Univerzitet u Beogradu

Matematički fakultet

Svođenje kriptografskih problema na problem SAT

— master teza —

Milan Šešum

Mentor: dr Predrag Janičić

Beograd

2008

Predgovor

Predmet ove teze je svođenje kriptografskih problema na problem SAT, na način koji je opisan u radu [JJ05]. U tom radu su, osim opisa, dati i rezultati koji su dobijeni razbijanjem heš funkcija MD4 i MD5 (*Message-Digest algorithm 4/5*). Takođe je napomenuto da se ovaj način razbijanja, zbog svoje univerzalnosti, može primeniti na druge heš funkcije i kriptografske algoritme. Cilj ove teze je da se ovaj način kriptoanalize (*logička kriptoanaliza*) primeni na neke poznate kriptografske algoritme.

Na samom početku rada na ovoj tezi, cilj je bio da se razvije kompletan mehanizam, na osnovu opisa koji je dat u radu [JJ05]. U toku razvoja mehanizma prvo je razbijena jedna trivijalna šifra (šifrat se od otvorenog teksta dobija cikličnim pomeranjem za određen broj mesta u neku od dve strane). Potom je rešavan sledeći problem: *Naći prirodan broj čiji je kvadrat dat.* Ovaj primer pokazuje koliko je korišćeni pristup univerzalan. Nakon toga su rekonstruisani rezultati koji su u [JJ05] dobijeni za heš funkciju MD5. Kao što je i očekivano, dobijeni su bolji rezultati, prvenstveno zbog korišćenja boljih, modernijih SAT rešavača ali i zbog bržeg hardvera. Na kraju je isti mehanizam primenjen i na kriptografske algoritme CMEA (*Cellular Message Encryption Algorithm*) i DES (*Data Encryption Standard*).

Kako je logička kriptoanaliza već primenjivana u razbijanju algoritma DES u radu [MM00], polazni cilj ove teze je bio izgradnja mehanizma opisanog u [JJ05] i njegova primena na algoritam DES. U [MM00] logička kriptoanaliza je primenjena na razbijanje algoritma DES na način koji zavisi od specifikacije algoritma DES, pa bi njegova uniformna primena na ostale kriptografske algoritme zahtevala dodatno vreme. S druge strane, uniforman pristup opisan u [JJ05] prevazilazi probleme prethodnog pa je bilo interesantno uporediti efikasnost ova dva pristupa.

S obzirom na kompleksnost algoritma DES, bilo je poželjno prvo primeniti uniforman pristup na neki jednostavniji kriptografski algoritam, pa tek onda na algoritam DES. Za ovo je odabran algoritam CMEA. Međutim, pretpostavka da je algoritam CMEA dovoljno jednostavan da se razbije uniformnim pristupom se u toku testiranja pokazala kao pogrešna.

Osnovna ideja pristupa je da se nepoznata vrednost, što je uglavnom ključ i/ili otvoreni tekst, ne predstavi kao niz bitova već kao niz iskaznih promenljivih. Poznate vrednosti, kao što su šifrat ili parovi — otvoreni tekst i odgovarajući šifrat, se predstavljaju kao nizovi logičkih konstanti. S obzirom da svaki kriptografski algoritam možemo da razmatramo kao niz logičkih operacija (svaka aritmetička operacija se može napisati kao niz logičkih), na ulazu, algoritmu se prosleđuje otvoreni tekst (i eventualno ključ) predstavljen na prethodno opisan način, a kao izlaz iz algoritma se dobija niz iskaznih formula koje *opisuju* tra-

nsformacije koje se vrše, u toku algoritma, nad bitovima otvorenog teksta. U toku rada je pretpostavljano da je, osim specifikacije algoritma, poznat i šifrat, čiji odgovarajući otvoreni tekst želimo da nađemo, ili parovi otvorenih tekstova i odgovarajućih šifrata, u slučaju da tražimo ključ. Na kraju je još potrebno dobijeni niz formula, izjednačiti sa poznatim šifratom. Dakle, potrebno je napraviti konjunkciju dobijenih formula, s tim da se prethodno negira svaka formula koja odgovara bitu 0 u šifratu. Na taj način dobija se iskazna formula koja je tačna za onu dodelu logičkih konstanti logičkim promenljivama koja odgovara vrednostima bitova u traženom otvorenom tekstu.

Jasno je da će ovaj pristup, uglavnom zbog svoje univerzalnosti, u većini slučajeva, dati lošije rezultate nego kriptoanaliza prilagođena konkretnom kriptografskom algoritmu. Međutim, s obzirom na veliki progres na polju SAT rešavača, opravdano je očekivati da opisani pistup, u vremenu koje sledi, dobije sve veći značaj.

Ovom prilikom bih zahvalio svom mentoru, dr Predragu Janičiću, koji me je uveo u oblast logičke kriptoanalize i koji je u velikoj meri zaslužan što je ova teza poprimila ovakav oblik. Takođe bih zahvalio ostalim članovima komisije, dr Miodragu Živkoviću, koji mi je svojim sugestijama pomogao u toku analize dobijenih rezultata, kao i mr Filipu Mariću, koji mi je svojim iskustvom i stručnim savetima pomogao u toku implementiranja kompletnog mehanizma. Posebnu zahvalnost dugujem Matematičkom institutu u Beogradu koji mi je omogućio da sva testiranja uradim na njihovom klasteru (IBM Cluster 1350).

Sadržaj

Glava 1

Uvod

Pojavom računara i razvojem komunikacionih tehnologija sve veća količina podataka se čuva u digitalnom obliku. Takođe, najveći deo komunikacije danas se obavlja putem interneta (*e-mail*), koji je veoma nesiguran[1]. Zbog toga se javlja potreba za što sigurnijim kriptografskim algoritmima koji će omogućiti čuvanje tih podataka u tzv. *šifrovanom* obliku. Time kriptoanaliza[2] dobija sve veći značaj jer nam daje mogućnost da proverimo sigurnost kriptografskog algoritma. Tačnije, daje mogućnost da proverimo kako se konkretan algoritam ponaša u odnosu na postojeće kriptoanalitičke napade. Naravno, nemogućnost razbijanja algoritma postojećim metodama ne garantuje apsolutnu sigurnost algoritma. Treba napomenuti da kriptoanaliza nikako nije rutinska stvar koja se svodi na primenu nekog od napada sa poznatog konačnog spiska. Spisak postojećih napada se stalno povećava, dodavanjem novih. A do novih napada se dolazi kompleksnim analizama postojećih kriptografskih algoritama. Postoje različiti pristupi oceni sigurnosti. Od njih, možda najvažnija je *praktična tajnost*. Za neki kriptografski algoritam se kaže da je praktično tajan ako je njegovo razbijanje *praktično neizvodljivo* sa raspoloživim (sadašnjim ili budućim) resursima.

Sve do sada poznate kriptoanalitičke tehnike su usko vezane za konkretne kriptografske algoritme. Naime, za svaki kriptografski algoritam je potrebno da kriptoanalitičar provede izvesno vreme pokušavajući da pronađe neke specifičnosti vezane za sam algoritam i da ih u kombinaciji sa postojećim ili eventualno novoosmišljenim tehnikama iskoristi u cilju razbijanja tog algoritma.

Prirodno se nameće pitanje da li postoji mogućnost da se razvije univerzalan mehanizam, takav da se pomoću njega mogu razbijati proizvoljni kriptografski algoritmi. Jedan pristup ovom problemu jeste da se nađe problem na koji bi se moglo svesti razbijanje proizvoljnog kriptografskog algoritma, ali da se to svođenje može uraditi u što kraćem vremenu. Pristup koji je primenjen u ovoj tezi garantuje da se pomenuto svođenje može obaviti u vremenu koje zavisi isključivo od broja instrukcija u konkretnom kriptografskom algoritmu. U ovoj tezi se kriptografski problemi svode na problem SAT.

Univerzalnost, do koje se dolazi na ovaj način, ima i svoju cenu. Cena je umanjena efikasnost u odnosu na neke specifične tehnike. Ovo pokazuju rezultati do kojih se došlo u toku rada na ovoj tezi.

[1] U smislu da postoje načini da komunikacija dve osobe bude *prisluškivana*.

[2] Kriptoanaliza je nauka o razbijanju šifrata, odnosno čitanju šifrovanih poruka.

Problem SAT je jedan od tipičnih predstavnika NP-kompletnih problema. Za sada se ne zna da li postoje polinomijalna rešenja za NP-kompletne probleme. Zbog toga, do pre nekoliko godina, bilo je besmisleno i pomisliti da se bilo kakav problem svodi na problem SAT. SAT rešavači su u poslednjoj deceniji doživeli zavidan progres, što je navelo na razmišljanje da se pronađe način da se razbijanje konkretnog kriptografskog algoritma svede na problem SAT. Ovo je na različite načine urađeno za DES u radu [MM00] kao i za MD4 i MD5 u radu [JJ05]. U ovoj tezi je razrađen mehanizam koji je opisan u radu [JJ05] i pokušano da se pomoću njega razbiju kriptografski algoritmi CMEA i DES.

Osnovna teza ovog rada je da je moguće uniformno kodiranje kriptografskih algoritama formulama iskazne logike i svođenje kriptoanalitičkih problema na problem iskazne zadovoljivosti.

Pristup koji je korišćen u ovoj tezi je prvi put opisan u radu [JJ05]. Međutim, u tom radu su formulama iskazne logike kodirane heš funkcije, a u zaključku je navedena pretpostavka da se isti pristup može iskoristiti i za kodiranje kriptografskih algoritama. Ova teza je prirodan nastavak prethodno pomenutog rada.

Nastavak rada je podeljen na sledeće glave:

- *Osnovni pojmovi i notacija.* U ovoj glavi su uvedeni osnovni pojmovi i oznake koji su korišćeni u ostatku rada, a u vezi su sa kriptografskim algoritmima ili iskaznom logikom i problemom SAT. Takođe je dat kratak opis heš funkcije MD5 i kriptografskih algoritama CMEA i DES.

- *Logička kriptoanaliza.* Ova glava sadrži formalno izvođenje formula iskazne logike na koje se svode kriptografski algoritmi. Takođe je dat opis dva različita pristupa ovom problemu. Jedan pristup je prvi put opisan u radu [MM00], a zasnovan je na konstrukciji logičke formule praćenjem koraka konkretnog kriptografskog algoritma. Drugi pristup je prvi put opisan u radu [JJ05], a zasnovan je na korišćenju jedne osobine objektno-orijentisanih programskih jezika — *preopterećivanja operatora*. Na kraju je dat pregled još jednog rada ([MZ]) koji jedan deo već postojećeg napada na heš funkciju MD5 svodi na rešavanje formula iskazne logike.

- *Implementacija.* U ovoj glavi je objašnjena kompletna implementacija mehanizma svođenja. Navedeni su i opisani svi novouvedeni tipovi podataka (klase) i razlozi njihovog uvođenja. Takođe su opisane glavne tehnike koje su korišćene, kao i unapređenja do kojih je došlo njihovim korišćenjem.

- *Eksperimentalni rezultati.* Ova glava sadrži rezultate dobijene primenom razvijenog mehanizma na kriptografske algoritme CMEA i DES, kao i na heš funkciju MD5. Ovde je, takođe, urađeno poređenje sa do sada postignutim rezultatima koji su objavljeni u [MM00] i [JJ05].

- *Zaključci i dalji rad.* Na kraju rada su, pored zaključka, data kratka razmatranja o tome šta bi se dalje moglo uraditi da bi se povećala efikasnost razvijenog mehanizma.

Glava 2

Osnovni pojmovi i notacija

U ovoj glavi će biti definisani osnovni pojmovi koji se koristite u daljem tekstu. Glava sadrži tri dela. Prvi deo, *Kriptografski algoritmi*, uvodi i objašnjava osnovne pojmove vezane za kriptografiju i kriptografske algoritme. Drugi deo, *Heš funkcije*, sadrži opis heš funkcija. U trećem delu, *Klase složenosti*, uvedeni su osnovni pojmovi vezani za klase složenosti i definisane su klase P i NP. Četvrti deo, *Problem SAT i SAT rešavači*, uvodi pojam problema SAT i govori o progresu koji je napravljen na polju SAT rešavača.

2.1 Kriptografski algoritmi

Kriptografski algoritmi predstavljaju funkcije koje preslikavaju jedan niz bitova u drugi niz bitova (ne nužno iste dužine). Ulazni niz bitova, tj. niz bitova koji se prosleđuje funkciji kao argument, zovemo *otvoreni tekst*. Primenu kriptografskog algoritma zovemo *šifrovanjem*, a rezultat šifrovanja, tj. izlazni niz bitova koji se dobija kao rezultat primene funkcije, *šifratom*. Ako otvoreni tekst označimo sa $\mathbf{P}$, šifrat sa $\mathbf{C}$ a funkciju šifrovanja sa E, proces šifrovanja možemo opisati sledećom jednakošću:

$$\mathbf{C} = E(\mathbf{P}).$$

Često je potrebno da postoji, u nekom smislu, inverzan postupak postupku šifrovanja koji zovemo *dešifrovanjem*. Ako funkciju dešifrovanja označimo sa D, tada važi:

$$\mathbf{P} = D(\mathbf{C})$$

kao i

$$\mathbf{P} = D(E(\mathbf{P})).$$

U realnim sistemima, praksa je da u kriptografskom algoritmu figuriše i neki niz bitova koji zovemo *ključ*. Ako ključ označimo sa $\mathbf{K}$, proces šifrovanja, korišćenjem ključa $\mathbf{K}$, možemo opisati sledećom jednakošću:

$$\mathbf{C} = E_{\mathbf{K}}(\mathbf{P}).$$

Kriptografski algoritmi u kojima se koriste ključevi se mogu podeliti na *simetrične* i *asimetrične* algoritme. *Simetrični algoritmi* su algoritmi kod kojih se i pri šifrovanju i pri dešifrovanju koristi isti ključ. Za njih važi:

$$\mathbf{P} = D_{\mathbf{K}}(\mathbf{C})$$

i

$$\mathbf{P} = D_{\mathbf{K}}(E_{\mathbf{K}}(\mathbf{P}))$$

Asimetrični algoritmi su algoritmi kod kojih se pri šifrovanju i pri dešifrovanju koriste različiti ključevi.

