Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Igor Alfirević

Tiger zgoščevalna funkcija

Tečaj iz kriptografije in računalniške varnosti

Seminarska naloga

Predavatelj:
Aleksandar Jurišić

Ljubljana, maj 2004

Kazalo

Povzetek

Dandanes popularne zgoščevalne funkcije kot so MD4 ali SNEFRU, so za razliko od blokovnih šifer zelo hitre. Čeprav so današnje zgoščevalne funkcije na prvi pogled uporabne v aplikacijah, ki zahtevajo hitro programsko zgoščevanje (izdelava povzetkov), najdemo pomanjkljivosti tudi pri teh algoritmih. Tako za MD4 kakor tudi za Snefru so odkrite pomankljivosti, kar poraja dvom v algoritme kot so MD5, SHA1, SNEFRU. Prav tako so vse navedene funkcije dizajnirane za 32 bitno arhitekturo, zato jih ni možno učikovito implementirati na 64 bitnih sistemih. V ta namen je bila razvita Tiger zgoščevalna funkcija, ki je optimizirana za delovanje na 64 bitnih sistemih in za katero do danes še ni odkrit učinkovit napad.

Uvod

Zgoščevalne funkcije uporabljamo za ugotavljanje verodostojnosti podatkov. Prvi način uporabe je preverjanje gesla. Prav tako se uporabljajo za preverjanje celovitosti sprejetega sporočila. Uporabne so tudi v druge namene (npr. generiranja semena za generator naključnih števil) vendar sta ti dve uporabi najpogostejši.

Težava nastane, kadar želimo geslo hraniti na sistemu, vendar, če nekdo dostopa do našega sistema, lahko prebere geslo in ga uporabi za dostop do nepooblaščenih podatkov. Očitno je, da gesla ne smemo shraniti kot čistopis. V teh primerih uporabimo zgoščevalne funkcije, za katere velja, da je računsko neizvedljivo iz povzetka poiskati začetno sporočilo. Pravimo, da je zgoščevalna funkcija enosmerna[8].

Uporabnik s pomočjo zgoščevalne funkcije izdela povzetek gesla ter ga shrani na sistem. Pri preverjanju gesla se izdela povzetek, ki se primerja z že shranjenim povzetkom. Če sta povzetka enaka je avtorizacija odobrena. Tudi če napadalcu uspe prebrati povzetek, mora poiskati takšno geslo, ki bo imelo enak povzetek ali na nek način iz povzetka izvedeti kakšno je bilo geslo kateremu povzetek pripada.

V drugem primeru želimo preveriti ali je prejeto sporočilo res enako poslanemu sporočilu. Tedaj mora pošiljatelj `A` poleg sporočila poslati še povzetek sporočila `H(A)`, kjer `H` predstavlja zgoščevalno funkcijo. Sprejemnik `B` sprejme sporočilo in povzetek. Iz prejetega sporočila izračuna nov povzetek ter ga primerja s prejetim povzetkom. Če sta povzetka enaka potem je sporočilo veljavno.

Ta način je občutljiv na vmesne napade, kjer oseba `C` prestreže sporočilo in povzetek osebe `A` ter pošlje osebi `B` 'podtaknjeno' sporočilo in ustrezen povzetek. Zato se zgoščevalne funkcije uporabljajo v kombinaciji z bločnimi šiframi, kjer zakodiramo le povzetek. Na ta način mora napadalec ugotoviti pošiljateljev privatni ključ, in šele nato lahko 'podtakne' lažno sporočilo.

V prvem delu seminarja so predstavljene osnovne lastnosti zgoščevalnih funkcij ter osnovni napadi, ki izkoriščajo lastnosti posameznih zgoščevalnih funkcij oziroma odkrivajo podrobno strukturo funkcij. Osrednji del se osredotoča na Tiger zgoščevalno funkcijo ter njeno implementacijo. V zaključku sledi še analiza funkcije in performančni testi Tiger funkcije v primerjavi z ostalimi zgoščevalnimi funkcijami.

