechodov blokom: 1
Veľkosť výstupu: 160/320-bit

Špecifikácia:
http://homes.esat.kuleuven.be/ cosicart/pdf/AB-9601/AB-9601.pdf
Využitie: MAC, HMAC, OpenPGP (RIPEMD-160)
Iné: RIPEMD-160 - súčasť štandardu ISO/IEC 10118-3
Navrhnuté ako bezpečná náhrada za MD4, MD5 a RIPEMD. Vzhľadom na rozsah konštrukcie v najstručnejšej forme (ktorú sa mi podarilo stvoriť) neuvádzam a ponechám na záujme čitateľa, či danú linku vzhliadne alebo nie. Z anatómie dodám len zopár faktov - spracováva 512-bitové bloky, metóda paddingu je identická tej z MD4, jeden prechod blokom pozostáva z 80 kôl, v ktorých sa aplikuje 2x vybraná (1 z 5 v závislosti od čísla kola) logická zložená funkcia s 4 slovným parametrom, 4x bitová rotácia a "veľa modulárnych sčítaní".

3.7 Rodina SHA

Názov: Secure Hash Algorithm
Autori: NSA, NIST

3.7.1 SHA-0

Rok uvedenia: 1992
Veľkosť bloku: 512-bit
Počet prechodov blokom: 1
Veľkosť výstupu: 160-bit
Špecifikácia: http://www.itl.nist.gov/fipspubs/fip180-1.htm
Iné: obmedzenie na dĺžku správy $< 2^{64}$ bajtov

Mierne upravuje MD5.

- Využíva 2 buffery z 5 32-bit slov $(A, B, C, D, E), (h_0, h_1, h_2, h_3, h_4)$
- Padding - rovnaký ako u MD5. m je správa pred paddingom, M po paddingu.
- Pomocné funkcie

$$\begin{array}{rcll} f(t,x,y,z) & = & (x \wedge y) \vee (\bar{x} \wedge z) & (0 \leq t \leq 19) \\ f(t,x,y,z) & = & x \oplus y \oplus z & (20 \leq t \leq 39) \\ f(t,x,y,z) & = & (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) & (40 \leq t \leq 59) \\ f(t,x,y,z) & = & x \oplus y \oplus z & (60 \leq t \leq 79). \\ S(n,X) & = & (X << n) \vee (X >> 32 - n) & // \text{ rotácia vľavo} \end{array}$$

- Pomocné konštanty pre jednotlivé "kolá"
  $K_t = 5a827999_{HEX} \quad (0 \leq t \leq 19)$
  $K_t = 6ed9eba1_{HEX} \quad (20 \leq t \leq 39)$
  $K_t = 8f1bbcdc_{HEX} \quad (40 \leq t \leq 59)$
  $K_t = ca62c1d6_{HEX} \quad (60 \leq t \leq 79)$
- Inicializácia:
  $h_0 = 67452301_{HEX}$
  $h_1 = efcdab89_{HEX}$
  $h_2 = 98badcfe_{HEX}$
  $h_3 = 10325476_{HEX}$
  $h_4 = c3d2e1f0_{HEX}$.
- Nech M je správa s paddingom. Rozdelíme ju na 512-bitové bloky ($M = M_1, M_2, M_N$). Spracovanie jedného 512-bitového bloku:
  1. rozdelíme M_i na 16 32-bitových slov ($W_0, W_1, \ldots, W_{15}$, W_0 je najviac vľavo).
  2. For t = 16 to 79
     $W_t = W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}$
     End
  3. $A = h_0, B = h_1, C = h_2, D = h_3, E = h_4$
  4. For t = 0 to 79
     $TEMP = S(5, A) + f(t, B, C, D) + E + W_t + K_t$
     $E = D, D = C, C = S(30, B), B = A, A = TEMP$
  5. $h_0 += A, h_1 += B, h_2 += C, h_3 += D, h_4 += E$
- $SHA(m) = h_0 h_1 h_2 h_3 h_4$

3.7.2 SHA-1

Rok uvedenia: 1995
Špecifikácia: http://tools.ietf.org/html/rfc3174
Iné: súčasť štandardu ISO/IEC 10118-3
SHA-1 sa oproti SHA-0 líši v jednom kroku kompresnej funkcie:
$W_t = S(1, W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$ // *SHA-0 tu nepoužíva bitovú rotáciu*

3.7.3 SHA-2

Rok uvedenia: 2004
Špecifikácia: http://tools.ietf.org/html/rfc4634

Spoločné pomocné funkcie (spomínané rotácie a posuny sú bitové)

$$\begin{array}{rcll} SHR(n,x) & = & x >> n & \textit{posun doprava} \\ ROTR(n,x) & = & (x >> n) \vee (x << (w-n)) & \textit{rotácia doprava} \\ ROTL(n,x) & = & (x << n) \vee (x >> (w-n)) & \textit{rotácia doľava} \\ CH(x,y,z) & = & (x \wedge y) \oplus (\bar{x} \wedge z) & \\ MAJ(x,y,z) & = & (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & \end{array}$$

SHA-256/224

Veľkosť bloku: 512-bit
Iné: 32-bitové slová

- Padding - rovnaký ako v MD5.
- Pomocné funkcie

$$\begin{array}{rcccccc} BSIG_0(x) & = & ROTR(2,x) & \oplus & ROTR(13,x) & \oplus & ROTR(22,x) \\ BSIG_1(x) & = & ROTR(6,x) & \oplus & ROTR(11,x) & \oplus & ROTR(25,x) \\ SSIG_0(x) & = & ROTR(7,x) & \oplus & ROTR(18,x) & \oplus & SHR(3,x) \\ SSIG_1(x) & = & ROTR(17,x) & \oplus & ROTR(19,x) & \oplus & SHR(10,x) \end{array}$$

