UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Marko Kastelic

DRUŽINA ZGOŠČEVALNIH FUNKCIJ SHA-2

Tečaj iz kriptografije in računalniške varnosti
Seminarska naloga

Predavatelj: prof. dr. Aleksandar Jurišić

Ljubljana, 2008

Kazalo

Poglavje 1

Uvod

Zgoščevalne funkcije se kot kriptografski primitiv pojavljajo v različnih kriptografskih sistemih, aplikacijah in protokolih. Zato je razumljivo, da s stališča varnosti in učinkovitosti pričakujemo veliko. Pojem zgoščevalnih funkcij v kriptografiji je širok, zato se bomo v nadaljevanju omejili na tako imenovane hitre oziroma namenske kriptografske zgoščevalne funkcije (v nadaljevanju samo zgoščevalne funkcije). Od ostalih[1] kriptografskih zgoščevalnih funkcij jih loči to, da so nastale oziroma bile načrtovane od začetka posebej za zgoščevanje. Veja hitrih zgoščevalnih funkcij se je začela z nastankom MD4 (avtor R. Rivest) leta 1992. Praktično vse kasnejše znane zgoščevalne funkcije, kot so MD5, SHA-0 in SHA-1, imajo MD4 za skupnega prednika. Tudi medgeneracijske razlike niso velike, kar današnje raziskovalce zelo skrbi, saj se lahko (grobo rečeno), tako prednosti kot slabosti prenašajo iz roda v rod. Leto 2004 je bilo za hitre zgoščevalne funkcije prelomno. Skupini kitajskih raziskovalcev (Wang et al.) je uspelo odkriti napad, ki je zlomil do tedaj najbolj uporabljeni (in po takratnih prepričanjih tudi varni) zgoščevalni funkciji SHA-0 in MD5. Kasneje so uspeli teoretično zlomiti tudi zgoščevalno funkcijo SHA-1. Logična posledica je bila (in še vedno je) iskanje dolgoročnega(ih) naslednika(ov). Žal v tem trenutku ni veliko alternativ. Družina zgoščevalnih funkcij SHA-2 predstavlja eno izmed njih, v naslednjih letih morda celo edino dovolj kvalitetno.

V naslednjih poglavjih bomo najprej omenili osnovne varnostne lastnosti zgoščevalnih funkcij, pristope k njihovem načrtovanju in področja uporabe ter naredili kratek zgodovinski pregled. Sledilo bo poglavje o napadih na hitre zgoščevalne funkcije. V 4. poglavju bo opisana družina zgoščevalnih funkcij SHA-2 s poudarkom na SHA-256 in njeni implementaciji. V 5. poglavju

[1] Poznamo tudi zgoščevalne funkcije, ki temeljijo na bločnih šifrah ali diskretnemu logaritmu in so v splšnem počasnejše od hitrih zgoščevalnih funkcij.

se bomo posvetili analizi varnosti in hitrosti družine SHA-2. Na koncu bodo povzete glavne lastnosti družine SHA-2 in prihodnost na podrpčju hitrih zgoščevalnih funkcij.

Poglavje 2

Hitre kriptografske zgoščevalne funkcije

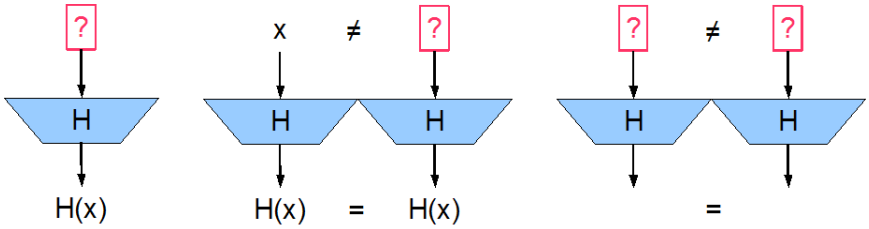
2.1 Osnovne lastnosti

Zgoščevalna funkcija h sprejme sporočilo $M \in \{0,1\}^*$ poljubne dolžine[1] in ga pretvori zaporedje fiksne dolžine $H = h(M) \in \{0,1\}^n$, ki ji pravimo zgostitev (povzetek, prstni odtis). Od zgoščevalne funkcije h pričakujemo naslednje (neformalne) lastnosti [9] (slika 2.1):

- Odpornost na predsliko (angl. Preimage resistance): Pri dani zgostitvi H v doglednem času ni možno najti sporočila M', da velja $h(M') = H$.
- Odpornost na 2. predsliko (angl. Second preimage resistance): Pri danem sporočilu M in zgostitvi $H = h(M)$ v doglednem času ni možno najti drugega sporočila $M' \neq M$ z enako zgostitvijo H.
- Odpornost na trke (angl. Collision resistance): V doglednem času ni možno najti dveh sporočil $M \neq M'$ z enako zgostitvijo H.

Zgoščevalna funkcija, ki izpolnjuje prvi dve lastnosti, je enosmerna. Enosmernost je pogoj za tretjo lastnost. Poleg zgornjih lastnosti morajo dobre zgoščevalne funkcije zadostiti tudi drugim, kot je na primer lastnost plazu (sprememba enega bita vhoda spremeni približno polovico bitov izhoda), lokalna enosmernost (pri dani zgostitvi je enako težko dobiti del vhodnega sporočila kot celotno vhodno sporočilo) in druge (več v [9]).

[1] V praksi je dolžina omejena z veliko konstanto.



Slika 2.1: Ponazoritev odpornosti zgoščevalnih funkcij na napade. Od leve proti desni so prikazani: napad s trki, napad s predsliko, napad z 2. predsliko.

2.2 Načrtovanje zgoščevalnih funkcij

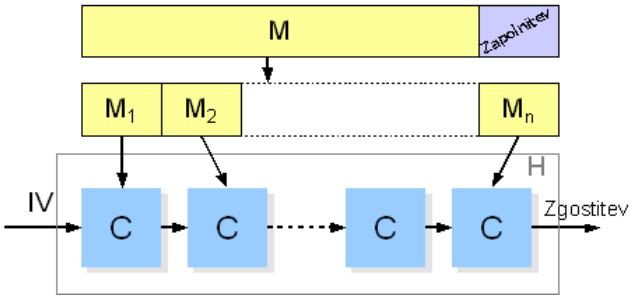
Načrtovanje in implementacija zgoščevalnih funkcij od MD4 naprej poteka po iterativnem pristopu Merkle-Damgard [4] (v nadaljevanju MD). Zgoščevalna funkcija mora procesirati poljubno dolgo sporočilo $M \in \{0,1\}^*$, zato ga najprej razdeli v bloke fiksne dolžine $M_1, M_2, \ldots, M_L \in \{0,1\}^m$ ter to zaporedje zaporedno procesira. V vsaki iteraciji uporabi tako imenovano kompresijko funkcijo:

$$c : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$$

Kompresijska funkcija c pretvori oziroma zmeša dva vhoda fiksne dolžine (blok sporočila M_i in verižno spremenljivko H_i) v izhod, ki je enako dolg kot eden izmed vhodov (od tod ime kompresijska). Pri prvi iteraciji je vhodna verižna spremenljivka enaka neki fiksni inicializacijski vrednosti (IV). Za vse nadaljnje iteracije je vhodna verižna spremenljivka enaka izhodu predhodne kompresijske funkcije. Izhod zadnje kompresijske funkcije je hkrati tudi rezultat zgoščevalne funkcije. V splošnem lahko postopek opišemo takole (slika 2.2):

$$\begin{aligned} H_0 &= IV\ , \\ H_{i+1} &= C(H_i, M_i),\ 0 \leq i < L \\ h(X) &= g(H_L). \end{aligned}$$

Izhodna tranformacija se ponavadi izpusti oziroma se uporabi funkcijo identitete $g(H_L) = H_L$. Nekatere zgoščevalne funkcije (na primer SHA-224/384) uporabijo izhodno transformacijo za skrajšanje dolžine rezultata. Če dolžina sporočila ni večkratnik dolžine bloka sporočila, je potrebno zadnji blok sporočila enolično dopolniti. Splošno pravilo je, da se najprej doda bit 1, sledi ustrezno število bitov 0 in na koncu še binarna predstavitev dolžine originalnega



Slika 2.2: Kontrukcija Merkle-Damgard. Vhod v kompresijsko funkcijo je m bitni blok sporočila M_i in n bitna verižna spremenljivka. Izhod kompresijske funkcije je nova vrednost vhodne verižne spremenljivke v naslednji iteraciji. Po zadnji iteraciji je izhod kompresijske funkcije rezultat zgoščevalne funkcije.

sporočila. Tak način dopolnjevanja skupaj z fiksno IV zagotavlja varnost[2] konstrukcije DM. Teorem Merkle-Damgard pravi, da če je IV fiksna in če dopolnitev vsebuje dolžino sporočila, potem je zgoščevalna funkcija h odporna na trke, če je uporabljena kompresijska funkcija c odporna na trke.