Poglavje 1

Zgoščevalne funkcije

1.1 Osnovne lastnosti

Zgoščevalne funkcije preslikajo poljubno dolg niz znakov v blok konstantne dolžine, ki je nekakšen prstni odtis oziroma povzetek vhodnega niza. Od zgoščevalne funkcije pričakujemo, da[10]:

- je nemogoče najti dve različni sporočili, ki bi ju preslikala v isti blok;
- isto sporočilo vedno preslika v enak blok;
- iz zgoščevalnega bloka ni mogoče restavrirati sporočila (od tu ime one-way hash function;
- vsaka sprememba v sporočilu povzroči spremembo zgoščevalnega bloka;

Zgoščevalne funkcije z navedenimi lastnostmi zapišemo z naslednjima definicijama[3]:

Definicija 1
Enosmerna zgoščevalna funkcija, je funkcija, ki zadostuje pogojema:

- *parameter X je lahko poljubne dolžine in rezultat $H(X)$ ima fiksno dolžino n bitov;*
- *funkcija mora biti enosmerna v smislu, da je računsko nemogoče poiskati X pri podanem $H(X) = Y$. Prav tako je računsko nemogoče, pri podanem $H(X)$ in X, poiskati tak $X' \neq X$, da velja $H(X') = H(X)$;*

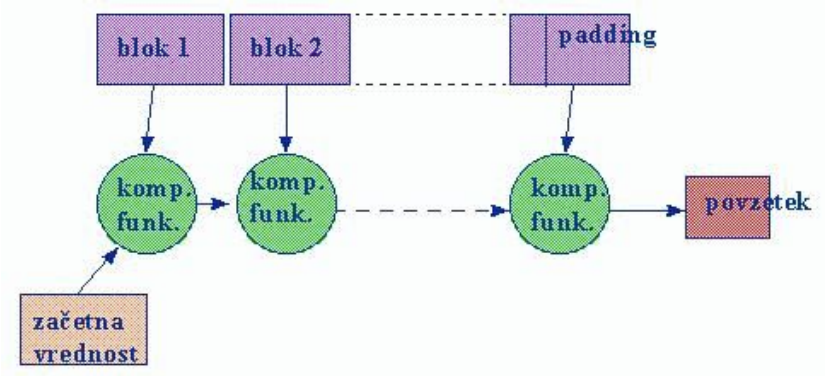
Definicija 2
Zgoščevalna funkcija, ki je odporna na trčenja zadostuje pogojem:

- *parameter X je lahko poljubne dolžine in rezultat $H(X)$ ima fiksno dolžino n bitov;*
- *zadoščati mora pogojem iz prve definicije*;
- *računsko ni mogoče poiskati dve različni sporočili, ki bi imeli enak rezultat;*

Rezultat mora torej enolično identificirati sporočilo. Zaradi tega so povzetki postali nepogrešljivi pri digitalnem podpisovanju, kjer zašifriramo samo povzetek, in kot indikatorji nespremenjenosti podatkov v postopkih za prenos podatkov. Ne smemo pa teh funkcij zamenjevati s kompresijskimi postopki (zip in podobnimi), kjer vedno lahko iz zgoščene datoteke dobimo nazaj prvotno datoteko.

1.2 Princip delovanja

Postopek se začne tako, da vhodno sporočilo razdelimo na bloke konstantne dolžine in konec dopolnimo do polnega bloka (padding). Potem zaporedoma obdelujemo bloke:



Slika 1.1: Sporočilo razbijemo na bloke fiksne dolžine (npr. 512 bitov), na koncu dodamo bite do polnega bloka.

Ena od možnosti je, da za zgoščevalno funkcijo uporabimo katerega od simetričnih algoritmov: vhodno sporočilo razbijemo na bloke take dolžine, kot ustrezajo algoritmu, prvi blok zašifriramo s ključem, vsak naslednji blok seštejemo (XOR) z

zašifriranim povzetkom prejšnjega bloka. Ta postopek je poznan pod imemom Message Authentication Code (MAC)[1]. Vendar se zaenkrat v glavnem uporabljajo posebej za to razviti algoritmi (MD5, SHA-1), ker so hitrejši.