- Pomocné konštanty - rovnako ako v SHA-0/1, aj tu je 80 pomocných konštánt $(K_0, K_1 \ldots K_{79})$, ktoré sú však pre každé kolo unikátne a vzhľadom na rozsah, rovnako ako inicializačné hodnoty stavových premenných, ich ponechám špecifikácií.
- Iterovanie

  $For\ i\ =\ 1\ to\ N$ // N - počet blokov
  $For\ t\ =\ 0\ to\ 15$
  $W_t = M(i)_t$

  $For\ t\ =\ 16\ to\ 63$
  $W_t = SSIG_1(W_{t-2}) + W_{t-7} + SSIG_0(t-15) + W_{t-16}$

  $a = H(i-1)_0 \qquad b = H(i-1)_1$
  $c = H(i-1)_2 \qquad d = H(i-1)_3$
  $e = H(i-1)_4 \qquad f = H(i-1)_5$
  $g = H(i-1)_6 \qquad h = H(i-1)_7$

  $For\ t\ =\ 0\ to\ 63$
  $T_1 = h + BSIG_1(e) + CH(e,f,g) + K_t + W_t$
  $T_2 = BSIG_0(a) + MAJ(a,b,c)$
  $h = g \quad g = f \quad f = e \quad e = d + T_1$
  $d = c \quad c = b \quad b = a \quad a = T_1 + T_2$

$$H(i)_0 = a + H(i-1)_0 \qquad H(i)_1 = b + H(i-1)_1$$
$$H(i)_2 = c + H(i-1)_2 \qquad H(i)_3 = d + H(i-1)_3$$
$$H(i)_4 = e + H(i-1)_4 \qquad H(i)_5 = f + H(i-1)_5$$
$$H(i)_6 = g + H(i-1)_6 \qquad H(i)_7 = h + H(i-1)_7$$

- $SHA-256(m) = H(N)_0 H(N)_1 \dots H(N)_7$
  $SHA-224(m) = H(N)_0 H(N)_1 \dots H(N)_6$

SHA-512/384

Veľkosť bloku: 1024-bit
Iné: 64-bitové slová, súčasť štandardu ISO/IEC 10118-3

- Padding - rovnaký ako v MD5 s tým rozdielom, že na 1024-bitové bloky, nuly sa dopĺňajú pokým dĺžka $\equiv 896 (MOD\ 1024)$, následne pridávaná 128-bitová binárna reprezentácia dĺžky správy pred paddingom.
- Pomocné funkcie

$$\begin{array}{rcccccc} BSIG_0(x) & = & ROTR(28,x) & \oplus & ROTR(34,x) & \oplus & ROTR(39,x) \\ BSIG_1(x) & = & ROTR(14,x) & \oplus & ROTR(18,x) & \oplus & ROTR(41,x) \\ SSIG_0(x) & = & ROTR(1,x) & \oplus & ROTR(8,x) & \oplus & SHR(7,x) \\ SSIG_1(x) & = & ROTR(19,x) & \oplus & ROTR(61,x) & \oplus & SHR(6,x) \end{array}$$

- Pomocné konštanty - rovnako ako v SHA-0/1, aj tu je 80 pomocných (ibaže 64-bitových) konštánt $(K_0, K_1 \dots K_7 9)$, ktoré sú však pre každé kolo unikátne a vzhľadom na rozsah, rovnako ako inicializačné hodnoty stavových premenných, ich ponechám špecifikácií.

- Iterovanie
  $For\ i\ =\ 1\ to\ N$ // N - počet blokov
  $For\ t\ =\ 0\ to\ 15$
  $W_t = M(i)_t$

  $For\ t\ =\ 16\ to\ 79$
  $W_t = SSIG_1(W_{t-2}) + W_{t-7} + SSIG_0(t-15) + W_{t-16}$

  $a = H(i-1)_0 \qquad b = H(i-1)_1$
  $c = H(i-1)_2 \qquad d = H(i-1)_3$
  $e = H(i-1)_4 \qquad f = H(i-1)_5$

$g = H(i-1)_6 \qquad h = H(i-1)_7$

$For\ t\ =\ 0\ to\ 79$
$T_1 = h + BSIG_1(e) + CH(e,f,g) + K_t + W_t$
$T_2 = BSIG_0(a) + MAJ(a,b,c)$
$h = g \quad g = f \quad f = e \quad e = d + T_1$
$d = c \quad c = b \quad b = a \quad a = T_1 + T_2$

$H(i)_0 = a + H(i-1)_0 \qquad H(i)_1 = b + H(i-1)_1$
$H(i)_2 = c + H(i-1)_2 \qquad H(i)_3 = d + H(i-1)_3$
$H(i)_4 = e + H(i-1)_4 \qquad H(i)_5 = f + H(i-1)_5$
$H(i)_6 = g + H(i-1)_6 \qquad H(i)_7 = h + H(i-1)_7$

- $SHA-512(m) = H(N)_0H(N)_1 \ldots H(N)_7$
  $SHA-384(m) = H(N)_0H(N)_1 \ldots H(N)_5$

3.8 Tiger

Autori: *Ross Anderson, Eli Biham*
Rok uvedenia: 1995
Veľkosť bloku: 512
Počet prechodov blokom: 3 (24 kôl vrámci jednej iterácie)
Veľkosť výstupu: 128-bit, 160-bit, 192-bit
Špecifikácia:
http://www.cs.technion.ac.il/ biham/Reports/Tiger/tiger/tiger.html
Licencia: Tiger nemá patentové obmedzenia a rovnako ani obmedzenia na používanie. Je voľne použiteľný, či už v referenčnej implementácií, inej implementácií alebo modifikácií referenčnej implementácie (pokiaľ stále implementujú Tiger). Autori žiadajú ľudí implementujúcich Tiger len o ich upovedomenie o použití Tiger-a a rovnako o citáciu pôvodu Tiger-a a jeho referenčnej implementácie.
Využitie: TigerTreeHash (p2p: DirectConnect, gnutella)
Iné: Optimalizované pre 64-bitové počítače. Dostatočne malé nároky na to, aby sa program vošiel do cache procesora.