2.3 Področja uporabe

Nekatera področja uporabe zgoščevalnih funkcij (samostojno ali v kombinaciji z drugimi kriptografskimi primitivi in protokoli):

- Preverjanje integritete podatkov.
- Digitalno podpisovanje.
- Shranjevanje in preverjanje gesel.
- Idetifikacija datotek.
- HMAC.
- Bločne šifre (Shacal, Bear, Lion).
- Psevdonaključni generatorji števil.
- Tekoče šifre (Seal).

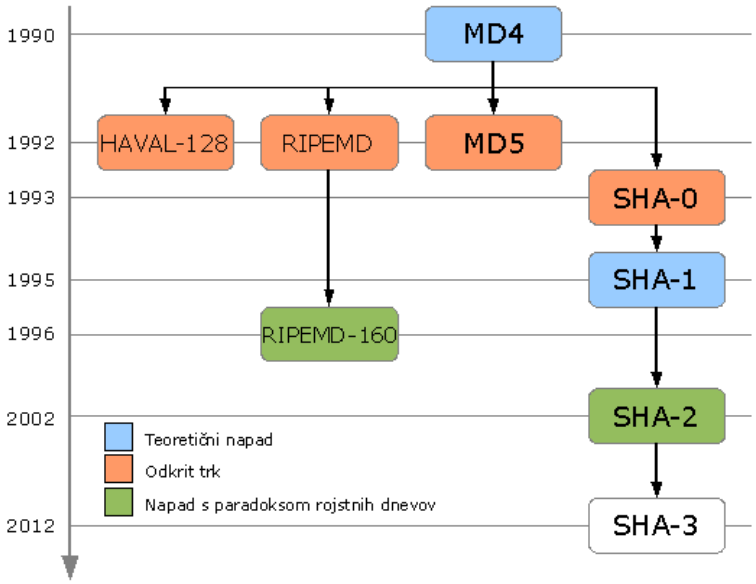
[2]MD ojačitev (angl. MD-strengthening).

- Razpoznavanje slabe kode (Malware).

2.4 Zgodovinski pregled

Razvojna pot hitrih kriptografskih zgoščevalnih funkcij se je začela z nastankom algoritma MD4. Kljub novemu načrtovalskemu pristopu in hitrosti (več kot 10 krat hitrejša od DES) se je izkazalo, da ni varna. Je pa ostala izhodišče za druge algoritme, ki so skupaj znani pod imenom MDx (slika 2.3):

- MD5 - razvil R.Rivest leta 1991.
- HAVAL - razvili Y. Zheng et al. leta 1992.
- SHA - prvi standard objavil NIST leta 1993.
- RIPEMD - razvit v okviru evropskega projekta RIPE leta 1992.



Slika 2.3: Zgodovina razvoja hitrih kriptografskih zgoščevalnih funkcij razreda MDx.

Sledi kratka zgodovina zgoščevalnih funkcij SHA. Prvi algoritem SHA je načrtovala National Security Ageny (NSA), objavil pa ga je NIST kot zvezni standard

(Federal Information Processing Standards - FIPS 180) leta 1993. Leta 1994 je NIST oznanil (brez podrobnosti), da obstaja tehnična pomanjkljivost pri SHA-0 zaradi katere je bil algoritem manj varen. Posledica tega je bil nov algoritem SHA-1 (standard FIPS 180-1). Leta 2002 je NIST objavil standard FIPS 180-2, ki poleg SHA-1 definira tudi tri nove zgoščevalne funkcije SHA-256, SHA-384 in SHA-512. Daljši povzetki naj bi povečali varnost proti napadom s paradoskom rojstnega dne in so po varnosti primerljivi bločno šifro AES (z 128, 192 in 256 bitnimi ključi po vrsti). Leta 2004 je NIST objavil tudi dodatek k standardu in sicer nov algoritem SHA-224, ki ima raven varnosti podoben trojnemu DES z 112 bitnim ključem. Slika 2.3 prikazuje tudi obstoj napadov na zgoščevalne funkcije MD-x.

Poglavje 3

Napadi na hitre zgoščevalne funkcije

3.1 Vrste napadov

Zgoščevalna funkcija z n bitnim izhodom ima "idealno" varnost, če ne obstaja napad, ki je boljši od napadov z grobo silo:

- Iskanje (druge) predslike zahteva 2^n operacij - napad s preiskovanjem celotnega prostora sporočil.
- Iskanje trka zahteva $2^{n/2}$ operacij - napad s paradoksom rojstnega dne[1].

Po drugi strani tako imenovani kriptoanalitični napadi poskušajo kršiti vsaj eno od varnostnih lastnosti zgoščevalne funkcije (poglavje 2). V večini primerov je to odpornost na trke, ker jih lahko iščemo z manj napora. Poznamo več vrst kriptoanalitičnih napadov, nekateri se osredotočajo na konstrukcijo zgoščevalnih funkcij (na primer iterativna konstrukcija MD), drugi na specifične zgoščevalne funkcije (na primer napadi na razred MDx). Nekateri generični napadi na konstrukcijo MD so (podrobnosti v [2, 9]):

- Napad s podaljševanjem (angl. Length Extension Attack).
- Sekundaren napad s trkom (angl. 2nd Collision Attack).
- Napad z več trki (angl. Multiple Collisions ali Joux-jev napad).

[1] Če želimo najti dva dva človeka z enakim rojstnim datumom, potrebujemo skupino 23 ljudi za 50% verjetnost zadetka.

- Napad na kaskade zgoščevalnih funkcij.
- Napad z več (drugimi) predslikami (angl. Multi (2nd)-preimage Attack).

Po teoremu Merkle-Damgard velja, da je zgoščevalna funkcija h je odporna na trke, če je kompresijska funkcija c odporna na trke. Zato se lahko napadalec osredotoči na kompresijsko funkcijo in poskuša najti dva bloka sporočila, ki imata enako vrednost izhodne verižne spremenljivke. Poleg iskanja navadnih trkov in predslik poznamo tudi napade na spremenjene različice algoritmov, kot so iskanje trkov in predslik pri prosti izbiri IV [9]. Prikazuje jih tabela 3.1. Trki s fiksno IV direktno napadejo specifikacijo zgoščevalne funckije in imajo

Vrsta napada	**Dano**	**Najdi**	**Lastnosti**
Predslika	H, Y	X	$c(H, X) = Y$
Druga predslika	H, X	X'	$c(H, X) = c(H, X')$
Trk (fiksna IV)	H	X, X'	$c(H, X) = c(H, X')$
Psevdo predslika	Y	H, X	$c(H, X) = Y$
Trk (naključna IV)	–	H, X, X'	$c(H, X) = c(H, X')$
Psevdo trk[2]	–	H, H', X	$c(H, X) = c(H', X)$

Tabela 3.1: Različne vrste napadov na kompresijsko funkcijo.

tudi praktične posledice [10]. Psevdo trki (angl. Pseudo collisions oziroma Free-start collision attack) in trki z naključno IV (angl. Semi-free-start collision attack) so sicer brez praktičnih posledic (konstrukcija MD fiksira začetno vrednost verižne spremenljivke), vendar ustvarjajo dvome o dolgoročni varnosti zgoščevalne funkcije. Morebitne odkrite slabosti pri napadih iz spodnjega dela tabele 3.1 so nezaželene[3], saj je v nekaterih primerih možno te omejene napade razširiti v pravi napad s trki (primer takega algoritma je MD-5). V literaturi najdemo tudi napad s skoraj trkom (angl. Almost/Near collision), ki najde dva bloka sporočila, pri katerih ima razlika izhodov kompresijske funkcije nizko Hammingovo utež. Napad ima smisel, ker pri n bitni zgoščevalni funkciji pričakujemo, da se izhoda kompresijske funkcije pri dveh različnih vhodnih blokih razlikujeta v povprečju za $n/2$ bitov. Če je to število veliko manjše, potem se kompresijska funckija ne obnaša naključno.