1.3 Velikost povzetka

Kako dolg pa naj bi bil povzetek? Ali je 128 bitov dovolj, da ne bo prišlo do enakih povzetkov različnih sporočil oziroma, da v postopku ne bo kolizij?

Da se izpeljati oceno, da bo ob n-bitov dolgih povzetkih s polovično verjetnostjo prišlo do kolizije ob približno $1.17n^{1/2}$ zgostitvah naključnih nizov. Za 128-bitne povzetke torej velja, da bo med 2^{64} povzetki naključnih nizov ena kolizija z verjetnostjo 50%. Ta dolžina je ocenjena kot najmanjša še ustrezna. Izkoriščanje kolizij pri prekratkih povzetkih imenujejo "birthday attack". Ime ima po paradoksu rojstnega dne (birthday paradox): če kot povzetek za vsakega človeka izberemo obletnico njegovega rojstnega dne (n je število dni v letu, $n = 365$), iz zgornje formule dobimo rezultat 22.3[9].

To pomeni, da bosta od 23 naključno izbranih ljudi z verjetnostjo 50% vsaj dva imela isti rojstni dan.

Pri izbiri zgoščevalne funkcije moramo biti predvsem pozorni na hitrost in varnost. Če želimo hiter in ne preveč zanesljiv algoritem potem lahko uporabimo MD5. V primeru, da pa želimo več povdarka na varnosti potem uporabimo SHA-512. Končna odločitev katero zgoščevalno funkcijo uporabiti je na razvijalcu. Izbira je namreč odvisna od konkretnega problema, ki se pojavi v praksi.

Poglavje 2

Napadi na zgoščevalne funkcije

Razlikujemo več vrst napadov:

- napadi na osnovi dolžine izhodne funkcije;
- napadi na osnovi lastnosti črne škatle, ki je zgoščevalna funkcija;
- kriptoanalitični napadi, ki se spuščajo v podrobno stukturo funkcije;

Varnost zgoščevalnih funkcij je hevristična, saj samo nekaj počasnih implementacij lahko prevedemo na številne teoretične probleme. Zato velja pravilo naj bomo pri izbiri zgoščevalne funkcije kar se, da konzervativni[3].

2.1 Napadi na osnovi dolžine izhodne funkcije

Ta napad je odvisen samo od velikosti n rezultata zgoščevalne funkcije. Predpostavka je, da se zgoščevlna funkcija obnaša kot generator naključnih števil.

Za napade, ki potrebujejo omejeno velkost spomina in ne preveč dostopov v pomnilnik se smatra 2^{70} operaciji na robu zmogljivega. Na podlagi Moorovega zakona pa je 2^{80} operacij, potrebnih za napad zadostna varnost za naslednjih 5 do 10 let. Vendar bo že na skrajni meji varnosti v naslednjih 15 letih.

Za aplikacije, ki naj bi veljale vsaj 20 let, je potrebna zgradba zgoščevalne funkcije, katera ne bo omogočala napade z manj kakor 2^{90} operacijami.

Poznani napadi:

- naključni napad (Random attack);
- napad s paradoksom rojstnega dne (Birthday attack)[3];

2.2 Napadi na osnovi lastnosti kompresijske funkcije

Ti napadi so zasnovani na lastnostih (high level properties) zgoščevalne funkcije.

Poznani napadi:

- napad s srečanjem na sredini (Meet-in-the-middle attack);
- napad s popravljanjem blokov (Correcting-block attack);
- napad s fiksno točko (Fixed point attack);

Napad s srečanjem na sredini je izpeljanka napada s paradoksom rojstnega dne. Uporabljamo ga pri zgoščevalnih funkcijah pri katerih je lahko poiskati inverz kompresijske funkcije. Omogoča nam, da poiščemo besedilo z enako vrednostjo povzetka v času $2^{n/2}$ namesto v času 2^n[3]. Najenostavnejši način obrambe je povečanje dolžine povzetka.

Pri napadu s popravljanjem blokov imamo sporočilo in povzetek sporočila. Z majhnimi spremembami sporočila poskušamo poiskati spremenjeno sporočilo z enakim povzetkov[2]. Napadu se izognemo z močno razpršenostjo funkcije. Torej en bit spremembe vpliva na veliko bitov povzetka.

