VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF THE TELECOMMUNICATIONS

REALIZACE ÚTOKU NA HAŠOVACÍ FUNKCI SHA-1

REALIZATION OF ATTACK ON SHA-1 HASH FUNCTION

BAKALAŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — MICHAL DZIURA
AUTHOR

VEDOUCÍ PRÁCE — Ing.JAN MALÝ
SUPERVISOR

BRNO, 2010

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Michal Dziura ***ID:*** 111035
Ročník: 3 ***Akademický rok:*** 2009/2010

NÁZEV TÉMATU:

Realizace útoku na hašovací funkci SHA-1

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte princip hašovací funkce SHA-1 a implementujte ji v jazyce C. Zjistěte na základě dostupné literatury slabiny této funkce a implementujte realizaci vybraných útoků, vedoucích k odhalení zdrojového textu nebo založených na cíleném zfalšování haše.

DOPORUČENÁ LITERATURA:

[1] Schneier, B.: Applied cryptography: protocols, algorithms, and source code in C, Wiley-India, 2007, ISBN: 9788126513690
[2] Eastlake D., Jones P.: US Secure Hash Algorithm 1 (SHA1) - RFC 3174 [online], 2001, <http://www.ietf.org/rfc/rfc3174.txt?number=3174>

Termín zadání: 29.1.2010 ***Termín odevzdání:*** 2.6.2010

Vedoucí práce: Ing. Jan Malý

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

Abstrakt

Bakalářská práce se zabývá hašovacími funkcemi, funkcí SHA-1 a její implementací v jazyce Java. Opisuje slabiny této funkce a implementuje realizaci vybraného útoku, vedoucího k odhalení zdrojového textu nebo založeného na cíleném zfalšování haše. V této práci jsou popsané vlastnosti hašovacích funkcí, kontrukce, jejich využití. Poslední část se zabývá opisem útoku na hašovací funkci SHA-1 se složitostí 2^{69}.

Kľúčové slová

Hašovací funkce, SHA-1, SHA-0, kolize, kód, šifrování, útok, heslo, šifra, metoda, složitost.

Abstract

The bachelor thesis is concerned with hash functions, SHA-1 function and its implementation in Java. The thesis describes weaknesses of this function and implements the implementation of the selected attack leading to the revelation of source text or attack based on targeted falsifying hash. This work also describes characteristics of hash functions, constructions and their use. The last part deals with a description of the attack on the hash function SHA-1 with the complexity of the 2^{69}.

Keywords

Hash function, SHA-1, SHA-0, collision, code, cryptography, attack, password, signature, method, complexity.

Bibliografická citace práce

DZIURA, M. *Realizace útoku na hašovací funkci SHA-1*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 64 s. Vedoucí semestrální práce Ing. Ján Malý

Prohlášení

Prohlašuji, že svoji bakalářskou práci na téma Realizace útoku na hašovací funkci SHA-1 jsem vypracoval samostatně pod vedením vedoucího semestrálního projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedeného semestrálního projektu dále prohlašuji, že v souvislosti s vytvořením tohoto projektu jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 1. júna 2010 .....

podpis autora

Poděkování

Děkuji vedoucímu práce Ing. Janu Malému za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mého semestrálního projektu.

V Brně dne 1. júna 2010 .....

podpis autora

Obsah

ZOZNAM OBRÁZKOV **10**

ZOZNAM TABULIEK **11**

1 ÚVOD **12**

2.1 JEDNOSMERNÉ FUNKCIE 13

2.2 KRYPTOGRAFICKÁ HAŠOVACIA FUNKCIA 14

2.3 VSTUP A VÝSTUP 15

2.4 JEDNOSMERNOSŤ A BEZKOLÍZOVOSŤ 15

2.5 ORÁKULUM A NÁHODNÉ ORÁKULUM 16

2.6 HAŠOVACIE FUNKCIE AKO NÁHODNÉ ORÁKULUM 16

2.7 KOLÍZIA 16

2.8 ZNEUŽITIE KOLÍZIE 16

2.9 NARODENINOVÝ PARADOX 17

2.10 ODOLNOSŤ PROTI KOLÍZIÁM PRVÉHO RÁDU, BEZKOLÍZOVOSŤ PRVÉHO RÁDU 18

2.11 ODOLNOSŤ PROTI KOLÍZIÁM DRUHÉHO RÁDU, BEZKOLÍZOVOSŤ DRUHÉHO RÁDU 18

2.12 BEZKOLÍZOVOSŤ 18

3.1 ZAROVNANIE A DOPLNENIE O DĹŽKU SPRÁVY 20

3.2 DAMGARD-MERKLOVO ZOSILNENIE 21

3.3 DAMGARD-MERKLOV ITERATIVNÝ PRINCIP MODERNÝCH HAŠOVACÍCH FUNKCIÍ 22

3.4 KONŠTRUKCIA KOMPRESNEJ FUNKCIE 22

3.5. KRYPTOGRAFICKÉ VYUŽÍVANIE HAŠOVACÍCH FUNKCIÍ 23

Digitálny odtlačok dát (digital fingerprint) *23*

Overenie integrity dát (Modification detection codes, MDC) *24*

Overovanie autenticity správ (Message authentication codes, MAC) *24*

Porovnávanie obsahu databáz *24*

Ukladanie hesiel *24*

Pseudonáhodné funkcie *25*

Pseudonáhodné generátory *25*

3.6 HAŠOVACIE FUNKCIE AKO STAVEBNÉ KAMENE KRYPTOGRAFICKÝCH PROTOKOLOV 25

Certifikácia a digitálne podpisy. *26*

Autentifikácia. *26*

3.7 SÚČASNE HAŠOVACIE FUNKCIE 26
MD2 ..... *26*
MD4 ..... *26*
MD5 ..... *27*
SHA-1 ..... *27*
SHA-2 ..... *28*

4 HAŠOVACIA FUNKCIA SHA-1 29

4.1 ITERAČNÉ HAŠOVACIE FUNKCIE 29
4.2 ITERAČNÉ ROVNICE SHA-1 30
4.3 PRÍKLADY POUŽITIA SHA-1 33
4.4 IMPLEMENTÁCIA SHA-1 34

5 ZÁKLADNÉ ÚTOKY 37

5.2 BRUTE FORCE - HRUBÁ SILA 37
5.3 IMPLEMENTÁCIA HRUBEJ SILY 39
5.4 DICTIONARY ATTACK - SLOVNÍKOVÝ ÚTOK 39
5.5 RAINBOW ATTACK (RAINBOW ÚTOK) 40

6 ÚTOK NA SHA-1 41

6.2 LOKÁLNE KOLÍZIE 41
6.3 DIFERENCIÁLNE CESTY 42
6.4 HĽADANIE PORUCHOVÝCH VEKTOROV S NÍZKOU HAMMINGOVOU VÁHOU 42
6.5 TECHNIKY NA SKONŠTRUOVANIE DIFERENCIÁLNYCH CIEST 44
6.6 ODVODENÉ PODMIENKY 45
6.7 MODIFIKAČNÉ TECHNIKY SPRÁVY 45
6.8 VÝBER NAJLEPŠIEHO PORUCHOVÉHO VEKTORA 46
6.9 POUŽITIE BLÍZKYCH KOLÍZIÍ NA NÁJDENIE KOLÍZIÍ 46
6.10 ZLOŽITOSŤ ANALÝZ A ĎALŠÍCH TECHNÍK 47
6.11 VÝSTAVBA ŠPECIFICKEJ DIFERENCIÁLNEJ CESTY 47
6.12 ODVODENIE PODMIENKY PRE A_I A M_I 49

7 ZÁVER 50

A. PRÍLOHA: TABUĽKY 55

B. PRÍLOHA 64

OBSAH PRILOŽENÉHO CD: 64

Zoznam obrázkov

Obr. 1. Jednosmerná funkcia 13
Obr. 2. Jednosmerná funkcia s padacími vrátkami 14
Obr. 3. Hašovacia funkcia 15
Obr. 4. Kompresná funkcia 21
Obr. 5. SHA-1 30

Zoznam tabuliek

Tabuľka 1. Veľkosť výstupov hašovacích funkcií v bitoch. 15
Tabuľka 2. Základné výhody a nevýhody jednotlivých autentizačných techník. 38
Tabuľka 3. 6-kroková lokálna kolízia. 41
Tabuľka 4. Podmienky pre poruchové vektory v SHA-1 v *t* krokoch..... 42
Tabuľka 5. Hammingova váha. 44
Tabuľka 6. Kolízia SHA-1 zredukovaná na 58 krokov.. 48
Tabuľka 7. Poruchové vektory SHA-1. 55
Tabuľka 8. Hľadanie zložitosti pre blízke kolízie. 56
Tabuľka 9. Hľadanie zložitosti pre jednoblokovú kolíziu..... 57
Tabuľka 10. Pravidlá pre počítanie podmienok v kolách 2-4. 58
Tabuľka 11. Počítanie počtu podmienok pre 80 krokov blízkej kolízie..... 58
Tabuľka 12. Diferenciálna cesta pre 58 krokov SHA-1 kolízie. 59
Tabuľka 13. Diferenciálna cesta pre 80 krokov SHA-1 kolízie. 61
Tabuľka 14. Súbor postačujúcich podmienok. 63

1 Úvod

Kryptografické hašovacie funkcie sú dôležitou súčasťou modernej kryptológie. V dnešnom svete pri rýchlom rozvoji, rozšírením internetu a telekomunikačných služieb rastie dôležitosť hašovacích funkcií stále viac. Uplatňujú sa predovšetkým k overeniu integrity dát a zaistenia autenticity. Hlavnou úlohou hašovacích funkcií je vygenerovať z reťazca ľubovoľnej konečnej dĺžky reťazec stálej dĺžky.

Cieľom tejto bakalárskej práce je predstaviť hašovacie funkcie. Opísať ich hlavné vlastnosti, konštrukciu a ich použitie v dnešnom svete. Je tu predstavená a bližšie opísaná hašovacia funkcia SHA-1. Zároveň je tu spracovaná jej iteračná rovnica a implementácia hašovacej funkcie SHA-1 je uložená na príslušnom CD. Implementácia je urobená v jazyku Java v programe IDE. Hlavným cieľom práce je zistiť na základe dostupnej literatúry slabiny funkcie SHA-1. Opísať vybraný útok, ktorý by viedol k odhaleniu zdrojového textu alebo založený na cielenom sfalšovaní hašu a implementovať ho.

V druhej kapitole sú zhrnuté vlastnosti hašovacích funkcií, ako sú napríklad jednosmernosť funkcie, odolnosť proti kolíziám prvého rádu, odolnosť proti kolíziám druhého rádu. V tejto kapitole sú taktiež definované pojmy náhodné orákulum a narodeninový paradox.

Konštrukcie hašovacích funkcií sú opísané v tretej kapitole. V jej podkapitolách nájdeme rozobrané témy ako Merkle-Dagmarovo zosilnenie, zarovnanie hašovacej funkcie, použitie hašovacích funkcií a sú tu v krátkosti opísané vybrané hašovacie funkcie (MD4, MD5, SHA-1, SHA-2).

V štvrtej časti tejto bakalárskej práce je detailnejšie rozobraná hašovacia funkcia SHA-1. Nájdete tu popis algoritmu, príklady kde sa táto hašovacia funkcia využíva a na príslušnom CD je implementovaná táto funkcia SHA-1.

Piata kapitola rozoberá primitívne útoky, ktoré môžu byť použité na hľadanie kolízie. Nachádzajú sa tam tri útoky – útok hrubou silou, slovníkový útok a Rainbow útok. Útok hrubou silou je zrealizovaný a nájdete ho na CD.

V poslednej kapitole je zhrnutý útok na SHA-1 so zložitosťou 2^{69}. Je v nej rozobraný a vysvetlený tento útok.

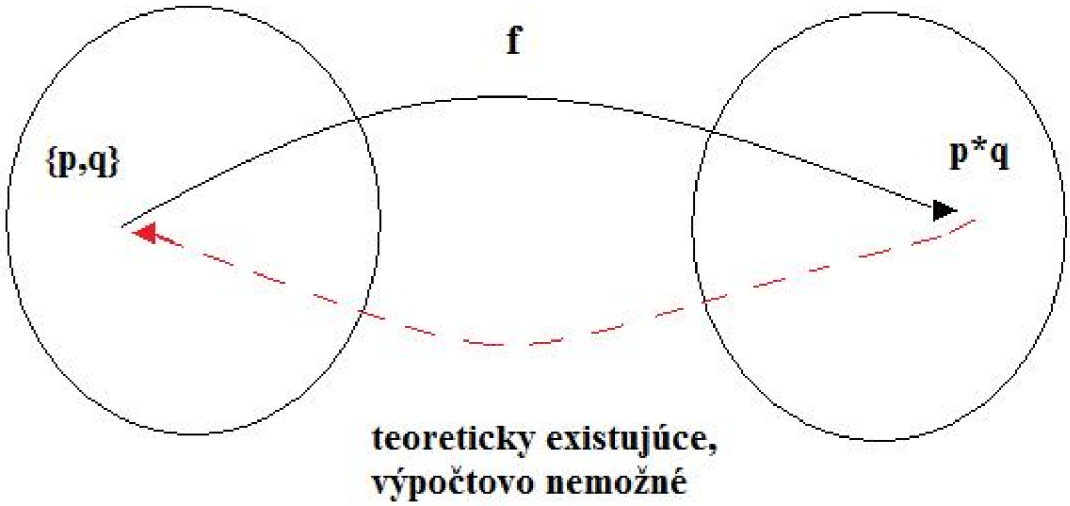
2 Hašovacie funkcie a ich vlastnosti

Hašovacie funkcie sú silným nástrojom súčasnej kryptológie. Sú jedným z hlavných kryptologických ideí počítačovej revolúcie a priniesli radu nových použití. V ich jadre sú pojmy jednosmernosť a bezkolízovosť.

2.1 Jednosmerné funkcie

Sú to funkcie f: X → Y, pre nich je jednoduché z akejkoľvek hodnoty $x \in$ X vypočítať $y = f(x)$, ale pre nejaký náhodný vybraný obraz $y \in f$(X) nie je možné (pre nás je to výpočtovo nemožné v rozumnom čase) odhaliť jej vzor $x \in$ X tak, aby $y = f(x)$. Pritom ale vieme, že takýto vzor existuje alebo ich je oveľa viac.

Príklad: Vynásobením dvoch veľkých prvočísiel, v aktuálnej dobe nepoznáme vhodnú rýchlu metódu ako tieto čísla separovať [12].



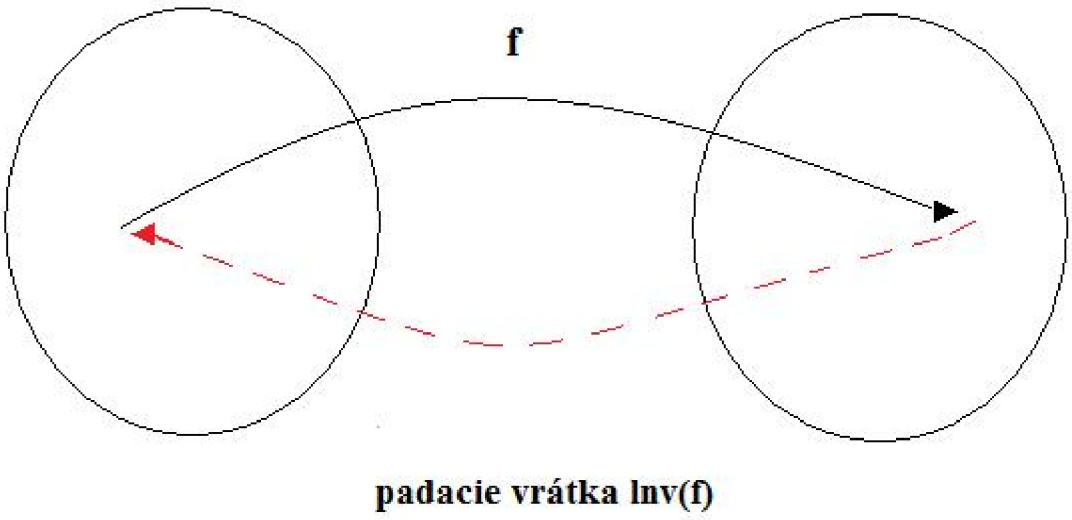
Obr. 1. Jednosmerná funkcia [12].