Obstajajo še t.i. napadi na veriženje (angl. Chaining attacks). Ti napadajo iterativno naravo zgoščevalnih funkcij, pri čemer se osredotočajo na kompresijsko funkcijo. Poznamo (podrobnosti v [9]):

[3]Znane tudi kot "Certificational weaknesses".

- Napad s fiksno točko (angl. Fixed point attack).
- Napad s popravljanjem blokov (angl. Correcting-block attack).
- Napad srečanje na sredini (angl. Meet-in-the-middle attack).
- "Herding" napad.

Zgornji napadi so pomembni, ker lahko na primer iskanje trkov in predslik pri kompresijski funkciji z napadom s popravljanjem blokov razširimo na zgoščevalno funkcijo.

Največ pozornosti pri kriptoanalizi hitrih zgoščevalnih funkcij ima diferenčna kriptoanaliza. Gre za statističen napad, ki išče relacije med vhodno-izhodnimi razlikami funkcije. Pri zgoščevalnih funkcijah trke najdemo takrat, ko je razlika na izhodu kompresijske funkcije enaka 0. Vhodna razlika je lahko v bloku sporočila ali v vhodni verižni spremenljivki. Napadi, ki so zlomili MD5, SHA-0 in SHA-1, so diferenčni napadi.

3.2 Praktičnost napadov

Napadi se v veliki večini osredotočajo na iskanje trkov, ki so bolj ali manj naključni [2]. Napadalec največkrat nima nadzora na izbiro sporočil M in M', zato tudi težko odkrije smiselno in nevarno sporočilo M'. Težave se lahko pojavijo, ko zgostitev sporočila ni znana vnaprej in ima napadalec možnost izbrati M in M' (tudi v primeru strukturnih omejitev sporočila). Lep primer je shema digitalnega podpisovanja, s katero podpišemo poveztek nekega sporočila. Napadalec bi lahko odkril par različnih sporočil, ki imajo enako zgostitev. Kasneje bi dal podpisati $h(M)$, podpis pa bi veljal tudi za M'. O nevarnosti trkov govorijo tudi rezultati raziskovalcev, ki so uspeli iskanje naključnih trkov pri MD5 pretvoriti v dobro usmerjene in praktične napade [2]:

- Kaminski in Mikle sta ustvarila dve samorazteglјivi datoteki, pri čemer je bila ena nevarna in je oprla stranska vrata.
- Lenstra, Wand in de Weger so ustvarili dva X.509 ceritfikata, ki trčita.
- Daum in Lucks sta s pomočjo if-then-else konstrukta pri jeziku PostScript ustvarila dva dokumenta, ki trčita. Podobno so dosegli Genhardt, Illies in Schindler pri opisnih jezikih PDF in TIFF brez if-then-else konstrukta.

Kljub zgornjim dosežkom so napadi z drugo predsliko bolj zaželeni, a tudi bolj zahtevni. Varnost digitalnih certifikatov temelji na odpornosti na napade z drugo predsliko. Certifikat vsebuje zgostitev nekega zaporedja bitov. Napadalec lahko ponaredi certifikat tako, da najde drugačno zaporedje z enako zgostitvijo. Idealno tak napad zahteva 2^n operacij, vendar s pomočjo Joux-ovega napada drugo predsliko poiščemo za faktor 2^k hitreje (pri SHA-1 je $k \leq 55$), pri čemer potrebujemo pomnilnik velikosti reda 2^k.

Poglavje 4

Družina zgoščevalnih funkcij SHA-2

Družino SHA-2 sestavljajo zgoščevalne funkcije SHA-224, SHA-256, SHA-384 in SHA-512. Tabela 4.1 prikazuje njihove glavne parametre. Različne dolžine povzetkov (224 do 512 bitov) omogočajo fleksibilno uporabo z drugimi kriptografskimi primitivi in protokoli, SHA-224/256 sta prilagojeni za 32 bitne arhitekture in uporabljata 32 bitne besede, SHA-384/512 pa sta prilagojeni za 64 bitne arhitekture.

Algoritem	**SHA-1**	**SHA-224**	**SHA-256**	**SHA-384**	**SHA-512**
Dolžina vhodnega sporočila (biti)	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Izhodna dolžina (biti)	160	224	256	384	512
Dolžina bloka (biti)	512	512	512	1024	1024
Dolžina besede (biti)	32	32	32	64	64
Število korakov	80	64	64	80	80

Tabela 4.1: Glavne značilnosti zgoščevalnih funkcij SHA-2 (zgoščevalna funkcija SHA-1 je dodana za primerjavo).

Notacija	Opis
$\wedge, \vee, \oplus, \neg$	Bitne operacije nad besedami
$A + B$	Seštevanje po modulu 2^w
$SHR^n(x)$	Bitni pomik v desno za n mest
$ROTR^n(x)$	Bitna rotacija v desno za n mest

Tabela 4.2: Operacije, ki jih uporabljajo zgoščevalne funkcije SHA-2. Vse opracije se izvajajo nad w bitnimi besedami ($w = 32$ pri SHA-224/256, $w = 64$ pri SHA-384/512).

4.1 Specifikacija SHA-256

Zgoščevalna funkcija SHA-256 lahko sprejme vhod maksimalne dolžine 2^{64} in na izhodu vrne 256 bitno zgostitev. V tabeli 4.2 so navedene operacije, ki jih uporablja (podrobnosti v [1]). Ima iterativno DM konstrukcijo, vsaka iteracija procesira blok sporočila dolžine 512 bitov. Pred izračunom zgostitve izvede 3 korake predprocesiranja:

1. Dopolnjevanje vhodnega sporočila M do večkratnika 512 bitov.
2. Razstavljanje dopolnjenega sporočila v bloke.
3. Nastavitev začetne vrednosti verižne spremenljivke.

Dopolnjevanje vhodnega sporočila poteka tako, da se na koncu sporočila najprej doda bit 1, sledi k ničelnih bitov in na koncu še binarna predstavitev dolžine originalnega sporočila. Število k je najmanjša nenegativna rešitev enačbe:

$$l + 1 + k \equiv 448 \bmod 512,$$

kjer je l dolžina sporočila M (v bitih). Nato se dopolnjeno sporočilo razčleni v N 512 bitnih blokov $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$. Vsak blok $M^{(i)}$ lahko predstavimo kot 16 32-bitnih besed $M_0^{(i)}, M_1^{(i)}, \ldots, M_{15}^{(i)}$.

Po inicializaciji algoritma se začne iterativno računanje zgostitve s kompresijsko funkcijo, ki kot vhod vzame 512 bitni blok sporočila in 256 bitno verižno spremenljviko ter vrne novo 256 bitno vrednost verižne spremenljivke. Za prvo iteracijo velja, da je vrednost verižne spremenljivke preddefinirana in fiksna ($H^{(0)}$), pri ostalih iteracijah pa je vhodna verižna spremenljivka enaka izhodu predhodne iteracije. Notranje stanje kompresijske funkcije predstavlja

tako imenovani register stanja, ki hrani trenutno zgostitev bloka sporočila. Sestavljen je iz 8 32 bitnih besed A,...,H.

Delovanje kompresijske funkcije pri SHA-256 lahko razdelimo v 2 fazi (slika 4.1):

- Postopek razširitve bloka sporočila[1].
- Računanje zgostitve.