Vychádza z Davies-Meyer-ovej konštrukcie[3].

[3] Verím, že ctený čitateľ mi odpustí nasledovný vizuálny prehrešok spôsobený snahou redukovať dĺžku výsledného materiálu

- využíva 3 64-bitové registre. Inicializačný vektor (h_0)
  $a = 0x0123456789ABCDEF$
  $b = 0xFEDCBA9876543210$
  $c = 0xF096A5B4C3B2E187$

- ```
  key_schedule {
   x0 -= x7 ^ 0xA5A5A5A5A5A5A5A5;
   x1 ^= x0;
   x2 += x1;
   x3 -= x2 ^ ((~x1)<<19);
   x4 ^= x3;
   x5 += x4;
   x6 -= x5 ^ ((~x4)>>23);
   x7 ^= x6;
   x0 += x7;
   x1 -= x0 ^ ((~x7)<<19);
   x2 ^= x1;
   x3 += x2;
   x4 -= x3 ^ ((~x2)>>23);
   x5 ^= x4;
   x6 += x5;
   x7 -= x6 ^ 0x0123456789ABCDEF;
  }

  save_abc {
   aa = a;
   bb = b;
   cc = c;
  }

  pass(a,b,c,mul) {
   round(a,b,c,x0,mul);
   round(b,c,a,x1,mul);
   round(c,a,b,x2,mul);
   round(a,b,c,x3,mul);
   round(b,c,a,x4,mul);
   round(c,a,b,x5,mul);
   round(a,b,c,x6,mul);
   round(b,c,a,x7,mul);
  }

  feedforward {  a ^= aa;  b -= bb;  c += cc; }

  round(a, b, c, x, mul) {
    c ^= x;
    a -= t1[c_0] ^ t2[c_2] ^ t3[c_4] ^ t4[c_6];
    b += t4[c_1] ^ t3[c_3] ^ t2[c_5] ^ t1[c_7];
    b *= mul;
  }
  ```

  potom jedna iterácia prechodu od h_i k h_{i+1} vyzerá nasledovne

  ```
  For i = 0 to N/16-1 do
    save_abc
    pass(a,b,c,5)
    key_schedule
    pass(c,a,b,7)
    key_schedule
  ```
  ```