Ďalšou skupinou jednosmerných funkcií sú jednosmerné funkcie s padacími vrátkami. Niekedy sa im taktiež hovorí jednosmerné funkcie so zadnými vrátkami. Je na nich založená asymetrická kryptografia.

Jednosmerné funkcie so zadnými vrátkami spôsobili revolúciu v modernej kryptológií [18]. Sú to jednosmerné funkcie f, ktoré sa dajú invertovať za predpokladu, že poznáme

znalosť zadných vrátok. Táto podmienka platí aj pre jednotlivé funkčné hodnoty a nielen pre transformáciu ako celok. Je to napríklad kľúča od poštovnej schránky. Každý človek môže do nej vhadzovať listy, ale iba majiteľ schránky ju môže vyberať.

Zandé vrátka nazývame znalosť kľúča, ktorý dokáže funkciu invertovať, t.j. systematicky vedieť vypočítať vzorky od predložených obrazov [18].

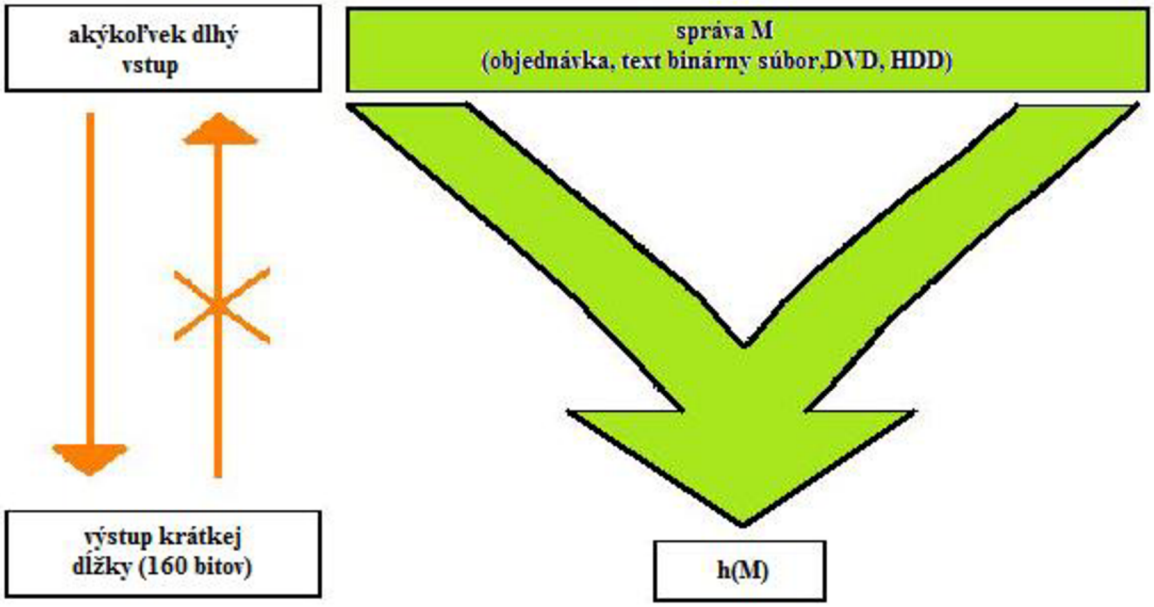


Obr. 2. Jednosmerná funkcia so zadnými vrátkami [18].

Hašovacie funkcie sú jednosmerné funkcie bez zadných vrátok [12].

2.2 Kryptografická hašovacia funkcia

Hašovacia funkcia bola pôvodne označovaná ako funkcia, ktorá priradzovala ľubovoľne veľkému vstupu krátku hašovaciu šifru pevnej dĺžky. Majme prirodzené čísla D, n a nech X je množina všetkých binárnych reťazcov dĺžky 0 až D (prázdny reťazec je platným vstupom a má dĺžku nula). Funkciu h: X $\rightarrow \{0,1\}^n$ pomenujeme hašovacou funkciou, ak je jednosmerná a bezkolizná. Každému binárnemu reťazcu z množiny X priradí binárny hašovací kód dĺžku n bitov [14].



Obr. 3. Hašovacia funkcia [12].

V praxi sa stretneme s $D = 2^{64} - 1$, $D = 2^{128} - 1$ a pod.. Vstupom hašovacej funkcie (z hľadiska dostupných moderných výpočtových aplikácií) ľubovoľný a prakticky neobmedzene dlhý binárny reťazec M [14]. Výstupom sa stáva hašovací kód $h(M)$, ktorý má pevnú, krátku a vopred stanovenú dĺžku n bitov. Pre dané M je ľahké vypočítať hašovací kód $h(M)$, ale naopak to nie je výpočtovo možné v praktickom čase (vyplýva to z vlastnosti jednocestnosti).

2.3 Vstup a výstup

Hašovacia funkcia h spracováva neobmedzene dlhé dáta M na krátky hašovací kód $h(M)$ [12].

Tabuľka 1. Veľkosť výstupov hašovacích funkcií v bitoch.

Hašovacia funkcia	MD5	SHA-1	SHA-256	SHA-512
Počet bitov	128	160	256	512

2.4 Jednosmernosť a bezkolízovosť

Hašovacia funkcia musí byť jednosmerná a bezkolizná [12]. Jednosmerná znamená, že z M sa dá vypočítať $h(M)$, ale obrátene je to výpočtovo nemožné v rozumnom čase.

Funkciu f: X → Y nazveme jednosmernou , ak z akejkoľvek hodnoty $x \in$ X je ľahké vypočítať $y = f(x)$, ale pre náhodnú vybranou hodnotu $y \in f$(X) nevieme (veľká výpočtová náročnosť) nájsť jej vzor $x \in$ X tak, aby $y = f(x)$ [12].

Funkciu f: X → Y nazveme bezkolíznou, ak je výpočtovo nemožné nájsť rôzne x, $x' \in$ X tak, že $f(x) = f(x')$ [12].

2.5 Orákulum a náhodné orákulum

Ľubovoľný stroj takto nazývame orákulum. Stroj na základe vstupu vie odpovedať nejakým výstupom. Má iba jedinú vlastnosť, a to takú, že rovnaký vstup odpovedá rovnakým výstupom. Náhodné orákulum je také orákulum, ktoré na nový vstup odpovedá námatkovým výberom výstupu z množiny existujúcich výstupov.

2.6 Hašovacie funkcie ako náhodné orákulum

Z hľadiska bezpečnosti by bolo dobré, aby sa hašovacia funkcia správala ako náhodné orákulum [12]. Odtiaľ odvodzujeme vlastnosti týkajúce sa bezpečnosti.

2.7 Kolízia

Kolízia alebo zrážka je situácia, ktorá nastane, ak máme 2 rôzne správy na vstupe a na výstupe sú vygenerované zhodné hašovacie hodnoty, kontrolné súčty alebo odtlačky prstov. Matematicky povedané, kolízny útok nájde 2 odlišné správy M_1 a M_2, tak aby pre výstupy platilo $h(M_1) = h(M_2)$. V klasickom kolíznom útoku nemá útočník kontrolu nad oboma vstupnými správami. Blízka kolízia sa skladá z dvoch správ, ktorých haš je takmer totožný a len pár bitov hašovacej hodnoty je odlišný.

2.8 Zneužitie kolízie

Existencia kolízií by sa dala pomerne ľahko zneužiť. Stačí uvážiť, že by sa mierne modifikácie skladali iba z rôznych zmien „netlačiteľných“ znakov (t.j. medzery by sa nahradzovali tvrdými medzerami, pridávali by sa a odoberali by sa prázdne riadky apod.) alebo z nahradzovaní podobných slov tak, aby text dával zmysel. Takto by sa dali dopredu získať 2 kolidujúce správy s rôznym, ale zmysluplným textom (napr. zmluvy). A vzhľadom

k tomu, že s hašovacími funkciami sa často používa v súvislosti s elektronickým podpisom (väčšinou nebýva elektronicky podpisovaná celá dlhá správa, ale iba jej *n*-bitový haš), mohol by „útočník“ podpísať pre seba menej výhodnú správu (t.j. jej haš). Neskôr by mohol kedykoľvek tvrdiť opak a za originál vydávať správu druhú, výhodnejšiu (jej haš je totožný).

2.9 Narodeninový paradox

Ak kolízie existujú, otázka bude znieť, aká veľká musí byť množina náhodných správ, aby v nej existovali s nie malou pravdepodobnosťou dve rôzne správy s rovnakým hašom, t.j. aby nastala kolízia. Narodeninový paradox rieši túto otázku, od neho sa odvodzuje bezpečnosť hašovacích funkcií. Rieši problém, koľko správ je potrebné zahašovať, aby s približne 50% pravdepodobnosťou nastala kolízia v množine vzniknutých hašovacích šifier. Pre 160-bitový hašovací kód očakávame $1/2 * 2^{160}$ správ, ale je to iba 2^{80} správ.

Majme množinu *M n* rôznych gúľ. Urobme výber *k* gúľ po jednej s vracaním späť do množiny *M*. Potom pravdepodobnosť, že sa vo výbere nájde aspoň jedna guľa viac ako jedenkrát, je [14]

$$P(n, k) = 1 - (n(n-1) \ldots (n-k+1)/ nk). \tag{2.1}$$

Pre

$$k = O(n^{1/2}), \tag{2.2}$$

a *n* idúce do nekonečna je $P(n, k)$ rovné približne $1 - e^{(-k2/2n)}$. Pre *n* veľké sa vo výbere

$$k = (2n \ln 2)^{1/2}, \tag{2.3}$$

prvkov z *M* s cca 50% pravdepodobnosťou nájdenou dva prvky zhodné [14].

Všeobecne pre hašovacie funkcie s *m* bitovým hašovacím kódom postačí zahašovať $2^{m/2}$ správ, aby sme našli kolíziu s približne 50% pravdepodobnosťou [14].

Príklad: Máme $P(265,23) = 0{,}507$, $P(365, 30) = 0{,}706$. Pre čísla $n = 365$ a $k = 23$ interpretujeme tvrdenie tak, že skupina vybraných 23 ľudí postačí k tomu, aby sa medzi nimi

s cca 50% pravdepodobnosťou objavila dvojica, sláviaca narodeniny rovnaký deň. U skupiny 30 ľudí je pravdepodobnosť už 70%. Tvrdenie sa zdá byť paradoxné, obvykle ho vnímame v zmysle „koľko ľudí je potrebné, aby sa k danému človeku našiel iný, sláviaci narodeniny v rovnaký deň“. V tejto podhodnocujúcej sa interpretácií hľadáme jedny konkrétne narodeniny, v žiadnom prípade, akékoľvek rovnaké narodeniny [14].

2.10 Odolnosť proti kolíziám prvého rádu, bezkolízovosť prvého rádu

Bezkolízovoosť („odolnosť proti kolíziám“) požaduje, aby nebolo možné vypočítať nájdenie ľubovoľných dvoch odlišných správ M a M' tak, že $h(M) = h(M')$. Ak táto situácia nastane, tak povieme, že sme našli kolíziu. Tejto bezkolízovosti hovoríme bezkolízovosť prvého rádu alebo bezkolízovosť.

Bezkolízovosť sa zásadným spôsobom využívaná v digitálnych podpisoch. Často veľmi dlhá správa sa priamo nepodpisuje (u MD5/SHA-1/SHA-256 až do dĺžky $D = 2^{64} - 1$ bitov), ale iba jej haš. Keďže bezkolízovosť zaručuje, že sa nemôže stať, aby sme našli dva dokumenty s rovnakým hašom, tak je nám to dovolené. Preto haš môžeme podpisovať.

2.11 Odolnosť proti kolíziám druhého rádu, bezkolízovosť druhého rádu

Z vlastnosti jednocestnosti vyplýva odolnosť hašovacej funkcie proti nájdeniu druhého vzoru [14].

Hašovacia funkcia h je odolná proti nájdeniu druhého vzoru, ak sa pre daný vzor x nedá výpočtovo nájsť druhý vzor $y \neq x$ tak, že $h(x) = h(y)$ [12].

Ak sa hašovacia funkcia bude f: $\{0,1\}^{D} \rightarrow \{0,1\}^{n}$ správať ako náhodné orákulum, bude komplikovanosť nájdenia vzoru k danej hašovacej šifre rovná 2^{n}.

Ak je možné nájsť kolízie menej zložito, hovoríme o prelomení hašovacej funkcie [14].

2.12 Bezkolízovosť

Ukážme si príklad na hašovacej funkcii MD5. Možných správ je veľa

$$(1 + 2^{1} + ... + 2^{D} = 2^{D+1} - 1), \tag{2.4}$$

kde $D = 2^{64} - 1$, a hašovacích šifier málo. U MD5 ich nájdeme iba 2^{128}. Musí preto existovať veľké množstvo správ, vedúce na rovnakú hašovaciu šifru – v priemere je to 2^{D-127}. Takže kolízií je veľké množstvo [14]. Nájdenie jedinej kolízie je nad naše výpočtové možnosti (aj s využitím narodeninového paradoxu). Z vlastností hašovacej funkcie vyplýva existencia ohromného počtu kolízií. A taktiež, nájdenie týchto kolízií je výpočtovo náročná a nemožná úloha.

3 Konštrukcie moderných hašovacích funkcií

U moderných hašovacích funkcií môže byť dĺžka správy veľmi veľká, napríklad $D = 2^{64} - 1$ bitov. Takúto správu musíme spracovávať po častiach, nie naraz. V komunikáciách je pochopiteľné, že správu dostávame po častiach. Správu nemôžeme z pamäťových dôvodov uložiť celú pre jedno rázové zhašovanie.

Takže je nutné hašovať správy po blokoch a sekvenčným spôsobom. Zo spracovania po blokoch plynie nutnosť zarovnania vstupnej správy na celistvý počet blokov pred hašovaním.

Predpokladajme, že máme hašovaciu funkciu o n-bitovom hašovacom kóde a správu M zarovnanú na m-bitové bloky $m_1, ..., m_N$ [12].

3.1 Zarovnanie a doplnenie o dĺžku správy

Zarovnanie má byť také, aby umožnilo jednoznačné odtrhnutie (jednoznačné oddelenie pôvodnej a doplnenej správy) [12]. Ak by to neplatilo, vznikali by jednoduché kolízie. Napríklad pri doplnení správy nulovými bitmi by nešlo rozoznať, koľko ich bolo doplnené, a či niektoré z nich nie sú platnými bity správy, ak by teda správa nulovými bitmi končila.