Razširitev bloka sporočila pretvori vhodni 512 bitni blok v razširjen blok, ki ga sestavlja 64 32 bitnih besed W_i (skupaj 2048 bitov). Proces je definiran z rekurzivno formulo:

$$W_i = \begin{cases} M_i & 0 \leq i < 16 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & 16 \leq i < 64 \end{cases} \tag{4.1}$$

kjer sta funkciji $\sigma_0(x)$ in $\sigma_1(x)$ pri SHA-256 definirani takole:

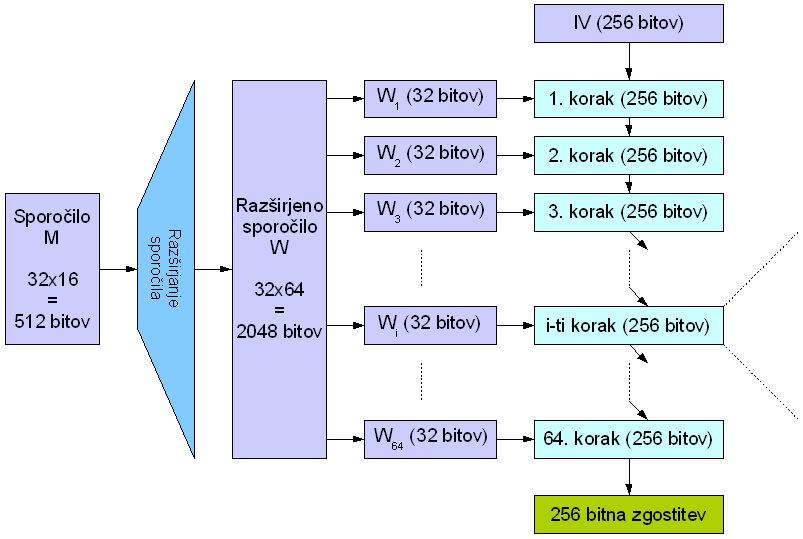
$$\begin{aligned} \sigma_0(x) &= ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x), \\ \sigma_1(x) &= ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x). \end{aligned}$$

Računanje zgostitve se izvede v 64 enakih zaporednih korakih, kjer je vsak korak (slika 4.2(b)) definiran z naslenjimi enačbami (konstanta K_i je za vsak korak različna, seštevanje je po modulu 2^{32}):

$$\begin{aligned} T_1 &= H_i + \Sigma_1(E_i) + f_{IF}(E_i, F_i, G_i) + K_i + W_i) \\ T_2 &= \Sigma_0(A_i) + f_{MAJ}(A_i, B_i, C_i) \\ A_{(i+1)} &= T_1 + T_2 \\ B_{(i+1)} &= A_i \\ C_{(i+1)} &= B_i \\ D_{(i+1)} &= C_i \\ E_{(i+1)} &= D_i + T_1 \\ F_{(i+1)} &= E_i \\ G_{(i+1)} &= F_i \\ H_{(i+1)} &= G_i \end{aligned} \tag{4.2}$$

Bolj natančno, v vsakem koraku se izračunata novi vrednosti dveh spremenljivk registra stanja. Operacija posodobitve stanja je odvisna od ostalih 6

[1] Angl. Message Expansion



Slika 4.1: Delovanje kompresijske funkcije pri SHA-256.

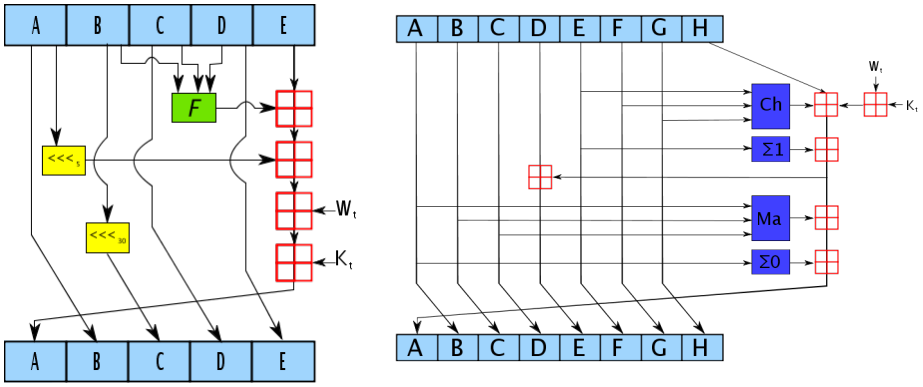
spremenljivk registra stanja, ene besede iz razširjenega sporočilnega bloka W_i $(0 \leq i < 64)$, koračne konstante K_i, dveh Boolovih (Selection in Majority) in dveh linerarnih $GF(2)$ funkcij:

$$\begin{aligned} f_{IF}(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ f_{MAJ}(x,y,z) &= (x \wedge y) \oplus (x \wedge y)(y \wedge z) \\ \Sigma_0(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \\ \Sigma_1(x) &= ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \end{aligned}$$

Opazimo lahko, da 4 zaporedni koraki posodobijo celoten register stanja: D in H, C in G, B in F ter A in E po vrsti. Skupno ga kompresijska funkcija posodobi 16 krat. Po 64 korakih kompresijska funkcija prišteje (po modulu 2^{32}) vhodni verižni spremenljivki trenutno vrednost registra stanja (angl. Feed forward) in tako dobimo novo vrednost verižne spremenljivke. Pri zadnji iteraciji je to tudi rezultat zgoščevalne funkcije.

Slika 4.2 prikazuje primerjavo med korakoma zgoščevalnih funkcij SHA-1 in SHA-256. Prva opazna razlika je ta, da ima SHA-256 daljše notranje stanje (8 proti 5 32 bitnih besed). Tudi kompleksnost koraka je pri SHA-256 večja, saj uporablja dodatni Boolovi in $GF(2)$ funkciji. Razlike med SHA-224 in SHA-256 niso velike [1] (analogno velja tudi za SHA-384 in SHA-512):

- Različne IV verižne spremenljivke H_0.



(a) SHA-1 (b) SHA-256

Slika 4.2: En korak transformacije pri SHA-1(a) in SHA-256(b). SHA-256 je bolj kompleksna, saj ima daljše notranje stanje in uporablja dodatne funkcije.

- Izhod zgoščevalne funkcije SHA-224 je 224 najbolj levih bitov izhoda zgoščevalne funkcije SHA-256.

Razlike med SHA-256 in SHA-512 pa so [1]:

- Različne dolžine besed (32/64 bitov) in velikosti blokov (512/1024 bitov).
- Različno število korakov kompresijske funckcije (64/80).
- Različne definicije funkcij $\sigma_0(x), \sigma_1(x), \Sigma_0(x)$ in $\Sigma_1(x)$.
- Različne IV (H_0) in koračne konstante (K_i).

4.2 Implementacija SHA-256

V tem poglavju bo prikazana implementacije zgoščevalne funkcije SHA-256[2] v jeziku ANSI C. Specifikacija FIPS180-2 [1] je primarna referenca za implementacijo, ker poleg opisa zgoščevalne funkcije vsebuje tudi 3 testne nize znakov za vsako zgoščevalno funkcijo iz družine SHA-2. Zgoščevalne funkcije družine SHA-2 so optimizirana za delovanje na big-endian arhitekturah.

Ker specifikacija [1] javnega programskega vmesnika (API) ne obravnava, smo kot predlogo uporabili dokument, ki definira API za kandidate natečaja SHA-3 [6] (več o SHA-3 v 6. poglavju):

[2]Z minimalnimi dopolnitvami kode dobimo implementacijo SHA-224 skoraj "zastonj".

```
HashReturn Init(hashState *state, int hashbitlen);

HashReturn Update(hashState *state, const BitSequence *data,
                  DataLength databitlen);

HashReturn Final(hashState *state, BitSequence *hashval);

HashReturn Hash(int hashbitlen, const BitSequence *data, DataLength
                databitlen, BitSequence *hashval);
```

Najpomembnejša podatkovna struktura je `hashState`, ki vsebuje vso potrebno informacijo za opis trenutnega stanja zgoščevalne funckije. Sestavljata ga register stanja in medpomnilnik za bloke vhodnega sporočila:

```
typedef struct {
   DataLength bitCount;
   uint32t hash[8];
   uint32t buffer[16];
} Sha256Context;

typedef struct {
   int hashbitlen;
   Sha256Context sha256Ctx;
} hashState;
```

Na najvišjem nivoju zgoščevanja je funkcija `Hash()`, ki za celovito obdelavo podatkov zaporedno pokliče naslednje tri funkcije:

- Funkcija `Init()` - skrbi za incializacijo strukture `hashState`, torej nastavitev začetne vrednosti registra stanja in prazenje medpomnilnika.
- Funkcija `Update()` - zaporedno razbija vhodno sporočilo na polne 512 bitne bloke in jih procesira s kompresijsko funkcijo. Morebitni ostanek sporočila, ki ne zapolni celega bloka, se shrani za nadaljnjo obdelavo. Funkcijo lahko pokličemo tudi večkrat zapored[3].
- Funkcija `Final()` - Obdela preostanek sporočila v nepolnem bloku. Najprej izvede dopolnitev bloka. Če je medpomnilnik preveč zaseden, da se dopolnitev izvede v celoti, se le-ta prenese v naslednji blok (kompresijska funkcija se pokliče največ dvakrat). Po zaključku funkcije je v registru stanja tudi končna vrednost zgostitve.