Prav tako je poznan napad s fiksno točko, kjer poiščemo točko za katero velja: $f(x_i, M_i) = x_i$. To pomeni, da obstoj sporočila M_i ne spremeni povzetka in lahko sporočilu dodamo M_i, kjerkoli je vmesno stanje x_i. Napadu se izognemo z dodajanjem zapisa o velkosti sporočila[3].

2.3 Kriptoanalitični napadi

Večina kriptoanalitičnih metod, ki jih lahko uporabimo pri blokovnih šifrah lahko uporabimo pri zgoščevalnih funkcijah. Diferenčna kriptoanaliza je ena od metod, ki se zelo uspešno uporablja pri napadih na zgoščevalne funkcije. Razvite so bile tudi posebne kriptoanalitične metode za napade na MD4 in MD5.

Poglavje 3

Opis zgoščevalna funkcije Tiger

3.1 Princip delovanja

Gre za hitro in močno zgoščevalno funkcijo, ki je optimizirana za delovanje na 64 bitnih sistemih. Predstavlja uravnovešenost med hitrostjo in varnostjo. Po hitrosti jo na 32 bitnih sistemih lahko primerjamo s SHA-1, na 64 bitnih sistemih pa je približno trikrat hitrejša. Prav tako naj bi bila hitrejša kakor SHA-1 na 16 bitni arhitekturi saj je SHA-1 optimiziran za delovanje na 32 bitnih sistemih [6][4].

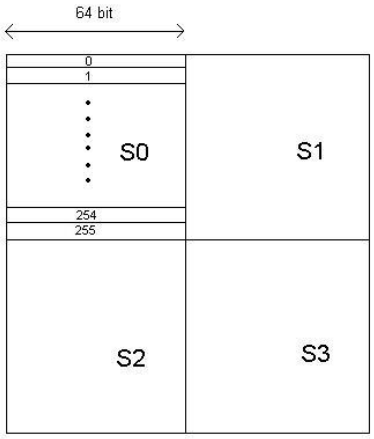
Osnovna operacija je sestavljena iz poizvedovanj v 4 S-Škatlah. Vsako S-Škatlo naslavljamo z osmimi biti kot rezultat pa dobimo 64 izhodnih bitov (Slika: 3.1).

Preostale operacije so:

- 64 bitno seštevanje
- 64 bitno odštevanje
- 64 bitno množenje z malimi konstantami (5,7 in 9)
- 64 bitno premikanje
- logične operacije (XOR in NOT)

Logične operacije ter operacije seštevanja in odštevanja so vsaj dvakrat počasnejše kakor ekvivalentne operacije na 32 bitnih sistemih. Enako zahtevnost ima le zamik vseh bitov. Operacije množenja z malimi konstantami pa so vsaj štiri do petkrat počasnejše.

Učinkovitost Tiger funkcije bazira predvsem na paralelnem izvajanju. Funkcije kot so MD in SNEFRU, za izračun naslednjega vmesnega stanja potrebujejo izračunano



Slika 3.1: S-Škatle naslavljamo z osmimi biti kot rezultat pa dobimo 64 izhodnih bitov.

predhodnje stanje. Na ta način ni možno efektivno izkoristiti procesorja saj cevovod v procerju ne pride do izraza. V vsakem ciklu Tiger funkcije, pa lahko pri osmih poizvedovanjih v S-Škatle, izvedemo poizvedovanja paralelno in na ta način izkoristimo prednosti cevovoda.

Velikost spomina, ki jo potrebuje Tiger je le malo večja kot štiri S-Škatle. V primeru, da imamo dovolj velik L1/L2 pomnilnik potem lahko imamo kar vse štiri S-Škatle v procesorskem pomnilniku. Na ta način izračuni potekajo kar dvakrat hitreje.

3.2 Kompatibilnost

V želji zadovoljiti kompatibilnost z zgoščevalnimi funkcijami iz družine MD4, tekstu dodamo '1', ki ji sledi niz '0'. Na koncu dodamo še 64 bitni niz, ki prestavlja dolžino celotnega teksta. Rezultat nato razbijemo na 512 bitne bloke.