Napríklad doplnená správa

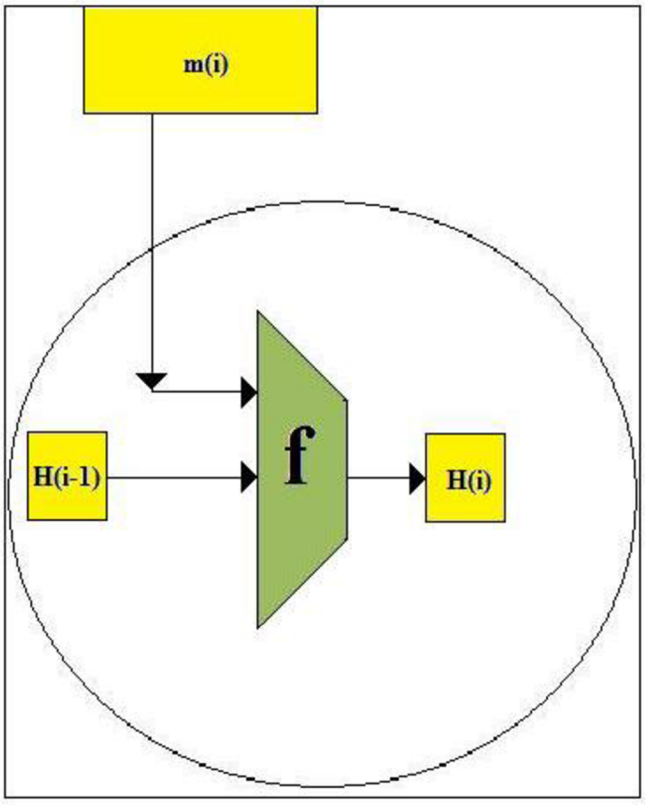
1011110011010100000000000000000000000000000
by bola doplnením správ
10111100110101
101111001101010
1011110011010100
10111100110101000
...
1011110011010100000000000000000000000000000
a všetky by preto viedli ku kolízií (mali by rovnaký hašovací kód) [12].

Zarovnávanie u hašovacích funkcií sa definuje ako doplnenie bitom 1 a až potom potrebným počtom bitov 0. To umožňuje jednoznačné odtrhnutie doplnku [12].

3.2 Damgard-Merklovo zosilnenie

Za počtom bitov je ešte potrebné pridať dĺžku pôvodnej správy. Ak by sme správu M doplňovali iba bitom 1 a nulovými bitmi do najbližšieho celistvého bloku a nepoužili doplnenie o dĺžku správy, potom by sme mohli nájsť druhý vzor M' tak, že $h(M') = h(M)$ so zložitosťou menšou než 2^n operácií. Toto doplnenie sa volá Damgard-Merklovo zosilnenie. Doplnenie pre blok dĺžky $m = 512$ bitov, ktoré je použité u hašovacích funkciách MD4, MD5, SHA-0, SHA-1, SHA-256.

Doplnenie sa urobí tak, že M je najprv doplnený bitom 1, potom čo najmenším počtom nulových bitov (0 - 447) tak, aby do kompletného násobku 512 bitov zostalo ešte 64 bitov. Týchto 64 bitov je nakoniec vyplnené 64-bitovým vyjadrením počtom bitov pôvodnej správy M. Dĺžka správy teda tiež vstupuje do hašovacieho procesu. Rezervácia 64 bitov na dĺžku správy umožňuje hašovať správy až do dĺžky $D = 2^{64} - 1$bitov [12].



Obr. 4. Kompresná funkcia (vykonáva mapovanie argumentu (vstupu) *m* ľubovoľnej bitovej dĺžky na hodnotu *h*(*m*) (výstup), ktorá má pevne určenú bitovú dĺžku).

3.3 Damgard-Merklov iterativný princip moderných hašovacích funkcií

Všetky moderné prakticky používané hašovacie funkcie používajú Damgard-Merkelov (*DM*) princíp iteratívnej hašovacej funkcie s využitím kompresnej funkcie [12].

Kompresná funkcia f spracováva súčasný blok správy m_i. Výsledkom je hodnota, ktorá označuje H_i a hovorí sa jej kontext. Hodnota kontextu nutne tvorí vstup do kompresnej funkcie v ďalšom kroku. Kompresná funkcia ma teda dva vstupy – kontext a aktuálny blok správy a jeden výstup – nový kontext [12].

Odtiaľ vznikla iterativná konštrukcia, popísaná vzorcom

$$H_i = f(H_{i-1}, m_i) \,, \tag{3.1}$$

$$H_0 = IV \,,$$

ktorá sa volá Damgard-Merklova konštrukcia.

Hašovanie prebieha po jednotlivých blokoch m_i v cyklu podľa i od 1 do N. Kompresná funkcia f v i-tém kroku (i = 1, ..., N) spracuje daný kontext H_{i-1} a blok správy m_i na nový kontext H_i. Šírka kontextu je často rovnaká ako šírka vstupného kódu [12].

Počiatočnej hodnote kontextu H_0 sa hovorí inicializačná hodnota (IV). Je to dodefinovaná konštanta daná v popise každej interatívnej hašovacej funkcie [12].

Názov kompresnej funkcie je vhodný, lebo funkcia f spracováva širší vstup (H_{i-1}, m_i) na oveľa kratšej H_i, teda blok správy m_i sa funkčne premietne do H_i, ale zároveň dochádza k strate informácie. Kontextom býva obvykle niekoľko 16-bitových alebo 32-bitových slov, u MD5 to sú štyri slová A, B, C, D (dohromady 128 bitov) [12].

Po zahašovaní posledného bloku m_N dostaneme kontext H_N. Z tohto posledného bloku potom zoberieme celú dĺžku alebo časť ako výsledný haš. Funkcia MD5 má šírku kontextu 128 bitov a výsledný haš tvorí 128 bitov kontextu H_N [12].

3.4 Konštrukcia kompresnej funkcie

Aby kompresná funkcia zabezpečila dokonalé premiešanie bitov správy a jednocestnosť, musí mať dostatočnú veľkosť. Pretože hašovacie funkcie sú jednocestné a musia sa chovať čo najviac ako náhodné orákulum. Je možnosť ich skonštruovať na základe

známych jednosmerných funkcií. Môžeme použiť vedomosti z oblasti blokových šifier. Kvalitná bloková šifra $E_k(x)$ sa pri pevnom kľúči k má správať ako náhodné orákulum. Ak poznáme akúkoľvek množinu vstupov – výstupov, t.j. otvorených – šifrovaných textov (x, y), nemôžeme odtiaľ určiť kľúč k. Vzhľadom ku kľúču je bloková šifra jednocestná. Pre každé x je funkcia $\rightarrow E_k(x)$ jednocestná. Odtiaľ plynie možnosť konštrukcie kompresnej funkcie: [12]

$$H_i = E_{mi}(H_{i-1}) \ , \tag{3.4}$$

kde E je kvalitná bloková šifra.

Predstavme si hašovanie krátkej správy, ktorá tvorí so zarovnaním jediný blok správy. Máme $H_1 = E_{m1}(IV)$. Vidíme, že z hodnoty hašovacieho kódu H_1 nemôžeme určiť m_1, takže máme zaručenú vlastnosť jednocestnosti.

U moderných funkcií sa používa ešte jedna operácia, Davies-Meyerova konštrukcia kompresnej funkcie, ktorá zosilňuje vlastnosť jednocestnosti pripočítaním vzoru pred výstupom [12]

$$H_i = f(H_{i-1}, m_i) = E_{mi}(H_{i-1}) \text{ xor } H_{i-1}. \tag{3.5}$$

Výstup je teda maskovaný vstupom, čo viac sťažuje prípadný spätný chod. Pretože blok správy m_i je poväčšine veľký (512 bitov). Kľúče blokových kódov tak dlhé nebývajú, použije sa blokový kód niekoľkokrát za sebou (iterácie), (iteračný) kľúč čerpá z bloku m_i [12].

3.5. Kryptografické využívanie hašovacích funkcií

Digitálny odtlačok dát (digital fingerprint)

Hašovacia funkcia vďaka svojej vlastnosti bezkolíznosti slúži ako funkcia, ktorá každému súboru či množine dát priradzuje jednoznačný identifikátor – digitálny odtlačok dát. Akokoľvek dlhé dáta sa dajú teda reprezentovať a identifikovať pomocou odtlačku, ktorý má len niekoľko stoviek bitov. O týchto odtlačkoch sa teda dá hovoriť podobne, ako napríklad o odtlačkov prstov u ľudí.

Overenie integrity dát (Modification detection codes, MDC)

Do tejto skupiny patria napríklad samoopravné kódy. To sú obvykle jednoduchšie funkcie, ich hlavným cieľom je ochrana voči chybám pri prenose informácie. Pri použití samoopravných kódov sa zo vstupnej informačnej postupnosti vytvárajú kódové slová, týmto procesom sa do prenosu vnáša nadbytočnosť. Redudancia sa využíva na zistenie chýb, ktoré vznikli v procese.

Overovanie autenticity správ (Message authentication codes, MAC)

Autenticita je daná vlastnosťou, že tajný kľúč, ktorý sa použil na výpočet autentifikačných údajov, pozná iba vysielač a príjemca správy. Integrita je zachovaná, pretože zmenou údajov sa zmení aj kontrolná suma. Autenticitu v prípade bezpečnostných protokolov vyšších vrstiev zväčša zaručujú certifikačné autority. Na nižších vrstvách je zabezpečovaná protokolmi na manažment kľúčov.

Porovnávanie obsahu databáz

Uveďme si príklad banky, ktorá všetky dáta zo všetkých účtov klientov ukladá do databázového systému, ktorý je on-line zálohovaný. Tým pádom sa vyskytuje na troch geograficky vzdialených miestach. Je nutné zistiť, či sa databázy zhodujú. Takže môžeme z troch databáz čítať jednotlivé záznamy a porovnávať ich. Znamenalo by to prenos celých databáz komunikačným systémom. To ale nemusí byť prakticky realizovateľné. Miesto toho stačí na troch miestach vypočítať odtlačky databáz . Ak sú hodnoty zhodné, tak databázy sa nelíšia ani o jeden bit.

Ukladanie hesiel

Ku kontrole hesiel v operačných systémoch sa využíva vlastnosť jendocestnosti. Uložia sa tu ich haše. Haše neodhalia heslá, z ktorých sú vypočítané, ale umožnia kontrolu ich správnosti pri prihlasovaní užívateľov do systému. Ak útočník zistí haš uloženého hesla užívateľa, nie je z tejto hodnoty schopný určiť užívateľovo heslo. Metoda solení sa používa

preto, aby sa vylúčil slovníkový útok. Útočník si predvypočíta haše pre často sa vyskytujúce slova a výrazy. Pri metóde solení sa pri uložení odtlačku hesla vygeneruje i náhodný reťazec (soľ), ktorý potom hašuje dohromady s vlastným heslom. Do databázy sa uloží dvojica. Je výpočtovo nemožné, aby si útočník predvypočítal slovník pre ľubovoľnú hodnotu soli.

Pseudonáhodné funkcie

Štandard PKCS#5 umožňuje využiť hašovaciu funkciu k tvorbe „náhodného" šifrovacieho kľúča z hesla pomocou pseudonáhodnej funkcie PRF ako kľúč = PRF (heslo) [14]. Predpis spočíva v hašovaní hesla a následne mnohonásobnom hašovanom výsledku. Počet hašovaní je dané konštantou c, jej hodnota sa doporučuje byť minimálne 1000, ale používa sa aj 2000. Výsledkom je krátky „náhodne vyzerajúci" kľúč DK, ktorý je možno využiť lepšie ako pôvodné heslo. Inak má pevnú dĺžku a taktiež z neho je možné využiť toľko bitov, koľko potrebujeme. Tu hrá kľúčovú rolu náhodnosť bitov v produkovaných hašov.

Pseudonáhodné generátory

Typické použitie hašovacích funkcií ako pseudonáhodných generátorov je v prípadoch, keď máme k dispozícií krátky reťazec (náhodných) dát (seed), typicky „krátky" 256-bitový náhodný šifrovací kľúč. Pritom potrebujeme z tohto vzorku získať pseudonáhodnou postupnosťou o veľkej dĺžke, napríklad 1 GB apod. Pseudonáhodné generátory, podobne ako v predchádzajúcom príklade, hrá kľúčovú úlohu náhodnosť bitov v produkovaných hašov. Napríklad štandard PKCS#1 v. 2.1 definuje pseudonáhodný generátor pomocou hašovacej funkcie H, s počiatočným nastavením seed takto: H (seed || 0x00000000), H (seed || 0x00000001), H (seed || 0x00000002)… [14].

3.6 Hašovacie funkcie ako stavebné kamene kryptografických protokolov

V praxi sa používajú hašovacie funkcie najmä v týchto prípadoch:

Certifikácia a digitálne podpisy.

Digitálny podpis je haš šifrovaný súkromným kľúčom pre podpisy. Overenie digitálneho podpisu je dešifrovanie podpisu pomocou verejného kľúča a porovnaním výsledku s hašom pôvodného dokumentu.

Certifikáty sú podpísané dokumenty. Zaisťujú súhlas verejných kľúčov s ďalšími informáciami (meno, adresa elektronickej pošty). Certifikačné autority vydávajú a podpisujú certifikáty.

Autentifikácia.

Je to služba, ktorá je používaná s cieľom nájsť pôvod informácie. Autentifikačná služba umožňuje verifikovať totožnosť užívateľa či systém, ktorý vytvoril danú informáciu. Autentifikácia je zaisťovaná pomocou digitálnych podpisov.

3.7 Súčasne hašovacie funkcie

MD2

Message Digest Algorithm 2 (MD2) je kryptografický haš funkcie, ktorý vyvinul Ronald Rivest v roku 1989. Algoritmus je optimalizovaný pre 8-bitové počítače. MD2 je špecifikovaný v RFC 1319. Hoci iné algoritmy boli navrhnuté, pretože ako MD4, MD5 a SHA, dokonca od roku 2009 aj MD2 zostane v použití infraštruktúry verejných kľúčov ako súčasť potvrdenia generované MD2 a RSA. 128-bitová haš hodnota každej správy tvorí výplň násobku dĺžky bloku na počítači (128 bitov alebo 16 bytov) a pripočítaním 16-bytov kontrolného súčtu. Pre vlastný výpočet, 48-bytov pomocného bloku a 256-bytov S-tabuľky vytvorenej nepriamo z číslic nepatrnej časti pí sú použité. Algoritmus preteká slučkou, kde permutácia každého bytu v pomocnom bloku 18-krát na všetkých 16 vstupných bajtoch je spracovaný. Potom, čo všetky bloky správy boli spracované, prvý čiastkový blok pomocných blokov sa stáva hašom hodnoty správy.

MD4

MD4 je algoritmus Message Digest (štvrté v rade) navrhol profesor Ronald Rivest z MIT v roku 1990. Realizuje kryptografický haš funkcie pre použitie kontroly integrity správ. Digest dĺžka je 128 bitov. Algoritmus ovplyvnil neskoršie návrhy, ako MD5, SHA a

RIPEMD algoritmy. MD4 je tiež používaná na výpočet NT- haš hesla v systéme Microsoft Windows NT, XP a Vista. MD4 sa skladá z týchto 48 operácií, zoskupených v troch kolách 16 operácií. *F* je nelineárna funkcia, jedna funkcia sa používa v každom kole. *Mi* označuje 32-bitový blok vstupu správy a *Ki* znamená 32-bitové konštanty, rôzne pre každú operáciu. Nedostatky v MD4 boli preukázané Denom Boerom a Bosselaersom v článku publikovanom v roku 1991. Prvá kolízny útok bol nájdený Hansom Dobbertinom v roku 1996. V auguste 2004, Wang et al. našiel veľmi efektívne zrážky útoku, vedľa útokov na hašovacej funkcie našiel neskôr vzory v rodine MD4/MD5/SHA/RIPEMD.