[3]Uporabno v primeru, ko vhodno sporočilo prihaja po delih.

Opazimo lahko, da se dopolnjevanje sporočila prestavi praktično na konec izračuna zgostitve (funkcija `Final()`), ker je pomembno samo pri zadnjem bloku. Na little-endian arhitekturah je potrebno izvesti še dodatno pretvorbo podatkov v pravilno obliko, ker algoritem predpostavlja big-endian arhitekturo. To pretvorbo lahko izvedemo znotraj kompresijske funkcije ali pa pred njenim klicem (torej v funkcijah `Update()` in `Final()` po(pri) branju vhodnega sporočila). Koda za pretvorbo:

```
#define REVERSEBYTEORDER32(w) { \
   (w) = (w >> 16) | (w << 16); \
   (w) = ((w \& 0xff00ff00) >> 8) | ((w \& 0x00ff00ff) << 8); \
}
```

Do sedaj smo se ukvarjali predvsem z implementacijo strukture DM, ki interno kliče kompresijsko funkcijo. Prva vloga kompresijske funkcije je razširanje vhodnega sporočila iz 16 v 64 besed, čemur ustreza pomnilnik velikosti 2048 bitov. Namesto tega lahko uporabimo medpomnilnik za vhodne podatke iz podatkovne strukture `Sha256Context` in ga obravnavamo kot krožni medpomnilnik (velikosti 512 bitov). Rekurzivna enačba za razširjanje (4.1) se zato spremeni (indekse računamo po modulu 16):

$$\begin{aligned} W_{i \bmod 16} &= \sigma_1(W_{(i+14) \bmod 16}) + W_{(i+9) \bmod 16} + \\ &\quad \sigma_0(W_{(i+1) \bmod 16}) + W_{i \bmod 16} \end{aligned}$$

kjer je $16 \leq i < 64$. Taka rešitev je sicer počasnejša, ker so potrebne dodatne operacije nad indeksi, vendar prihranimo 48 dodatnih besed. Druga vloga kompresijske funckije je posodobitev stanja zgoščevalne funckije, kjer se 64 krat izvede en korak (4.3). Koda za jedro kompresijske funckije:

```
do {
   T1 = h + Sigma1256(e) + CH(e, f, g) + sha256K[j] + W[j];
   T2 = Sigma0256(a) + MAJ(a,b, c);
   h = g; g = f; f = e; e = d + T1;
   d = c; c = b; b = a; a = T1 + T2;
   j++;
} while (j < 16);

do {
   MSGEXPANSION256(j);
   T1 = h + Sigma1256(e) + CH(e, f, g) + sha256K[j] + W[j & 15];
   T2 = Sigma0256(a) + MAJ(a, b, c);
   h = g; g = f; f = e; e = d + T1;
   d = c; c = b; b = a; a = T1 + T2;
   j++;
} while (j < 64);
```

Osnovna implementacija je s tem zaključena. Na spletu lahko najdemo kar nekaj odprtokodnih implementacij družine SHA-2 (na primer OpenSSL[4], Crypto++[5] in nekaj implementacij posameznikov). Vse so lahko dober vir informacij o algoritmu in o različnih možnostih optimizacije. A z izboljšanjem hitrost izvajanja algoritmov lahko zmanjšamo njegovo prenosljivost na različne platforme. Pri prenosljivosti poznamo težave z vrstnim redu shranjevanja operandov (little/big endian), velikostjo operandov, ki jih prevajalniki oziroma arhitektura podpirata (64/32 bitni), in različnimi standardi jezikov ter prevajalnikov. Potrebna učinkovitost neke implementacije je odvisna od njene uporabe. V večini primerov zelo hitra implementacija ni potrebna. Sicer je nujno dele algoritma implementirati v zbirniku, kjer lahko dodatno izkoristimo posebne strojne ukaze (npr. MMX, SSE). Žal tako zmanjšamo prenosljivost. Obstaja več splošnih priporočil za optimizacijo že pri kodiranju, kot so razvijanje zank, minimizacija števila skokov in izogibanje skokom v zankah, uporaba hitrejših bitnih operacij, inverzija zank in podobno.

Kompresijska funkcija je kritična glede hitrosti, saj se posodobitev stanja izvede 64 krat za vsak blok sporočila. Optimiziramo jo lahko na primer z metodo ravijanja zank. Omenili smo že, da se v vsakem koraku izračunata samo dve spremenljivki registra stanja. Namesto menjave spremenljivk s prirejanjem vrednosti, spremenljivke rotiramo. V izhodiščni položaj se vrnejo vsakih 8 korakov. Koda za izboljšan korak:

```
#define SHA256ONEROUND(a, b, c, d, e, f, g, h, k, w) \
   h += Sigma1256((e)) + CH((e), (f), (g)) + k + w; \
   d += h; \
   h += Sigma0256((a)) + MAJ((a), (b), (c))
```

in spremenjena kompresijska funkcija z razvito zanko:

```
...
SHA256ONEROUND(a,b,c,d,e,f,g,h,sha256K[ 0],W[ 0]);
...
SHA256ONEROUND(b,c,d,e,f,g,h,a,sha256K[15],W[15]);
SHA256ONEROUND(a,b,c,d,e,f,g,h,sha256K[16],MSGEXPANSION256( 0));
SHA256ONEROUND(h,a,b,c,d,e,f,g,sha256K[17],MSGEXPANSION256( 1));
SHA256ONEROUND(g,h,a,b,c,d,e,f,sha256K[18],MSGEXPANSION256( 2));
SHA256ONEROUND(f,g,h,a,b,c,d,e,sha256K[19],MSGEXPANSION256( 3));
SHA256ONEROUND(e,f,g,h,a,b,c,d,sha256K[20],MSGEXPANSION256( 4));
SHA256ONEROUND(d,e,f,g,h,a,b,c,sha256K[21],MSGEXPANSION256( 5));
SHA256ONEROUND(c,d,e,f,g,h,a,b,sha256K[22],MSGEXPANSION256( 6));
SHA256ONEROUND(b,c,d,e,f,g,h,a,sha256K[23],MSGEXPANSION256( 7));
```

[4]http://www.openssl.org
[5]http://www.cryptopp.com

```
SHA256ONEROUND(a,b,c,d,e,f,g,h,sha256K[24],MSGEXPANSION256( 8));
...
SHA256ONEROUND(b,c,d,e,f,g,h,a,sha256K[63],MSGEXPANSION256(15));
```

Pohitritev je odvisna od velikosti predpomnilnikov, saj postane kode z razvijanjem nekajkrat daljša. Če nismo pazljivi, lahko pri procesorjih z majhnimi predpomnilniki dosežemo nasprotni učinek, ker so prenosi ukazov iz pomnilnika bolj pogosti. Nekaj osnovnih lastnosti implementacije lahko hitro ocenimo. Operacije, ki jih algoritem uporablja, so preproste. Po drugi strani pa podatkovne odvisnosti ne dopuščajo veliko paralelizma. Izračun naslednjega koraka potrebuje znane vrednosti spremenljivk A,...,H iz prejšnjega koraka, kar vsili zaporednost. Cevovod procesorja zaradi te lastnosti ni dobro izkoriščen. Poraba pomnilnika je približno enaka velikosti podatkovne strukture `hashState` - 800 bitov oziroma 100 bajtov. To je za današnje predpomnilnike malo in je lahko celotna struktura blizu procesorja.

Poglavje 5

Analiza SHA-256

5.1 Varnost

Primerjava družine zgoščevalnih funkcij SHA-2 s predhodnico SHA-1[7]:

- Vsak korak kompresijske funkcije uporabi linearni $GF(2)$ funkciji Σ_0 in Σ_1, ki omogočajo hitrejšo razpršitev kot leve bitne rotacije.
- Vsak korak uporabi obe nelinearni funkciji *Majority* in *Choice* namesto ene izmed treh.
- Vsak korak izračuna dve spremenljivki registra stanja namesto ene.
- Vsi koraki so enaki (razen koračne konstante K_t) v primerjavi s štirimi različnimi tipi korakov, kjer vsak uporablja svojo koračno konstanto.
- Uporaba seštevanja po modulu in funkcij σ_0 in σ_1 pri razširjanju bloka sporočila omogoča hitrejšo razpršitev kot bitna operacija XOR.