```
  pass(b,c,a,9)
  feedforward
end /* slučky na i */
```

ri značí *i*-ty bajt z registra *r*. ^ označuje XOR, << a >> logické bitové posuny, operátory $X\ op =\ Y$ to isté, čo v jazyku C, teda $X\ =\ X\ op\ Y$

- výsledným hashom je hodnota uložená v *abc* po poslednom volaní $feedforward$

Tiger/128

Výsledná hodnota je prvých 128 bitov z výsledku Tiger/192. Použitý len kvôli kompatibilite s MD4, MD5, RIPEMD, variantom Snefru a pár ďalším hashovacím funkciám založeným na blokových šifrách.

Tiger/160

Výsledná hodnota je prvých 160 bitov z výsledku Tiger/192. Použitý kvôli kompatibilite s SHA a SHA-1;

Tiger2

Líšia sa od pôvodnej Tiger akurát spôsobom paddingu. Tiger2 využíva rovnaký ako MD5/SHA.

3.9 WHIRLPOOL

Autori: *Vincent Rijmen, Paulo S. L. M. Barreto*
Rok uvedenia: 2000
Veľkosť bloku: 512-bit
Počet prechodov blokom: 10
Veľkosť výstupu: 512-bit
Špecifikácia:
http://planeta.terra.com.br/informatica/paulobarreto/whirlpool.zip
Licencia: WHIRLPOOL nie je (a nikdy nebude) patentovaný. Môže byť bezplatne použitý na ľubovoľné účely. Referenčná implementácia je voľne dostupná."
Iné: súčasť štandardu ISO/IEC 10118-3, obmedzenie dĺžky vstupu na

2^{256}

Pôvodná verzia je označovaná ako *WHIRLPOOL-O*, revízia z roku 2001 ako *WHIRLPOOL-T* a revízia 2003 - *WHIRLPOOL*.

- Merkle-ho hashovacia funkcia s blokovou šifrovaciou funkciou W (512-bitový kľúč) využívajúcou *konečné pole*, *substitučnú tabuľku* (S-Box), *rozptylovú maticu*.

  S-Box: pôvodne generovaný náhodne (čo komplikovalo hardware-ovú implementáciu), v 2001 pozmenený, vylepšené kryptografické vlastnosti aj možnosť hardware-ovej implementácie.
  Rozptylová matica[4]: 8x8, ktorej koeficienty tvorí 8-ica konštánt, ktorých rotáciou doprava dostávame jednotlivé riadky matice. Pôvodná 8-ica (1, 1, 3, 1, 5, 8, 9, 5) bola pri revízií v roku 2003 nahradená konštantami (1, 1, 4, 1, 8, 5, 2, 9).

- Počas výpočtu sa interné stavové dáta držia v matici a - 8x8 x 8-bit. i-ty prechod funkcie W sa skladá z

  1. $(\gamma(a))$ paralelnej nelineárnej substitúcie prvkov matice stavovej podľa S-Boxu
  2. $(\pi(a))$ rotácia prvkov stavovej matice vrámci stĺpco - j-ty stĺpec - "rotáciaö j prvkov nadol $(0 \leq j \leq 7)$
  3. $(\theta(a))$ rozptylovanie prvkov podľa rozptylovej matice (
  4. $(\sigma[k](a))$ bitový súčet aktuálnej matice s kľúčom
  5. prenásobením maticou pre i-ty prechod (generovanej z čísla prechodu a S-Boxu)

- Padding - jeden 1 bit, potrebný počet 0 bitov na doplnenie dĺžky a získanie nepárneho počtu 256-bitových blokov. Následne pridanie 256-bitového bloku s doprava zarovnanou binárnou reprezentáciou dĺžky pôvodného reťazca. Výsledná správa M je potom rozdelená na t blokov $(M = m_1, \ldots m_t)$

- Iterácie postavené na schéme Miyaguchi-Preneel

$$\begin{array}{rcll} \eta_i & = & \mu(m_i), & \\ H_0 & = & \mu(IV), & // \ IV \text{ - 512-bitový inicializačný vektor} \\ H_i & = & W[H_{i-1}](\eta_i) \oplus H_{i-1} \oplus \eta_i & 1 \leq i \leq t \end{array}$$

- $WHIRLPOOL(M) \equiv \mu^{-1}(H_t)$.

[4] diffusion matrix

Kapitola 4

História útokov

4.1 Stručný prehľad typov útokov

4.1.1 Brutal force

Konštrukciou útoky z kategórie *brutal force* neoznačujeme ako "state of art", no určite stoja za povšimnutie. V dobe, keď spojením niekoľko tisíc chipov optimalizovaných na počítanie danej hashovacej funkcie a s investíciou na úrovni miliónov dolárov je reálne dosiahnuť (pre smrteľníka) neuveriteľný výpočtový výkon schopný s optimalizáciou (čo už nespadá do kategórie BF) lámania rôznych "slabších šifier" (takto bol zlomený napr. DES s 56-bitovým kľúčom). Výkon dosiahnuteľný hardware-om v dohľadnej dobe a vlastnosti aj nižšie uvedených útokov sú kľúčovými aspektmi pri voľbe dĺžok kľúča pre "bezpečné"hashovacie (aj šifrovacie) funkcie.

Keď sme už narazili na hrubú silu - zaujímavou voľbou, z hľadiska využitia dostupného hardware na účely "domáceho lámania", sa javí byť využitie GPU grafických kariet. Dôvodov je hneď niekoľko:

- výpočty bežiace na grafickom procesore (GPU), na rozdiel od tých bežiacich na "centrálnom" procesore (CPU), nie sú prerušované ostatnými úlohami systému

- disponuje viacerými programovateľnými pipeline-ami ("rúrami") poskytujúcimi istý stupeň paralelizmu, ktorý je pre naše potreby vhodnejší, ako ten poskytovaný CPU. Thready sú paralelné len naoko (pokiaľ ich procesor skutočne hardware-ovo nepodporuje). Viacjadrové CPU, prípadne ich väčší počet by zdalnivo mohli

byť dobrým riešením, no po krátkom zamyslení zistíme, že riešenie postavené na viacerých grafických procesoroch s viac jadrami a "rúrami" je oveľa dostupnejšie a lacnejšie.

- relatívne silné framework-y na prácu s GPU sú voľne k dispozícií (napr. *nVidia CUDA*).