Pre upotrebe algoritama zasnovanih na ključu, bezbednost kriptografskog algoritma se zasnivala na njegovoj tajnosti. Uvođenjem ključa, bezbednost algoritma se počela zasnivati na tajnosti ključa, dok algoritmi više nisu morali biti tajni. Na taj način, javnost je uključena u razotkrivanje nedostataka postojećih algoritama i pronalaženje novih, bezbednijih algoritama.

Simetrični algoritmi se mogu uslovno podeliti u sledeće dve kategorije:

- *lančane šifre* — ovoj kategoriji pripadaju algoritmi koji otvoreni tekst obrađuju bit po bit;
- *blokovske šifre* — ovoj kategoriji pripadaju algoritmi koji otvoreni tekst dele na grupe bitova koje zovemo *blokovi*, a potom ga obrađuju blok po blok.

Kod asimetričnih algoritama postoje dva različita ključa. Jedan ključ se koristi za šifrovanje i obično se zove *javni ključ*, dok se drugi koristi za dešifrovanje i zove se *tajni ključ*. Obično je poznat javni ključ, kao i specifikacija algoritma. Poruku, prethodno šifrovanu asimetričnim algoritmom korišćenjem javnog ključa, ima smisla poslati samo osobi koja ima odgovarajući tajni ključ, jer je samo ta osoba može dešifrovati.

Pokušaj kriptoanalize se zove *napad*. Postoji nekoliko tipova kriptoanalitičkih napada, prema količini informacija koje napadač poseduje. U svim varijantama se podrazumeva da napadač poznaje algoritam za šifrovanje. Neki od napada su:

- *Napad na osnovu šifrata.* Kriptoanalitičaru je poznat jedan ili više šifrata koji su dobijeni istim algoritmom za šifrovanje. Cilj je otkriti odgovarajuće otvorene tekstove ili ključ koji je korišćen pri šifrovanju.
- *Napad na osnovu parova (otvoreni tekst, šifrat).* U ovom slučaju, kriptoanalitičaru su poznati parovi otvorenih tekstova i odgovarajućih šifrata, a cilj je otkriti odgovarajući ključ ili pronaći način da se za svaki sledeći šifrat može izračunati odgovarajući otvoreni tekst.
- *Napad na osnovu izabranog otvorenog teksta.* Ovaj napad je sličan prethodnom. Dakle, poznati su parovi otvorenih tekstova i odgovarajućih šifrata, s tom razlikom da kriptoanalitičar ima mogućnost da sâm bira otvorene tekstove. U ovom slučaju, kriptoanalitičar, pažljivim odabirom otvorenih tekstova, može da otkrije mnogo više informacija o ključu.

U potpoglavljima koja slede, biće dat opis kriptografskih algoritama čije će svođenje na problem SAT biti urađeno u okviru ove teze.

2.1.1 Algoritam CMEA

Algoritam CMEA je jedan od četiri dela koji čine standard, kreiran od strane Telekomunikacione Industrijske Asocijacije (TIA), koji je služio za šifrovanje glasa, autentifikaciju i drugo, a koji je korišćen u mobilnoj telefoniji u Americi.

Za razliku od ostala tri dela, CMEA se koristila za šifrovanje kontrolnih poruka. Objavljena je 1991, a jedan napad je opisan u [Kir07], koji je zasnovan na jednom od prvih napada na algoritam CMEA, koji je opisan u [WSK].

CMEA je blokovska šifra promenljive dužine ulaza (ulaz i izlaz su iste dužine) i 64-bitnog ključa. Za različite veličine ulaza, šifra CMEA uzima različite oblike. Veličina ulaza je n bajtova, za $n \geq 2$. U praksi se najčešće koriste oblici koji odgovaraju ulazu dužine $n \in \{2, 3, 4, 5, 6\}$. Uvedimo sledeće oznake

- $P_0P_1 \dots P_{n-1}$, otvoreni tekst dužine n bajtova,
- $C_0C_1 \dots C_{n-1}$, šifrat dužine n bajtova,
- $K_0K_1 \dots K_7$, osam bajtova ključa.

Ako znakovi $+$ i $-$ označavaju sabiranje, odnosno oduzimanje po modulu 256, a znakovi $\wedge$, $\vee$, $\veebar$ označavaju logičke veznike za konjunkciju, disjunkciju i ekskluzivnu disjunkciju, respektivno, definišimo funkciju $T : Z_{256} \to Z_{256}$ na sledeći način:

$$T(x) = C(((C(((C(((C((x \veebar K_0) + K_1) + x) \veebar K_2) + K_3) + x) \veebar$$

$$\veebar K_4) + K_5) + x) \veebar K_6) + K_7) + x$$

gde je funkcija $C : Z_{256} \to Z_{256}$ zadata tabelom 2.1. Vrednost $C(x)$ se nalazi

Tabela 2.1: Tabela vrednosti funkcije C.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	d9	23	5f	e6	ca	68	97	b0	7b	f2	0c	34	11	a5	8d	4e
1	0a	46	77	8d	10	9f	5e	62	f1	34	ec	a5	c9	b3	d8	2b
2	59	47	e3	d2	ff	ae	64	ca	15	8b	7d	38	21	bc	96	00
3	49	56	23	15	97	e4	cb	6f	f2	70	3c	88	ba	d1	0d	ae
4	e2	38	ba	44	9f	83	5d	1c	de	ab	c7	65	f1	76	09	20
5	86	bd	0a	f1	3c	a7	29	93	cb	45	5f	e8	10	74	62	de
6	b8	77	80	d1	12	26	ac	6d	e9	cf	f3	54	3a	0b	95	4e
7	b1	30	a4	96	f8	57	49	8e	05	1f	62	7c	c3	2b	da	ed
8	bb	86	0d	7a	97	13	6c	4e	51	30	e5	f2	2f	d8	c4	a9
9	91	76	f0	17	43	38	29	84	a2	db	ef	65	5e	ca	0d	bc
10	e7	fa	d8	81	6f	00	14	42	25	7c	5d	c9	9e	b6	33	ab
11	5a	6f	9b	d9	fe	71	44	c5	37	a2	88	2d	00	b6	13	ec
12	4e	96	a8	5a	b5	d7	c3	8d	3f	f2	ec	04	60	71	1b	29
13	04	79	e3	c7	1b	66	81	4a	25	9d	dc	5f	3e	b0	f8	a2
14	91	34	f6	5c	67	89	73	05	22	aa	cb	ee	bf	18	d0	4d
15	f5	36	ae	01	2f	94	c3	49	8b	bd	58	12	e0	77	6c	da

u preseku m-te vrste i n-te kolone, gde su m i n vrednosti koje su određene sa 4 bita veće, odnosno manje težine 8-bitnog broja x (na primer $C(90) = C(01011010) = C(5, 10) = 5f$).

Sledeći kôd predstavlja specifikaciju šifre CMEA:

$y_0 = 0$
for $i = 0 \dots n-1$
  $P_i^{'} = P_i + T(y_i \veebar i)$
  $y_{i+1} = y_i + P_i^{'}$

$$
\begin{aligned}
&\text{for } i = 0 \ldots \lfloor \tfrac{n}{2} \rfloor - 1 \\
&\quad P_i^{''} = P_i^{'} \veebar (P_{n-1-i}^{'} \vee 1)
\end{aligned}
$$

$$
\begin{aligned}
&z_0 = 0 \\
&\text{for } i = 0 \ldots n-1 \\
&\quad z_{i+1} = z_i + P_i^{''} \\
&\quad C_i = P_i^{''} - T(z_i \veebar i)
\end{aligned}
$$

U ovoj tezi je razmatran slučaj kada je $n = 2$. U tom slučaju, dužina, kako otvorenog teksta, tako i šifrata je 16 bitova. Stoga, ako ulaz označimo sa P_0P_1, a izlaz sa C_0C_1, algoritam CMEA dobija sledeći oblik:

$$
\begin{aligned}
P_0^{'} &= P_0 + T(0) \\
P_1^{'} &= P_1 + T(P_0^{'} \veebar 1)
\end{aligned}
$$

$$
\begin{aligned}
P_0^{''} &= P_0^{'} \veebar (P_1^{'} \vee 1) \\
P_1^{''} &= P_1^{'}
\end{aligned}
$$

$$
\begin{aligned}
C_0 &= P_0^{''} - T(0) \\
C_1 &= P_1^{''} - T(P_0^{''} \veebar 1)
\end{aligned}
$$

Za više informacija u vezi sa algoritmom CMEA i standardom čiji je on deo, videti u [TIA].

2.1.2 Algoritam DES

Kompanija IBM je oko 1970. godine razvila kriptografski algoritam, zvani *Lucifer*, sa kojim se pojavila na konkursu za projektovanje standardnog algoritma za šifrovanje. Algoritam DES je nastao oko 1975. godine iz algoritma Lucifer, smanjivanjem ključa (Lucifer je imao ključ dužine 128, a DES 56 bitova). U leto 1998. godine nesigurnost DES-a je demonstrirana tako što je specijalizovani računar od 250000 dolara za 56 sati odredio ključ za DES, na osnovu jednog poznatog para otvorenog teksta i odgovarajućeg šifrata. Nakon toga su usledile i softverske realizacije napada. Jedan od njih, u kojem se koristi iskazna logika, je opisan u [MM00].

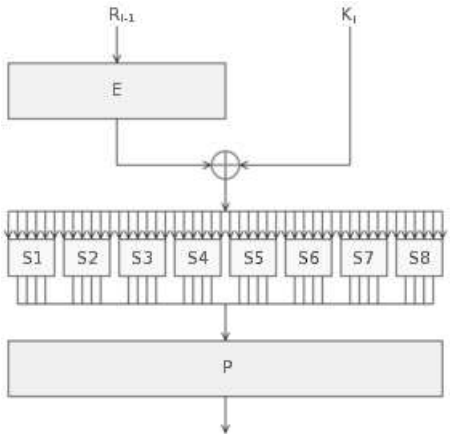
Kao i CMEA, algoritam DES je takođe blokovska šifra. Algoritam DES na ulazu prihvata blokove fiksirane dužine, 64 bita. Na izlazu se dobija šifrat iste dužine. Ključ koji se koristi u algoritmu je dužine 56 bitova. Međutim, algoritmu se prosleđuje ključ kao 64-bitni neoznačeni ceo broj, a potom se kao prvi korak algoritma, permutuju bitovi ključa uz izbacivanje svakog osmog bita. Na taj način se dobija ključ dužine 56 bitova. Osim toga, na početku algoritma se permutuju i sami bitovi otvorenog teksta (tzv. *inicijalna permutacija*). Algoritam se sastoji od 16 rundi, koje se razlikuju samo po podključu koji se koristi. U i-toj rundi ($1 \leq i \leq 16$) se koristi 48-bitni podključ, K_i, koji se od osnovnog ključa dobija na sledeći način: 56-bitni ključ se razbija na dva 28-bitna dela koji se potom, nezavisno, ciklički pomeraju ulevo za jednu ili dve pozicije (zavisno od runde); potom se dva dela spajaju i od tih 56 bitova se 48-bitni ključ dobija tako što se sa određenih 48 pozicija uzmu odgovarajući bitovi.

Kada je određen ključ koji se koristiti u i-toj rundi, izlaz iz prethodne runde (u prvoj rundi to je sam otvoreni tekst) deli se na levu i desnu polovinu, koje ćemo označiti sa L_{i-1} i R_{i-1}. Korišćenjem uvedenih oznaka i-ta runda može se opisati sledećim jednakostima:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \veebar f(R_{i-1}, K_i) \end{aligned}$$

Spajanjem dva dobijena dela, L_i i R_i, dobija se 64-bitni ulaz za sledeću rundu.

Funkcija f ima dva parametra, 32-bitni R_{i-1} i 48-bitni K_i. Prvo se *proširujućom permutacijom* blok označen sa R_{i-1} proširuje na 48 bitova. Potom se nalazi ekskluzivna disjunkcija proširenog bloka i ključa K_i. Dobijenih 48 bitova se smanjuje na 32 bita pomoću osam supstitucionih blokova. Dobijena 32 bita se na kraju permutuju i tako se dobija vrednost funkcije f. Ako proširujuću permutaciju označimo sa E, a supstitucione blokove sa S_k $(1 \leq k \leq 8)$, slika 2.1 ilustruje funkciju f.



Slika 2.1: Funkcija *f*.

Redukcija 48 na 32 bita se obavlja na sledeći način. Polaznih 48 bitova se deli na 8 blokova dužine 6 bitova. Svaki blok predstavlja ulaz u jedan supstitucioni blok. Svaki od osam supstitucionih blokova predstavlja matricu izlaznih vrednosti čije su dimenzije 4×16. Na osnovu ulaznih 6 bitova se određuje izlazna vrednost iz supstitucionog bloka. Izlazna vrednost se nalazi u preseku vrste i kolone čiji su redni brojevi određeni sa dva bita veće težine i preostala četiri bita, respektivno.

Posle 16 rundi dobija se 64-bitna poruku čijih 32 bita veće težine je L_{16}, a 32 bita manje težine je R_{16}. Na samom kraju algoritma se 64 bita izlazne poruke permutuju inverznom permutacijom inicijalne permutacije.

S obzirom da je rad na DES algoritmu samo deo ove teze, ovde nisu date tabele koje opisuju permutacije i supstitucione blokove. Za više informacija u vezi sa DES algoritmom, kao i pomenute tabele videti, na primer, [Ziv00].

2.2 Heš funkcije

U ovoj glavi će biti definisani osnovni pojmovi vezani za heš funkcije, a potom i specifikacija heš funkcije MD5 koja se svodi na problem SAT u ovoj tezi.

Heš funkcije transformišu ulaz proizvoljne dužine u izlaz fiksirane dužine. Izlaznu vrednost zovemo *heš vrednost* (nekada i *digitalni potpis*).

Osnovni zahtev koji svaka heš funkcija koja se koristi u kriptografiji mora da zadovoljava jeste da heš vrednost *ne otkriva* nikakve informacije o polaznoj poruci. Ako se u polaznoj poruci invertuje samo jedan, proizvoljan, bit, trebalo bi da se odgovarajuća heš vrednost drastično razlikuje od prvobitne. Dodatni uslov koji heš funkcije treba da zadovoljavaju je da je teško naći dve poruke koje imaju istu heš vrednost. U suprotnom, ako sa *hash* označimo heš funkciju, a sa x i y označimo dva moguća ulaza i ako važi da je $hash(x) = hash(y)$, kažemo da je došlo do *kolizije*.