Gilbert in Handschuch sta v [7] pokazala, da uspešnega drsnega napada (angl. Slide attack) na SHA-1 ne moremo uporabiti pri SHA-256, ker vsak korak kompresijske funkcije uporablja enolično koračno konstanto. Napada s trki na SHA-0 prav tako ne moremo uporabiti pri SHA-256 zaradi uporabe funkcije σ_i pri razširjanju sporočila (rotacije in pomiki). Hkrati sta ugotovila, da je poenostavljena različica algoritma (uporaba simetričnih IV in koračnih konstant, seštevanje po modulu nadomesti XOR) presenetljivo slaba, saj pri simetričnih vhodih dobimo simetrične zgostitve. Napadi na SHA-2 v zadnjem času so diferenčni napadi z iskanjem na krajše, a sicer nespremenjene različice algoritma SHA-256/512. Tabela 5.1 prikazuje dosežke pri kriptoanalizi krajših

zgoščevalnih funkcij SHA-256/512. Čeprav so napadi vedno bolj uspešni, trenutno ne ogrožajo celotne zgoščevalne funkcije.

Poznamo dve vrsti diferenčnih napadov na SHA-2:

- Linearni napadi - napadi na linearizirano različico algoritma z uporabo lokalnih trkov.
- Nelinearni napadi - napadi na nespremenjeno različico algoritma z uporabo lokalnih trkov.

Iskanje lokalnih trkov ponavadi poteka pri lineariziranih različicah zgoščevalne funkcije. Potem se izračuna verjetnost, da dobljen lokalni trk velja tudi za dejansko zgoščevalno funkcijo. Avtorja v [8] sta predstavila 4 lastnosti zgoščevalne funkcije SHA-2, ki se izkoriščajo pri napadih s krajšim številom korakov:

1. Pri SHA-2 obstaja 9 koračni lokalni trk (z uporabo diferenciala XOR) z verjetnostjo 2^{-39}ali manj. Po drugi strani so z uporabo aditivnega diferenciala našli 9 koračni trk z verjetnostjo 1, kar kaže na odpornost kompresijske funkcije na linearne diferenciale. Zato je verjetnost linearnega napada pri SHA-2 je zelo majhna v primerjavi z verjetnostjo nelinearnega napada. Glede na to, da so linearni napadi pri SHA-0 in SHA-1 uspešni, lahko sklepamo na kriterij pri načrtovanju SHA-2. Z uporabo seštevanja po modulu je postala kompresijska funkcija odporna na linearne napade.

2. V vsakem koraku se nelinearno posodobita samo spremenljivki registra stanja A in E . Obstaja pa preprosta relacija, kjer lahko nadziramo vrednost spremenljivke E v koraku i z vrednostmi spremenljivke A v korakih i do $i-4$:

$$E_i = A_{i-4} + A_i - \Sigma_0(A_{i-1}) - f_{MAJ}(A_{i-1}, A_{i-2}, A_{i-3}). \qquad (5.1)$$

   Preprosta relacija med spremenljivkama in primerno diferencialno obnašanje funkcije f_{MAJ} zmanjša uporabnost posodobitve dveh spremenljivk.

3. Funkciji $\Sigma_0$in Σ_1 sta obrnljivi linearni transformaciji. Obe imata 0 in -1 kot fiksni točki ($\Sigma(x) = x$), kar poveča verjetnost nelinearnih lokalnih trkov. Še več, obe funkciji imata skupne fiksne točke pri vseh algoritmih družine SHA-2.

4. Napadi se gradijo z uporabo tehnike pokvari-popravi. Med koraka i in $i+8$ se vstavi 9 koračni lokalni trk. Vse razlike v besedah sporočila so po koraku $i+8$ enake nič.

Avtor	**SHA-x**	**N**	**Verj.**	**Št. klicev**	**Lokalni trk**	**Vrsta napada**
Mendel et al.	256	18	/	/	GH[1]	Linearni
Sanadhya, Sarkar	256	18	/	/	SS[2]	Linearni
Nikolić, Biryukov	256	20	$\frac{1}{3}$	/	NB[3]	Nelinearni
Nikolić, Biryukov	256	21	2^{-19}	/	NB	Nelinearni
Sanadhya, Sarkar	256/512	18,20	1	1	SS	Nelinerani
Sanadhya, Sarkar	256	21	2^{-15}	/	SS	Nelinerani
Sanadhya, Sarkar	256/512	21	1	1	SS	Nelinerani
Sanadhya, Sarkar	256/512	22	1	1	SS	Nelinerani
S. Indesteege et al.	256	23	/	2^{18}	NB	Nelinerani
S. Indesteege et al.	256	24	/	$2^{28.5}$	NB	Nelinerani
Sanadhya, Sarkar	256	23	/	$2^{12.5}$	SS/NB	Nelinerani
Sanadhya, Sarkar	256	24	/	$2^{28.5}$	SS/NB	Nelinerani
Sanadhya, Sarkar	256	24	/	$2^{14.5}$	SS/NB	Nelinerani

Tabela 5.1: Napadi na skrajšane različice algoritmov družine SHA-2 (vir [5]). N predstavlja število korakov, napor pri iskanju pa lahko izrazimo z verjetnostjo uspeha ali pa s številom klicev skrajšane kompresijske funkcije. Pri zadnjem napadu potrebujemo tabelo z 2^{32} vnosi po 8 B.

Avtorja v [8] tudi pokažeta, da je možno preprosto dokazati nenaključnost katerekoli zgoščevalne funkcije družine SHA. Wang in Yu (Eurocrypt 2008) sta dokazala enako, torej nenaključnost kompresijske funckije do koraka 39 algoritma SHA-256 z uporabo 33 koračne diferencialne poti. Lastnost naključnosti ne vpliva na varnostne lastnosti iz 1. poglavja. Vendar se zgoščevalne funckije uporabljajo v različnih situacijah in na različne načine, kot je na primer generiranja naključnega niza znakov. Ker se kompresijska funkcija SHA ne obnaša tako naključno, kot bi pričakovali, velja previdnost pri uporabi.

Idejo pokvari-popravi uporabimo za iskanje lokalnega trka. Če v koraku i vpeljemo spremembo sporočila (popačenje), potem lahko s spremembami besed v naslednjih korakih dosežemo, da se prvotno sprememba izniči v koraku $i+8$. Tako dobimo 9 koračni lokalni trk. Napad NB in njegove izpeljanke za iskanje r koračnega trka lahko v grobem opišemo takole:

1. Izberi primeren i in vstavi lokalni trk med koraka i in $i+8$.
2. Zagotovi, da so razlike v besedah $\delta W_j = 0$ za $j = i+9, ..., r-1$, kar pripelje do trka v r korakih. Torej, vpeljana razlika ne vpliva na spremenljivke stanja v koraku j, če je j daleč od i.

Napad NB je uporabil en lokalen trk, ki ga izbere tako, da ne povzroči razlik v besedah sporočila od konca lokalnega trka do koraka 20. Tako dobimo 20 koračni trk za SHA-256 z enako verjetnostjo kot je verjetnost enega lokalnega trka. S podobnimi idejami raste število korakov kompresijske funckije. Spreminjajoče vrednosti spremenljivk med potekom lokalnega trka nimajo vpliva v kasnejših korakih.

5.2 Performančna analiza

Hitrost družine SHA-2 smo primerjali z nekaterimi znanimi zgoščevalnimi funkcijami pri dveh odprtokodnih implementacijah kriptografksih primitivov in protokolov: Crypto++ in OpenSSL. Čeprav ne moremo pričakovati od tako obsežnih knjižnic, da so vsi algoritmi performančno optimizirani, lahko vsaj primerjamo zgoščevalne funkcije znotraj posamezne knjižnice, kjer naj bi veljali enaki pogoji za vse.

Rezultati pri knjižnici Crypto++ kažejo (glej tabelo 5.2), da sta algoritma SHA-256/512 na 32 bitni x86 platformi počasnejša od vseh ostalih razen zgoščevalne funkcije Whirpool. Primerjava z SHA-1 da približno 1,5-2 kratno razliko v korist SHA-1, pri MD5 je razlika še večja. Na 64 bitni platformi je položaj podoben s to razliko, da se povečajo zmogljivosti zgoščevalne

funkcije SHA-512, ki je optimizirana za 64 bitne okolja. Čeprav je v primerjavi z MD5 približno 2 krat počasnejša, razlika v primerjavi s SHA-1 ni velika. Druga knjižnica s širokim naborom zgoščevalnih funkcij je OpenSSL.