Ako príklad uvediem nedávno zverejnený antwerpský projekt *Fastra* (*http://fastra.ua.ac.be/*), kde s investíciou výšky cca. 4000 euro získate spojením 4 relatívne výkonných voľne dostupných grafických kariet súčasnosti (*nVidia GeForce 9800GX2*) výkon 8 grafických procesorov s dokopy 1024 subprocesormi schopných pracovať paralelne. Keďže nemalá časť dosiaľ zverejnených útokov hľadajúcich kolízie sa po zúžení množiny kandidátov prikloní k preberaniu zvyšných možnosti, potom možnosť skrátenia tejto doby (veľmi zjednodušene povedané) 1024 krát určite stojí aspoň za povšimnutie.

4.1.2 Birthday attack

Alebo tiež *narodeninový útok* je všeobecný útok na hashovacie funkcie (teda nie na konkrétny typ konštrukcie) hľadajúci kolízie na hashovacej funkcií $f : X \to Y$ nasledovne:

1. zvolíme si niekoľko rôznych vzorov $x_1, x_2, \ldots x_k \in X$
2. vypočítame ich obrazy $f(x_1), f(x_2), \ldots f(x_k)$
3. zotriedime otlačky a hľadáme v nich kolízie

Predpokladáme, že výpočet obrazu nám trvá konštantný čas, potom časová zložitosť útoku je $O(k)$ (bod 3 vieme vzhľadom na konštantnú dĺžku otlačkov tiež riešiť v lineárnom čase od počtu vzorov - k).
V našom prípade je teda podstatný počet testovaných vzorov, od čoho sa odvíja pravdepodobnosť úspechu nájdenia kolízie.

[1] str. 59: *Narodeninový útok je dôvod, prečo sa v praxi požaduje dvojnásobná dĺžka výstupu hašovacej funkcie v porovnaní s dĺžkou kľúča symetrických šifrovacích algoritmov. Zložitosť všeobecného útoku na šifrovací algoritmus, teda útoku úplným preberaním, je* $O(2^k)$*, ak množina kľúčov je* $\{0,1\}^k$*. Na vynútenie rovnakej zložitosti všeobecného útoku na hašovaciu funkciu, teda narodeninového útoku, je potrebná množina hodnôt hašovacej funkcie* $O(2^{2k})$.

4.1.3 Rainbow tables

Máme známy *hash_hodnoty*, poznáme hashovaciu funkciu h, chceme zistiť takú *hodnotu*, že $h(hodnota) = hash\_hodnoty$. Útok publikovaný švajčiarskym kryptografom Phillipe Oechslin-om bol inšpirovaný myšlienkou "Time-Memory Trade-Off" Martina Hallmana. Využívaná je najmä na útoky na hashované dáta známej povahy, kedy vieme pôvodnú množinu potenciálne zobrazovaných hodnôt zredukovať na menšiu množinu reálne zobrazovaných hodnôt (H). Úplné preberanie zredukovanej množiny je časovo náročné, a ak aj realizovateľné, tak všetky dvojice $[hodnota, hash\_hodnoty]$ si s najväčšou pravdepodobnosťou do pamäte uložiť nemôžeme. Tam zväčša môžeme uložiť len zlomok z počtu všetkých možných hashov. Pokiaľ si ale ten zlomok vhodne vyberieme, môže nám to stačiť. Niečo predrátame predom a zapamätáme, niečo dorátame potom pri samotnej aplikácií - spomínaný Time-Memory Trade-Off. Rozdelíme si zredukovanú množinu na množstvo disjunktných podmnožín zvolenej početnosti a budeme si pamätať len reprezentantov daných množín tak, aby sme vedeli dané množiny následne zrekonštruovať. Rozdeľovanie realizujeme predom a výsledky ukladáme, rekonštrukciu množiny realizujeme pri samotnej aplikácí. Ako?

Predspracovanie: K známej hashovacej funkcii si zostrojíme funkciu mapujúcu výsledky hashovacej funkcie na prvky z redukovanej množiny. Označme ju $reduce : 2^l \rightarrow h$, kde l je počet bitov výstupu hashovacej funkcie, $h = |H|$. Budeme zostrojovať tzv. *reťazce*. Konštrukcia jedného reťazca vyzerá nasledovne:

- p_1 - nejaká počiatočná *hodnota*, $p_1 \in H$
- $p_i = reduce(h(p_{i-1}))$ - $p_i \in H$
- $hash_n = h(p_n)$

Takto si pre daný reťazec symbolizujúci n-rôznych hesiel a k ním prislúchajúcich hashov $(p_1, hash_1, p_2, hash_2, \ldots, p_n, hash_n)$, ktoré sme pred chvíľou vyrátali, uložíme dvojicu $[p_1, hash_n]$. Predrátame dostatočné množstvo reťazcov a uložíme do *rainbow table*.

Hľadanie: Nech sa *hodnota* nachádzala v danom reťazci. Potom zjavne

$$hodnota \rightarrow (hash_i \rightarrow p_i)^k \rightarrow hash_n \ (0 \leq k < n)).$$

Nám je však známy iba $hash\_hodnoty$ a postupnosť

$$hash\_hodnoty \rightarrow p_i \rightarrow hash_i \rightarrow^* p_n \rightarrow hash_n.$$

Zistenie nejakej $hodnoty^*$ s otlačkom $hash\_hodnoty$ už nerobí problém. Poznáme dvojicu $[p_1, hash_n]$, a teda aj

$$p_1 \rightarrow hash_1 \rightarrow^* p_l \rightarrow hash_l = hash\_hodnoty.$$

Hľadaná $hodnota^*$ je teda p_l. Všeobecne - ak máme predspracované vhodné reťazce a rozumne organizované rainbow tables (n je dĺžka reťazca, m počet reťazcov), potom časová zložitosť zisťovania $hodnoty^*$ k známemu $hashu\_hodnoty$ (keď nerátame inicializáciu rainbow tables) je $O(m \times log_2\ n + n)$ (hľadanie reťazca + rekonštrukcia $hodnoty^*$).

Postihuje: Systémy, ktoré používajú "čisté hashovacie funkcie tzn. napr. heslá pre autentifikáciu sú v systéme ukladané len ako hashsum pôvodného hesla (napr. Windows NT/2000/XP - LanManager, unixy v dávnych dobách).

Obrana: Jednou z možných ochrán je pridávanie tzv. *salt-u* (náhodnej hodnoty) k vstupu. Doterajšia situácia sa zmení z $hash\_hodnoty = h(hodnota)$ na $hash\_hodnoty = h(hodnota, salt)$. Pokiaľ je salt slutočne pri tvorbe hesla náhodný (samozrejme, následne si ho treba pri skladovanom hesle držať), potom sa rainbow tables stávajú na riešenie tohto problému nepoužiteľné, keďže reťazce pre funkciu s iným saltom vyzerajú inak a počet potenciánych možností sa rapídne zvyšuje (predrátavanie stráca zmysel).