Heš funkcije imaju široku primenu:

- *Heš tabele.* Heš tabele predstavljaju veoma efikasnu strukturu podataka. Osnovne dve operacije su umetanje i traženje. Ideja je da se za svaki element, koji se umeće ili traži u heš tabeli, njegova pozicija pronalazi izračunavanjem heš vrednosti tog elementa (na osnovu unapred zadate heš funkcije).

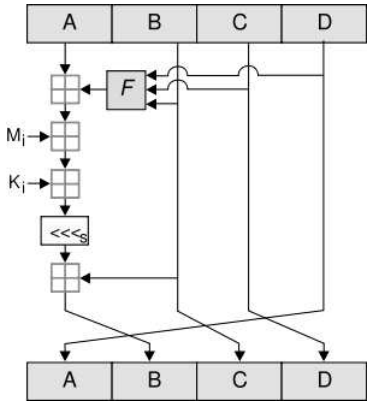
- *Kriptografija.* Heš funkcije se ne koriste kao klasični kriptografski algoritmi (zbog nepostojanja inverzne funkcije). U skoro svim računarskim sistemima korisničke lozinke se čuvaju u šifrovanom obliku. Kada korisnik ukuca svoju lozinku u toku procesa prijavljivanja na sistem, sistem šifruje datu lozinku i upoređuje dobijeni šifrat sa postojećim. U ovu svrhu se nekada koriste heš funkcije[1]. Na taj način, teorijski, postoji mogućnost da se na sistem korisnik loguje sa jednim korisničkim imenom i dve različite lozinke (u slučaju da postoji mogućnost kolizije), ali se dodatnim mehanizmima može dovoljno povećati bezbednost sistema.

- *Kontrolne sume (eng. Checksum).* Zbog nesavršenosti internet veza, pri preuzimanju datoteka, može da dođe do grešaka. Te greške mogu biti prijavljene od strane operativnog sistema u toku rada sa tim datotekama. Zbog toga, poželjno je imati mogućnost provere validnosti preuzetih datoteka. U novije vreme, većina servera sa kojih se mogu preuzeti datoteke (eng. *file servers*) postoje niske pridružene svakoj datoteci (eng. *checksum*) koje predstavljaju heš vrednost sadržaja datoteke (dobijena primenom funkcije MD5).

Heš funkcija MD5. MD5 je funkcija koja preslikava ulaznu poruku, promenljive dužine, u izlaz fiksirane dužine, 128 bitova. Na početku algoritma, ulazna poruka se proširuje do dužine deljive sa 512 na sledeći način. Prvo se na kraj dodaje bit 1. Za njim sledi onoliko nula koliko je potrebno da dobijena (proširena) poruka bude dužine jednake 448 po modulu 512. Na kraju se dodaje 64-bitni broj koji predstavlja dužinu neproširene poruke, s tim da se koristi redosled smeštanja bajtova sa desnog kraja (eng. *little endian*).

[1] Na primer, na operativnom sistemu Linux koristi se heš funkcija MD5.

Svaki korak MD5 algoritma se karakteriše 128-bitnim stanjem, koje delimo na četiri 32-bitne reči koje označavamo sa A, B, C i D. Na samom početku algoritma, ove četiri reči su inicijlizovane na neke konkretne vrednosti. Svaki 512-bitni blok poruke utiče na izmenu 128-bitnog stanja. Procesiranje jednog bloka se sastoji od četiri runde, gde se svaka runda predstavlja kombinaciju 16 sličnih operacija koje su bazirane na nelinearnoj funkciji F, sabiranju po modulu 2^{32} i bitskom pomeranju u levo. Slika 2.2 ilustruje jednu operaciju u okviru runde. 512 bitova poruke se deli na 16 manjih blokova dužine 32, od kojih svaki učestvuje u jednoj od 16 operacija koje čine rundu. Na slici 2.2 je jedan tako dobijeni 32-bitni blok označen sa M_i. Postoje četiri različite funkcije



Slika 2.2: Jedna MD5 operacija. $<<<_s$ označava operaciju bitskog pomeranja za s pozicija u levo, a $\boxplus$ označava operaciju sabiranja po modulu 2^{32}.

i svaka se koristi u jednoj rundi.

$$\begin{aligned} F(X,Y,Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\ G(X,Y,Z) &= (X \wedge Z) \vee (Y \wedge \neg Z) \\ H(X,Y,Z) &= X \veebar Y \veebar Z \\ I(X,Y,Z) &= Y \veebar (X \vee \neg Z) \end{aligned}$$

Operatori $\wedge$, $\vee$ i $\veebar$ predstavljaju logičke operatore konjunkcije, disjunkcije i ekskluzivne disjunkcije.

2.3 Klase složenosti

U toku analize složenosti nekog algoritma, često je najvažnije asimtotsko ponašanje tog algoritma. Ovde koristimo tzv. *O-notaciju.*

Definicija 1 *Ako postoje pozitivna konstanta c i prirodan broj n_0 takvi da za funkcije f i g nad prirodnim brojevima važi*

$$f(n) \leq c \cdot g(n) \text{ za sve vrednosti } n \text{ veće od } n_0$$

onda pišemo

$$f = O(g)$$

i čitamo "f je veliko 'o' od g".

Iako sintaksa u kojoj upotrebljavamo oznaku O asocira na to da je O funkcija, moramo napomenuti da O nije funkcija već označava čitavu klasu funkcija.

Definicija 2 *Ako je $T(n)$ vreme izvršavanja algoritma A (čiju veličinu ulaza karakteriše prirodan broj n), ako važi $T = O(g)$, onda kažemo da je algoritam A* vremenske složenosti *(ili* reda*) g i da pripada klasi $O(g)$.*

Definicija 3 *Za problem odlučivanja sa ulaznom vrednošću n kažemo da je polinomijalne složenosti ako je njegovo vreme izvršavanja $O(P(n))$ gde je $P(n)$ polinom po n. Klasu polinomijalnih algoritama označavamo sa* P.

Kada koristimo termin algoritam, mislimo na *deterministički algoritam.* Kod determinističkih algoritama, u svakom trenutku, zna se koji je sledeći korak koji treba primeniti. Za razliku od njih, *nedeterministički algoritmi* mogu u nekim svojim delovima da nedeterministički biraju između dva različita koraka. Precizne definicije ovih pojmova se mogu naći u [Ziv00].

Definicija 4 *Klasu svih problema odlučivanja za koje postoje nedeterministički algoritmi čije je vreme izvršavanja polinomijalno (po veličini ulaza) zavemo klasa* NP.

Intuitivno, pojam *svodljivosti* možemo uvesti na sledeći način. Za problem A kažemo da je svodljiv na problem B ako se algoritam koji rešava problem B može iskoristiti za rešavanje problema A sa određenim dodatnim vremenom i prostorom potrebnim za svođenje problema.

Definicija 5 *Za problem X kažemo da je* NP-težak problem *ako je svaki problem iz klase* NP *u polinomijalnom vremenu svodljiv na X.*

Definicija 6 *Za problem X kažemo da je* NP-kompletan problem *ako pripada klasi* NP *i ako je* NP*-težak.*

Svi NP-kompletni problemi su međusobno ekvivalentni — *za neki* NP*-kompletan problem postoji efikasan algoritam akko za svaki* NP*-kompletan problem postoji efikasan algoritam* (pod efikasnim algoritmom ovde podrazumevamo algoritam polinomijalne složenosti). Široko je rasprostranjeno verovanje da ne postoji efikasan algoritam za bilo koji NP-kompletan problem, mada ovo do sada nije dokazano. Za stotine ili čak hiljade problema (zavisno od načina klasifikacije) se zna da su NP-kompletni, što ovu oblast čini vrlo značajnom. Kada bi bio pronađen polinomijalni algoritam koji rešava neki NP-kompletan problem, to bi, na osnovu definicije 3 i međusobne ekvivalentnosti NP-kompletnih problema, značilo da je P = NP. Generalno vlada uverenje da je P $\neq$ NP.

2.4 Problem SAT i SAT rešavači

U ovom poglavlju će biti definisani osnovni pojmovi iskazne logike, a potom će biti definisan problem SAT i biće rečeno nešto o načinima njegovog rešavanja.

2.4.1 Osnovni pojmovi iskazne logike

Alfabet (azbuka) je konačan, neprazan skup simbola. Svaki konačan niz simbola nekog alfabeta predstavlja reč nad tim alfabetom.

Definicija 7 *Neka je alfabet* Σ *unija sledeća četiri skupa:*

1. *prebrojivog skupa* iskaznih slova P*;*
2. *skupa* logičkih veznika $\{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$ *pri čemu je* $\neg$ *unarni veznik, a* $\wedge$*,* $\vee$*,* $\Rightarrow$ *i* $\Leftrightarrow$ *su binarni veznici;*
3. *skupa* logičkih konstanti $\{\top, \bot\}$*;*
4. *skupa* pomoćnih simbola $\{(,)\}$.

Skup iskaznih formula *(ili* jezik iskazne logike*) predstavlja najmanji podskup skupa svih reči nad azbukom* Σ *za koji važi:*

- *iskazna slova (elementi skupa* P*) i logičke konstante su iskazne formule;*
- *ako su* f *i* g *iskazne formule, onda su* $(\neg f)$*,* $(f \wedge g)$*,* $(f \vee g)$*,* $(f \Rightarrow g)$ *i* $(f \Leftrightarrow g)$ *takođe iskazne formule.*

Iskazna slova zovemo i *iskazne promenljive* ili *iskazne varijable*. Elemente skupova P i $\{\top, \bot\}$ zovemo *atomičkim iskaznim formulama*. *Literal* je iskazna formula koja je atomička ili je negacija atomičke iskazne formule. *Klauza* je disjunkcija literala.

Definicija 8 *Skup* podformula *formule* A *je najmanji skup formula koje zadovoljavaju sledeće uslove:*

- *svaka iskazna formula* A *je podformula sama sebi,*
- *ako je* A *jednako* $\neg B$*, onda je svaka podformula formule* B *istovremeno i podformula formule* A*; ako je* A *jednako* $B \wedge C$*,* $B \vee C$*,* $B \Rightarrow C$ *ili* $B \Leftrightarrow C$*, onda je svaka podformula formule* B *i svaka podformula formule* C *istovremeno i podformula formule* A*.*

Definicija 9 *Za formulu kažemo da je u* konjunktivnoj normalnoj formi (KNF) *ukoliko je konjunkcija disjunkcija literala tj. ukoliko je oblika*

$$\bigwedge_i \bigvee_j l_{ij}$$

gde su l_{ij} *literali.*

Funkcije $v : P \to \{0, 1\}$, koje svakoj iskaznoj promenljivoj dodeljuje vrednost 0 (netačno) ili 1 (tačno), zovemo *valuacijama*. Svaka valuacija v jedinstveno određuje funkciju I_v koja slika skup svih iskaznih formula u dvočlani skup $\{0, 1\}$. Funkciju I_v zovemo *interpretacijom* za valuaciju v.

Definicija 10 *Funkcija* I_v *definiše se na sledeći način:*

- $I_v(p) = v(p)$*, za svaki element* p *skupa* P*;*

- $I_v(\top) = 1$ *i* $I_v(\bot) = 0$*;*
- $I_v(\neg f) = 1$ *ako je* $I_v(f) = 0$ *i* $I_v(\neg f) = 0$ *ako je* $I_v(f) = 1$*;*
- $I_v(f \wedge g) = 1$ *ako je* $I_v(f) = 1$ *i* $I_v(g) = 1$*;* $I_v(f \wedge g) = 0$ *inače;*
- $I_v(f \vee g) = 0$ *ako je* $I_v(f) = 0$ *i* $I_v(g) = 0$*;* $I_v(f \vee g) = 1$ *inače;*
- $I_v(f \Rightarrow g) = 0$ *ako je* $I_v(f) = 1$ *i* $I_v(g) = 0$*;* $I_v(f \Rightarrow g) = 1$ *inače;*
- $I_v(f \Leftrightarrow g) = 1$ *ako je* $I_v(f) = I_v(g)$*;* $I_v(f \Leftrightarrow g) = 0$ *inače.*

Vrednost funkcije $I_v(f)$ zovemo i *vrednošću iskazne formule* f *u interpretaciji* I_v.

Definicija 11 *Ako je* $I_v(f) = 1$*, onda kažemo da je iskazna formula* f *tačna u interpretaciji* I_v*. U suprotnom, ako je* $I_v(f) = 0$*, kažemo da je iskazna formula* f *netačna u interpretaciji* I_v*.*

Definicija 12 *Za valuaciju v kažemo da je zadovoljavajuća za iskaznu formulu* f *ako je* $I_v(f) = 1$*.*

Definicija 13 *Za iskaznu formulu f kažemo da je* zadovoljiva *ako za nju postoji zadovoljavajuća valuacija. Formula f je* valjana *ili* tautologija *ako je za nju svaka valuacija zadovoljavajuća. Ako ne postoji valuacija koja je zadovoljavajuća za datu formulu, onda kažemo da je ta iskazna formula* nezadovoljiva *ili* kontradikcija*.*

Da bismo dokazali da iskazna formula f nije tautologija, potrebno je i dovoljno dokazati da je $\neg f$ zadovoljiva. Takođe, da bismo dokazali da iskazna formula f nije kontradikcija, potrebno je i dovoljno dokazati da je $\neg f$ zadovoljiva. Dakle, svi problemi (da li je data iskazna formula tautologija, nezadovoljiva ili kontradikcija) se mogu svesti na problem da li je neka formula zadovoljiva, koji se zove *probem iskazne zadovoljivosti* ili problem SAT (od eng. *satisfiability*).

Definicija 14 *Za dve iskazne formule f i g kažemo da su* logički ekvivalentne *i pišemo* $f \equiv g$*, ako je svaka zadovoljavajuća valuacija za formulu f, zadovoljavajuća i za formulu g, i obrnuto.*

Ako je iskazna formula f logički ekvivalentna iskaznoj formuli g i iskazna formula g je u konjunktivnoj normalnoj formi, tada kažemo da je formula g konjunktivna normalna forma formule f. U opštem slučaju, za datu formulu postoji više sintaksno različitih formula u konjunktivnoj normalnoj formi koje su KNF polazne formule.