Algoritem	**Intel Core 2 (1,83 GHz)**		**AMD Opteron (2,4 GHz)**	
	MB/s	**Cikli/B**	**MB/s**	**Cikli/B**
MD5	258	6,8	376	6,1
SHA-1	155	11,3	216	10,6
SHA-256	81	21,5	106	21,5
SHA-512	99	17,6	178	12,9
Tiger	217	8,0	320	7,2
Whirpool	58	30,0	63	36,1
RIPEMD-160	108	16,1	165	13,9
RIPEMD-320	111	15,8	174	13,1
RIPEMD-128	155	11,3	255	9,0
RIPEMD-256	159	11,0	263	8,7

Tabela 5.2: Primerjava zmogljivosti zgoščevalnih funkcij iz knjižnice Crypto++ na dveh procesorjih: Intel Core 2 in AMD Opteron (oba imata little-endian arhitekuro). Prvi test se je izvajal v okolju Windows XP (32 bitno) na enem jedru procesorja Intel Core 2, uporabljen pa je bil prevajalnik Microsoft Visual C++ 2005 SP1 (vklopljena optimizacija za hitrost) in ukazi zbirnika x86/MMX/SSE2 (pri SHA-512, Tiger in Whirpool). Drugi test se je izvajal v okolju Unix (64 bitno), uporabljen je bil prevajalnik GCC 4.1.1 (optimizacija -O2) in ukazi zbirnika x86-64/MMX/SSE2 (pri Whirpool).

Projekt eBASH[4] (ECRYPT Benchmarking of All Submitted Hashes) primerja učinkovitost zgoščevalnih funkcij tako, da meri čas zgoščevanja sporočila dolžine 0,1,...,4096 bajtov na različnih procesorjih z različnimi prevajalniki in različnimi parametri prevajanja. Tabela 5.4 prikazuje meritve na procesorju Intel Core 2 Quad, ki ima little-endian arhitekturo. Zgoščevalne funckije SHA-2 so v primerjavi z SHA-1 in MD-5 počasnejše (za faktor 2 ali več), tako v primeru 32 kot 64 bitnih okolij. SHA-512 se je v 32 bitnem okolju izkazala slabo, sploh pri krajšnih sporočilih, medtem ko so se njene prednosti pokazale v 64 bitnem okolju. SHA-256 se je celo malo bolje izvajala v 64 kot v 32 bitnem (to lahko pripišemo izkoriščanju širših podatkovnih poti), vendar razlike pri daljših sporočilih niso velike. Na procesorju PowerPC G5 (big-endian arhitektura) je

[4] `http://bench.cr.yp.to/ebash.html`

Algoritem	Dolžina sporočila					
	Dolgo	**4096B**	**1536B**	**576B**	**64B**	**8B**
MD5	5,44 5,48	5,58 5,62	5,82 5,85	6,44 6,44	14,34 14,34	78,75 69,75
SHA-1	/ 7,76	/ 7,92	/ 8,30	/ 9,39	/ 23,20	/ 122,62
SHA-256	23,39 20,30	23,83 20,75	24,57 21,52	26,72 23,62	55,41 49,78	274,50 240,75
SHA-512	98,43 13,28	126,30 13,91	133,80 14,92	140,34 16,25	287,02 39,80	2279,25 312,75
Whirpool	98,43 44,74	99,77 45,27	102,03 46,32	108,23 49,08	186,75 83,76	723,38 322,88
RIPEMD-160	/ 14,22	/ 14,59	/ 15,11	/ 16,64	/ 36,42	/ 178,88

Tabela 5.3: Primerjava zmogljivosti zgoščevalnih funkcij iz knjižnice OpenSSL na procesorju Intel Core 2 Quad Q6600 (2400 MHz, little-endian arhitektura, prevajalnik GCC 4.2.3). Vse vrednosti so v ciklih/B. V primeru dveh vrednosti je zgornja vrednost izmerjena v 32 bitnem, spodnja pa v 64 bitnem okolju Linux. Za dolgo sporočilo je rezultat ekstrapoliran glede na razliko časa zgoščevanja sporočila dolžin 4096B in 2048B.

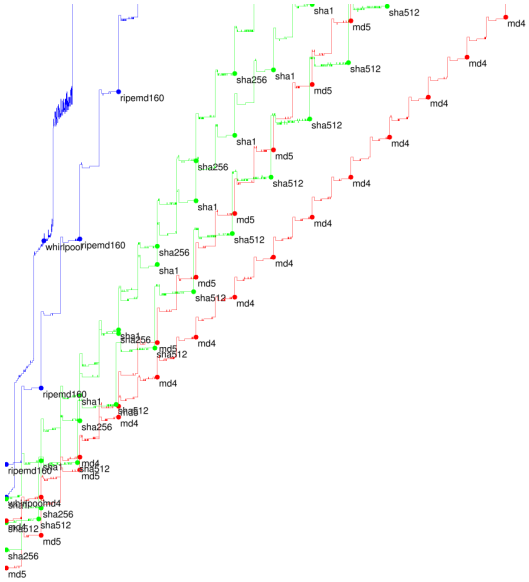
položaj drugačen, saj je SHA-512 pri daljših sporočilih celo hitrejši od MD5 in SHA-1 tako v 32 kot 64 bitnem načinu izvajanja (razlike sicer niso velike). Presenetljivo je SHA-256 počasnejši od vseh treh za faktor 1,5.

Sliki 5.2 in 5.1 grafično prikazujeta primerjavo zmogljivosti zgoščevalnih funkcij na little in big-endian arhitekturah (po vrsti). Opazimo lahko, da je na big-endian arhitekturah razlika med algoritmi MD5, SHA-1 in SHA-2 manjša kot na little-endian arhitekturah. Razlog je v tem, da je družina SHA-2 prilagojena za big-endian arhitekturo in zato ni potrebno pretvarjati podatkov v pravilno obliko.

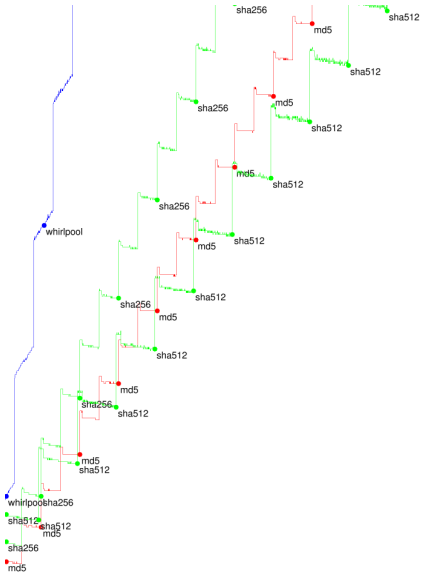
V splošnem lahko trdimo, da se večja kompleksnost kompresijske funkcije (in s tem večje varnosti) pri družini SHA-2 odraža tudi v manjši hitrosti v primerjavi s SHA-1 in MD5. V nekaterih primerih je ta razlika tudi faktor 2 ali več, čeprav rezultati kažejo, da so razlike odvisne od arhitekture procesorja, dolžine podatkov, prevajalnika in parametrov prevajanja.

Algoritem	**Dolžina sporočila**					
	Dolgo	**4096B**	**1536B**	**576B**	**64B**	**8B**
MD5	16,55 18,52	16,88 18,93	17,42 19,57	18,96 21,35	37,50 44,06	172,50 202,50
SHA-1	16,82 /	17,72 /	19,26 /	23,33 /	75,00 /	457,50 /
SHA-256	22,27 25,37	22,76 25,96	23,55 26,91	25,73 29,38	52,50 60,94	247,50 277,50
SHA-512	14,65 14,62	15,38 15,45	16,60 16,72	18,12 18,44	45,94 47,81	360,00 397,50
Whirpool	68,00 70,55	68,58 71,16	70,00 72,58	73,65 76,67	122,81 128,44	472,50 480,00
RIPEMD-160	38,17 /	39,33 /	41,29 /	46,46 /	113,44 /	600,00 /

Tabela 5.4: Primerjava zmogljivosti zgoščevalnih funkcij iz knjižnice OpenSSL na procesorju IBM PowerPC G5 970 (2000 MHz, big-endian arhitektura, prevajalnik GCC 4.0.1). Vse vrednosti so v ciklih/B. V primeru dveh vrednosti je zgornja vrednost izmerjena v 32 bitnem, spodnja pa v 64 bitnem okolju. Za dolgo sporočilo je rezultat ekstrapoliran glede na razliko časa zgoščevanja sporočila dolžin 4096B in 2048B.