Velikost povzetka in vmesnih stanj je velika 192 bitov. Ta velikost je izbrana iz naslednjih razlogov:

- Zaradi uporabe 64 bitnih besede, naj bo dolžina deljiva z 64;
- Z namenom zagotoviti kompatibilnost s SHA-1 mora biti velikost vsaj 160 bitov;
- Vsi uspešni napadi na obstoječe zgoščevalne funkcije napadajo vmesna stanja. Napadalci izberejo dve kolizijski vrednosti za vmesno stanje in to razširijo na celotno funkcijo. Vendar ti napadi ne bi uspeli, če bi vmesna stanja bila večja;

Kadar uporabljamo vseh 192 bitov povzetka lahko Tiger funkcijo poimenujemo Tiger/192. V primeru da želimo s Tiger funkcijo nadomestiti že obstoječe zgoščevalne funkcije se priporočata dve različici:

- Tiger/160: Za rezultat uporabimo samo prvih 160 bitov Tiger/192 funkcije. To inačico uporabljamo za kompatibilnost s SHA in SHA-1;
- Tiger/128: Za rezultat uporabimo samo prvih 128 bitov Tiger/192 funkcije. To inačico uporabljamo za kompatibilnost z MD4, MD5, RIPE-MD in za različice SNEFRU funkcije.

Predpostavka je, da so vse tri funkcije brez trčenj ter da se trčenje za Tiger/N ne da poiskati z manj kot $2^{N/2}$ operacijami. Prav tako se domneva, da je enosmerna in da sprememba enega bita teksta spremeni rezultat[6].

3.3 Specifikacije

Vse operacije Tiger funkcije so izvedene na 64 bitnih besedah. Za vmesne registre uporabljamo tri registre a,b,c. Te registre inicializiramo na začetno vrednost h_0.

a=0x0123456789ABCDEF
b=0xFEDCBA9876543210
c=0xF096A5B4C3B2E187

Vsak zaporeden blok dolžine 512 bitov razbijemo na osem 64 bitnih besed $x0, x1, x2, ...x7$. Naslednji izračuni spremenijo vrednosti h_i v h_{i+1}. Ti izračuni vsebujejo tri prehode. Med vsakim prehodom imamo še inverzno operacijo, ki preprečuje napade z redkimi vhodi. Sledi še faza, kjer nove vrednosti registrov a,b,c kombiniramo z inicializiranimi vrednostmi. Tako preidemo v novo stanje registrov h_{i+1} :

```
save_abc
pass(a,b,c,5)
key_schedule
pass(a,b,c,7)
key_schedule
pass(a,b,c,9)
feedforward
```

kjer so:

save_abc shrani trenutno stanje h_i

```
aa = a;
bb = b;
cc = c;
```

pass(a,b,c,mul)

```
round(a,b,c,x0,mul);
round(b,c,a,x1,mul);
round(c,a,b,x2,mul);
round(a,b,c,x3,mul);
round(b,c,a,x4,mul);
round(c,a,b,x5,mul);
round(a,b,c,x6,mul);
round(b,c,a,x0,mul);
```

kjer je `round(a,b,c,x,mul)` definirano:

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

Pri tem smo uporabili sintakso C jezika. `c_i` je i-ti byte c registra ($0 \leq i \leq 7$), znak `^` pa predstavlja XOR operacijo.

key_schedule

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

kjer so `<<` in `>>` logične shift operacije.

feedforward

```
a^=aa;
b-=bb;
c+=cc;
```

Vrednosti registrov a,b in c odražajo 192 bitno vmesno stanje h_{i+1}.

3.4 Generiranje S-Škatel

Algoritem za generiranje S-Škatel, ki jih uporablja Tiger, uporablja ravno Tiger funkcijo za generiranje psevdo naključnih podatkov za generiranje psevdo naključnih S-Škatel.

Algoritem prikazan v dodatku `B` uporablja 512 bitni vhodni parameter in število ponovitev. Inicializacijska vrednost S-Škatel je postavljena kot enotska matrika, ter inicializacijsko stanje vmesnega stanje enako kakor pri Tiger funkciji.