Definicija 15 *Za dve formule, f i g, kažemo da su* ekvizadovoljive (ili SAT ekvivalentne) *ako važi da je f zadovoljiva ako i samo ako je g zadovoljiva.*

2.4.2 Problem SAT

Problem iskazne zadovoljivosti (SAT) je problem odlučivanja da li je data iskazna formula u konjunktivnoj normalnoj formi zadovoljiva. Drugim rečima, problem SAT je problem odlučivanja da li za datu iskaznu formulu postoji valuacija u kojoj je data iskazna formula tačna.

Eksperimentalni rezultati sugerišu da kod problema SAT postoji *fazna promena* između zadovoljivosti i nezadovoljivosti kada se odnos L/N (L je broj klauza, a N je broj iskaznih slova) povećava. Osim toga, rezultati sugerišu da postoji *prelomna tačka* ili *tačka fazne promene* c_0 za koju važi:

$$\lim_{N\to\infty} s(N,[cN]) = \begin{cases} 100\%, & c < c_0 \\ 0\%, & c > c_0 \end{cases}$$

gde je $s(N,L)$ *funkcija zadovoljivosti* koja preslikava kolekciju klasa problema SAT (gde se sve iskazne formule razvrstavaju po klasama na osnovu broja klauza i broja iskaznih slova) u interval $[0,100]$ na osnovu procenta zadovoljivih formula u konkretnoj klasi. Eksperimentalno je pokazano da među svim problemima SAT postoji obrazac *jednostavno-teško-jednostavno* kada se vrednost L/N povećava. Za male vrednosti L/N, problemi su relativno jednostavni za bilo koju proceduru odlučivanja (jer postoji veliki broj zadovoljavajućih valuacija); za velike vrednosti L/N, za probleme je relativno jednostavno pokazati da su nezadovoljivi. Najteže instance problema SAT za sve procedure odlučivanja se nalaze upravo u regionu fazne promene. Do sada ni za jedan tip problema SAT nije teorijski određena prelomna tačka, pa čak nije dokazano ni njeno postojanje (sa izuzetkom 2-SAT problema).

Prvi problem za koji je dokazano da je NP-kompletan je upravo problem SAT (*Kukova teorema*). Od tada je za većinu NP-kompletnih problema NP-kompletnost dokazana svođenjem na problem SAT. Osim toga, značaj problema SAT je otkriven i u mnogim oblastima računarstva kao što su dizajn hardvera, veštačka inteligencija, verifikacija softvera i td. S obzirom da je problem SAT NP-kompletan, to znači da svi do sada poznati metodi rešavanja problema SAT imaju eksponencijalnu složenost. Neki od tih metoda su Davis-Putnam-Logemann-Lovelandova procedura (DPLL), metod rezolucije i metod tabloa.

Za više informacija o problemu SAT pogledati [Jan04].

2.4.3 SAT rešavači

Programi koji služe za rešavanje SAT problema zovu se *SAT rešavači*. Postoje potpuni i nepotpuni SAT rešavači. Potpuni rešavači se uglavnom zasnivaju na algoritmu DPLL [Jan04]. Napredak SAT rešavača u novije vreme, pripisuje se unapređenjima osnovnog DPLL algoritma kroz sužavanje prostora pretrage tehnikama učenja (eng. *learning*), nehronološkog povratka u pretrazi (eng. *backjumping*), zatim pametnim strukturama podataka i pametnim heurističkim komponentama. Pored ovoga, koristi se napredno pretprocesiranje.

U poslednjih nekoliko godina se održavaju takmičenja SAT rešavača. Na internet adresi [SATComp] se mogu naći dosadašnji rezultati. Neki od SAT rešavača koji su na poslednjem takmičenju postigli zapaženije rezultate su `Rsat`, `Picosat`, `Minisat` i dr. Na Matematičkom fakultetu Univerziteta u Beogradu, poslednjih nekoliko godina se razvija SAT rešavač `ArgoSAT`. U radu [FM08] su

detaljno opisani metodi koji se koriste pri rešavanju problema SAT i koji, samim tim, predstavljaju sastavni deo danas najefikasnijih SAT rešavača.

2.4.4 Generisanje konjunktivne normalne forme

Prevođenje proizvoljne iskazne formule u KNF nad istim skupom promenljivih zahteva eksponencijalno vreme, čak i za prilično jednostavne formule[2].

Generisanje KNF date iskazne formule zahteva puno vremena, pa transformisanje date formule u KNF ne treba vršiti ako nije neophodno. Na primer, u ovoj tezi je potrebno da se nađe formula koja je u KNF i u nekom smislu ekvivalentna datoj iskaznoj formuli (ne mora da bude logički ekvivalentna). Preciznije, potrebno je da su te dve formule ekvizadovoljive (videti definiciju 15).

Dobijanje SAT-ekvivalentne formule je moguće prevođenjem date iskazne formule u *Cajcin (Tseitsin) definicionu normalnu formu* pri čemu se koriste dodatne promenljive. Neka je Φ data iskazna formula. Označimo sa $Sub(\Phi)$ skup svih podformula polazne formule Φ. Za svaku neatomičnu podformulu $\Psi \in Sub(\Phi)$ uvodimo novu iskaznu promenljivu p_Ψ. U slučaju da je Ψ atomična, uzimamo da je $p_\Psi = \Psi$. Cajcin definiciona normalna forma dobija se na sledeći način:

$$p_\Phi \wedge \bigwedge_{\substack{\phi \in Sub(\Phi) \\ \phi = p_{\phi_1} \otimes p_{\phi_2}}} (p_\phi \Leftrightarrow (p_{\phi_1} \otimes p_{\phi_2})) \wedge \bigwedge_{\substack{\phi \in Sub(\Phi) \\ \phi = \neg\phi_1}} (p_\phi \Leftrightarrow \neg p_{\phi_1}),$$

gde $\otimes$ označava binarni logički veznik. Da bi dobijena formula bila u KNF, potrebno je eliminisati logički veznik ekvivalencije zajedno sa ostalim veznicima. U tome pomažu pravila koja su data u tabeli 2.2.

Tabela 2.2: Pravila za transformaciju u Cajcin definicionu formu.

Tip formule	Odgovarajuća zamena
$\phi = \neg\phi_1$	$(p_\phi \vee p_{\phi_1}) \wedge (\neg p_\phi \vee \neg p_{\phi_1})$
$\phi = \phi_1 \wedge \phi_2$	$(p_\phi \vee \neg p_{\phi_1} \vee \neg p_{\phi_2}) \wedge (\neg p_\phi \vee p_{\phi_1}) \wedge (\neg p_\phi \vee p_{\phi_2})$
$\phi = \phi_1 \vee \phi_2$	$(\neg p_\phi \vee p_{\phi_1} \vee p_{\phi_2}) \wedge (p_\phi \vee \neg p_{\phi_1}) \wedge (p_\phi \vee \neg p_{\phi_2})$
$\phi = \phi_1 \veebar \phi_2$	$(\neg p_\phi \vee p_{\phi_1} \vee p_{\phi_2}) \wedge (\neg p_\phi \vee \neg p_{\phi_1} \vee \neg p_{\phi_2}) \wedge$ $\wedge (p_\phi \vee \neg p_{\phi_1} \vee p_{\phi_2}) \wedge (p_\phi \vee p_{\phi_1} \vee \neg p_{\phi_2})$

Glavni nedostatak transformacije date iskazne formule u Cajcin definicionu normalnu formu se ogleda u tome što se broj klauza i iskaznih promenljivih u velikoj meri poveća u odnosu na broj istih u polaznoj formuli.

[2] Primer takve formule je $(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee \ldots \vee (p_n \wedge q_n)$.

Glava 3

Logička kriptoanaliza

U radu [MM00] razmatrana je mogućnost upotrebe iskazne logike u *kodiranju svojstava niskog nivoa* kriptografskih problema. Pokazano je da je nalaženje modela (zadovoljavajuće valuacije) za iskaznu formulu dobijenu na osnovu konkretnog kriptografskog algoritma ekvivalentno pronalaženju ključa primenom kriptoanalitičkog napada na taj algoritam. Ovaj metod je u ovom radu nazvan *logička kriptoanaliza* i primenjen je na kriptografski algoritam DES. Delom kao nastavak ovog razmatranja, nastao je rad [JJ05] u kojem je dat opis sasvim drugačijeg pristupa osnovnom problemu (kodiranja osobina kriptografskih algoritama logičkim formulama), s tom razlikom da je ovde pokazano da se osim na kriptografske algoritme, logička kriptoanaliza može primeniti i na heš funkcije.

Ova glava sadrži četiri dela. Prvi deo, *Svođenje kriptografskih problema na problem SAT*, sadrži izvođenje formula iskazne logike na koje se svode kriptografski problemi. Drugi deo, *Pristup zasnovan na logičkim kolima*, predstavlja pregled pristupa ovom svođenju koji je opisan u [MM00]. U trećem delu, *Uniformno kodiranje bazirano na zadatoj implementaciji* je opisan potpuno drugačiji pristup koji je korišćen i u ovoj tezi (implementacioni detalji su dati u glavi 4). Na kraju, u poslednjem delu, prikazan je napad na heš funkcije koji koristi svođenje na iskazne formule ali samo kao jedan svoj deo.

3.1 Svođenje kriptografskih problema na problem SAT

Zbog mogućnosti primene u kriptografiji, heš funkcije se često ubrajaju u kriptografske algoritme. Međutim, između kriptografskih algoritama i heš funkcija postoji izvesna razlika. Većina kriptografskih algoritama, koji su danas u upotrebi, je zasnovana na tajnosti ključa, dok kod heš funkcija ključ ne postoji. Pod razbijanjem kriptografskog algoritma nekada se podrazumeva nalaženje ključa pod uslovom da su poznati jedan par ili više parova otvorenih tekstova i odgovarajućih šifrata. Zbog nepostojanja ključa, ovakav napad je nemoguće primeniti na heš funkciju. S druge strane, postoje napadi na heš funkcije koji se ne primenjuju na kriptografske algoritme (nalaženje kolizija). Zbog navedenih razloga, u ovom poglavlju je posebno opisano izvođenje iskaznih formula kojima se kodiraju heš funkcije, a posebno je ovo urađeno za kriptografske algoritme.

3.1.1 Kodiranje heš funkcija

Označimo sa *hash* heš funkciju čije su izlazne vrednosti fiksne dužine N. Sa $p_1p_2\dots p_M$ označimo ulaznu poruku dužine M ($M \leq N$)[1] koju heš funkcija transformiše u izlaznu sekvencu bitova $h_1h_2\dots h_N$ ($hash(p_1p_2\dots p_M) = h_1h_2\dots h_N$). Postavlja se pitanje kako ovu transformaciju izraziti pomoću iskazne formule. Za početak, očigledno je da rezultujući bitovi h_i zavise od bitova polazne poruke p_j; tačnije, rezultujući bitovi se mogu izraziti kao iskazne formule u kojima se pojavljuju bitovi polazne poruke. Dobijene formule su, očekivano, kompleksne zbog toga što i same izražavaju kompleksnost heš funkcija. Međutim, i pored toga efektivno dobijanje tih formula jeste moguće.

Kod napada na heš funkciju, podrazumeva se da je poznata sama heš funkcija ($hash$), sekvenca $h_1h_2\dots h_N$ (heš vrednost) i dužina polazne poruke, M. Označimo sa $H_i(p_1p_2\dots p_M)$ formulu koja odgovara izračunavanju bita h_i heš vrednosti. Za navedene podatke, cilj je otkriti vrednosti p_1, p_2 $\dots$ p_M, takve da važi $hash(p_1p_2\dots p_M) = h_1h_2\dots h_N$. Drugim rečima, traži se valuacija v za koju važi $I_v(H_i(p_1, p_2, \dots, p_M)) = h_i$ ($i = 1, 2, \dots, N$) gde I_v označava interpretaciju određenu valuacijom v. Zahtev koji tražena valuacija treba da ispuni, može biti opisan sledećom jednakošću:

$$I_v(H_i(p_1, p_2, \dots, p_M)) = \begin{cases} 1, & \text{ako je } h_i = 1 \\ 0, & \text{ako je } h_i = 0 \end{cases}.$$

Dalje, uvedimo sledeću oznaku:

$$\overline{H_i}(p_1, p_2, \dots, p_M) = \begin{cases} H_i(p_1, p_2, \dots, p_M), & \text{ako je } h_i = 1 \\ \neg(H_i(p_1, p_2, \dots, p_M)), & \text{ako je } h_i = 0 \end{cases}.$$

Očigledno, formula $\overline{H_i}$ je tačna u valuaciji v akko su u toj valuaciji promenljivama p_i pridružene vrednosti za koje važi da je $H_i(p_1, p_2, \dots, p_M) = h_i$. U opštem slučaju, takvih valuacija ima više od jedne, a među njima je i valuacija koja je određena polaznom porukom koju pokušavamo da nađemo. Skup valuacija može biti redukovan zahtevom da ona bude zadovoljavajuća za sve formule $\overline{H_i}$ ($i = 1, 2, \dots, N$). Ako iskaznu formulu $\mathcal{H}$ definišemo sa:

$$\mathcal{H}(p_1, p_2, \dots, p_M) = \bigwedge_{i=1,2,\dots,N} \overline{H_i}(p_1, p_2, \dots, p_M),$$

razbijanje polazne heš funkcije svodimo na problem nalaženja valuacije za koju je formula $\mathcal{H}$ zadovoljiva. Problem nalaženja takve valuacije je, u smislu današnjih SAT rešavača, identičan problemu određivanja da li je formula $\mathcal{H}(p_1, p_2, \dots, p_M)$ zadovoljiva. Naime, zadovoljivost date iskazne formule se ispituje tako što se pokušava naći zadovoljavajuća valuacija. Međutim, generalno, to još uvek ne znači da smo dobili formulu ($\mathcal{H}$) koja je zadovoljiva samo za jednu valuaciju. Zbog ranije objašnjenog problema kolizije koji je prisutan kod heš funkcija, postoji mogućnost da za dve različite ulazne poruke $p_1p_2\dots p_M$ i $q_1q_2\dots q_M$ važi da je

$$hash(p_1p_2\dots p_M) = hash(q_1q_2\dots q_M),$$

[1] Kvalitetna heš funkcija treba da se ponaša kao generator slučajnih brojeva. Odavde sledi da je za $M = N$ mala verovatnoća da će generator slučajnih brojeva biti permutacija, tj. da neće doći do kolizije. Međutim, za $M > N$ pojava kolizije je neizbežna, bez obzira na heš funkciju. Stoga je ovde pretpostavljeno da je $M \leq N$, da bi se mogućnost pojave kolizije smanjila.