MD5

V kryptografii, MD5 (Message-Digest algorithm 5) je široko používaná kryptografická haš funkcia so 128-bitovou hodnotu hašu. Ako internetový štandard (RFC 1321), MD5 bola zameraná na širokú škálu bezpečnostných aplikácií, a je tiež bežne používaná na kontrolu integrity súborov. Avšak, to sa ukázalo, že MD5 nie je odolná proti nárazom, ako taká, MD5 nie je vhodná pre aplikácie, ako sú SSL certifikáty, alebo digitálne podpisy, ktoré sa opierajú o túto vlastnosť. MD5 haš je zvyčajne vyjadrená ako 32-miestne hexadecimálne číslo.
MD5 navrhol Ron Rivest v roku 1991, aby nahradil predchádzajúcu hašovaciu funkciu MD4. V roku 1996 bola nájdená chyba s dizajnom MD5. I keď to nebolo jednoznačná slabosť, kryptografici začali odporúčať používanie iných algoritmov, ako je napríklad SHA-1. V roku 2004 boli objavené ďalšie závažné nedostatky, pričom ďalšie použitie algoritmu z bezpečnostných dôvodov bolo sporné. V roku 2007 skupina vedcov vrátane Arjen Lenstra popísala, ako vytvoriť pár súborov, ktoré zdieľajú rovnaký MD5 kontrolný súčet. US-CERT of the US Department of Homeland Security povedal MD5 „by mala byť považovaná za kryptograficky rozbitá a nevhodná pre ďalšie použitie“.

SHA-1

Tento štandard špecifikuje Secure Hash Algorithm, SHA-1, na výpočet skrátený reprezentáciou správy alebo dátovým súborom. Keď sa správa o ľubovoľnej dĺžke $i < 264$ bitov, je vstup, SHA-1 produkuje 160-bitový výstup tzv. Message Digest (hašovacia hodnota). Message Digest potom môže byť vstup do Digital Signature Algorithm (DSA), ktorý vytvára alebo overuje podpis správy. Podpísanie message digest skôr ako správa často

zlepšuje účinnosť procesu, pretože message digest je zvyčajne oveľa menšia ako veľkosť správy. V rovnakom algoritme musia byť použité overovateľom z digitálneho podpisu, ktorý použil tvorca digitálneho podpisu. SHA-1 je technickou revíziou SHA (FIPS 180). Táto revízia zvyšuje zabezpečenie poskytovanou touto normou. SHA-1 je založená na princípoch podobných tým, ktoré používajú profesor Ronald L. Rivest z MIT pri navrhovaní MD4 Message Digest Algorithm, a je úzko modelovaná po tomto algoritme.

SHA-2

SHA2 je označenie 4 funkcií: SHA-256, SHA-384, SHA-512, SHA-224. SHA-256 je funkcia pracujúca s 32-bitovými slovami. SHA-224 je odvodená varianta s kratším hašom a inými inicializačnými konštantami a vektormi. SHA-512 pracuje so 64-bitovými slovami. SHA-384 je jej odvodená varianta s kratším hašom a inými inicializačnými vektormi.

4 HAŠOVACIA FUNKCIA SHA-1

Funkcia SHA-1 (Secure Hash Algorithm) je vytvorená z funkcie MD4 a navrhol ju Americký úrad (National Institute of Standards and Technology – NIST) pre použitie vo vládnych inštitúciách. Pôvodný návrh funkcie SHA bol v roku 1995 modifikovaný na SHA-1 [7].

Zmena v algoritme SHA-1 je veľmi malá. V hlavnej slučke algoritmu bola pridaná jedno-bitová rotácia. Z pohľadu bezpečnosti však táto zmena má veľký význam. Táto zmena zvýšila odolnosť pôvodnej SHA voči kolíziám. Pôvodná SHA teda nie je bezpečná [7].

Algoritmus SHA–1 umožňuje spracovať originálnu správu s maximálnou dĺžkou menšou než 2^{64} bitov a predstavuje výstupný hašovací kód s dĺžkou 160 bitov. Vstupná správa je spracovaná po blokoch s veľkosťou 512 bitov [8].

4.1 Iteračné hašovacie funkcie

SHA-1 patrí medzi tzv. iteračné hašovacie funkcie, ktorých základný model je uvedený na obr.4.1.

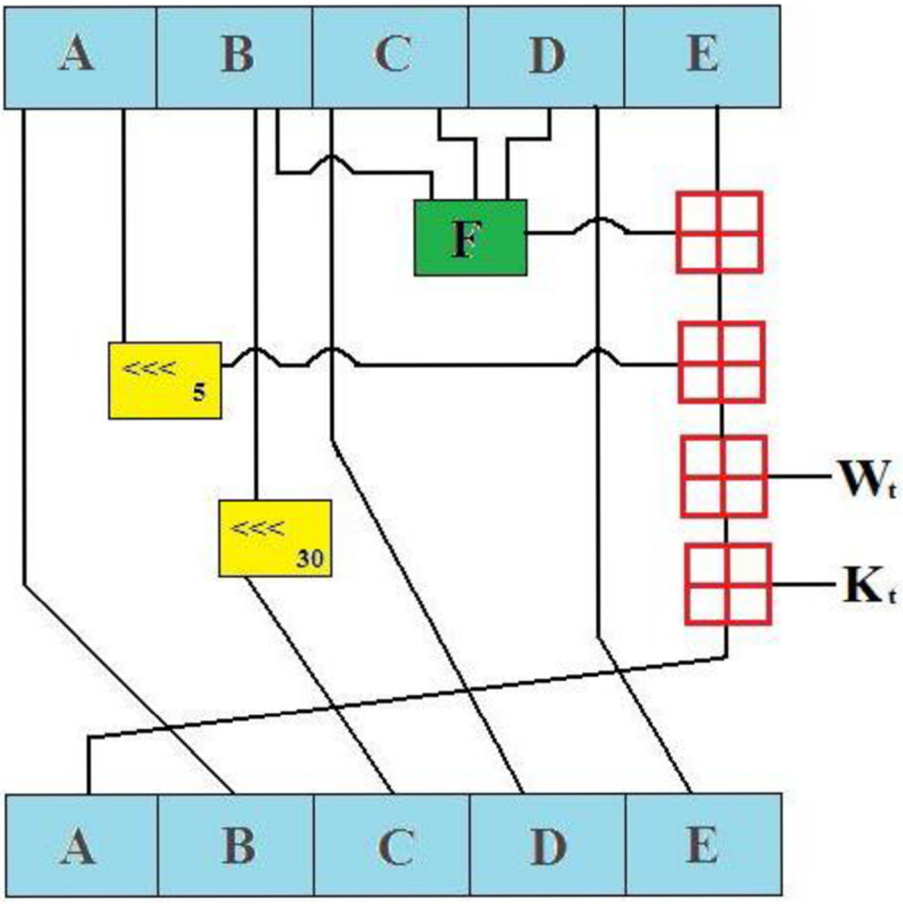
Vstup hašovacej funkcie x má prakticky ľubovoľnú dĺžku. Vstupná správa x je v procese predspracovania (preprocessing) rozdelená na r-bitové bloky x_i. Toto predspracovanie zahrňuje aj pripojenie dodatočných bitov (padding) tak, aby celková dĺžka správy bola presným násobkom dĺžky bloku . V tomto kroku sa tiež doplňuje informácia o dĺžke pôvodnej správy x. Každý r-bitový blok x_i je privedený v iteračnom procese na vstup iteračnej hašovacej funkcie f, ktorá vypočíta v každom iteračnom kroku výstupnú n-bitovú hodnotu H_i (tzv. zreťazená premenná). Hodnota H_i je funkciou predchádzajúcej výstupnej hodnoty H_{i-1} a aktuálnej vstupnej hodnoty x_i. Iteračné spracovanie správy $x = x_1x_2...x_t$ možné zapísať v tvare [7]

$$H_0 = IV,$$

$$H_i = f(H_{i-1}, x_i),\ 1 \leq i \leq t\ , \tag{4.1}$$

$$h(x) = g(H_t)\ ,$$

pričom *IV* je preddefinovaná štartovacia hodnota. Voliteľná výstupná transformácia môže byť využitá na mapovanie *n*-bitovej premennej H_t na *m*-bitovú výstupnú hodnotu. Veľmi často je $g(H_t) = H_t$ [7].



Obr. 5. SHA-1. Jedna iterácia v rámci SHA-1 kompresnej funkcie: *A*, *B*, *C*, *D*, *E* sú 32 bitové slová, *F* je nelineárna funkcia, ktorá je mení, <<<*n* označuje bitovú rotáciu doľava o *n* miest, jednotlivé operácie sa líšia *n*, *W*t je rozšírená správa slova v kroku *t*, *K*t je konštanta kroku *t*, ⊞ označuje pridanie modulo 2^{32}.

4.2 Iteračné rovnice SHA-1

ALGORITMUS**:** Secure Hash Algorithm – SHA-1

VSTUP: bitový reťazec *x* dĺžky $0 \leq b < 2^{64}$

VÝSTUP: 160-bitová hašovacia hodnota správy *x*

1. Definícia konštánt

Definujme päť 32-bitových *IV* (prvé štyri sú z algoritmu MD4)

h_1 = 0*x*67452301

h_2 = 0xefcdab89

h_3 = 0x98badcfe

h_4 = 0x10325476

h_5 = 0xc3d2e1f0

a 32-bitové iteračné aditívne konštanty [7]

y_1 = 0x5a827999

y_2 = 0x6ed9eba1

y_3 = 0x8f1BBcdc

y_4 = 0xca62c1d6

2. Predspracovanie

Doplňme x tak, že bitová dĺžka je násobkom 512 podľa nasledujúceho pravidla. Najskôr doplňme bit 1 a za ním r - 1 (≥ 0) bitov 0 pre najmenšiu možnú hodnotu tak, aby celková bitová dĺžka správy s doplnenými bitmi bola celočíselným násobkom 512 zmenšeným o 64. Nakoniec doplňme 64 bitovú hodnotu, ktorá vyjadruje hodnotu b mod 2^{64}. Túto 64 bitovú hodnotu doplňme ako dve 32-bitové hodnoty, pričom menej významové slovo je doplnené ako prvé. Nech m je počet 512-bitových blokov vo výslednom bitovom reťazci, t.j. platí

$$b + r + 64 = 512m = 32 \cdot 16m \,, \tag{4.2}$$

Upravený vstup sa tak skladá z 16m 32-bitových slov $x_0, x_1, ..., x_{16m-1}$. Inicializujme zreťazené premenné [7]

$$(H_1, H_2, H_3, H_4, H_5) \leftarrow (h_1, h_2, h_3, h_4, h_5) \,,$$

3. Iteračná slučka

Pre i= 0,1... m-1 skopírujme i-tý blok šestnástich 32-bitových slov do dočasných premenných $X[j] \leftarrow x_{16i+j}$, $0 \leq j \leq 15$ a tieto premenné spracujme v nasledujúcich štyroch 20 krokových iteráciách:

Expandujme 16 slovný blok na 80 slovný blok, označme $X_j = X[j]$ a vykonajme

$$X_j \leftarrow ((X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) <<<1),$$
$$j = 16,17 \ldots 79$$

Inicializujme pracovné premenné A, B, C, D, E

$$(A, B, C, D, E) \leftarrow (H_1, H_2, H_3, H_4, H_5)$$

a vykonajme nasledujúce iterácie

Iterácia 1 j = 0,1... 19:

$$t \leftarrow ((A<<<5) + f(B, C, D) + E + X_j + y_1)$$
$$(A, B, C, D, E) \leftarrow (t, A, B<<<30, C, D)$$

Iterácia 2 j = 20,21... 39:

$$t \leftarrow ((A<<<5) + h(B,C, D) + E + X_j + y_2)$$
$$(A, B, C, D, E) \leftarrow (t, A, B<<<30, C, D)$$

Iterácia 3 j = 40,41...59:

$$t \leftarrow ((A<<<5) + g(B, C, D) + E + X_j + y_3)$$
$$(A, B, C, D, E) \leftarrow (t, A, B<<<30, C, D)$$

Iterácia 4 $j = 60,61...79$:

$$t \leftarrow ((A<<<5) + h(B, C, D) + E + X_j + y_4)$$
$$(A, B, C, D, E) \leftarrow (t, A, B<<<30, C, D)$$

Zmena zreťazených premenných

$$(H_1, H_2, H_3, H_4, H_5) \leftarrow (H_1 + A, H_2 + B, H_3 + C, H_4 + D, H_5 + E)$$

Označenie $X <<< s$ označuje cyklickú rotáciu slova W o s bitov doľava [7].

4. Výstup

Výstupná 160 bitová hašovaná hodnota je (operátor $X \| Y$ označuje spojenie dvoch slov)[7]:

$$H_1 \| H_2 \| H_3 \| H_4 \| H_5.$$

4.3 Príklady použitia SHA-1

Digitálny podpis

Existuje niekoľko prakticky využiteľných schém digitálneho podpisu. V prípade ich praktického využitia sa namiesto podpisu pôvodnej digitálnej správy x, ktorá môže mať prakticky neobmedzenú dĺžku podpisuje podstatne kratšia hašovacia hodnota (tzv. Message Digest), čo v prípade funkcia SHA-1 budeme zapisovať v tvare

$$\text{Message Digest} = \text{SHA}(x)\ . \tag{4.3}$$

Generátor pseudonáhodných čísel

Hašovacie funkcie sú vzhľadom na ich jednocestnosť obľúbeným stavebným blokom pri vytváraní kryptograficky bezpečných generátorov pseudonáhodných čísel. Jednocestnosť zabezpečí nemožnosť spustiť spätný chod generátora aj pri zistení interného stavu generátora.

Typickým príkladom je PRNG použitý v štandarde DSS (Digital Signature Standard). DSA PRNG realizuje všetky aritmetické operácie modulo 2^N pričom $160 \leq N \leq 512$ a je ho možné opísať takto:

1.) PRNG udržiava stavovú premennú X_i.

2.) PRNG môže využiť voliteľný vstup W_i. V prípade, že tento vstup neexistuje, je použitá hodnota $W_i = 0$.

3.) PRNG generuje výstup s využitím vzťahov [7]

$$\text{vystup}[i] = \text{SHA}(W_i + X_i \bmod 2^N)\,, \tag{4.4}$$

$$X_{i+1} = X_i + \text{vystup}[i] + 1 (\bmod 2^N). \tag{4.5}$$

4.4 Implementácia SHA-1

Táto implementácia je urobená v jazyku Java v programe IDE (jazyk bol zvolený dohode s vyučujúcim, keďže jazyk Java a jazyk C sú si podobné). Vstupné slovo sa zadáva cez Properties → Run → Arguments. Program bol úspešne vyskúšaný a jeho výsledky boli porovnané s programami na internetových stránkach uvedené v literatúre [26] a [29]. V tomto programe je umožnené hašovať slová a obrázky.

Deklarácia triedy `public class Main`:

```
public class Main {
public static byte[] hash(byte[] array) {}
private static void processMi(byte[] block) {}
private static int ft(int iRound) {}
private static int Kt(int iRound) {}
public static void main(String[] args) throws
FileNotFoundException, IOException{}
public static byte[] getBytesFromFile(String name) throws
FileNotFoundException, IOException{}
private static int rotate(int iVal, int iRot) {}
public static String printIt(byte[] print) {}
}
```

```
public static byte[] hash(byte[] array)
```

Na začiatku v metóde `hash` potrebujeme vytvoriť pole správnej dĺžky, tak aby bitová dĺžka tohto poľa bola celočíselným násobkom 512 a nakoniec zmenšíme o 64. Urobíme to tak, že zväčšíme dĺžku poľa `array` tak, aby bola deliteľná 64-`mi` (64 bytov = 512 bitov). Ak máme pole `array` veľkosť 491 bitov, potrebujeme pole zarovnať na hodnotu 64 bytov. K dĺžke poľa 491 bitov pridáme 64 bytov. Následne odčítame od tejto hodnoty zvyšok po delení 64-mi (v tomto prípade je to 491 % 64 = 21), takže dostaneme pole 512 bitov + 64 bytov – 21 bytov. Takto sme vytvorili pole `paddedMsg`, ktoré má dĺžku poľa deliteľnú 64 bytmi, ale obsahuje iba prázdne hodnoty. Potrebujeme pole naplniť a naplníme ho hodnotami, ktoré obsahuje pole `array`. `PaddedMsg` má veľkosť 512 bitov (zarovnané na 64 bytov), ale pole `array` má iba 491 bitov. Hodnoty poľa `paddedMsg` (`paddedMsg[491]` – `paddedMsg[511]`) sú prázdne. Teraz nastavíme hodnotu `paddedMsg[491]`, čím vytvoríme značku konca správy. A následne nastavíme hodnoty posledných 8 bytov správy (posledných 21 bytov je prázdnych).