Nato naključno premeša vsak bajt-stolpec v vsaki S-Škatli z zamenjavo dveh vrednosti. To naredi vsakič kadar na novo izračuna vrednost funkcije, ki je odvisna od rezultatov vseh prej zamenjanih vrednosti v vseh S-Škatle in vseh kolonah. Na ta način je zamenjava vsakega bajta v koloni odvisna od prejšnjih zamenjav v vseh kolonah in ne samo v isti koloni.

Za generiranje S-Škatel za Tiger funkcijo je bilo preizkušenih več variant podanega algoritma. Rezultate so ocenjevali na podlagi naslednjih kriterijev:

1. Vse vrednosti S-škatle naj bodo različne. Še več, noben par naj nima več kot tri enake bajte.
2. Vsaka kolona je permutacija vseh 256 vrednosti.
3. Kolone vseh S-Škatel naj bodo čimbolj različne in naj imajo nek velik cikel.
4. Nobeni dve razliki vrednosti S-Škatel ($S_i(t_1) \bigoplus S_i(t_2)$ in $S_j(t_3) \bigoplus S_j(t_4)$) naj ne bi imeli več kakor 4 enake bajte.
5. Hitrost algoritma naj bo relativno hitra, da omogoči aplikacijam generiranje S-Škatel v teku aplikacije.
6. Struktura S-Škatel je bila izbrana tako da zmanjša linearno in difernčno odvisnost.
7. Naključni parameter naj si bo možno lahko zapomniti.

Za naključni parameter je bil izbran naslov članka, ki opisuje Tiger funkcijo "Tiger - A Fast New Hash Function, by Ross Anderson and Eli Biham". Naslov članka je dolg ravno 64 znakov. Število prehodov je pet[5].

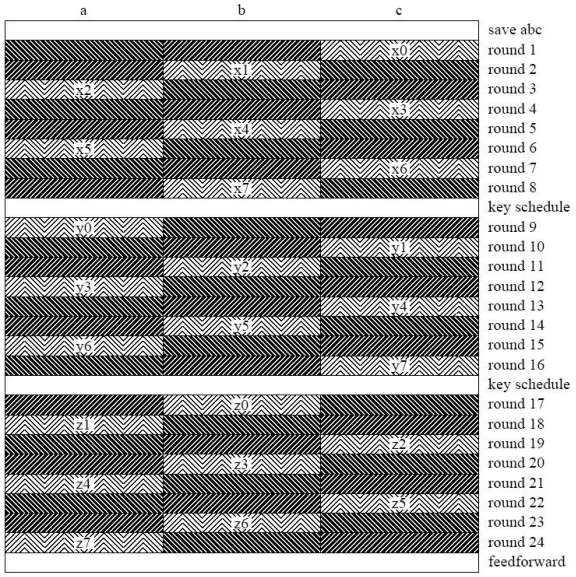
Poglavje 4

Analiza funkcije Tiger

4.1 Analiza

Slika 4.1 prikazuje grafični prikaz delovanja Tiger funkcije. Temna področja predstavljajo registre, ki so bili spremenjeni, poševne črte pa ponazarjajo bajte spremenjene v svetlih področjih.

Spremenljivke $y0, y1, ..., y7$ in $z0, z1, ..., z7$ pa ponazarjajo vrednosti $x0, x1, ..., x7$ v drugem in tretjem ciklu. Zadnja vmesna vrednost h_i se vzame za rezultat funkcije Tiger/192.



Slika 4.1: Grafični prikaz delovanja Tiger funkcije.

Nelinearnost funkcije izhaja predvsem iz S-Škatel kjer iz 8 vhodnih bitov dobim 64

izhodnih bitov. Ta metoda je veliko bolj zanesljivejša kakor kombinacije seštevanja in logičnih operacij. Prav tako vpliva na vse izhodne bite in ne samo sosednje bite.

Po prvih treh rundah vsak bit teksta vpliva na vse tri registre, kar je mnogo hitreje kakor pri ostalih funkcijah.