što znači da postoje bar dve različite valuacije za koje je formula $\mathcal{H}$ zadovoljiva. To dalje znači da postoji mogućnost da će SAT rešavač naći bitove ulazne poruke koje nismo očekivali. Sledi da se na ovaj način može automatizovati proces pronalaženja kolizija kod heš funkcija.

Pronalaženje kolizija kod heš funkcija. Problem traženja kolizija kod heš funkcija se može formulisati na sledeći način: za datu heš funkciju treba naći dva niza $p_1p_2 \dots p_M$ i $p'_1p'_2 \dots p'_M$ koji imaju istu heš vrednost. Ovo se može kodirati sledećom formulom iskazne logike:

$$\bigwedge_{i=1,2,\dots,N} (H_i(p_1, p_2, \dots, p_M) \Leftrightarrow H_i(p'_1, p'_2, \dots, p'_M)) \wedge \neg \bigwedge_{i=1,2,\dots,M} (p_i \Leftrightarrow p'_i).$$

3.1.2 Kodiranje kriptografskih algoritama

Osnovna ideja ostaje ista: sekvence bitova koji čine otvoreni tekst, šifrat i ključ, označimo ih sa $\mathbf{P}$, $\mathbf{C}$ i $\mathbf{K}$ respektivno, razmatrati kao nizove iskaznih promenljivih, P, C, K koje imaju vrednost *tačno* ako je odgovarajući bit jednak 1, odnosno vrednost *netačno* ako je odgovarajući bit jednak 0. Ako sa $\mathcal{H}(P, K, C)$ označimo iskaznu formulu koja je dobijena kodiranjem osobina kriptografskog algoritma, za nju važi da je tačna za onu valuaciju koja je određena sekvencama bitova za koje važi $\mathbf{C} = E_{\mathbf{K}}(\mathbf{P})$.

Postoje različiti tipovi kriptoanalitičkih napada na osnovu poznatih informacija. Pri razbijanju kriptografskih problema u ovoj tezi, podrazumeva se da je poznat jedan par ili više parova otvorenih tekstova sa odgovarajućim šifratom, a da je cilj na osnovu tih podataka rekonstruisati ključ. Stoga, ako sa v_P i v_C označimo istinitosne vrednosti iskaznih promenljivih, redom, otvorenog teksta (P) i šifrata (C), tada je potrebno da naći zadovoljavajuću valuaciju formule $\mathcal{H}(v_P, K, v_C)$, jer upravo ona određuje vrednost ključa K. U slučaju DES algoritma, dužina ključa je 56, pa će u formuli $\mathcal{H}(v_P, K, v_C)$ da bude upravo toliko iskaznih promenljivih.

U slučaju da je poznato n različitih parova otvorenih tekstova i odgovarajućih šifrata, pretraga bi se, možda, mogla ubrzati tako što se razmatra sledeća formula:

$$\bigwedge_{i=1}^{n} \mathcal{H}({v_P}^i, K, {v_C}^i).$$

Na ovaj način je problem pronalaženja ključa kriptografskog algoritma sveden na problem zadovoljivosti. Pronalaženje valuacije (v_K) koja nepoznatim iskaznim promenljivama ključa dodeljuje istinitosne vrednosti takvih da je formula $\mathcal{H}(v_P, v_K, v_C)$ tačna je ekvivalentno razbijanju kriptografskog algoritma u slučaju da je poznat otvoreni tekst i odgovarajući šifrat. S obzirom da su kriptografski algoritmi dizajnirani tako da budu teški za razbijanje, dobijene formule će biti teške za ispitivanje zadovoljivosti.

3.2 Pristup zasnovan na logičkim kolima

Ovaj pristup je prvi put opisan u radu [MM00], a zasniva se na sledećem: konstruiše se logičko kolo koje predstavlja implementaciju konkretnog kriptografskog algoritma, a potom se na osnovu dobijenog kola konstruiše iskazna formula.

S obzirom da je konstrukcija logičkog kola koje implementira i neke jednostavne funkcije, prilično složena operacija, neautomatizovana konstrukcija logičkog kola koje odgovara na primer, kriptografskom algoritmu DES bi bio veliki i kompleksan posao. Zbog toga je automatizacija neophodna. Za više detalja o ovom pristupu primenjenom na algoritam DES videti [MM00].

Osnovna ideja je da se u toku prolaska kroz algoritam DES generišu formule koje odgovaraju svakoj operaciji koja je sastavni deo DES-a. Međutim, ovo nije jedina ideja koja se sledi u toku rada. Naime, neke operacije se mogu odmah kodirati iskaznim formulama bez konstrukcije odgovarajućih logičkih kola. Na primer, ako je potrebno permutovati bitove u nekoj reči, nema potrebe da se pravi logičko kolo koje implementira ovu permutaciju.

Proces generisanja iskazne formule se može podeliti na sledeća tri dela:

1. **Kodiranje generalne strukture algoritma DES.** Koristeći oznake uvedene u potpoglavlju 2.1.2, iskazne formule koje opisuju generalnu strukturu i-te runde algoritma DES se mogu zapisati u sledećem obliku:

$$\begin{array}{lcl} L_i & \Leftrightarrow & R_{i-1} \\ R_i & \Leftrightarrow & L_{i-1} \veebar F_i \\ X_i & \Leftrightarrow & E(R_{i-1}) \veebar K_i \\ F_i & \Leftrightarrow & P(S_i) \end{array}$$

   Izraz $E(R_{i-1})$ predstavlja primenu proširujuće permutacije na 32-bitni vektor R_{i-1}. U praksi, ovo znači da su neki bitovi vektora R_{i-1} dupli-rani. Vektor F_i predstavlja izlaz iz funkcije f koja je predstavljena slikom 2.1. Sa K_i je označen vektor iskaznih promenljivih koje predstavljaju i-ti podskup inicijalnog 56-bitnog ključa koji je odabran u i-toj rundi, dok X_i predstavlja 48-bitni ulaz u S-blokove. Izraz $P(S_i)$ predstavlja permutaciju izlaza iz S-blokova.

2. **Kodiranje permutacija i generisanja podključeva.** Kao što je već rečeno, postoje odstupanja od osnovne ideje i neće biti konstruisano logičko kolo koje predstavlja implementaciju permutacije.

   Označimo sa $\overline{A}$ rezultat primene matrice permutacije M, koju ćemo predstaviti kao vektor, na vektor iskaznih promenljivih A. Ako je M_j j-ti element matrice M, tada je j-ti element vektora $\overline{A}$ ($\overline{A_j}$) jednak M_j-om elementu vektora A (A_{M_j}), tj.

$$\overline{A_j} = A_{M_j}.$$

   Da bi se generisao podključ K_i, polazi se od inicijlnog 64-bitnog ključa i na njega se primenjuje permutacija (koja izbacuje svaki osmi bit) i na taj način dobija se 56 bitova. Ovih 56 bitova se dele na dve polovine, koje se nezavisno jedna od druge ciklično pomeraju za određeni broj mesta (u zavisnosti od runde). Potom se dva dela ponovo spajaju i još jednom permutuju. Od tako dobijenih 56 bitova, sa konkretnih 48 pozicija se uzimaju odgovarajući bitovi koji čine podključ K_i. Odavde sledi da je potrebno da se kodiraju u iskazne formule operacije permutacije i cikličnog pomeranja. S obzirom da je kodiranje permutacije već objašnjeno, ostaje još da se objasni kako se kodira ciklično pomeranje. Pretpostavimo da se u i-toj rundi obavlja ciklično pomeranje za s_i mesta. Potom, za j-ti bit

($j \in \{1, 2, \ldots, 28\}$), u 28-bitnom vektoru koji se ciklički pomera, pozicija p_j računa se sledećim algoritmom:

$$p'_j = j - s_i$$

if $p'_j \leq 0$
  $p_j = 28 - |p'_j|$
else
  $p_j = p'_j$

Nakon što su nove pozicije izračunate, one se mogu smestiti u matricu a ciklično pomeranje se može obaviti kao permutovanje.

3. **Kodiranje S-blokova.** Rečeno je da se svaki od osam supstitucionih blokova, uglavnom, predstavlja kao matrica čiji elementi, koje kodiramo sa 4 bita, indeksiramo pomoću 6 bitova. U radu [MM00] je za svaki S-blok napravljen PLA (eng. *Programmable Logic Array*) koji implementira funkciju koja je predstavljena konkretnim S-blokom. Kada se, na kraju, konstruiše za svaki S-blok odgovarajući PLA, dobijaju se formule sledećeg oblika:

$$C_i^{hk} \Leftrightarrow \bigwedge_j \overline{X_i^j}, \quad k = 1, 2, \ldots, N_i^h \tag{3.1}$$

$$S_i^h \Leftrightarrow \bigvee_{j=1}^{N_i^h} C_i^{hj}, \quad h = 1, 2, \ldots, 32$$

S obzirom na kompleksan zapis ovih formula, sledi kraće objašnjenje. Sa S_i^h je označen h-ti izlazni bit iz S-blokova u i-toj rundi. Ovaj bit će imati vrednost 1 u slučaju da ulaznih 48 bitova (X_i) ima jednu od, u opštem slučaju, više vrednosti. Broj tih različitih vrednosti je određen logičkim tablicama koje opisuju konstruisane PLA, a ovde je označen sa N_i^h, jer je on, u opštem slučaju, različit za svaki izlazni bit svake runde. Za j-ti od tih N_i^h različitih ulaza, C_i^{hj} se konstruiše tako što se napravi konjunkcija, eventualno negiranih, iskaznih promenljivih koje čine ulaz. Zbog toga je uvedena oznaka:

$$\overline{X_i^j} = \begin{cases} X_i^j, & \text{ako je } j\text{-ti bit u mogućem ulazu 1} \\ \neg X_i^j, & \text{ako je } j\text{-ti bit u mogućem ulazu 0} \end{cases}.$$

U jednačini 3.1 bi se moglo očekivati da za indeks j važi $j \in \{1, 2, \ldots, 48\}$. Međutim, za svako h, S_i^h zavisi samo od određenih 6 bitova ulaza (X_i). Tačnije, S_i^h zavisi od $X_i^{(\lfloor \frac{h-1}{4} \rfloor + 1) \cdot j}$ za $j \in \{1, 2, \ldots, 6\}$.

Nakon što su svi koraci algoritma kodirani logičkim formulama, konjunkcija svih dobijenih formula predstavlja upravo formulu $\mathcal{H}(P, K, C)$. Zanemarujući detalje i ostavljajući samo jedan indeks i, koji označava indeks tekuće runde, u

upotrebi, i-ta runda DES-a se može predstaviti sledećim formulama:

$$\begin{aligned} L_i &\Leftrightarrow R_{i-1} \\ R_i &\Leftrightarrow L_{i-1} \veebar F_i \\ F_i &\Leftrightarrow P(S_i) \\ S_i &\Leftrightarrow \bigvee C_i \\ C_i &\Leftrightarrow \bigwedge X_i \\ X_i &\Leftrightarrow E(R_{i-1}) \veebar K_i \end{aligned}$$

U radu [MM00] su osim opisa ovog pristupa dati i rezultati dobijeni pri pokušaju razbijanja DES algoritma. Opisanim pristupom su razbijane oslabljene verzije algoritma DES, dobijene variranjem broja rundi $(1, 2, 3, 4, 8, 16)$. Takođe je variran i broj poznatih parova otvorenih tekstova i odgovarajućih šifrata $(1, 2, 4, 8)$. Prvo su generisani parovi otvorenih tekstova i odgovarajućih šifrata (za slučajno generisani ključ), a potom su na opisani način generisane iskazne formule (u kojima promenljivama odgovaraju bitovi ključa). Za rešavanje dobijenih iskaznih formula su korišćena tri različita SAT rešavača — `TABLEAU`, `SATO` i `Loop-Back CSP`. Najbolji rezultati su postignuti korišćenjem `Loop-Back CSP` koji je jedini razbio 3 runde algoritma DES za 75.02s, uz poznavanje 8 parova otvorenih tekstova i odgovarajućih šifrata.

Očigledno je da se ovaj pristup može primeniti i na druge kriptografske algoritme jer se za sve operacije koje čine proizvoljan kriptografski algoritam mogu konstruisati logička kola koja predstavljaju implementaciju tih operacija. Nakon toga, dobijanje odgovarajućih iskaznih formula je trivijalno. Međutim, i pored toga što je ovaj pristup primenljiv na proizvoljan kriptografski algoritam, za svaki konkretan kriptografski algoritam potrebno je prateći njegovu specifikaciju konstruisati logičko kolo koje implementira baš taj algoritam. Drugim rečima, za svaki kriptografski algoritam je potrebno napisati program koji konstruiše iskaznu formulu koja opisuje logičko kolo koje odgovara samom algoritmu. U sledećem poglavlju će biti dat opis jednog elegantnijeg pristupa koji se na mnogo jednostavniji način primenjuje na proizvoljan kriptografski algoritam.

3.3 Uniformno kodiranje bazirano na zadatoj implementaciji

U ovom poglavlju biće ukratko predstavljen jedan uniforman pristup problemu kodiranja kriptografskih problema formulama iskazne logike, prvi put opisan u radu [JJ05]. U ovom poglavlju je dat samo kratak opis osnovne ideje i eksperimentalnih rezultata koji su opisani u pomenutom radu jer je cela sledeća glava posvećena implementaciji ove ideje. Za razliku od poglavlja 3.1, u kojem je pravljena razlika između heš funkcija i kriptografskih algoritama (zasnovanih na ključu), u ovoj glavi ćemo pod kriptografskim algoritmom podrazumevati sve kriptografske algoritme kao i heš funkcije.