Na úplný koniec správy `paddedMsg[511]` zapíšeme dĺžku pôvodnej správy v bitoch a postupne nastavíme hodnoty `paddedMsg[510]` – `paddedMsg[504]` na hodnoty `1Len` (v `1Len` je dĺžka pôvodnej správy v bitoch) posunuté stále o 8 bitov doprava. Po nastavení hodnôt definujeme päť 32 bitových zásobníkov (H_0, H_1, H_2, H_3, H_4). Potom si vytvoríme 64 blokov (`block[0]` - `block[63]`), každý blok má veľkosť 1 byte. Postupne ukladáme do blokov hodnoty poľa `paddedMsg`(`block[1] = paddedMsg[1]` - `block[63] = paddedMsg[63]`). Vo chvíli, keď sa v premennej `i` objaví hodnota 63 tak prebehne metóda `processMi(block)`. Teraz máme dokončenú celú rozšírenú správu a posledným krokom v tejto metóde je posun bytov doprava kvôli pretypovaniu bytov.

```
private static void processMi(byte[] block)
```

Na začiatku tejto metódy sa do poľa `words` uložia hodnoty poľa `int`, to znamená 1 slovo, čo je 32 bitov. Prvých 16 slov (hodnôt) zachováva pôvodný obsah 512-bitového bloku (512bitov = 16 · 32 bitov). Ak premenná `i` nadobúda hodnoty menšie ako 16, tak pole `words[i]` naplníme hodnotami viď program, ale aby sme dodržali poradie bitov musíme naplniť najprv najnižšie bity. Ak hodnota premennej `i` je 16 - 79 tak sa pole `words[i]` naplní pomocou rotácií viď program. Následne do vytvorenej premennej `Temp` uložíme hodnotu

(`rotate(A, 5) + ft(i) + E + words[i] + Kt(i)`) a ďalšími operáciami sa nám zmenia premenné `A`, `B`, `C`, `D`, `E`. Napr. do premennej `A` sa uloží hodnota premennej `Temp`. Tieto kroky v tejto metóde prebehnú 80 krát. Po skončení tejto metódy sú všetky slová naplnené hodnotami.

```
private static int ft(int iRound)
```

Po dvadsiatich kolách sa mení výpočet, v ktorom sa nachádzajú premenné `B`, `C`, `D`.

```
private static int Kt(int iRound)
```

Po dvadsiatich kolách vracia hodnotu.

```
public static void main(String[] args)
```

Vytvorenie a vypísanie haša. Ak chceme zahašovať reťazec je potrebné pred reťazec napísať –t. Ak by sme chceli zahašovať nejaký obrázok tak potrebujeme pred zadať –b a nasledovne napísať cestu k súboru a názov súboru. Vstupné slovo alebo obrázok sa zadáva cez Properties → Run → Arguments.

```
private static int rotate(int iVal, int iRot)
```

Metóda implementuje cyklický posun doľava.

```
private static String printIt(byte[] print)
```

Metóda, ktorá vráti reťazec k vytlačeniu, zostaví ho z poľa bytu.

5 Základné útoky

V nižšie opísaných útokov sa budeme zameriavať na hľadanie kolízie.

5.2 Brute force - hrubá sila

Medzi najjednoduchší útok patrí hrubá sila (brute force). Hlavnou myšlienkou tohto útoku je vyskúšanie všetkých možností. Tento útok sa používa na hádanie, snaží sa teda vyskúšať všetky heslá aké existujú. Medzi znaky, ktoré sa využívajú patria malé písmená: a, b, c, d, e, ... u, x, y, z. Ďalej sú to čísla: 0, 1, 2, 3,..., 8, 9. Nesmieme zabudnúť tak tiež na veľké písmená a je možné používať aj interpunkčné znamienka, ktoré sa ale v heslách veľmi nepoužívajú. Použitím tejto metódy máme zaručené, že bude odhalené každé heslo. Hlavnou vlastnosťou tohto útoku je ale čas, ktorý je na toto odhalenie hesiel potrebný. Nájdenie hesla môže trvať aj niekoľko desiatok rokov. Pomocou obmedzenia hľadacieho priestoru môžeme tento čas skrátiť.

Vyrátajme si možnosti v jednom znaku. V abecede nájdeme 25 písmen. Musíme ale rozlišovať veľké a malé písmená. Ďalej pripočítame čísla, znaky s diakritikou a interpunkčné znamienka. Po spočítaní týchto znakov nám výsledok bude dávať hodnotu zhruba 100 možností. Keďže sa interpunkčné znamienka a znaky s diakritikou takmer v heslách nepoužívajú, z toho vyplýva, že možností je menej. Teda v jednom znaku sa môže nachádzať 100 možností. Keby heslo malo dva znaky, tak potrebujeme 100 × 100 pokusov, čo je 10 000 pokusov. A samozrejme by sme nemali zabudnúť na pokusy s jedným znakom. Tie by sme k tomuto výsledku mali prirátať, pretože nevieme aké dlhé heslo je. Zoberme si heslo o dĺžke 7 znakov. Dostaneme, že by sme potrebovali 100 000 000 000 000 pokusov. Musíme ešte pripočítať ďalšie možnosti na 6 znakov, 5, 4, ... Keď to spočítame, tak dostaneme 101 010 101 010 100 možností.

V brute force rozlišujeme dva stavy online a offline. Online znamená, že program sa niekam prihlasuje, napr. cez službu FTP a teda sa daný FTP server pripojí a vyskúša sa prihlásiť, pošle login a ako heslo pošle napríklad nejaké slovo zo slovníka. Pri stave offline ide o to, že zašifrované heslo je nám známe a už sa nepokúšame nikam prihlásiť. Celkové hádanie prebehne lokálne, čo je rádovo rýchlejšie. V metóde hrubej sily sa používa zvyčajne offline.

Heslá slúžia k overeniu indentity užívateľa. Tokeny sú rôzne identifikačné karty, preukazy a podobne predmety, slúžia k overeniu identity alebo identifikáciu užívateľa na

základe niečoho, čo má. Biometrické systémy rozhodujú o užívateľovej identinte na základe niečoho, čím je. Sú to behaviorálne (napríklad hlas, dynamika podpisu, dynamika stlačenia kláves) a fyziologické (odtlačok prstu, geometrie ruky, sietnice,...) vlastnosti.[1]

Tabuľka 2. Základné výhody a nevýhody jednotlivých autentizačných techník

	Vlastnosti	
Heslá	výhody	Ľahko použiteľné, užívatelia sú na nich zvyknutí, žiaden špeciálny hardware
	nevýhody	Silné heslo vs. ľudská pamäť, dajú sa odpozorovať
Tokeny	výhody	Nedajú sa odpozorovať a ani zabudnúť v zmyslu hesiel
	nevýhody	Dajú sa stratiť, odcudziť, potrebný dodatočný hardware
Biometriky	výhody	Sú viazané užívateľom, nedajú sa odcudziť, stratiť, odpozorovať
	nevýhody	Nízka dôvera užívateľov, väčšinou potrebný špeciálny hardware

Bezpečnosť hesla sa dá určiť 3 základnými požiadavkami. Sú to jeho dĺžka, zložitosť a doba, po ktorej musí byť heslo zmenené. Požiadavka dĺžky je jasná, čím dlhšie heslo, tým väčší priestor, ktorý musí byť prehľadaný pri útoku. Pre obranu pred útokom je potrebná dostatočná zložitosť hesla. Ďalej je to vhodná doba zmeny hesla, ktorú určuje administrátor. Existuje pár základných pravidiel pre zostavenie silného hesla (tieto pravidlá sa často rozchádzajú) [1]:

1. Heslo má mať aspoň 8 znakov (v závislosti na použitých znakoch).
2. Heslo nesmie obsahovať zmysluplné slovo.
3. Heslo nesmie byť spojené s informáciami o užívateľovi.
4. Heslo obsahuje malé i veľké písmená.
5. Heslo obsahuje číslice a špeciálne znaky (napríklad ?, !, @, ...)

Ako už bolo povedané, útok hrubou silou spočíva v testovaní všetkých možných slov nad zvolenou abecedou s prípadnými obmedzujúcimi podmienkami. Ak máme heslo o n znakov a použitá abeceda zahrňuje m symbolov, potom možné slová dĺžky n je m^n. [1]

5.3 Implementácia hrubej sily

```
public class Collision
```

Pre zastavenie programu slúžia klávesy ctrl + c, a zároveň po tomto prerušení sa objaví výpis, koľko hodnôt sa stihlo počas trvania programu vygenerovať a otestovať. V programe sa vytvorí `map`, do ktorej sa ukladajú náhodne vygenerované hodnoty a ich haše. Tieto hodnoty sa porovnávajú a tým sa zisťuje, či nenastane kolízia. Ak nastane zhoda dvoch hašových hodnôt, program vypíše: Našiel som dvojicu s rovnakým hašom. Je to... a ... (vypíše sa nájdená dvojica).

5.4 Dictionary attack - Slovníkový útok

Slovníkový útok (dictionary attack) sa často používa na zlomenie hesla. Táto metóda je modifikáciou útoku hrubej sily. Základom pre realizáciu tohto útoku je to, že heslo ktoré si človek vybral, nie je niečo nezmyselné. Teda očakáva sa heslo, ktoré dáva zmysel. Ľudia často ako heslo využívajú rodné čísla, trvalé bydlisko, alebo aj svoje meno. Názov tohto útoku je odvodený od slovníka, ktorý sa tu používa. V tomto slovníku sa nachádzajú slová, ktoré majú význam, ale aj slová ktoré nie sú spisovné. Napríklad ako heslo sa dá použiť poiuLKJH. Sú to prvé dva riadky klávesnice z pravej strany. Aj v tomto útoku rozlišujeme, či ide o stav online alebo offline.

Slovníkový útok skúša slová, ktoré sa nachádzajú v slovníku. Predpokladá sa, že slovníkový útok bude úspešný, pretože veľa ľudí si vyberá krátke hesla (7 znakov) alebo sú príliš jednoduché. Tento útok sa vyžíva v dvoch situáciách. Ide o dešifrovanie kľúča a obchádzanie autentifikačného mechanizmu. Crack patrí medzi najvyužívanejšie programy v tejto oblasti. Používa niekoľko slovníkov a súbor pravidiel, podľa ktorých sa hľadá heslo. Tento program využíva aj informácie z užívateľových súborov.[11]

Útoky tejto metódy nebývajú často úspešne, ak je heslo poskladané z viacerých slov alebo obsahuje malé a veľké písmena zároveň s číslami. Pri takýchto systémoch hesiel je účinná metóda hrubej sily, ale k nájdeniu takéhoto hesla môže trvať dlhú dobu, pre dosiahnutie požadovaného výsledku. Zraniteľnosť hesla môže byť znížená takmer na nulu tak, že sa obmedzí počet pokusov v časovom rámci a zároveň je heslo zvolené múdro. Napríklad ak sa obmedzíme hádanie hesla len na 3 pokusy, alebo po každom neúspešnom pokuse musí uplynúť 10 minúť, aby sme mohli skúsiť iné heslo. Slovníkový útok je často využívaný

spamermi. Spameri využívajú zoznamy účtov a skúšajú všetky známe domény a snažia sa ich prejsť. Celý proces môže prebehnúť rýchlo na veľkých serveroch organizácií.[28]

5.5 Rainbow attack (Rainbow útok)

Idea celého postupu je jednoduchá: prečo pri lámaní každého hesla vždy počítať odpovedajúci hašovací kód, keď ich môžeme pre rôzne vstupy vypočítať vopred? Zohnať použiteľnú tabuľku predvypočítaných hodnôt je jednoduché, ale nič nám však nebráni vytvoriť si ju samostatne. Generovanie vlastných tabuliek taktiež netrvá práve krátku dobu, ale následné lámanie hašovacieho kódu zaberie iba pár sekúnd (samozrejme za predpokladu použitia správnej abecedy). Výhodou používania rainbow tabuliek je ich podpora v rade aplikácií slúžiacich k lámaní hesiel.[2]

Rainbow tabuľky účinne riešia problém kolízie s obyčajným hašovacím reťazcom nahradením jednotného redukovania funkcie R so sekvenciou súvisiace redukovanie funkcií R_1 cez R_k. Týmto spôsobom sa podľa poradia dva reťazce zrazia a spoja, musia zasiahnuť rovnakú hodnotu na rovnakej iterácií. Preto budú konečné hodnoty v každom reťazci zhodné. Výsledným spracovaním priebehu môžeme zoradiť reťazce v tabuľke a odstrániť „duplikát“ reťazcov, ktoré majú rovnakú konečnú hodnotu ako iné reťazce. Nové reťazce sú generované na vyplnenie tabuľky.

6 Útok na SHA-1

V tejto kapitole opisujeme útok na SHA-1, v ktorom sú zavedené nové techniky a stratégie. Tie nám pomáhajú odstrániť prekážky na nájdenie kolízií v SHA-1. Použitím týchto metód je možné obmedziť zložitosť na menej ako 2^{69} hašovacích operácií. Tento útok je prvý, ktorý bol realizovaný na plnú SHA-1 (80-krokov). Analýza útoku je postavená na pôvodnom diferenciálnom útoku na SHA-0, blízkej kolízií SHA-0, multiblokovej kolíznej technike a technike modifikovanej správy.

6.2 Lokálne kolízie

Lokálna kolízia je kolízia počas niekoľkých krokov hašovacej funkcie. Pre SHA-1 platí, že lokálna kolízia má 6 krokov. Predpokladajme, že diferenciálna správa v bite j sa prvýkrát objavuje v kroku i (napríklad, $\Delta m_{i-1,j} = 1$). Rozdiel bude mať vplyv na reťazenie premenných a, b, c, d, e v najbližších piatich krokoch za sebou. Diferenciálne cesty ako lokálne kolízie nájdeme v tabuľke č.3[32].

Tabuľka 3. 6-kroková lokálna kolízia od kroku *i*. Mierou rozdielu je +. Sčítanie v exponente je modulo 32. „nc“ znamená žiadny prenos. Δf je rozdiel výsledku Booleovskej funkcie [32].

Krok	Δm	Δa	Δb	Δc	Δd	Δe	Podmienky
i	2^j	2^j					nc
i + 1	2^{j+5}		2^j				
i + 2	2^j			2^{j+30}			nc, $\Delta f = 2^j$
i + 3	2^{j+30}				2^{j+30}		nc, $\Delta f = 2^{j+30}$
i + 4	2^{j+30}					2^{j+30}	nc, $\Delta f = 2^{j+30}$
i + 5	2^{j+30}						nc

Kolízia diferenciálnej cesty pre SHA-0 vyberá $j = 2$ tak že $j + 30 = 32$ sa stane eliminujúcou pre možný efekt v posledných troch krokoch. Táto podmienka

$$m_{i,2} = \neg m_{i+1,7}\,, \tag{6.1}$$

pomáha vyrovnať úplne reťazenie premenných rozdielov v druhom kroku lokálnej kolízie, kde $m_{i,j}$ označuje j-tý bit správy slova m_i. Podmienka správy v treťom kole

$$m_{i,2} = \neg m_{i+2,2}\,, \tag{6.2}$$

pomáha vyrovnať rozdiel spôsobený nelineárnou funkciou v treťom kroku lokálnej kolízie. Lokálna kolízia nie je závislá na rozšírení správy, platí to pre SHA-0 a aj pre SHA-1. Preto túto lokálnu kolíziu môžeme použiť ako základ pri skonštruovaní kolízie a blízkych kolízií pre SHA-0 a SHA-1 [32].