4.1.4 Poisoned block attack

KSHF sa využívajú aj pri digitálnych podpisoch dokumentov, kedy človek "podpisuje" namiesto celého veľkého dokumentu iba výsledok po zahashovaní v domnení, že je dostatočnou zárukou, že podpísaný dokument nemôže nikto zmeniť. Práve táto rutina je cieľom útoku "otráveným blokom". Pokiaľ niekto dokáže zameniť blok v dokumente za iný blok s rovnakým hashom (samozrejme správnym spôsobom, v závislosti od hashovacej funkcie), potom zjavne vyhral. Stručná ilustrácia takéhoto útoku - časť PostScriptu, ktorý má rovnaký hash no vie vypísať 2 rôzne texty:

$$(R1)\ (R2)\ eq\ \{vypisprvehotextu\}\ \{vypisdruhehotextu\}\ ifelse.$$

4.2 HAVAL

***17. Aug 2004*: X. Wang, D. Feng, X. Lai, H. Yu**

Stručný popis: ohlásené 3 kolízie pre HAVAL (128 bit, 3 prechody)
Zdroj: http://eprint.iacr.org/2004/199.pdf

Útok pozmenením pôvodnej správy. Nech

$$M' = M + \triangle C, \triangle C = (2^{i-1}, 0, 0, \ldots, 0, 2^{i-12}, 0, \ldots 0, 2^{i-8}, 0, \ldots)$$

pričom nenulové pozície sú $0, 11$ a 18. Dokopy 2^6 volaní HAVALu, s $i = 0, 1, \ldots, 31$. Výsledný efekt: $HAVAL(M) = HAVAL(M')$ pre nájdené prípady.

4.3 Rodina MD

4.3.1 MD2

- *Máj 1995* - N. Rogier, P. Chauvaud, "The compression function of MD2 is not collision free", Selected Areas in Cryptography – SAC'95, Ottawa, Canada, May 18–19, 1995 (workshop record).

- *1997* - N. Rogier, P. Chauvaud, MD2 is not Secure without the Checksum Byte, Designs, Codes and Cryptography, 12(3), pp245-251, 1997.

- *2004* - F. Muller - nájdený preimage attack so zložitosťou $2^1 04$ applikácií kompresnej funkcie. "MD2 už nemôže byť považovaná za bezpečnú jednosmernú hashovaciu funkciu".

- *2005* - Lars R. Knudsen and John Erik Mathiassen, Preimage and Collision Attacks on MD2. FSE 2005.

4.3.2 MD4

- *1991* - prvé slabosti funkcie publikované dvojicou B. den Boer, A. Bosselaers

- *1992* - vznikla MD5 ako náhrada MD4
- *1996* - H. Dobbertin - kolízie s komplexnosťou 2^{22}
- X. Wang, ... - útok s pravdepodobnosťou úspechu medzi 2^{-2} a 2^{-6} a komplexnosťou 2^8 volaní MD4.
  http://www.infosec.sdu.edu.cn/paper/md4-ripemd-attck.pdf[1]

Nemálo z útokov na MD5 sa dá v upravenej forme aplikovať aj na MD4. V prípade hlbšieho záujmu o MD4 ale aj ďalšie od nej odvodené funkcie odporúčam:
M. Daum: Cryptanalysis of Hash Functions of the MD4-Family
http://www.cits.rub.de/imperia/md/content/magnus/dissmd4.pdf.

4.3.3 MD5

- *5. Júl 1993* - B. den Boer a A. Bosselaers zverejnili asi prvú známu pseudokolíziu kompresnej funkcie MD5 (s komplexnosťou 2^{16}). S rôznymi IV[2] dosiahli výsledok s rozdielom iba na 4-bitoch (po aplikovaní kompresnej funkcie).
  http://homes.esat.kuleuven.be/ cosicart/pdf/AB-9300.pdf
- *Aug 1996* - H. Dobbertin ohlásil pseudokolíziu (v tomto prípade kolíziu s rozdielnymi inicializačnými vektormi) na oslabenej verzií MD5 (s komplexnosťou 2^{34}), čo spôsobilo dostatočný rozruch a zapríčinilo postupné nahrádzanie MD5 bezpečnejšími funkciami. *ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf*
- *1. Mar - 24. Aug 2004* - *MD5CRK* - projekt distribuovaného lámania MD5 hrubou silou, postavený na princípe birthday attack. Skončil bez úspechu po tom, ako...
- *17. Aug 2004* - X. Wangová a kol. prezentovali kolízie získané analytickou metodou, nie však metódu. Útok trval hodinu na klastri IBM p690.
- *1. Mar 2005* - A. Lenstra, X. Wangová, B. de Weger demonštrovali konštrukciu 2 certifikátov X.509 s rozdielnymi verejnými kľúčmi, no rovnakými MD5 sumami. Zverejnili aj obe dvojice verejných a súkromných kľúčov.

[1] v čase finalizácie tejto práce bol obsah na danej adrese nedostupný
[2] inicializačnými vektormi

- *5. Mar 2005* - V. Klíma prezentoval svoju rýchlejšiu metódu - 8 hodín na bežnom PC.
  http://cryptography.hyperlink.cz/md5/MD5_ collisions.pdf

- *9. Mar 2005* - X. Wangová a kol. dodatočne prezentoval svoju (už pomalšiu metódu s komplexnosťou 2^39) + postačujúce podmienky na kolízie.
  http://www.infosec.sdu.edu.cn/paper/md5-attack.pdf
  dodatočne sa ukázalo, že sú nesprávne

- *10. Aug 2005* - Yajima a Shimoyama upravili nesprávne podmienky, *http://eprint.iacr.org/2005/263.pdf*

- *7. Nov 2005* - Sasaki a kol. upravili nesprávne podmienky, *http://eprint.iacr.org/2005/400.pdf*

- *23. Nov 2005* - Liang a Lai - vyvrátili Wangovej podmienky, prezentovali nové a pravdepodobne definitívne správne.
  http://eprint.iacr.org/2005/425.pdf

- Aktuálny Klímov algoritmus beží pod 30 sekúnd na *bežnom PC*.

Metóda modulárnej diferencie