Kao što je već rečeno, sigurnost svakog kvalitetnijeg kriptografskog algoritma se ne zasniva na tajnosti algoritma. Zbog toga, specifikacije većine danas poznatih i korišćenih kriptografskih algoritama su dostupne bilo u deskriptivnoj formi, bilo u formi konkretne implementacije na nekom od danas popularnih programskih jezika. Većina kriptografskih algoritama se sastoji od velikog broja

logičkih operacija, pa je *ručno* konstruisanje iskazne formule praktično nemoguće. U nastavku će biti opisan mehanizam koji je implementiran u okviru rada na ovoj tezi, a koji omogućava automatsko generisanje iskazne formule zasnovane na postojećoj implementaciji kriptografskog algoritma na programskom jeziku C/C++. Ovaj pristup je prvi put primenjen na heš funkcije MD4 i MD5 u [JJ05]. Međutim, zbog svoje univerzalnosti, on se može primeniti za kodiranje u iskazne formule ne samo heš funkcija, već i drugih kriptografskih algoritama (u okviru ove teze to je urađeno za CMEA i DES).

Osnovna ideja ovog pristupa je korišćenje jednog svojstva objektno orijentisanih programskih jezika — *preopterećivanju operatora* (eng. *operator overloading*). Preopterećivanje operatora je specifičan slučaj polimorfizma, gde se operatori koji se koriste na uobičajen način u programiranju (na primer `+`, `-` ili `+=`), mogu razmatrati kao polimorfne funkcije. Drugim rečima, njihovo značenje može biti različito za različite tipove operanada. Ovo omogućava da se istim operatorima daje različita semantika (koja zavisi, kao što je rečeno, od tipa operanada), što napisani kôd ponekad može da čini manje čitljivim i težim za razumevanje. Nepažljivim korišćenjem ovog svojstva, u jezicima koji ga poseduju, može da dođe do grešaka koje se teško otkrivaju. Zbog toga, preopterećivanje operatora treba koristiti isključivo kada je neophodno.

Nakon opisa ovog pristupa, u radu [JJ05] su dati rezultati dobijeni razbijanjem heš funkcija MD4 i MD5. U poglavlju 2.2 je rečeno da se heš funkcija MD5 sastoji od 4 slične runde. U radu [JJ05] je variran kako broj rundi $(1, 2, 3, 4)$, tako i veličina ulaza $(1, 2, \ldots, 16$ bitova). Za svaki broj rundi i za svaku veličinu ulaza, generisan je određen broj parova poruka i odgovarajućih heš vrednosti. Potom je za svaki par na opisan način generisana iskazna formula za čije se rešavanje korišćen SAT rešavač `zChaff`. Kao što je i očekivano, kako se veličina ulaza povećavala, vreme potrebno rešavaču `zChaff` da reši dobijenu formulu se eksponencijalno povećavalo. Rezultati su dati za dužine ulazne poruke do 16 bitova jer je za duže poruke, vreme rešavanja odgovarajuće formule bilo veće od 10000s (što je bilo vremensko ograničenje postavljeno od strane autora). Za ulaznu poruku dužine 16 bitova je heš funkcija MD5 razbijena za nekoliko hiljada sekundi.

3.4 Primene u klasičnoj kriptoanalizi

Dva pristupa, opisana u prethodnim poglavljima, demonstriraju kako se rešavanje problema iskazne logike može iskoristiti u kriptoanalizi nekog kriptografskog algoritma. Oni su, u određenom smislu, veoma slični jer ne ulaze u smisao i ne koriste nikakve specifičnosti konkretnog kriptografskog algoritma. Takođe, u oba pristupa se kompletan kriptografski algoritam kodira formulom iskazne logike čijim rešavanjem dobijamo nepoznate vrednosti (bitove ključa, otvorenog teksta, ...). Međutim, ovo nije jedina moguća primena iskazne logike u kriptoanalizi. Na primer, u radu [MZ] je modifikovan već postojeći napad[2] na familije heš funkcija MDx i SHA-x (zasnovan na nalaženju kolizija), tako što je jedan deo tog napada izmenjen korišćenjem svođenja na probleme iskazne logike.

[2]Grupa kineskih kriptografa je otkrila *ranjivost* nekoliko značajnih heš funkcija (među njima su i MD4 i MD5). Međutim, konstruisani napad se sastojao od koraka koji svi nisu mogli biti automatizovani, a sadržavali su veliki broj stanja koje je bilo potrebno održavati. U radu [MZ] je iskorišćen ovaj napad uz otklanjanje uočenih nedostataka.

Sledi kratak opis ovog napada.

Notacija. Kao što je već rečeno, napad je zasnovan na nalaženju kolizija. Tačnije, potrebno je naći dve 512-bitne poruke, $M = (m_0, \ldots, m_{15})$ i $M' = (m'_0, \ldots, m'_{15})$ (gde su m_k i m'_k dužine 32 bita), za koje važi:

$$H(M) = H(M')$$

gde je H data heš funkcija. Sa (a_i, b_i, c_i, d_i) označimo stanje u i-toj rundi (na slici 2.2 je stanje označeno četvorkom (A, B, C, D)) pri šifrovanju poruke M, a sa (a'_i, b'_i, c'_i, d'_i) stanje pri šifrovanju poruke M'. Razlikovaće se dva tipa *razlika* (eng. *differentials*):

$$\Delta^{+} a_i = a_i - a'_i \pmod{2^{32}} \text{ i slično za } \Delta^{+} m_i, \Delta^{+} b_i, \Delta^{+} c_i, \Delta^{+} d_i$$

i

$$\Delta^{\oplus} a_i = a_i \oplus a'_i \text{ i slično za } \Delta^{\oplus} m_i, \Delta^{\oplus} b_i, \Delta^{\oplus} c_i, \Delta^{\oplus} d_i.$$

Uvedimo oznaku $\circ$ koja će označavati i $+$ (koristi se kod heš funkcija familije MDx) i $\oplus$ (koristi se kod heš funkcija familije SHA-x). Neka je

$$\Delta^{\circ}_i = (\Delta^{\oplus} a_i, \Delta^{\oplus} b_i, \Delta^{\oplus} c_i, \Delta^{\oplus} d_i).$$

Tada sekvencu $\Delta^{\circ}_0, \Delta^{\circ}_1, \ldots$ zovemo staza različitosti (eng. *differential path*).

Opis napada. Generički napadi na heš funkciju H tretiraju funkciju H kao *crnu kutiju*. Ideja jednog od njih je korišćena i u ovom napadu: pronalaženjem dve poruke x i y za koje važi $x = y \circ \delta$ (za neko fiksirano δ) i $H(x) = H(y)$ pronađena je kolizija. Kompleksnost problema nalaženja kolizije na ovaj način je 2^n, gde je $n = 128$ (odnosno 160) za familiju heš funkcija MDx (odnosno SHA-x). Ova kompleksnost se, pažljivim odabirom vrednosti δ, kao i korišćenjem izvesnih heuristika, može svesti na 2^{42}.

Kompletan napad koji je korišćen u [MZ] se sastoji od četiri odvojene celine:

1. Pažljivo odabrati $\Delta^{\circ} m_0, \ldots, \Delta^{\circ} m_{15}$.
2. Pažljivo odabrati $\Delta^{\circ}_0, \ldots, \Delta^{\circ}_{r-1}$, gde je r broj rundi (48 za MD4, 64 za MD5 ili 80 za SHA).
3. Pronalaze se uslovi koje trebaju da ispunjavaju poruka $M = (m_0, \ldots, m_{15})$ kao i međustanja $a_i, \ldots, d_i$ koji su dovoljni da bi se moglo garantovati (sa velikom verovatnoćom) da par poruka M, $M' = (m_0 \circ \Delta^{\circ} m_0, \ldots, m_{15} \circ \Delta^{\circ} m_{15})$ zadovoljava odabranu stazu različitosti $\Delta^{\circ}_0, \ldots, \Delta^{\circ}_{r-1}$.
4. Pronaći poruku M koja ispunjava uslove iz prethodne tačke.

Prva dva dela napada se uglavnom rade ručno ili primenom neke heuristike. U trećem delu je potrebno definisati uslove koje trebaju da zadovoljavaju promenljive stanja (a_i, b_i, c_i, d_i). Ti uslovi su oblika: *bit najmanje težine u c_7 treba da bude 0*, ili *jedanaesti bit u a_7 treba da bude 1*, a ponekad ih ima i nekoliko stotina (zavisi od konkretnog algoritma — za MD4 ih ima 122). Međutim, većina ovih uslova se tiče promenljivih koje se javljaju u prvih 25 rundi (upravo zbog toga nije moguće korišćenjem samo ove tehnike razbiti, na

primer, heš funkciju MD5). U četvrtom delu, u postojećem napadu se koriste tehnike modifikacije jedne ili više poruka (eng. *(single/multi)-message modification techniques*), opisane u [MZ]. Međutim, u toku ovog dela napada potrebno je održavati informaciju o velikom broju stanja i promenljivih, pa je ovaj problem prevaziđen kodiranjem kompletne staze različitosti iskaznom formulom, da bi se nalaženje poruke koja ga zadovoljava prepustilo SAT rešavaču.

Rezultati. U radu [MZ] su objavljeni rezultati koji su dobijeni pri pokušaju razbijanja heš funkcija MD4, MD5 i SHA-0. U toku eksperimenata, variran je broj rundi (u poglavlju 2.2 je rečeno da heš funkcija MD5 ima 4 runde od kojih svaka ima po 16 operacija, dok je u radu [MZ] pod rundom podrazumevana svaka operacija — zbog toga u ovom radu MD5 ima 64 runde) i dobijeni su sledeći rezultati:

- heš funkcija MD4 je razbijena cela;
- razbijeno je 46 (od 64) rundi heš funkcije MD5;
- razbijeno je 35 (od 80) rundi heš funkcije SHA-0.

Iniciran, pre svega, ovim radom, u februaru 2008. godine je objavljen predlog teze [JCV]. Za razliku od [MZ], ova teza će se baviti kriptoanalizom isključivo heš funkcija iz familije MD-x. Cilj ove teze je konstruisanje alata koji će služiti kriptoanalitičarima u razotkrivanju slabosti heš funkcija, a biće zasnovan na pristupu predstavljenom u [MZ]. Takođe, cilj je uspešno generisanje kolizije za kompletne heš funkcije MD4 i MD5.

Glava 4

Implementacija

U prethodnoj glavi su opisana dva različita pristupa problemu svođenja kriptografskih problema na problem SAT. S obzirom da je pristup pod nazivom *uniformno kodiranje bazirano na zadatoj implementaciji* široko primenljiv, a primenjen do sada samo na heš funkcije, ova teza je zasnovana na tom pristupu. Ova glava predstavlja pregled svih korišćenih implementacionih tehnika korišćenih u toku rada na ovoj tezi.

U prvom poglavlju, *Osnovni tipovi*, dat je opis osnovnih tipova koji su implementirani, kao i razlozi zbog kojih su uvedeni. Drugo poglavlje sadrži opis implementacije osnovne ideje ovog rada — preopterećivanja operatora. Treće poglavlje je, na neki način, nastavak prve glave — u ovoj glavi su predstavljeni problemi na koje se nailazilo u toku rada, kao i metodi kojima su ti problemi prevaziđeni. Opisani su novi tipovi i razlozi njihovog uvođenja. U četvrtom poglavlju je opisan proces povezivanja implementiranih tipova sa postojećim realizacijama kriptografskih algoritama i heš funkcija. Peto poglavlje detaljno opisuje konstrukciju formule u KNF na osnovu date formule. Takođe je data specifikacija datoteka u DIMACS formatu. U šestom poglavlju je opisano kako se pomoću postojećih SAT rešavača mogu rešavati dobijene formule koje su prethodno zapisane u datoteku u odgovarajućem formatu. Na kraju, sedmo poglavlje govori o korišćenju SAT rešavača u rešavanju konstruisanih iskaznih formula.

4.1 Osnovni tipovi

U standardnim implementacijama heš funkcija i kriptografskih algoritama, uglavnom se sve operacije primenjuju na operande koji su nizovi bitova čija je dužina 8, 16 ili 32 jer se ti operandi mogu u programskim jezicima predstaviti postojećim primitivnim tipovima kao što su neoznačeni 8-bitni karakter (*char*), 16-bitni ceo broj (*short*) i 32-bitni ceo broj (*int*)[1]. U slučaju da je u algoritmu potrebno raditi sa dužim nizovima bitova, onda se oni u konkretnoj implementaciji predstavljaju kao nizovi nekoliko elemenata koji su jednog od postojećih tipova. Na primer, kod algoritma MD5 izlazni niz bitova je dužine 128, pa bi se on mogao u konkretnoj implementaciji predstaviti, na primer, kao niz

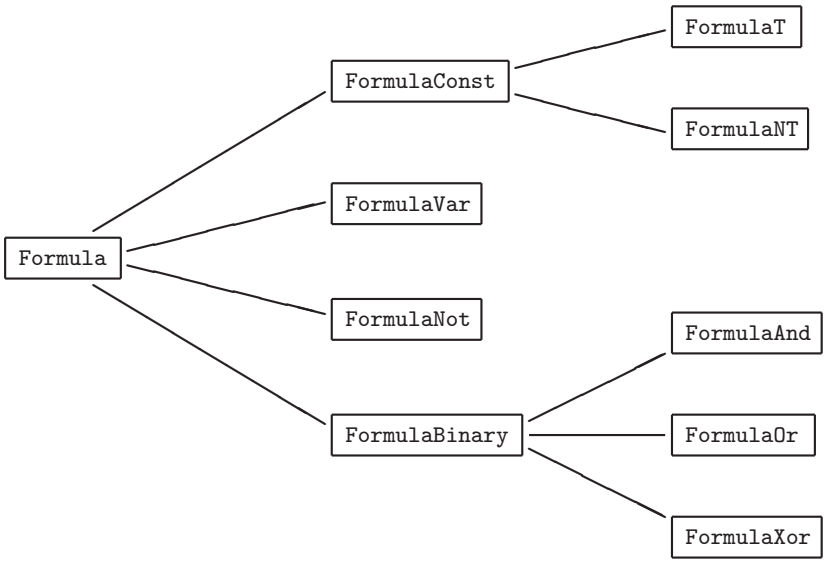
[1] Treba biti oprezan jer broj bitova kojima se predstavlja određen tip podataka zavisi od mnogih faktora (konkretnog računara, prevodioca, itd.) — ovde su date najčešće vrednosti.

od četiri 32-bitna neoznačena cela broja. Kao što je rečeno u prethodnoj glavi, osnovna ideja pristupa je da se nizovi bitova otvorenog teksta, šifrata i ključa, razmatraju kao nizovi iskaznih promenljivih. Osim toga, potrebno je obezbediti da se svaka operacija koja učestvuje u algoritmu, kodira na odgovarajući način iskaznom formulom. To je omogućeno konstruisanjem novog tipa podataka koji je implementiran u vidu klase `Word` čiji su privatni članovi prikazani na slici 4.1.

```
class Word {
public:
    ...
private:
    Formula** bitArray; /* Niz pokazivaca na elemente tipa Formula. */
    unsigned int size; /* Duzina reci. */
};
```

Slika 4.1: Privatni čalnovi klase `Word`.