6.3 Diferenciálne cesty

Diferenciálnou cestou nazývame zreťazenie (konkatenáciu) lokálnych kolízií . Ak chceme zrealizovať túto cestu potrebujeme nájsť podmienky pre lokálne kolízie. Podmienky pre lokálne kolízie môžu byť určené prostredníctvom 80 bitového 0-1 vektora $x = (x_0, ..., x_{79})$, tzv. poruchový vektor. Pre 80 premenných x_i, akýchkoľvek 16 po sebe nasledujúcich premenných určí zvyšok. Máme teda 16 voľných premenných určených 2^{16} možnosťami. Hľadáme „dobrý“ vektor, umožňujúci znížiť časovú zložitosť výpočtu na 2^{16}. V prípade SHA-1, každý vstup x_i v poruchovom vektore je 32 bitové slovo, skôr ako jeden bit. Vektory takto definované vyhovujú pre použite na výpočet SHA-1 hašu.
To znamená, že pre $i = 16, ..., 79$,

$$x_i = (x_{i-3} \oplus x_{i-8} \oplus x_{i-14} \oplus x_{i-16}) \lll 1. \tag{6.3}$$

Ak poruchový vektor splní potrebné podmienky potom vedie ku kolízií. Tieto podmienky sú zhrnuté v tabuľke č.4. V SHA-1 je zložité nájsť dobrý poruchový vektor s malou Hammingovou váhou, pretože prehľadávaný priestor je veľmi obsiahly [33].

Tabuľka 4. Podmienky pre poruchové vektory v SHA-1 v *t* krokoch [33].

	Podmienky	Účel
1.	$x_i = 0$ pre $i = t - 5, ..., t - 1$	Produkuje kolízie v poslednom kroku t
2.	$x_i = 0$ pre $i = -5, ..., -1$	Zabránenie skráteným kolíziám v prvých pár krokoch
3.	Rozdielne hodnoty na rovnakej bitovej pozícií v prvých 16-tich premenných	Vyhnutie sa nemožnej kolíznej cesty vzhľadom k IF

6.4 Hľadanie poruchových vektorov s nízkou Hammingovou váhou

Dôležitým krokom v tejto časti je nájdenie dobrého poruchového vektora. Vyhľadávanie sa môže stať jednoduchším použitím rozšírenej rekurzívnej správy. Pretože je

tam 16 32-bitových voľných premenných, vyhľadávací priestor môže mať veľkosť 2^{512}. Všimneme si, že týchto 80 poruchových vektorov x_0, ..., x_{79} môžeme chápať ako 80 na 32 (veľkosť matice) matíc, kde každý vstup je samotný 0/1 bit. Poznámka, pre matice s malou Hammingovou váhou, nenulové vstupy sa budú pravdepodobne koncentrovať do niekoľkých po sebe idúcich stĺpcov matice. Z toho dôvodu môžeme vybrať prvé dva vstupy $x_{i,\,j-1}$, $x_{i,\,j}$ v matici a nechať dva 16-bitové stĺpce začínajúce na $x_{i,\,j-1}$, $x_{i,\,j}$ robiť variácie prostredníctvom všetkých 2^{32} možností. Existuje 64 možností pre i (i = 0, 1, ...,63) a 32 možností pre voľby j (j = 1, 2, ..., 32). V skutočnosti s rovnakými i a rozdielnymi možnosťami j sa produkujú poruchové vektory, ktoré sú rotované navzájom okolo seba a mali by mať rovnaké Hammingove váhy. Nastavením j = 2, môžeme minimalizovať vykonaný účinok. Celkovo, veľkosť hľadaného priestoru je nanajvýš $64 \times 2^{32} = 2^{38}$ [33].

Najprv vyhľadáme pre najlepšie vektory predpovedajúce jedenoblokové kolízie. Pre plnú SHA-1, je najlepšie získať nastavenie $x_{64,\,2} = 1$ a $x_{i,\,2} = 0$ pre i = 65, .., 79. V tabuľke č.7 je uvedený výsledný poruchový vektor. Najlepšie poruchové vektory pre SHA-1 znižujú t-kroky, sú rovnaké s prvými 80 t vynechanými vektormi. Až do 75 krokov je Hammingova váha stále dostatočne malá na to, aby útok s menšou zložitosťou ako 2^{80} bol povolený. Výsledky týchto variant sú uvedené v tabuľke č.9. V hľadaní poruchových vektorov budeme pokračovať aj naďalej, aby sa nám podarilo prelomiť plnú SHA-1. Vypočítame viac vektorov po 80 kroku použitím rovnakej SHA-1 správy rozšírenej (expanzívnej) rovnice viď tabuľka č.7 [33].

Potom hľadáme všetky možné 80-vektorové intervaly (x_i, ..., + x_{i+79}). Pre výstavbu blízkej kolízie môžeme použiť súbor 80 vektorov s malou Hammingovou váhou. V skutočnosti sme našli celkom 12 dobrých sád vektorov, a to nám dáva určitú voľnosť vybrať tú, ktorá dosiahne najlepšiu zložitosť (iné kritéria ako len Hammingovu váha).

Nakoniec porovnávame minimálnu Hammingovu váhu poruchových vektorov nájdených experimentálne, keď boli prijaté rôzne podmienky, viď tabuľka č.5. Predpoklad, že priemerná pravdepodobnosť v 2-4 kolách je 2^{-3}. Reálny poruchový vektor by mal mať Hammingovu váhu nižšiu ako je prahová hodnota 27. Kolízny diferenciál má vyššiu pravdepodobnosť ako 2^{-80}, ktorý môže mať za následok útok rýchlejší ako s teoretickú hranicu 2^{80}. Vidíme, že odstránenie všetkých podmienok má veľký vplyv na znižovanie Hammingovej váhy poruchových vektorov [33].

Tabuľka 5. Hammingova váha (pre kolá 2-4) najlepšie poruchové vektory pre varianty SHA-1 nájdené experimentálne. Porovnanie sa vykonáva v rôznych podskupinách podmienok uvedených v tabuľke č.4. Zápis 1BC označuje jeden blok kolízie, 2BC je druhý blok kolízie, NC znamená blízke kolízie [33].

krok	Existujúce výsledky				Nové výsledky	
	SHA-1		SHA-1 w/o Kolo 1		SHA-1	
	podmienky		podmienky		podmienky	
	1,2,3	2,3	1,2	2	1	-
	1BC	NC, 2BC	1BC	NC, 2BC	1BC	NC, 2BC
47	**26**	12	24	12	5	5
53	42	20	**16**	16	10	7
54	39	**24**	36	16	10	7
60					14	11
70					14	17
75					**26**	21
80					31	**25**

6.5 Techniky na skonštruovanie diferenciálnych ciest

Keď vektor nespĺňa podmienky z tabuľky č.4, potom skonštruovanie diferenciálnych reálnych ciest, je oveľa ťažšie. Je to najzložitejšia časť útokov na SHA-1. Nájdenie diferenciálnych ciest je hlavná časť analýzy. Bez nich by sme nenašli žiadne skutočné kolízie.

Myšlienky nových analytických metód:

- Použitie „odčítania“ namiesto „neekvivalencie“ ako meradla rozdielu uľahčujúce presné analýzy.
- Využitie špeciálnych diferenciálnych vlastností IF. Najmä, keď vstupný rozdiel je 1, výstupný rozdiel môže byť 1, -1 alebo 0. Z toho dôvodu môže funkcia zachovať, zahadzovať alebo absorbovať vstupné rozdiely, dávajúce dobrú flexibilitu pre konštrukciu diferenciálnych ciest.
- Využitie nosného efektu. Keďže

$$2^j = - 2^j - 2^{j+1} \ldots - 2^{j+k-1} + 2^{j+k} , \qquad (6.4)$$

pre všetky k, jediný bitový rozdiel j môže byť rozšírený do niekoľko bitov. Táto vlastnosť umožňuje zaviesť špeciálne bitové rozdiely.

- Použitie rôznej správy rozdielu pre 6 krokov lokálnej kolízie. Napríklad: (2^j, 2^{j+5}, 0, 0, 0, $2^{j+30)}$ je reálna správa rozdielov pre miestne kolízie v prvom kole.
- Zavedieme extra bitové rozdiely vyrobené nemožnými bitovými rozdielmi po sebe idúcich miestnych kolízií, ktoré zodpovedajú nasledujúcim poruchám v prvých 16 krokoch, alebo vyrovnáme bitové rozdiely reťazných premenných vyrobených skrátením miestnych kolízií.

Blízke kolízie diferenciálnej cesty pre prvý blok sú uvedené v tabuľke č.13 [33].

6.6 Odvodené podmienky

Teraz môžeme odvodiť podmienky pre správy a reťazenie premenných. Ak má správa rozšírenia posunutú operáciu, porucha nastáva v iných bitových pozíciách ako v 2. bite správy slov (viď tabuľka č.7). V SHA-0 poruchy začínajú v 2. bite. Ak sa to stane v XOR kolách (kolo 2 a 4), bude počet podmienok stúpať z 2 až na 4 pre každú miestnu kolíziu. To sa môže odraziť na počte podmienok, ak nie sú dobre riešené.

Popisujeme užitočnú techniku pre využitie dvoch skupín správových rozdielov zodpovedajúce dvom po sebe idúcim poruchám v rovnakom kroku, aj pre výrobu jednej 6 krokovej lokálnej kolízie. Ak je porucha v oboch bitoch, v bite 1 a v bite 2 x_i, môžeme nastaviť znaky správových rozdielov Δm_i, ktoré majú byť opakom týchto dvoch bitov. Týmto spôsobom, aktuálne správy rozdielu môžu byť považované za jeden bit rozdielu v polohe 1, od $2^1 - 2^0 = 2^0$. Preto počet podmienok môže byť redukovaných zo 4 + 2 = 6 na 4 [33].

6.7 Modifikačné techniky správy

Použitím základných modifikačných techník správy sa môže zmeniť vstupná správa tak, aby sa podmienky pre reťazenie premenných konali v prvých 16 krokoch. Správy môžeme upraviť aj tak, aby platili podmienky v 17- 22 krokoch.

Modifikovaná správa by mala spĺňať všetky podmienky správy, aby sme splnili diferenciálne cesty. Všetky správy podmienok môžu byť vyjadrené ako rovnica bitových premenných v m_0, m_1, ... m_{15} (slovné správy pred správou expanzie). Pretože z 1-bitového posunu v správe rekurzie si všetky rovnice navzájom neodporujú. Predpokladajme, že by sme chceli opraviť 10 podmienok od 17 do 22 kroku zmenou posledných 6 slovných správ m_{10}, m_{11}, ..., m_{15}. Existuje 32 reťazných premenných podmienok, spolu so všetkými 47 správami rovnice z kroku 11 na krok 16, celkový počet podmienok v krokoch 11-16 je 79. Intuitívne, tento ponechávaný priestor správy veľkosti 2^{113} je dostatočne veľký na zmenu niektorých bitových správ pre opravu 10 podmienok [33].

6.8 Výber najlepšieho poruchového vektora

Ak sú podmienky odvodené a modifikované správy použité, môžeme analyzovať zložitosť počítaním zostávajúceho počtu podmienok v kolách 2 - 4. Počítanie závisí na logickej funkcii. Umiestnenie poruchy nastáva v každom kole. Miestne kolízie za hranicami kôl potrebujú iné zaobchádzanie. Ďalšie informácie môžete nájsť v tabuľke č.10.

Určené poruchové vektory (viď tabuľka č.7) zistíme pre 80 krokov v blízkej kolízií, minimálna Hammingova váha je 25 za použitia 80 vektorov s indexom (15,9). Avšak, minimálny počet je 71 podmienok, za použitia 80 vektorov s indexom (17,96). Je to preto, že podmienky v krokoch 79 a 80 môžu byť ignorované pre účely blízkych kolízií, a podmienky v kroku 21 môžu byť vykonané držaním. Ak použijeme minimálneho počtu podmienok ako kritérium výberu, tak vyberáme vektory s indexom (17,96) ako poruchové vektory pre výstavbu 80 krokovej blízkej kolízie [33].

6.9 Použitie blízkych kolízií na nájdenie kolízií

Ak využijeme myšlienku multiblokových kolízií, môžeme vytvoriť 2 bloky kolízií použitím blízkych kolízií. VMD5 je zložitosť nájdenia prvého bloku blízkej kolízie vyššia ako nájdenie kolízie v druhom bloku.

Nech M_0 a M'_0 sú dve správy blokov a

$$\Delta h_1 = h'_1 - h_1, \tag{6.5}$$

je výstup rozdielu pre 80 krokovej blízkej kolízie. Ak sa pozrieme bližšie na poruchové vektory, nachádzajú sa tam 4 poruchy v posledných 5 krokoch, ktoré sa budú šíriť do $\Delta h1$ a stávajú sa vstupným rozdielom v počiatočných hodnotách pre druhý blok [33].

Existujú dve techniky, ktoré používame na výstavbu diferenciálnych ciest pre druhé bloky správ M_1 a M'_1. Najprv, sme použili techniky opísané v podkapitole 5.5, takže Δh_1 môže byť "absorbovaná" v prvých 16 krokoch diferenciálnych ciest. V druhom kroku, musíme nastaviť podmienky na M_1 tak, že výstup rozdielu Δh_2 bude mať opačné znaky pre každý z rozdielov v Δh_1. Inými slovami, sme si určili značky (podmienky) tak, aby

$$\Delta h_2 + \Delta h_1 = 0, \tag{6.6}$$

čo znamená, že kolízia nastane po druhom bloku správy. Týmito podmienkami nezvýšime počet podmienok pre výslednú diferenciálnu cestu, neovplyvňuje to zložitosť. Blízke kolízie v druhom bloku správy, možno nájsť s rovnakou zložitosťou ako kolízie na prvom bloku [33].

6.10 Zložitosť analýz a ďalších techník

Pomocou modifikačnej techniky môžeme opraviť podmienky krokov 17-22. Po prvé, môžeme predspracovať a opraviť súbor správ v prvých 10 krokoch a nechať zvyšok ako voľné premenné. Existuje 70 podmienok v krokoch 23-77. Pre tri podmienky v krokoch 23-24, používame „skoré zastavenie techniky“. Urobíme výpočet až po krok 24 a potom vyskúšame, či tri podmienky uvedené v krokoch 23-24 sú platné. K tomu je potrebné v 12 kroku operácie vrátane zmenenej správy opravovanie podmienok krokov 17-22. To sa rovná asi dvom SHA-1 operáciám. Celková zložitosť nájdenia blízkej kolízie pre plnú SHA-1 je približne 2^{68} výpočtov. Celková zložitosť nájdenia úplnej kolízie SHA-1 je teda približne 2^{69}. Výsledky sú zhrnuté v tabuľke č.8 a č.9 [33].