Vsi napadi na funkcije iz družine MD in SNEFRU se osredotočajo na šibkost vmesnih stanj. Z 192 bitno dolžino vmesnega stanja so ti napadi onemogočeni.

Korak `key schedule` (razporejanje) nam zagotavlja, da sprememba majhnega števila bitov v tekstu, pri večkratnih ponovitvah spremeni veliko bitov. Skupaj z močno razpršitvijo omogoča obrambo pred diferenčno analizo ter napadom s popravljanjem blokov.

Prav tako množenje registra `b` v vsaki rundi prispeva k odpornosti na takšne napade, saj zagotavlja da so vhodni biti S-Škatle v prejšnem ciklu premešani tudi v drugih S-Škatlah.

Korak `feedforward` je namenjen preprečevanju "Meet in the middle birthday attack"napadu.

4.2 Hitrostna primerjava

Hitrostna primerjava je bila izvedena s Crypto++^{TM} Library 5.1 knjižnico, ki vsebuje implementacijo več različnih zgoščevalnih funkcij. Test je bil izveden na Pentium 3 procesorju, zmogljivosti 800 MHz z 256 MB pomnilnika. Prikazano je več različnih funkcij z nameno prikazati jasno sliko hitrosti Tiger hash funkcije glede na druge zgoščevalne funkcije4.2. Vsak test je bil izveden 5 krat, predstavljen rezultat pa povprečje teh petih rezultatov [7].

Algorithm	Bytes Processed	Time Taken	Megabytes(2^20 bytes)/Second
CRC-32	67108864	0.961	66.597
Adler-32	67108864	0.921	69.490
MD2	262144	0.701	0.357
MD5	33554432	1.262	25.357
SHA-1	8388608	1.192	6.711
SHA-256	4194304	1.292	3.096
SHA-512	2097152	0.731	2.736
HAVAL (pass=3)	8388608	0.971	8.239
HAVAL (pass=4)	4194304	0.751	5.326
HAVAL (pass=5)	4194304	0.962	4.158
Tiger	16777216	1.191	13.434
RIPE-MD160	4194304	0.711	5.626
Panama Hash (little endian)	8388608	0.701	11.412
Panama Hash (big endian)	8388608	0.992	8.065

Slika 4.2: Primerjava hitrosti zgoščevalnih funkcij.

Zaključek

Ekspanzijo Tiger funkcije lahko pričakujemo šele s porastom števila 64 bitnih sistemov. Takrat lahko pričakujemo tudi več varnostnih analiz Tiger funkcije. Pomanjkanje literature na temo Tiger, je poglavitni vzrok, zakaj sem se v osrednjem delu seminarske naloge osredotočil na članka "Tiger: A Fast New Hash Function"in "Generation of the S box of Tiger".

Literatura

[1] S. A. Vanstone A. J. Menezes, P. C. Van Oorschot, *Handbook of applied cryptography*, October 1996.

[2] K.U. Leuven B. Preneel, *Correcting-block attack*, `http://www.win.tue.nl/~henkvt/correctingblockv2.pdf`.

[3] K.U. Leuven B. Preneel, *Hash functions*, January 18 2004.

[4] advisor Joseph N. Gregg PhD Eric C. Seidel, *Parallel computation via multiple processors, vector processing, and multi-cored chips*, `http://homepage.mac.com/macdomeeu/lu/Modern_Cryptography.pdf`, December 30 2002.

[5] Eli Biham Ros Anderson, *Generation of the s boxes of tiger*, `http://www.cs.technion.ac.il/~biham/Reports/Tiger/gen-sboxes.ps.gz`, 1996.

[6] Eli Biham Ros Anderson, *Tiger a fast new hash function*, `http://www.cs.technion.ac.il/~biham/Reports/Tiger/tiger.ps`, 1996.

[7] Britt Savage, *A guide to hash algorithms*, `http://www.giac.org/practical/GSEC/Britt_Savage_GSEC.pdf`, April 18, 2003.

[8] Bruce Schneier, *Applied cryptography, second edition: Protocols, algorthms, and source code in c*, second edition ed., Wiley Computer Publishing, January 1996.

[9] D. R. Stinson, *Cryptography theory and practice*, first edition ed., Wiley Computer Publishing, March 1995.