Bitovi koji čine objekat klase `Word` su pokazivači na elemente tipa `Formula`, koji predstavljaju iskazne formule. Ove formule upravo oslikavaju sve promene koje se dese u toku algoritma nad odgovarajućim bitovaima. Tip `Formula` je kao i prethodni tip realizovan u vidu klase. `Formula` predstavlja baznu klasu čitave hijerarhije klasa koje predstavljaju sve moguće tipove formula. Ta hijerarhija je predstavljena na slici 4.2.



Slika 4.2: Hijerarhija klasa kojima se predstavljaju formule.

4.2 Preopterećivanje operatora

Nakon konstruisanja potrebnih tipova, treba obezbediti kodiranje operacija konkretnog algoritma. Kao što je rečeno, ovo će biti urađeno korišćenjem osobine preopterećivanja operatora (za novonapravljeni tip `Word`). Kao prvo, potrebno je preopteretiti logičke veznike `&` (konjunkcija), `|` (disjunkcija), `^` (ekskluzivna disjunkcija) i `~` (negacija). Na primer, operator `&` će izračunavati konjunkciju dva objekta koji su tipa `Word`, na očekivan način (tj. onako kako se određuje konjunkcija dva elementa neoznačenog celobrojnog tipa). Dakle, pri preopterećivanju `&` operatora, biće kreiran novi objekat tipa `Word` čiji će svaki bit biti novi objekat tipa `Formula` (i to konkretnog tipa `FormulaAnd` koji opisuje formulu koja predstavlja logičku konjunkciju). Slika 4.3 prikazuje implementaciju preopterećenog operatora konjunkcije.

```
Word Word::operator & (const Word &w) const {

    Word res(size); /* Rezultat konjunkcije. */

    /* Racunamo konjunkciju. */
    for (unsigned int i = 0; i < size; i++)
        res.setFormulaAt(i, Formula::makeAnd(bitArray[i], w.bitArray[i]));

    return res; /* Vracamo izracunatu konjunkciju. */
}
```

Slika 4.3: Implementacija operatora &.

Kôd bi verovatno bio jasniji da je u ovom kôdu jedina linija u `for`-petlji

```
res.setFormulaAt(i, new FormulaAnd(bitArray[i], w.bitArray[i]));
```

Međutim, u slučaju da je bar jedan od dva operanda logička konstanta (za koju takođe postoji odgovarajuća klasa `FormulaConst` i izvedene klase `FormulaT` i `FormulaNT`), može da dođe do redukcije. Na primer, ako je u nekom koraku algoritma potrebno naći konjunkciju dve formule `f1` i `f2`, tada se to može uraditi sledećom linijom kôda:

```
f = new FormulaAnd(f1, f2);
```

Međutim, ako je bar jedna od formula `f1` i `f2` logička konstanta, onda je nepotrebno kreirati novu formulu tipa `FormulaAnd` jer se ova konjunkcija može izračunati na sledeći način:

if `f1` je konstanta
  if `f1` je **false**
    `f = f1`
  else
    `f = f2`
else if `f2` je konstanta
  if `f2` je **false**
    `f = f2`
  else
    `f = f1`

```
else
   f = new FormulaAnd(f1, f2)
```

Eliminacija konstanti je, takođe, poželjna jer se u suprotnom povećava broj promenljivih pri izgradnji Cajcin definicione forme. Zbog toga je implementirana grupa statičkih proizvodnih metoda klase `Formula` koje su upravo implementirane na opisani način:

```
static Formula* makeNot(Formula *f);
static Formula* makeOr(Formula *f1, Formula *f2);
static Formula* makeXor(Formula *f1, Formula *f2);
static Formula* makeAnd(Formula *f1, Formula *f2);
```

Slika 4.4 prikazuje implementaciju prve od njih.

```
Formula* Formula::makeNot(Formula *f) {

    Formula *tmp;
    /* U slucaju da je u pitanju logicka konstanta ... */
    if (dynamic_cast<FormulaConst*>(f) != NULL) {
        /* ... tacnije, konstanta cija je vrednost 'tacno' ... */
        if ((dynamic_cast<FormulaConst*>(f))->GetValue() == true)
            /* ... negacija je logicka konstanta 'netacno'. */
            tmp = new FormulaNT();
        /* ... tacnije, konstanta cija je vrednost 'netacno' ... */
        else
            /* ... negacija je logicka konstanta 'tacno'. */
            tmp = new FormulaT();

        ...->Remove(f);
    }
    /* U slucaju da je u pitanju formula koja nije logicka konstanta ... */
    else
        /* ... nalazimo negaciju te formule. */
        tmp = new FormulaNot(f);

    return ...->Get(tmp);
}
```

Slika 4.4: Implementacija statičke metode `makeNot`.

Aritmetički operatori se preopterećuju na sličan način kao i logički, ali su stvari za nijansu komplikovanije. Razlog je to što se operacije ne mogu jednostavno primeniti na parove odgovarajućih bitova jer vrednost svakog bita zavisi od svih prethodnih bitova. Slika 4.5 prikazuje implementaciju preopterećenog operatora sabiranja.

4.3 Unapređenja

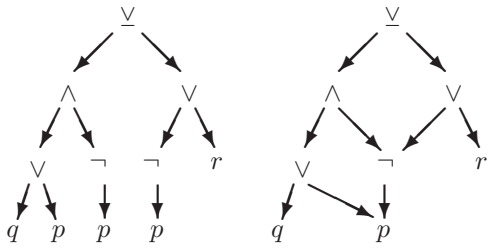
U prvoj verziji implementacije, svaki put kada je trebalo napraviti novu formulu, nije vođeno računa o tome da li možda ta formula već postoji sačuvana u memoriji. Zbog toga je vrlo često dolazilo do pravljenja duplikata. Posledica toga su bili neočekivano veliki memorijski zahtevi. Već kod problema *korenovanja* (*Naći prirodan broj čiji je kvadrat dat.*), nakon nalaženja definicione normalne forme i zapisivanja formule u datoteku, za 8-bitne brojeve datoteka je bila veća

```
Word Word::operator + (const Word &w) {

    Word plusWord(w.getSize());

    Formula *c = ...->Get(new FormulaNT()); /* Prenos. */
    Formula *sumF;

    /* Izracunavamo sumu pocevsi od bita najmanje tezine. */
    for(int i = size - 1; i >= 0; i--) {
        Formula *andF = NULL, *orF = NULL, *xorF = NULL;

        andF = Formula::makeAnd(bitArray[i], w.bitArray[i]);
        orF = Formula::makeOr(bitArray[i], w.bitArray[i]);
        xorF = Formula::makeXor(bitArray[i], w.bitArray[i]);

        c->IncRefCount();
        sumF = Formula::makeXor(xorF, c); /* Suma odgovarajuceg para bitova i prenosa. */

        /* Izracunata suma se postavlja kao i-ti bit rezultata. */
        plusWord.setFormulaAt(i, sumF);

        c->DecRefCount();
        /* Izracunavamo novi prenos. */
        c = Formula::makeOr(andF, Formula::makeAnd(c, orF));
    }

    ...->Remove(c); /* Brisemo poslednji prenos. */

    return plusWord; /* Vracamo izracunatu sumu. */
}
```

Slika 4.5: Implementacija operatora +.

od 1GB. Pri rešavanju dobijene formule SAT rešavačem, prijavljena je greška pri alokaciji memorije. Za generisanje datoteke koja će sadržavati opis formule bilo je potrebno i do nekoliko minuta.

Zbog toga je upotrebljena tehnika deljenja zajedničkih termova iskorišćena u radu [FM05]. Ideja je da se u pažljivo odabranoj strukturi podataka čuvaju pokazivači ka svim do tog trenutka napravljenim formulama. Na primer, stablo formule koje je predstavljeno slikom 4.6a se transformiše u usmereni aciklični graf na slici 4.6b. Kada je potrebno napraviti novu formulu, najpre se pretražuje pomenuta struktura. Ako nova formula ne postoji, pokazivač na novu formulu se ubacuje u strukturu. U suprotnom, koristi se postojeća formula. Implementacija funkcije koja ovo radi, predstavljena je slikom 4.7. S obzirom da je potrebno



Slika 4.6: Deljenje podformula u formuli $((p \vee q) \wedge \neg p) \veebar (\neg p \vee r)$.

da operacije ubacivanja i pretraživanja budu veoma efikasne, prirodno je da se za čuvanje pokazivača na formule koristi heš tabela. Rad sa ovom heš tabelom je omogućen metodama klase `FormulaFactory` čiji je privatni član pomenuta heš tabela. S obzirom da je potrebno da u sistemu uvek postoji tačno jedan objekat klase `FormulaFactory`, ova klasa je implementirana po obrascu *unikat* (eng. *singleton*) [GHJV95].

```
Formula* Get(Formula *f) {

    /* Prolazimo kroz hes tabelu u potrazi za datom formulom. */
    FormulaPointerSet::const_iterator i = existingFormulas.find(f);

    /* U slucaju da smo nasli formulu u hes tabeli ... */
    if ( i != existingFormulas.end() ) {

        if (*i != f)
            delete f;

        /* ... vracamo pokazivac na postojecu formulu. */
        return *i;
    }
    /* U slucaju da nismo nasli formulu u hes tabeli ... */
    else {
        /* ... ubacujemo pokazivac na novu formulu u tabelu. */
        existingFormulas.insert(f);
        return f;
    }
}
```

Slika 4.7: Implementacija funkcije `Get` u klasi `FormulaFactory`.

Sve do sada opisane tehnike i uvedene klase (`Formula`, `Word` i `FormulaFactory`) su bile dovoljne za kodiranje heš funkcije MD5. Međutim, kod kriptografskih algoritama CMEA i DES pojavio se novi problem. Problem su predstavljale tabela 2.1 koja se koristi u algoritmu CMEA, kao i nekoliko matrica koje se koriste u algoritmu DES (na primer, matrice koje opisuju supstitucione S-blokove). U konkretnoj kriptografskoj implementaciji, elementima ovih matrica se pristupalo pomoću operatora indeksiranja. Indeks, kao i vrednost u matrici, je neoznačen ceo broj (neobavezno iste dužine — ovo je slučaj na primer, kod S-blokova u algoritmu DES). Problem je u tome što se indeksi računaju u toku izvršavanja algoritma na osnovu bitova poznatog otvorenog teksta i ključa. S obzirom da je pretpostavljeno da je ključ nepoznat, njegove bitove razmatramo kao iskazne promenljive. To znači da će u toku rada algoritma, indeksi biti izračunati kao iskazne formule u kojima se pojavljuju iskazne promenljive koje predstavljaju bitove ključa.

Biće opisano kako je ovaj problem rešen u algoritmu CMEA (za DES je rešen analogno). Označimo sa w ulaznih 8 bitova, a sa $r_1r_2\ldots r_8$ izlaznih 8 bitova funkcije C. Tada važi da je $C(w) = r_1r_2\ldots r_8$. U tabeli 2.1 ima 256 elemenata kojima se pristupa indeksima iz skupa $I = \{0, 1, 2, \ldots, 255\}$. Za svako $k \in \{1, 2, \ldots, 8\}$, bitu r_k mogu da budu pridružena dva skupa:

$$I_k^0 = \{i \in I,\ \text{k-ti bit broja } C(i) \text{ je } 0\} \text{ i}$$

$$I_k^1 = \{i \in I,\ \text{k-ti bit broja } C(i) \text{ je } 1\}.$$

Tada važi

$$r_k = \begin{cases} 0, & \text{ako je } w \in I_k^0 \\ 1, & \text{ako je } w \in I_k^1 \end{cases}.$$

Međutim, potrebno je r_k izraziti kao iskaznu formulu nad promenljivama koje čine ulaznu reč w. S obzirom da je $I = I_k^0 \cup I_k^1$, ako je $I_k^1 = \{x_1, x_2, \ldots, x_j\}$, r_k može da se zapiše na sledeći način:

$$r_k = \begin{cases} 1, & (w = x_1) \vee (w = x_2) \vee \ldots \vee (w = x_j) \\ 0, & \text{inače} \end{cases}.$$

Ostaje još da se za svako $l \in \{1, 2, \ldots, j\}$ izrazi $w = x_l$ zapišu u obliku iskazne formule. Kako su vrednosti x_l poznate, dovoljno je negirati formule koje predstavljaju bitove u reči w kojima odgovaraju 0-bitovi u x_l, a potom naći konjunkciju svih tih formula. Na primer, ako je $w = p_1 p_2 \ldots p_8$, onda izrazu $w = x_l$ odgovara iskazna formula $p_1^l \wedge p_2^l \wedge \ldots \wedge p_8^l$, gde je:

$$p_i^l = \begin{cases} p_i, & \text{ako je i-ti bit u } x_l \text{ jednak 1} \\ \neg p_i, & \text{ako je i-ti bit u } x_l \text{ jednak 0} \end{cases}.$$

Sada r_k može da se zapiše u sledećem obliku:

$$r_k = \bigvee_{l=1}^{j} (p_1^l \wedge p_2^l \wedge \ldots \wedge p_8^l).$$

Ovo je realizovano konstruisanjem klase `WordArray`. Na slici 4.8 je prikazana deklaracija ove klase. Privatni članovi ove klase su matrica `I`, koja odgovara nizu

```
class WordArray {
public:
        WordArray(const unsigned char *X, unsigned int n);
        Word operator[](Word w);
private:
        unsigned char I[8][256];
        unsigned char count[8];
};
```

Slika 4.8: Deklaracija klase `WordArray`.

skupova I_k^1, dok je drugi član niz `count` koji sadrži brojeve elemenata skupova I_k^1. Kao što se vidi, klasa ima samo dva metoda, a to su konstruktor i operator indeksiranja. Razlog za to je što je ova klasa uvedena samo da bi operator indeksiranja bio preopterećen. Pri definiciji promenljive ovog tipa poziva se konstruktor koji za svaki izlazni bit r_k konstruiše skup indeksa I_k^1 i elemente tog skupa smešta u niz $I[k]$. Konstruktoru se prosleđuje kao prvi argument tabela koja će biti na opisani način *indeksirana* indeksima koji su tipa `Word`. Drugi argument predstavlja broj elemenata tabele. Kada se na kreirani objekat primeni operator indeksiranja, na osnovu indeksa (koji je tipa `Word`) će biti kreiran novi objekat tipa `Word` na gore opisani način. Konkretnije, koristeći gore navedene oznake, indeks je reč w, a novokreirani objekat (povratna vrednost operatora indeksiranja) je $r_1 r_2 \ldots r_8$.