6.11 Výstavba špecifickej diferenciálnej cesty

Nech $a_{i,j}$ označujú j-tý bit premennej a_i a

$$\Delta a_i = a'_i - a_i, \tag{6.7}$$

značí rozdiel. Používame $a_i(j)$ na označenie

$$a_i(j) = a_i + 2^{j-1}, \tag{6.8}$$

bez bitového vykonávania, a $a_i(-j)$, ktoré udáva, že

$$a_i(-j) = a_i - 2^{j-1}, \tag{6.9}$$

bez bitového vykonávania. V tabuľke č.7 sú poruchy vektora krokov 23-80 k vytvoreniu 58 kroku diferenciálnej cesty [33].

Tabuľka 6. Kolízia SHA-1 zredukovaná na 58 krokov. Všimnite si, že pravidlá predspracovania sa neuplatňujú na správy a compress(h_0, M_0) = compress(h_0, M'_0) = h_1.

h_0: 67452301 efcdab89 98badcfe 10325476 c3d2e1f0
M_0: 132b5ab6 a115775f 5bfddd6b 4dc470eb 0637938a 6cceb733 0c86a386 68080139 534047a4 a42fc29a 06085121 a3131f73 ad5da5cf 13375402 40bdc7c2 d5a839e2
M'_0: 332b5ab6 c115776d 3bfddd28 6dc470ab e63793c8 0cceb731 8c86a387 68080119 534047a7 e42fc2c8 46085161 43131f21 0d5da5cf 93375442 60bdc7c3 f5a83982
h_1: 9768e739 b662af82 a0137d3e 918747cf c8ceb7d4

Cesta prvých 16 krokov je uvedená v tabuľke č.12. Zvyšné cesty sú skonštruované z bežných miestnych kolízií. Existujú dve hlavné komplikácie, v prvých 16 krokoch. Následne si ukážeme vysporiadanie s 2 uvedenými problémami [33]:

1. Správy rozdielov poruchy sú zahájené v krokoch -5 až -1. Tieto rozdiely sú

$$m_0(30),\ m_1(-5, 6, -30, 31),\ m_2(-1, 30, -31).$$

2. Nasledujúce poruchy v rovnakej bitovej pozícií v prvých 16 krokoch. Tam sú dve takéto sekvencie:

$$1.\ x_{1,2},\ x_{2,2},\ x_{3,2},$$

$$2.\ x_{8,2},\ x_{9,2},\ x_{10,2}.$$

Viac sa treba zamerať na hodnoty Δa_i bez vykonania rozšírenia. Najprv zvážime propagáciu rozdielu $m_1(-5, 6)$. Tá vyrába nasledujúce rozdiely:

$$a_2(5) \rightarrow a_3(10) \rightarrow a_4(15) \rightarrow a_5(20) \rightarrow a_6(25).$$

Tieto rozdiely sa šíria cez b, c, d a tieto rozdiely sa spojili do premennej e:

$$e_6(3) \rightarrow e_7(8) \rightarrow e_9(13) \rightarrow e_9(18) \rightarrow e_{10}(23).$$

Rozdiely v b, c, d sú ľahko riešené, pretože oni môžu byť absorbované podľa logickej funkcie. Potrebujeme dávať pozor na premenné a a e. Rozdiel $a_6(25)$, rovnako ako päť rozdielov v e_i je zrušených v kroku po nasledujúcom kroku, v ktorom sa prvýkrát objavia. Týmto spôsobom sa nebudú šíriť ďalej. Zrušenie sa vykonáva buď existujúcimi rozdielmi v

inej premennej alebo extra rozdielmi z prevodu účinku(efektu). Napríklad sme si rozšírili a_8(-18) na a_8(18, 19, ...,- 26), takže a_8(25, -26) môže produkovať bitový rozdiel c_{10}(23, -24), na vyrovnanie e_{10}(23) a a_8(-26) produkuje b_9(-26), aby sa vyrovnalo e_9(26). Po sebe idúce poruchy sú riešené rôznymi spôsobmi. Pre prvú sekvenciu platí, že stredná porucha m_2(2) je v kombinácii s m_2(1) tak, že porucha je presunutá z bitu 2 na bit 1. Pre druhú sekvenciu je stredné narušenie m_9(2) vyvážené na c_9(2), ktoré pochádza z rozdielu a_7(4) [33].

6.12 Odvodenie podmienky pre a_i a m_i

Táto metóda na odvodenie premenných sa zhoduje s analýzou SHA-0, je zložitejšia pre správy, lebo sa týka bitových pozícií v správe, detaily sú vynechané.

Pre zjednodušenie výpočtu, najprv nájdeme čiastočnú správu. To možno vykonať pomocou správ modifikačnej techniky v systematickom spôsobe. Tým získame štyri voľné premenné, menovite m_{12}, m_{13}, m_{14}, m_{15}. Ďalej môžeme písať každému m_i ($i \geq 16$) v závislosti na štyroch voľných premenných pomocou správy expanzie rekurzie. Podmienky pre tieto m_i sa potom prekladajú na základe podmienok pre m_{12}, m_{13}, m_{14}, m_{15} a tieto bity budú určené počas kolízie vyhľadávania [33].

7 Záver

Cieľom tejto bakalárskej práce bolo zistiť na základe dostupnej literatúry slabiny funkcie SHA-1. Opísať vybraný útok, ktorý by viedol k odhaleniu zdrojového textu alebo založený na cielenom sfalšovaní hašu a implementovať ho.

Na začiatku tejto bakalárskej práce bola rozobraná základná teória o hašovacích funkciách. Postupne ste sa mohli oboznámiť s vlastnosťami hašovacích funkcií. V ďalšej kapitole boli spomenuté a vysvetlené konštrukcie hašovacích funkcií spolu s Damgard-Merklovým zosilnením. Zmienili sme sa aj o tom, že ich používame v kryptografických protokoloch a v závere 3 kapitoly sme uviedli príklady hašovacích funkcií a ich stručnú charakteristiku. Ďalej sme podrobnejšie opísali hašovaciu funkciu SHA-1. SHA-1 patrí medzi iteračné rovnice. Táto rovica sa skladá z definície konštánt, predspracovania, iterčnej slučky a výstupu. V predposlednej kapitole sú opísané primitívne útoky: útok hrubou silou, slovníkový útok a Rainbow útok.

Posledná kapitola sa zaoberá útokom na SHA-1. V tejto časti je teoreticky opísaný útok, ktorý prezentovala pani X. Wangová a jej tým. Je tu uvedené ako vznikajú lokálne kolízie, ako môžeme nájsť diferenciálne cesty na uskutočnenie lokálnej kolízie. Vysvetľuje sa tu, čo sú poruchové vektory a dáva návod, ako skonštruovať útok na SHA-1 pomocou Wangovej techník. Tieto techniky vychádzajú z techník z útoku na SHA-0, a pre SHA-1 je tento útok vylepšený.

V praktickej časti bola implementovaná hašovacia funkcia SHA-1. Bola naprogramovaná v jazyku Java v programe IDE (jazyk bol zvolený po dohode s vyučujúcim). Tento program bol vyskúšaný a jeho výstupy boli porovnané s programami na internetových stránkach uvedené v literatúre. Ďalej na priloženom CD je implementovaný program na nájdenie kolízií (hrubá sila). Ako hlavný program tejto bakalárskej práce mal byť program, ktorý útočí na SHA-1, teda so zložitosťou 2^{69} opísaný v kapitole 6. Keďže potrebné podrobnejšie informácie o tomto útoku nebolo možné nájsť z dôvodu nedostatku verejných informácií z dôvodu nedostupnosti článku [31], tak sa mi nepodarilo z nadobudnutých výsledkov a informácií z článkov implementovať overenie funkčnosti tohto útoku.

Zoznam použitej literatúry

[1] BITTO, Ondřej. Lámání hesel v praxi. *Lupa* [online]. 2005, 1, [cit. 2010-05-24]. Dostupný z WWW: <http://www.lupa.cz/clanky/lamani-hesel-v-praxi-1/>.

[2] BITTO, Ondřej. Lámání hesel v praxi. *Lupa* [online]. 2005, 4, [cit. 2010-05-26]. Dostupný z WWW: <http://www.lupa.cz/clanky/lamani-hesel-v-praxi-4/>.

[3] *Blog.enfinity.sk* [online]. 2009 [cit. 2010-05-26]. Zraniteľnosť elektronických platieb (nakupujeme zadarmo). Dostupné z WWW: <http://blog.enfinity.sk/bezpecnost/zranitelnost-elektronickych-platieb/>.

[4] Brute force attack. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, , last modified on 20.5.2010 [cit. 2010-05-24]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Password_cracking?#Brute_force_attack>.

[5] *Cotse.net privacy servis* [online]. 2004 [cit. 2010-05-24]. Dictionary Attack. Dostupné z WWW: <http://mail.cotse.net/dictionary_attacks.html>.

[6] *Faqs.org* [online]. 2001 [cit. 2010-05-24]. RFC 3174 - US Secure Hash Algorithm 1 (SHA1) . Dostupné z WWW: <http://www.faqs.org/rfcs/rfc3174.html>.

[7] *Fei.tuke.sk* [online]. Košice : 2003 [cit. 2010-05-24]. HAŠOVACIE (JEDNOCESTÉ) FUNKCIE. Dostupné z WWW: <http://www.kemt.fei.tuke.sk/predmety/KEMT414_AK/_materialy/Cvicenia/kryp_8.pdf>.

[8] *Fei.tuke.sk* [online]. Košice : 2009 [cit. 2010-05-24]. VYBRANÉ TYPY HAŠOVACÍCH FUNKCIÍ. Dostupné z WWW: <http://www.kemt.fei.tuke.sk/Predmety/KEMT414_AK/_materialy/Prednasky/10%20Vybrane%20typy%20hasovacich%20funkcii.pdf>.

[9] GALOVIČ, Pavol. *REKURZÍVNE DEKÓDOVANIE BLOKOVÝCH KÓDOV* [online]. Bratislava, 2007. 5 s. Oborová práce. STU. Dostupné z WWW: <http://www.nuc.elf.stuba.sk/svoc2007/CDROM/23%20Telekomunikacie%20VI/TEVI_DD_Galovic.pdf>.

[10] GLABB, Ryan, et al Multi-mode Operator for SHA-2 Hash Functions. In *Multi-mode Operator for SHA-2 Hash Functions* [online]. [s.l.] : [s.n.], 2006 [cit. 2010-05-24]. Dostupné z WWW: <http://hal.archives-ouvertes.fr/docs/00/12/55/21/PDF/ersa06.pdf>.

[11] Hesla. *Bezpečnost v systémech UNIX* [online]. 1997, 7, 4, [cit. 2010-05-26]. Dostupný z WWW: <http://www.ics.muni.cz/zpravodaj/articles/97.html>.

[12] KLÍMA, Vlastimil. *Hašovací funkce, principy, příklady a kolize* [online]. verze 1. [s.l.] : [s.n.], 19. 3. 2005 [cit. 2010-05-24]. Dostupné z WWW: <http://crypto-world.info/klima/2005/cryptofest_2005.htm>

[13] KLÍMA, Vlastimil. Nalézání kolizí MD5 - hračka pro notebook. *Personal page: Vlastimil Klima* [online]. 2005, 1, [cit. 2010-05-24]. Dostupný z WWW: <http://cryptography.hyperlink.cz/md5/MD5_kolize.pdf>.

[14] KLÍMA, Vlastimil Operační mody blokových šifer a hašovací funkce. In *Základy moderní kryptologie - Symetrická kryptografie III.* [online]. verze: 1.4,. [s.l.] : [s.n.], 20. 4. 2005 [cit. 2010-05-24]. Dostupné z WWW: <http://www.karlin.mff.cuni.cz/~tuma/ciphers09/Symetricka_kryptografie_III.pdf>

[15] KLÍMA, Vlastimil; ROSA, Tomáš. Bezpečnost mobilních telefonů GSM : GSM pod tlakem klonování. *Chip*. 1998, 7, s. 134 - 136. Dostupný také z WWW: <http://cryptography.hyperlink.cz/1998/1998.html>.

[16] KLÍMA, Vlastimil; ROSA, Tomáš. Zesílení hašovací funkce třídy SHA-1. *Kryptologie pro praxi* [online]. 200ž, 35, [cit. 2010-05-24]. Dostupný z WWW: <http://crypto-world.info/klima/2006/ST_2006_07_19_19.pdf>.

[17] KLÍMA, Vlastimil. Tunely v hašovacích funkcích: kolize MD5 do minuty. *Personal page: Vlastimil Klima* [online]. 2006, 2, [cit. 2010-05-24]. Dostupný z WWW: <http://cryptography.hyperlink.cz/2006/tunely.pdf>.

[18] KLÍMA, Vlastimil Základy moderní kryptologie – Symetrická kryptografie I.. In *Základy moderní kryptologie – Symetrická kryptografie I.* [online]. verze: 1.3,. [s.l.] : [s.n.], 5.4.2005 [cit. 2010-05-24]. Dostupné z WWW: <http://www.karlin.mff.cuni.cz/~tuma/nciphers/Symetricka_kryptografie_I.pdf>.

[19] KUBRAN, Marek. *Http://fornax.elf.stuba.sk/~kubiki/projekty/TCP_IP.pdf* [online]. [s.l.], 2002. 25 s. Semestrální práce. STU. Dostupné z WWW: <http://fornax.elf.stuba.sk/~kubiki/projekty/TCP_IP.pdf>.

[20] MD2 %28cryptography%29. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, , last modified on 6.5.2010 [cit. 2010-05-24]. Dostupné z WWW: <http://en.wikipedia.org/wiki/MD2_%28cryptography%29>

[21] MD4. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, , last modified on 7.4.2010 [cit. 2010-05-24]. Dostupné z WWW: <http://en.wikipedia.org/wiki/MD4>.

[22] MD5. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, , last modified on 24.5.2010 [cit. 2010-05-24]. Dostupné z WWW: <http://en.wikipedia.org/wiki/MD5>.

[23] *Microsoft.cz* [online]. 1997 [cit. 2010-05-24]. Kryptografie veřejných klíčů. Dostupné z WWW: <http://www.microsoft.cz/products/internet/articles/crypto.htm>.

[24] MIRONOV, Ilya; ZHANG, Lintao Applications of SAT Solvers to Cryptanalysis of Hash Functions. In *Applications of SAT Solvers to Cryptanalysis of Hash Functions* [online]. Seattle : [s.n.], 2006 [cit. 2010-05-24]. Dostupné z WWW: <http://research.microsoft.com/en-us/people/mironov/sat-hash.pdf>. ISBN 3-540-37206-7.

[25] NEMEČKAY, Marek. *BEZPEČNOSŤ V OBLASTI INFORMAČNÝCH TECHNOLÓGIÍ*. Bratislava, 2005. 70 s. Diplomová práce. UNIVERZITA

KOMENSKÉHO. Dostupné z WWW: <http://sprite.edi.fmph.uniba.sk/~nemeckay/files/dp_bezpecnost_v_it.pdf>.

[26] *Online-convert.com* [online]. 2010 [cit. 2010-05-29]. Generate a SHA-1 hash. Dostupné z WWW: <http://hash.online-convert.com/sha1-generator>.

[27] SECURE HASH STANDARD. *U.S. DEPARTMENT OF COMMERCE* [online]. 17.4.1995, 1, [cit. 2010-05-24]. Dostupný z WWW: <http://www.itl.nist.gov/fipspubs/fip180-1.htm>.

[28] *Searchsecurity.com* [online]. 2005 [cit. 2010-05-26]. Dictionary Attack. Dostupné z WWW: <http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci1081943,00.html>.

[29] *Timestamp Generator.com* [online]. 2010 [cit. 2010-05-29]. Text to SHA1 Conversion. Dostupné z WWW: <http://www.timestampgenerator.com/tools/sha1-generator/>.