Diferenčná metóda sama o sebe bola uvedená duom E.Biham a A.Shamir pri analýze bezpečnosti DES. Keďže základné operácie, ktoré hashovacie funkcie používajú sú v podstate modulárna aritmetika a logické operácie, potom logicky nástroje na analýzu rozdieloch stavov v jednotlivých krokoch / prechodoch kompresnej funkcie sú modulárna diferencia a XOR. Tzn. metódou pracného odhaľovania vzťahov medzi dátami a stavmi prechodovej funkcie postupne vznikali nutné a postačujúce podmienky pre diferenčné schémy.

Metóda tunelov

Zdroj: *http://cryptography.hyperlink.cz/2006/tunnels.pdf*

Publikovaná Vlastimilom Klímom pôvodne v roku 2005, následne postupne upravovaná, aj v závislosti na momentálne dostupnej publikovanej sade postačujúcich podmienok pre kolízie. Tunely predstavujú akúsi, “cestu naprieč bitmi jednotlivých iterácií"s tým, že hľadáme bity, ktorých zmena sa v ďalších iteráciách neprejaví/vieme ju kompenzovať vhodnou substitúciou ďalších bitov vstupu - viacnásobná modifikácia

pôvodného vstupu postavená na spomínaných tuneloch. Oproti diferenčnej metóde je menej náročná na výpočtový výkon a podľa autora všeobecne použiteľná. Kolíziu MD5 na bežnom notebooku Pentium 1,6 GHz dosiahne včase menšom ako 1 minútu.

Klíma sa nazdáva, že použitie jeho metódy tunelov je v mierne upravenej forme možné na ľubovoľnú funkciu z rodiny odvodených od MD4.

iné dostupné útoky na MD5

Vyššie popísaný “poisoned block attack", rovnako na internete je dostupná sada vytvorených rainbow tables pre MD5.

4.4 PANAMA

- *2001* - V. Rijmen publikoval útok zložitosti 2^{82} s minimálnymi pamäťovými nárokmi.
- *2007* - J. Daemen, G. Von Assche - zložitosť 2^6, využívajúc ku kolízií ľubovoľný prefixový reťazec a následné zreťazenie s blokmi s vhodne zvolenou diferenciou. Z útoku vyplýva, že PANAMA sa v danú chvíľu za bezpečnú považovať nedá.
  http://radiogatun.noekeon.org/panama/PanamaAttack.pdf

4.5 RIPEMD

- *17. Aug 2004*: X. Wang, D. Feng, X. Lai, H. Yu - 2 kolízie pre RIPEMD (*http://eprint.iacr.org/2004/199.pdf*)

  Útok má obdobnú povahu, ako útok na HAVAL-128, rovnako sa jedná o zmenu pôvodnej správy.

$$M' = M + \triangle C$$
$$\triangle C = (0, 0, 0, 2^{20}, 0, 0, 0, 0, 0, 0, 2^{18} + 2^{31}, 0, 0, 0, 0, 2^{31}).$$

4.6 Rodina SHA

Napriek faktu, že rodina SHA mala byť posilneným odvodencom od MD*, časť návrhových čŕt majú spoločnú, čo ich miestami činí byť náchylné podliehať rovnakým typom útokov.

4.6.1 SHA-0

- *1998* - Florent Chabaud, Antoine Joux: Differential Collisions in SHA-0. CRYPTO 1998. pp56-71, komplexnosť 2^{61} (teória)
- *2004* - Biham, Chen - near-collision komplexnosť 2^{40}
- *2005* - Biham a kol. - kolízia, komplexnosť 2^{51}
- *2005* - Wangová a kol. - kolízia, komplexnosť 2^{39}

4.6.2 SHA-1

- *2005* - Biham a kol. - kolízia na oslabenom modeli (40 kôl), komplexnosť "malá"
- *2005* - Biham a kol. - kolízia na oslabenom modeli (58 kôl), komplexnosť 2^{75} (teória)
- *2005* - Wangová a kol. - kolízia na oslabenom modeli (58 kôl), komplexnosť 2^{63}
- *2005* - Wangová a kol. - kolízia, komplexnosť 2^{33} (teória)

4.7 Tiger

***feb 2006*: J. Kelsey, S. Lucks - oslabený model**

Zdroj: *http://th.informatik.uni-mannheim.de/People/Lucks/papers/Tiger_FSE_v10.pdf*
Dva útoky - prvý využíva model oslabený z pôvodných 24 kôl na 16 kôl na iteráciu s prácou ekvivalentnou 2^{44} volaní kompresnej funkcie na nájdenie kolízie.
Druhý ukazuje "takmer" kolíziu (s dvoma správami s Hammingovou vzdialenosťou 6) na modeli s 20 kolami s náročnosťou $\leq 2^{48}$ volaní kompresnej funkcie.

Okrem poukázania na možnú zraniteľnosť hashovacej funkcie Tiger však tento útok ukázal, že diferenčné metódy používané na lámanie funkcií odvodených od MD4 sa dajú v málo upravenej forme použiť aj na lámanie funkcií s odlišným návrhom (teda nie sú špecifické len pre MD4).

Kapitola 5

Blízka budúcnosť KSHF

Väčšina dnes používaných hashovacích funkcií využíva na zosilnenie konštrukciu z blokových šifier, ktorých bezpečnosť však bola postavená na iných kritériach ako hashovacie funkcie. Blokové šifry ich bezpečnosť opierajú aj o informáciu, ktorá zostáva tajná. Pri hashovacích funkciách však nič tajné nie je a blokové šifry predstavujú skôr "znáhodnenie" ako reálne bezpečnostné zosilnenie. Útočník tak pozná vstup, výstup, môže analyzovať výpočtový proces. Na tomto princípe bola postavená väčšina popisovaných útokov na konkrétne hashovacie funkcie. Kedže doteraz používané koncepcie sa javia byť pre dnešnú a najbližšiu dobu slabé, je tu medzinárodná iniciatíva o stvorenie nových koncepcií. Na túto tému je však dosiaľ publikovaného len málo.

Rovnako väčšina dnešných funkcií vychádza z využitia jednoduchých a rýchlych logických operácií, ktoré sú v konečnom dôsledku popísateľné diferenčnými schémami. Tie sa taktiež frekventovane využívajú a ich postupné odhalenie pre danú funkciu závisí viac-menej od "neprehladnosti" resp. komplikovanosti jej návrhu. Prinesie nám budúcnosť funkcie, ktoré na tomto princípe postavené nie sú? Uvidíme :)