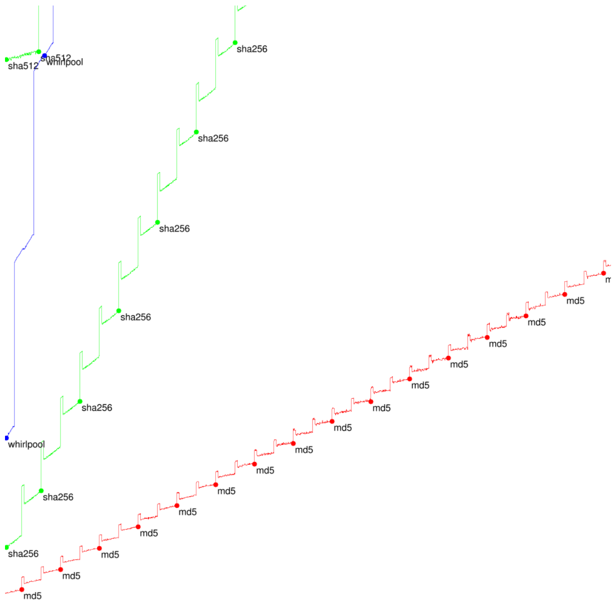


(a) ppc32

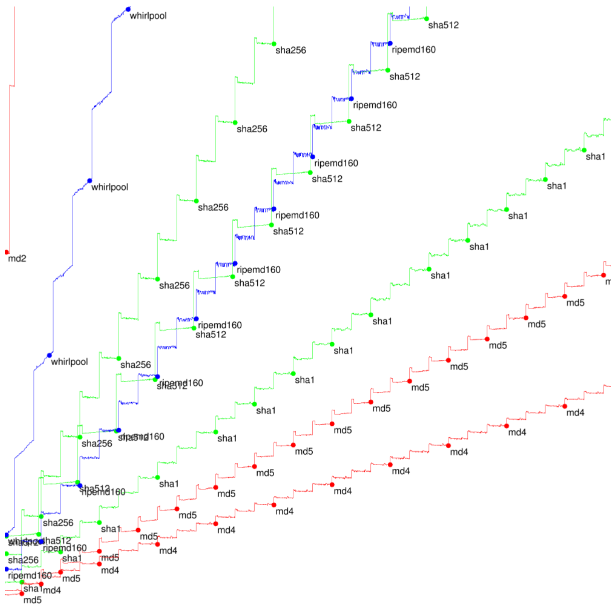


(b) ppc64

Slika 5.1: Graf časa izvajanja (0 - 20000 ciklov) v odvisnosti od dolžine sporočila (0 - 2000 B) za različne zgoščevalne funkcije na 32 bitni (a) in 64 bitni (b) PowerPC platformi. Diagonala grafa predstavlja 10 ciklov/B. Stopničasta oblika krivulje je posledica dejstva, da zgoščevalne funkcije obdelujejo sporočilo v blokih.



(a) x86



(b) amd64

Slika 5.2: Graf časa izvajanja (0 - 20000 ciklov) v odvisnosti od dolžine sporočila (0 - 2000 B) za različne zgoščevalne funkcije na 32 bitni (a) in 64 bitni (b) Intel platformi. Diagonala grafa predstavlja 10 ciklov/B. Stopničasta oblika krivulje je posledica dejstva, da zgoščevalne funkcije obdelujejo sporočilo v blokih.

Poglavje 6

Povzetek in prihodnost

Po uspešnih napadih na MD5 in SHA-1 se je pokazalo, da 160 bitni povzetki dolgoročno ne zagotavljajo zadostne varnosti. Velikost povzetka je sicer odvisna od okolja uporabe, vendar v varnostno kritičnih okoljih potrebujemo vsaj 256 bitov dolge povzetke. Kot kažejo rezultati kriptoanalize, slepo daljšanje povzetkov avtomatično ne zagotavlja večje varnosti. Zato je potrebna migracija, a možnosti ni veliko. Poleg zgoščevalnih funkcij RIPEMD-x, Tiger in Whirpool je družina SHA-2 je ena izmed alternativ. Njene prednosti so:

1. Je standard. NIST (National Institute of Standards and Technology) umika SHA-1 v korist SHA-2 (do leta 2010).
2. Fleksibilnost pri uporabi z drugimi kriptografskmi primitivi in protokoli, ker nudi različne dolžine povzetkov in podporo za 64 bitne arhitekture.
3. Dosedanji napadi niso ogrozili polnih različic algoritmov SHA-2. Hkrati so odporni na napade, ki so zlomili SHA-1 in MD5.

Kot slabosti lahko omenimo:

- Manjša hitrost, predvsem na še zelo razširjenih 32 bitnih little-endian platoformah.
- Strukturna krhkost, saj tudi majhne spremembe algoritma odkrijejo nekatere slabosti.
- Iterativna konstrukcija MD in ostale slabosti, ki izhajajo iz načrtovalskih pristopov zgoščevalnih funkcij MDx.

Zgoščevalni funkciji Tiger in Whirpool kljub konstrukciji MD drugače uporabljata kompresijsko funkcijo, s čimer naj bi bili bolj odporni na analitične napade. Žal do tega trenutka nista dobili dovolj pozornosti in analiz, da lahko postaneta pravi alternativi SHA-2 in RIPEMD-x. NIST priporoča uporabo SHA-2 za vse nove aplikacije, hkrati pa se zaveda, da je to le začasni nadomestek. Zato je oktobra 2007 objavil razpis za projekt SHA-3 (podobno kot projekta eStream in AES). Cilj projekta je najti ustrezne alternative današnjim zgoščevalnim funkcijam do leta 2012. Nekatera pričakovanja:

- Raven varnosti vsaj enaka SHA-2 z močno izboljšano učinkovitostjo.
- Izogibnje slabim lastnostim konstrukcije DM (zaželjene so tudi druge konstrukcije).
- Visoka stopnja paralelizacije.
- Fleksibilnost za uporabo z drugimi kriptografskimi primitivi in protokoli, dolžine povzetkov 224,256,384 in 512 bitov.
- Napad na SHA-2 ne sme voditi v napad na SHA-3.
- Nastavljivi varnostni parametri (na primer število korakov).

Literatura

[1] National Institute of Standards and Technology (NIST), "FIPS 180-2: Secure Hash Standard (SHS)," feb. 2004. Dostopno na http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[2] R. Weis, S.Lucks, "Cryptographic Hash Functions, Recent Results on Cryptoanalysis and their Implications on System Security," SANE 2006. Dostopno na http://www.sane.nl/sane2006/program/final-papers/R10.pdf

[3] A. K. Lenstra, "Further progress in hashing cryptanalysis," feb. 2005. Dostopno na http://cm.bell-labs.com/who/akl/hash.pdf

[4] S. Lucks, "Design Principles for Iterated Hash Functions," sep. 2004. Dostopno na http://eprint.iacr.org/2004/253.pdf

[5] S. K. Sanadhya, P. Sarkar, "Attacking Step Reduced SHA-2 Family in a Unified Framework," jun. 2008. Dostopno na http://eprint.iacr.org/2008/271.pdf

[6] NIST, "ANSI C Cryptographic API Profile for SHA-3 Candidate Algorith Submissions," feb. 2008. Dostopno na http://www.csrc.nist.gov/groups/ST/hash/documents/SHA3-C-API.pdf

[7] H. Gilbert, H. Handschuch, "Security Analysis of SHA-256 and her sisters," v zborniku Selected Areas in Cryptography, Ottawa, Canada, avg. 2003, str. 175-193.

[8] S. K. Sanadhya, P. Sarkar, "Some Observations on Strengthening the SHA-2 Family," maj 2008. Dostopno na http://eprint.iacr.org/2008/272.pdf

[9] A. Menezes, P. van Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, okt. 1996, pogl. 9.

[10] I. Mironov, "Hash Functions: Theory, attacks, and applications," nov. 2005. Dostopno na `http://research.microsoft.com/users/mironov/papers/hash_survey.pdf`