[30] VONDRUŠKA, Pavel. Hledá se náhrada za kolizní funkce …. *Crypto-World* [online]. 15.5.2006, 5/2006, [cit. 2010-05-24]. Dostupný z WWW: <http://crypto-world.info/casop8/crypto05_06.pdf>. ISSN 1801-2140.

[31] WANG, Xiaoyun. The Collision attack on SHA-0. *Crypto* [online]. 1997, [cit. 2010-05-24].

[32] WANG, Xiaoyun; YIN, Yiqun; YU, Hongbo. Efficient Collision Search Attacks on SHA-0. *Crypto* [online]. 2005, 16, [cit. 2010-05-24]. Dostupný z WWW: <http://www.springerlink.com/content/da906llfxwnlmlx2/fulltext.pdf>.

[33] WANG, Xiaoyun; YIN, Yiqun; YU, Hongbo. Finding Collisions in the Full SHA-1. *Crypto* [online]. 2005, 36, [cit. 2010-05-24]. Dostupný z WWW: <http://people.csail.mit.edu/yiqun/SHA1AttackProceedingVersion.pdf>.

A. Príloha: Tabuľky

Tabuľka 7. Poruchové vektory SHA-1. 96 vektorov x_i(i = 0, ..., 95) spĺňa SHA-1rozšírenú rekurzívnu správu, ale žiadne ďalšie podmienky. Druhý index je potrebný iba pre číslovanie 80 vektorov, ktoré boli vybrané pre skonštruovanie najlepšej 80-krokovej blízkej kolízie [33].

Index *i*	*Index*	Vektor x_{i-1}	Index *i*	*Index*	Vektor x_{i-1}	Index *x*	*Index*	Vektor x_{i-1}
1		e0000000	33	17	80000002	65	49	2
2		2	34	18	0	66	50	0
3		2	35	19	2	67	51	0
4		80000000	36	20	0	68	52	0
5		1	37	21	3	69	53	0
6		0	38	22	0	70	54	0
7		80000001	39	23	2	71	55	0
8		2	40	24	2	72	56	0
9		40000002	41	25	1	73	57	0
10		2	42	26	0	74	58	0
11		2	43	27	2	75	59	0
12		80000000	44	28	2	76	60	0
13		2	45	29	1	77	61	0
14		0	46	30	0	78	62	0
15		80000001	47	31	0	79	63	0
16		0	48	32	2	80	64	0
17	1	40000001	49	33	3	81	65	4
18	2	2	50	34	0	82	66	0
19	3	2	51	35	2	83	67	0
20	4	80000002	52	36	2	84	68	8
21	5	1	53	37	0	85	69	0
22	6	0	54	38	0	86	70	0
23	7	800000001	55	39	2	87	71	10
24	8	2	56	40	0	88	72	0
25	9	2	57	41	0	89	73	8
26	10	2	58	42	0	90	74	20
27	11	0	59	43	2	91	75	0
28	12	0	60	44	0	92	76	0
29	13	1	61	45	2	93	77	40
30	14	0	62	46	0	94	78	0
31	15	80000002	63	47	2	95	79	28
32	16	2	64	48	0	96	80	80

Tabuľka 8. Hľadanie zložitosti pre blízke kolízie (NC), a dva bloky kolízie (2BC) SHA-1zredukované na *t*-kroky. „Štart & koniec“ sa odvoláva na index pre poruchové vektory v tabuľke č.7. Zložitosť odhadu sa vysvetlí ako zrýchlenie pomocou technika včasného zastavenia. A odhad pre 78-80 krokov berie do úvahy ako zrýchlenie pokročilých modifikačných technikách.

t-krok SHA-1	Štart & koniec index DV	HW v kolách 2-4	Podmienky v kolách 2-4	Zložitosť NC	Zložitosť 2BC
80	17, 96	27	71	2^{68}	2^{69}
79	17, 95	26	71	2^{68}	2^{69}
78	17, 94	24	71	2^{68}	2^{69}
77	16, 92	23	71	2^{68}	2^{69}
76	19, 94	22	69	2^{66}	2^{67}
75	20, 94	21	65	2^{62}	2^{63}
74	21, 94	20	63	2^{60}	2^{61}
73	20, 92	20	61	2^{58}	2^{59}
72	23, 94	19	59	2^{56}	2^{57}
71	24, 94	18	55	2^{52}	2^{53}
70	25, 94	17	52	2^{49}	2^{50}
69	26, 94	16	50	2^{48}	2^{49}
68	27, 94	16	48	2^{46}	2^{47}
67	28, 94	16	45	2^{43}	2^{44}
66	29, 94	15	41	2^{39}	2^{40}
65	30, 94	13	40	2^{38}	2^{39}
64	29, 92	14	37	2^{35}	2^{36}
63	32, 94	12	35	2^{33}	2^{34}
62	33, 94	11	34	2^{32}	2^{33}
61	32, 92	11	31	2^{29}	2^{30}
60	29, 88	12	29	2^{27}	2^{28}
59	30, 88	10	28	2^{26}	2^{27}
58	29, 86	11	25	2^{23}	2^{24}
57	32, 88	9	23	2^{21}	2^{22}
56	33, 86	8	22	2^{20}	2^{21}
55	32, 86	8	19	2^{17}	2^{18}
54	33, 86	7	18	2^{16}	2^{17}
53	34, 86	7	18	2^{16}	2^{17}
52	32, 83	7	15	2^{13}	2^{14}
51	33, 83	6	14	2^{12}	2^{13}
50	34, 83	6	14	2^{12}	2^{13}

Tabuľka 9. Hľadanie zložitosti pre jednoblokovú kolíziu SHA-1 zníženej na *t*-kroky. Vysvetlenie tabuľky je rovnaké ako pre tabuľku č.8.

SHA-1 redukovaná na *t*-kroky	Štart & koniec bod DV	HW v kolách2-4	Podmienky v kolách 2-4	Hľadanie zložitosti
80	1, 80	31	96	2^{93}
79	2, 80	30	95	2^{92}
78	3, 80	30	90	2^{87}
77	4, 80	28	88	2^{85}
76	5, 80	27	83	2^{80}
75	6, 80	26	81	2^{78}
74	7, 80	25	79	2^{76}
73	8, 80	25	77	2^{74}
72	9, 80	25	77	2^{74}
71	10, 80	24	74	2^{71}
70	11, 80	24	71	2^{68}
69	12, 80	22	68	2^{66}
68	13, 80	21	62	2^{60}
67	14, 80	19	58	2^{56}
66	15, 80	19	55	2^{53}
65	16, 80	18	51	2^{49}
64	17, 80	18	48	2^{46}
63	18, 80	16	48	2^{46}
62	19, 80	16	45	2^{43}
61	20, 80	15	41	2^{39}
60	21, 80	14	39	2^{37}
59	22, 80	13	38	2^{36}
58	23, 80	13	35	2^{33}
57	24, 80	12	31	2^{29}
56	25, 80	11	28	2^{26}
55	26, 80	10	26	2^{24}
54	27, 80	10	24	2^{22}
53	28, 80	10	21	2^{19}
52	29, 80	9	17	2^{15}
51	30, 80	7	16	2^{14}
50	31, 80	7	14	2^{12}

Tabuľka 10. Pravidlá pre počítanie podmienok v kolách 2-4.

Krok	Porucha v bite 2	Porucha v iných bitoch	Komentáre
19	0	1	Pre a_{21}
20	0	2	Pre a_{21}, a_{22}
21	1	3	Podmienka a_{20} je „skrátená“
22-36	2	4	
37	3	4	
38-40	4	4	
41-60	4	4	
61-76	2	4	
77	2	3	Podmienky sú „skrátené“ začínajúce 77 krokom
78	2	2	
79	(1)	(1)	Podmienky pre kroky 79, 80 môžu byť ignorované v analýze.
80	(1)	(1)	

Špeciálne pravidlá počítania:

1. Ak dve poruchy začínajú v oboch bitoch, bite 2 a bite 11 v rovnakom kroku, potom majú výsledok j v 4 podmienkach.
2. Pre kolo 3, dva po sebe nasledujúce poruchy v rovnakej bitovej pozícií slúžia len pre 6 podmienok (skôr než 8). Toto je kvôli vlastnosti MAJ funkcie.

Tabuľka 11. Príklad: Počítanie počtu podmienok pre 80 krokov blízkej kolízie. „Index“ sa odvoláva na druhý index v tabuľke č.7.

Index	Počet podmienok	Komentáre
21	4 – 1 – 1 = 2	4 podmienky: a_{20}, a_{21}, a_{22}, a_{23} -a_{20} spôsobené skrátenie -a_{21} použitie modifikácie
23, 24, 27, 28		
32, 35, 36	2 × 7 = 14	
25, 29, 33, 39	4 × 4 = 16	
43, 45, 47, 49	4 × 4 = 16	
65, 68, 71, 73, 74	4 × 45= 20	
77	3	Skrátené
79	0	2 podmienky ignorované
80	0	1 podmienka ignorovaná
celkovo	71	

Tabuľka 12. Diferenciálna cesta pre 58 krokov SHA-1 kolízie. Všimnite si, že x_i (i = 0... 15) sú poruchové vektory pre prvých 16 krokov, ktoré zodpovedajú 16 vektorom indexu 23 až 38 v tabuľke č.7. Δ zložky udávajú pozície rozdielov a ich znamienko. Napríklad, rozdiel 2^j je uvedený ako (j + 1) a -2^j ako =(j + 1).

Krok i	x_{i-1}	Δm_{i-1}	Δa_i		Δb_i	Δc_i	Δd_i	Δe_i
			Žiaden prenos	Prenos				
1	80000001	30	30	-30, 31				
2	2	-2 -5, 6 -30 31	2 5 -30 31	-2, 3 5 -30 -31, 32	Δa_1			
3	2	-1, -2 -7 30, -31	1 10	1 10	Δa_2	$\Delta a_1^{«30}$		
4	2	-7 30	-2 15 -5	2, -3 -15, 16 5, -6	...	$\Delta a_2^{«30}$	$\Delta a_1^{«30}$	
5	0	-2, 7 30, 31 32	20 28 -1 -10	-20, 21 -28, 29 -1 10, 11, -12		...	$\Delta a_2^{«30}$	$\Delta a_1^{«30}$
6	0	-2 -30, -31	25 15	25 -15, 16			...	$\Delta a_2^{«30}$
7	1	1, 32	1 8 4, -21	1 -8, -9, 10 4, -21				...
8	0	-6	-18	-18, ..., -26	...			
9	80000002	1, 2	-2, 32 -9	-2, 32 9, ..., -19		...		
10	2	-2 -5, 7 31					...	
11	80000002	7, 31	2, -32 9	2, -32 9	...			...
12	0	-2 -5, -7 -30 31, -32			Δa_{11}	...		

Krok i	x_{i-1}	Δm_{i-1}	Δa_i		Δb_i	Δc_i	Δd_i	Δe_i
			Nevykonáva sa	Vykonáva sa				
13	2	-30, -32	-2	-2		$\Delta a_{11}^{«30}$	...	
14	0	7, 32			Δa_{13}		$\Delta a_{11}^{«30}$	...
15	3	1, 30	1	1		$\Delta a_{13}^{«30}$		$\Delta a_{11}^{«30}$
16	0	-6, -7 30			Δa_{15}		$\Delta a_{13}^{«30}$	

Tabuľka 13. Tabuľka 13. Diferenciálna cesta pre 80 krokov SHA-1 kolízie. x_i (i = 0... 19) je poruchový vektor pre prvých 20 krokov, ktoré zodpovedajú 20 vektorom s indexom 1 až 20 v tabuľke č.7. Δ zložky udávajú pozície rozdielov a ich znamienko. Napríklad, rozdiel 2^j je uvedený ako (j + 1) a -2^j ako –(j + 1).

Krok i	x_{i-1}	Δm_{i-1}	Δa_i		Δb_i	Δc_i	Δd_i	Δe_i
			Žiaden prenos	Prenos				
1	40000001	30	30, 31	30, 31				
2	2	-2, -4 6 -30, -31, 32	2 6 30	-2, 3 -6, -7, 8 -30, -31, 32	Δa_1			
3	2	1, 2 -7 30	-1 4 11	-1 4 -11, -12, -13, 14	Δa_2	$\Delta a_1^{«30}$		
4	80000002	7 29, -30 -32	-2, 9 16 -32	-2, 9 -16, -17, -18, 19 -32	...	$\Delta a_2^{«30}$	$\Delta a_1^{«30}$	
5	1	1, -2 -5, 7 29, 31, 32	-5 21 -28	5, -6 -21, 22 28		...	$\Delta a_2^{«30}$	$\Delta a_1^{«30}$
6	0	-2, -6 29, 31 32	11 16 26	-11, -12, 13 -16, 17 -26, 27			...	$\Delta a_2^{«30}$
7	80000001	30	1 -4, -6 32	1 -4, 6, -7 32				...
8	2	-2, -5, -6 30, 31	-19	-19, ..., -26	...			
9	2	1,- 2, -7 -30, -31	-2 -10	-2 10, ..., -20		...		
10	2	7 -30	2	2			...	
11	0	2, -7 -30, 31, -32	9	-9, 10	...			...
12	0	2 -30, -31	-4	-4		...		
13	1	1 32	1	1			...	
14	0	-6						...
15	80000002	-1, 2	-32	-32				

Krok i	x_{i-1}	Δm_{i-1}	Δa_i		Δb_i	Δc_i	Δd_i	Δe_i
			Nevykonáva sa	Vykonáva sa				
16	2	2, 5, -7 -31	2	2	Δa_{15}			
17	80000002	-7 31	-2 32	-2 32	Δa_{16}	$\Delta a_{15}^{«30}$		
18	0	-2, -5, 7 30, 31, 32				$\Delta a_{16}^{«30}$	$\Delta a_{15}^{«30}$	
19	2	30 32	2	2			$\Delta a_{16}^{«30}$	$\Delta a_{15}^{«30}$
20	0	-7 32			Δa_{19}			$\Delta a_{16}^{«30}$

Tabuľka 14. Súbor postačujúcich podmienok a_i pre diferenciálne cesty uvedené v tabuľke č.13. Označenie „*a*“ stojí pri podmienke $a_{i,j} = a_{i-1,j}$ a „*b*“ označuje podmienku $a_{19,30} = a_{18,32}$.

Reťazné premenné	Podmienky pre bity			
	32 – 25	24 – 17	16 – 9	8 – 1
a_1	a00-----	--------	1-----aa	1-0a11aa
a_2	01110---	------1-	0aaa-0--	011-001-
a_3	0-100---	-0-aaa0-	--0111--	01110-01
a_4	10010---	a1---011	10011010	10011-10
a_5	001a0---	--01-000	10001111	-010-11-
a_6	1-0-0011	1-1001-0	111011-1	a10-00a-
a_7	0---1011	1a0111--	101—010	-10-11-0
a_8	-01---10	000000aa	001aa111	---01-1-
a_9	-00-----	10001000	0000000-	---11-1-
a_{10}	0-------	1111111-	11100000	0-----0-
a_{11}	--------	------10	11111101	1-a—1--
a_{12}	0-------	--------	--------	10--11--
a_{13}	--------	--------	--------	11----10
a_{14}	-0------	--------	--------	----0-1-
a_{15}	10------	--------	--------	----1-0-
a_{16}	--1-----	--------	--------	----0-0-
a_{17}	0-0-----	--------	--------	------1-
a_{18}	--1-----	--------	--------	----a---
a_{19}	--b-----	--------	--------	------0-
a_{20}	--------	--------	--------	-----a--
a_{21}	--------	--------	--------	-------1

B. Príloha

Obsah priloženého CD:

- Elektronická verzia bakalárskej práce.
- Zdrojové súbory programov vytvorených v IDE.