5.1 V. Klíma a kol. - Special Nested MAC

Koncept vznikol vrámci projektu českého NBÚ "Bezpečná hašovací funkce". Konštrukcia vychádza z Bellare-Coronových NMAC a HMAC, ktoré mali autorom dokázanú odolnosť v nekonečne. Autori vypracovali tesné kvantitatívne horné a dolné odhady odolnosti pre konečné rozmery. Odhady ukazujú, že na nájdenie kolízií pre NMAC/HMAC potrebuje útočník približne toľko isto operácií, ako pre náhodné oráku-

la (ktoré hashujú rovnaké vstupné dáta na rovnaké hodnoty).
Novým prvkom v konštrukcií je tzv. "špeciálna bloková šifra". Oproti používaným blokovým šifrám sa má líšiť možnosťou manipulovať s kľúčom a homogenitou pri spracovaní všetkých vstupných bitov. Navrhovaný tvar špeciálnej blokovej šifry:

$$f : \{0,1\}^K \rightarrow \{0,1\}^n : k \rightarrow E_k(Const_0).$$

Konštrukcia SNMAC nahrádza v konštrukcií NMAC náhodné orákulum f, špeciálnou blokovou šifrou f, analogicky pre g.
$m = m_1, m_2, \ldots m_L$; Operátor . značí zreťazenie.

- m_i - dĺžka K bitov
- h_i - dĺžka n bitov
- h_0
- $h_i = f(h_{i-1}, m_i)$
- $NMAC(m) = g(h_L)$

NMAC

- m_i - dĺžka $K - n$ bitov
- h_i - dĺžka n bitov
- h_0
- $h_i = E_{h_{i-1}.m_i}(Const_0)$
- $SNMAC(m) = E_{h_L.NULL}(Const_1)$

SNMAC založená na NMAC

SNMAC sa dá konštruovať aj z HMAC nahradením klasickej blokovej šifry "špeciálnou".

5.2 31. Október 2008

Nenaznačuje spomienky na minulosť či nebodaj istú dávku čierneho humoru, ale naopak, našu potenciálne svetlú budúcnosť. Je to termín uzavretia súťaže vypísanej NIST-om, súťaže o kandidáta na SHA-3. Očakáva sa od nej napríklad:

- kompatibilita s SHA-2 čo sa týka veľkosti vstupov a výstupov, rovnako jednoprechodovosť
- odolnosť voči metódam útokov použiteľných na rodinu SHA-2
- bezpečnosť dosahujúcu aspoň úroveň špecifikovanú v FIPS 180-2 avšak dosiahnutej s oveľa vyššiou efektivitou
- možnosť širokého nasadenia, aj za cenu nie 100% efektívnej realizácie vo všetkých prípadoch

Podmienky sú zaujímavé, uvidíme čo ponúkne budúcnosť.

Kapitola 6

Záver

Záverov vyplývajúcich z tejto práce je pre bežného človeka viac. Rada "zostať opatrnejší" a neveriť slepo niečomu, čo pôsobí bezpečne asi iba preto, lebo sa to vyhýba nášmu chápaniu. Absolútna bezpečnosť neexistuje a spoliehať sa, že materiál, ktorý som dnes podpísal "ultrabezpečným" digitálnym podpisom nebude zneužiteľný a modifikovateľný až do mojej smrti je síce pekná, no pravdivá asi len v prípade, že kosa sa skrýva niekde za najbližším rohom.

Inými slovami povedané: aktuálne KSHF sa verejne javia byť bezpečné dnes, no nakoľko ich bezpečnosť zväčša nie je dokázateľná a opiera sa len o zložitosť návrhu a komplikovanosť tvorby diferenčných schém, ich zlomenie je len otázkou času, ... možno padnú zajtra, možno už dávno padli, možno ...

V prípade, že používame digitálne podpisy, preštudovať si k nim príslušné materiály intenzívnejšie sledovať (resp. dožadovať sa informácií aspoň od vystavovateľa certifikátu pre digitálny podpis) dianie a stav bezpečnosti technológie, ktorú používame.

Kľúčové slová: informovať sa bezpečnosti technológií čo používate, pred použitím "n-krát ($n \in \mathbb{N}, n \geq 2$) merať a raz strihať"

Literatúra

[1] M. Stanek: Základy kryptológie, verzia. 0.16, 2004,
http://www.dcs.fmph.uniba.sk/ stanek/crypto/main2.pdf

[2] Y. Zheng, J. Pieprzyk, J. Seberry: HAVAL - A One-Way Hashing Algorithm with Variable Length of Output (Extended Abstract), 1993,
http://labs.calyptix.com/files/haval-paper.pdf

[3] B. Kaliski: The MD2 Message-Digest Algorithm, RFC 1320, Apríl 1992,
http://tools.ietf.org/html/rfc1319

[4] R. Rivest: The MD4 Message-Digest Algorithm, RFC 1320, Apríl 1992,
http://tools.ietf.org/html/rfc1320

[5] R. Rivest: The MD5 Message-Digest Algorithm, RFC 1321, Apríl 1992,
http://tools.ietf.org/html/rfc1321

[6] J. Daemen, C. Clapp: "Fast Hashing and Stream Encryption with PANAMA", FSE 1998,
http://homes.esat.kuleuven.be/ jlano/stream/papers/panamadc.pdf

[7] Secure Hash Standard, 17. Apríl 1995, FIPS PUB 180-1,
http://www.itl.nist.gov/fipspubs/fip180-1.htm (v. 1993 May 11), navštívené naposledy 13. Jún 2008

[8] US Secure Hash Algorithm 1 (SHA1), 2001,
http://tools.ietf.org/html/rfc3174

[9] US Secure Hash Algorithms (SHA and HMAC-SHA), 2006,
http://tools.ietf.org/html/rfc4634

[10] X. Wang, D. Feng, X. Lai, H. Yu: Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, 17. August 2004,
http://eprint.iacr.org/2004/199.pdf

[11] V. Klima: Tunnels in Hash Functions: MD5 Collisions Within a Minute (extended abstract), IACR ePrint archive Report 2006/105, 18. Máj. 2006,
http://eprint.iacr.org/2006/105.pdf
http://cryptography.hyperlink.cz/2006/tunnels.pdf

[12] V.Klíma: A New Concept of Hash Functions SNMAC Using a Special Block Cipher and NMAC/HMAC Constructions, IACR ePrint archive Report 2006/376, Október 2006,
http://eprint.iacr.org/2006/376
http://cryptography.hyperlink.cz/SNMAC/SNMAC_EN.pdf