[10] Center Vlade RS za informatiko, *Zgoščevalne funkcije*, `http://www.sigov.si/tecaj/kripto/kr-zgo.htm`.

Dodatek A

Implementacija Tiger hash funkcije

```
word64 t1[256] = {...};
word64 t2[256] = {...};

word64 t3[256] = {...};
word64 t4[256] = {...};

TIGER_compression_function (state, block) word64 state[3];
unsigned word64 block[8]; {
  word64 a = state[0], b = state[1], c = state[2];
  word64 x0=block[0], x1=block[1], x2=block[2], x3=block[3],
         x4=block[4], x5=block[5], x6=block[6], x7=block[7];
  word64 aa, bb, cc;

#define save_abc aa = a; bb = b; cc = c;

#define round(a,b,c,x,mul) \
      c ^= x; \
      a -= t1[((c)>>(0*8))&0xFF] ^ t2[((c)>>(2*8))&0xFF] ^ \
           t3[((c)>>(4*8))&0xFF] ^ t4[((c)>>(6*8))&0xFF] ; \
      b += t4[((c)>>(1*8))&0xFF] ^ t3[((c)>>(3*8))&0xFF] ^ \
           t2[((c)>>(5*8))&0xFF] ^ t1[((c)>>(7*8))&0xFF] ; \
      b *= mul;


#define pass(a,b,c,mul) \
      round(a,b,c,x0,mul) \
      round(b,c,a,x1,mul) \
```

```
      round(c,a,b,x2,mul) \
      round(a,b,c,x3,mul) \
      round(b,c,a,x4,mul) \
      round(c,a,b,x5,mul) \
      round(a,b,c,x6,mul) \
      round(b,c,a,x7,mul)

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

#define feedforward a ^= aa; b -= bb; c += cc;

#define compress \
      save_abc \
      pass(a,b,c,5) \
      key_schedule \
      pass(c,a,b,7) \
      key_schedule \
      pass(b,c,a,9) \
      feedforward

  compress;
  state[0] = a; state[1] = b; state[2] = c;
}
```

Dodatek B

Algoritem generiranja S-škatel

```
/*This function generates the Sboxes of Tiger,by caling         */
/*gen("Tiger-AFastNewHashFunctionbyRosAndersonandEliBiham",5);  */
/*Thiscodeiswritenforlitle-endiancomputers.                     */
/*small changes (indicated in the code) arerequiredfor         */
/*big-endian computers.                                         */
/*word64 and word32 are unsigned int/long/long long             */
/*of the given sizes in bits                                    */
/* The type definition might vary on various machines.          */


   typedef unsigned long long int word64;
   typedef unsigned int word32;
   typedef unsigned char byte;
   typedef unsigned char octet[8];

   extern word32 table [4*256][2];
   static octet * table_ch=(octet*) table;
   gen(word str[1],int pases{
      word64 state[3];
      octet*_ch = (octet*) state;
      byte tempstr[64]inti,j;
      int cnt;
      int sb,col;
      int abc;

      state[0] = x123456789ABCDEFLL;
      state[1] = xFEDCBA9876543210L
      state[2] = xF096A5B4C3E2187LL
      for(j=0; j<64; j++)
```

```
      tempstr[j]=((byte*)str[j]);
   /*Onbig-endian computers the above line should be: */
   /*tempstr[j^7) = ((byte*)str[j]); */
   for(i=0; i<1024; i++{
      for(col = 0; c < ol8; col++){
         table_ch[i][col]= i&255;
      }
   }
   abc=2;
   for(cnt = 0; cnt < pases; cnt++){
      for(i = 0;i < 256;i++)
         for(sb = 0; sb < 1024; sb+=256){
            abc++;
            if(abc == 3){
               abc = 0;
               tiger_compres(tempstr,state);
            }
            for(col=0; col < 8; col++){
               byte tmp = table_ch[sb+i][col];
               table_ch[sb+i][col] = table_ch[sb+state_ch[abc][col]][col];
               table_ch[sb + state_ch[abc][col]][col]=tmp;
            }
         }
   }
}
```