4.4 Povezivanje mehanizma sa postojećim implementacijama

Kada je opisani mehanizam razvijen, njegovo povezivanje sa postojećim implementacijama heš funkcija i kriptografskih algoritama na programskom jeziku C/C++ je jednostavno. Sve što je potrebno je u implementaciji konkretnog kriptografskog algoritma zameniti svaku pojavu objekta koji je tipa neoznačenog celog broja objektom koji je tipa `Word`. Ovde treba biti oprezan da se pri zameni ne menjaju tipovi objekata koji ne učestvuju direktno u izračunavanju. Primeri takvih objekata su indeksi, brojači, konstante itd. Ako se greškom takva zamena ipak napravi, to ne bi trebalo bitno da utiče na proces generisanja formule već bi samo usporilo proces. Na slici 4.9 je prikazan deo kôda koji implementira heš funkciju MD5. U implementacijama u kojima se pojavljuju

```
Word F(Word x, Word y, Word z) {
        return (x & y) | (~x & z);
}

void FF(Word& a, Word b, Word c, Word d, Word x, unsigned int s, unsigned int ac) {
        a += F(b, c, d);
        a += x;
        a += ac;
        a = rotate_left(a, s);
        a += b;
}
```

Slika 4.9: Primena tipa klase `Word` na deo kôda heš funkcije MD5.

tabele koje se, posle zamene, indeksiraju indeksima koji su tipa `Word`, potrebno je, za svaku tabelu, kreirati objekat tipa `WordArray` (pri kreiranju objekta, konstruktoru proslediti tabelu kao prvi argument) i zameniti indeksiranja tabele, indeksiranjem novokreiranog objekta. Na slici 4.10 je prikazan deo kôda, koji implementira kriptografski algoritam CMEA i koji opisuje ove izmene (na slici 4.10, promenljiva `_Table` je niz 256 vrednosti iz tabele 2.1).

```
WordArray Table(_Table, 256);

Word C(Word index) {
        return Table[index];
}

Word T(Word x) {
        return C(((C(((C(((C((x ^ K[0]) + K[1]) + x) ^ K[2]) + K[3]) + x) ^
                K[4]) + K[5]) + x) ^ K[6]) + K[7]) + x;
}
```

Slika 4.10: Primena tipa klase `WordArray` na deo kôda kriptografskog algoritma CMEA.

Na kraju je potrebno napisati program koji koristi ceo opisani mehanizam. Opšti oblik takvog programa je prikazan na slici 4.11. Nakon poslednje linije kôda koji je prikazan na ovoj slici, promenljiva `ciphertext` sadrži niz formula čijom konjunkcijom se dobija formula čiju zadovoljivost želimo da ispitamo (fo-

```
main() {
        Word plaintext,  /* Otvoreni tekst. */
             ciphertext, /* Sifrat. */
             key;        /* Kljuc. */

        ...

        /* Inicijlizujemo kljuc (kao niz iskaznih promenljivih). */
        key.init();

        /* Ucitavamo otvoreni tekst i smestamo u promenljivu plaintext. */
        ...
        /* Ucitavamo sifrat i smestamo u pomocnu promenljivu. */
        ...

        /* Pozivamo funkciju sifrovanja. */
        encode(plaintext, key, &ciphertext);

        /*
         * Izjednacavamo dobijene formule sa ucitanom vrednoscu sifrata
         * tako sto negiramo odgovarajuce formule (one koje odgovaraju
         * 0-bitovima u ucitanom sifratu).
         */
        ciphertext.negCorrespondingTo(...);

        ...
}
```

Slika 4.11: Opšti oblik programa koji koristi opisani mehanizam.

rmula $\mathcal{H}$ iz potpoglavlja 3.1.1 i potpoglavlja 3.1.2). Kao što je već rečeno, dobijena formula će biti rešavana SAT rešavačem. Međutim, SAT rešavači rade sa formulama koje su u KNF. Stoga je potrebno dobijenu formulu transformisati u ovaj oblik i o tome govori sledeće poglavlje.

4.5 Konstrukcija KNF i zapis u DIMACS format

U radu [JJ05] je za konstruisanje formule u KNF koja je ekvizadovoljiva sa polaznom formulom iskorišćena Cajcin definiciona normalna forma. Ovde je napomenuto da je konstruisanje KNF date formule, na način koji je dat u [Jan04] moguće samo za male formule i da je neupotrebljiv za formule koje se dobijaju programima opisanim u prethodnom poglavlju. Zbog toga je, i pored njenih nedostataka, Cajcin definiciona normalna forma upotrebljena i u ovom radu. S obzirom da je pri konstrukciji Cajcin definicione normalne forme formula čuvana u obliku koji je u skladu sa formatom datoteke u koju će ona biti upisana, najpre će biti dat opis formata datoteke a potom i objašnjenje kako je ova transformacija urađena.

Jedan od formata koje prepoznaje većina SAT rešavača je DIMACS format. Sintaksa DIMACS formata je vrlo jednostavna. U DIMACS formatu postoje samo jednolinijski komentari i oni počinju slovom `c`. Nakon, eventualno, vodećih komentara, prva linija datoteke treba da bude sledećeg oblika:

```
p cnf n m
```

gde `n` označava broj iskaznih promenljivih, a `m` broj klauza. Svaka sledeća linija predstavlja jednu klauzu. Iskazna slova se identifikuju celim brojevima iz intervala $[1, n]$, ispred kojih, eventualno, stoji predznak za negaciju $(-)$. Pozitivan

broj označava promenljivu koja se identifikuje tim brojem, a negativan broj označava negaciju promenljive čiji je identifikator apsolutna vrednost tog broja. Klauza se predstavlja kao niz celih brojeva različitih od nule koji se završava nulom. Dakle, nula označava kraj tekuće klauze.

Primer 1 *Opis iskazne formule* $(p_2 \vee \neg p_1 \vee p_3) \wedge (\neg p_1 \vee p_2)$ *u DIMACS formatu može biti sledećeg oblika:*

```
c Primer zapisa formule u DIMACS formatu.
p cnf 3 2
2 -1 3 0
-1 2 0
```

Nakon poslednje linije kôda koji je prikazan na slici 4.11, promenljiva `ciphertext` sadrži niz formula čija konjunkcija predstavlja formulu čiju Cajcin definicionu normalnu formu treba konstruisati. Međutim, umesto toga će biti konstruisane Cajcin definicione normalne forme svih formula koje su sadržane u promenljivoj `ciphertext`, a potom će biti konstruisana njihova konjunkcija. Na primer, ako promenljiva `ciphertext` sadrži niz formula $\Phi_1, \Phi_2, \ldots, \Phi_n$, tada će konjunkcija Cajcin definicionih normalnih formi biti (videti potpoglavlje 2.4.4):

$$\bigwedge_{k=1}^{n} \Big(p_{\Phi_k} \wedge \bigwedge_{\substack{\phi \in Sub(\Phi_k) \\ \phi = p_{\phi_1} \otimes p_{\phi_2}}} (p_\phi \Leftrightarrow (p_{\phi_1} \otimes p_{\phi_2})) \wedge \bigwedge_{\substack{\phi \in Sub(\Phi_k) \\ \phi = \neg \phi_1}} (p_\phi \Leftrightarrow \neg p_{\phi_1}) \Big) \tag{4.1}$$

Međutim, ovde će, u opštem slučaju, doći do ponavljanja nekih konjunkata (na primer, ako neke dve formule Φ_i i Φ_j imaju istu podformulu). Ovo može biti izbegnuto ako se konstruiše skup podformula svih formula Φ_k $(k = 1, 2, \ldots, n)$. Označimo taj skup sa S. Tada važi:

$$S = \bigcup_{k=1}^{n} Sub(\Phi_k).$$

Sada se, umesto formule (4.1), može razmatrati formula (4.2).

$$\bigwedge_{k=1}^{n} p_{\Phi_k} \wedge \bigwedge_{\substack{\phi \in S \\ \phi = p_{\phi_1} \otimes p_{\phi_2}}} (p_\phi \Leftrightarrow (p_{\phi_1} \otimes p_{\phi_2})) \wedge \bigwedge_{\substack{\phi \in S \\ \phi = \neg \phi_1}} (p_\phi \Leftrightarrow \neg p_{\phi_1}) \tag{4.2}$$

Da bi ova formula bila konstruisana, neophodno je još odrediti skup S. Međutim, ovaj skup je već poznat jer se svi njegovi elementi nalaze sačuvani u okviru klase `FormulaFactory` (pogledati 4.3).

S obzirom na strukturu DIMACS formata, elementi Cajcin definicione normalne forme će biti čuvani u vektoru

```
vector<vector<int> > conj;
```

koji predstavlja niz konjunkta (dakle, svaki elemenat vektora je takođe vektor). Kako je već rečeno u potpoglavlju 2.4.4, za svaku podformulu je potrebno uvesti novu promenljivu. Međutim, s obzirom na specifikaciju DIMACS formata, nije potrebno eksplicitno uvoditi nove promenljive već samo pozvati funkciju koja će svim podformulama (a to su svi elementi u heš tabeli) dodeliti jedinstveni

identifikator (zbog toga klasa `Formula` ima privatni član `id`), s tim da će formulama koje su tipa `FormulaVar` (koje predstavljaju nepoznate bitove ključa ili otvorenog teksta) biti dodeljeni najmanje vrednosti $(1, 2, \ldots)$. Ova funkcija ima mogućnost da pri dodeli identifikatora formulama, sačuva informaciju o broju promenljivih (a to je najveći dodeljeni identifikator). Sada se jasno vidi razlog zašto je bilo potrebno redukcijom formula ukloniti sve nepotrebne konstante. Da to nije urađeno, i za svaku konstantu bi bilo potrebno uvesti novu promenljivu.

Vektor `conj` se popunjava prateći formulu 4.2 na sledeći način:

- Prvi element konjunkcije se obrađuje prolaskom kroz niz formula koje sadrži promenljiva `ciphertext`, tako što se za svaku formulu kreira vektor koji sadrži samo jedan element, a to je identifikator te formule.

- Drugi i treći element konjunkcije se obrađuje prolaskom kroz heš tabelu, tako što se za svaku formulu kreiraju 2, 3 ili 4 vektora (u zavisnosti od tipa formule) i popune se identifikatorima podformula te formule prateći tabelu 2.2. Na primer, ako je u pitanju formula tipa `FormulaNot`, označimo je sa `f` i ako je njena podformula `f1` (tj. ako je `f` $= \neg$ `f1`), tada će biti kreirana dva vektora: [`f->GetId()`, `f1->GetId()`] i [`-f->GetId()`, `-f1->GetId()`] (`GetId()` je javni metod klase `Formula` koji vraća identifikator konkretne formule).

Nakon što je vektor `ciphertext` kreiran, potrebno je formulu zapisati u datoteku. Prvo se zapisuje niska `p cnf` za kojom sledi broj promenljivih (ovaj podatak je dobijen nakon završene dodele identifikatora formulama) i broj klauza (to je upravo dužina vektora `conj`). Nakon toga, u petlji se ispisuje jedan po jedan element vektora `conj`. Svaki element ovog vektora je vektor celih brojeva pa se ispisuju elementi tog vektora. Na kraju se upisuje simbol 0 jer DIMACS format zahteva da se tako svaka linija završava.

4.6 Korišćenje SAT rešavača

Zadovoljivost formule zapisane u datoteci koja je u DIMACS formatu, može se ispitati većinom današnjih SAT rešavača. Uglavnom svim SAT rešavačima se datoteka u DIMACS formatu može proslediti kao argument komandne linije.

U toku rada na ovoj tezi, uočeno je da je odabir SAT rešavača koji će biti korišćen za rešavanje dobijenih formula, od velikog značaja. Posle kraćeg testiranja na dobijenim formulama, najbolje se pokazao rešavač `Picosat` [Pico], pa je u toku rada on dalje korišćen. Ako je formula zapisana u datoteci `formula.cnf`, poziv `Picosat`-a bi mogao da izgleda ovako (`$` je odzivni znak):

```
$ picosat formula.cnf
```

Nakon završetka rada, program ispisuje poruku o tome da li je formula zadovoljiva ili nije. U slučaju da je zadovoljiva, ispisuje zadovoljavajuću valuaciju. Ispis valuacije može biti različit kod različitih rešavača. Ako je `Picosat` pozvan na prethodni način, u slučaju da nađe zadovoljavajuću valuaciju za formulu, ispisaće niz onoliko celih brojeva koliko polazna formula ima promenljivih. Ukoliko je konkretan broj negativan, to znači da odgovarajuća promenljiva ima vrednost 0 (netačno). U suprotnom ima vrednost 1 (tačno). Na primer, ako je izlaz `Picosat`-a sledećeg oblika:

```
-1 -2 3 -4 5 6 -7 8
```

tada je polazna formula zadovoljiva u valuaciji v koja je predstavljena tabelom 4.1 (sa p_i su ovde označene promenljive koje se pojavljuju u formuli).

p	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
$v(p)$	0	0	1	0	1	1	0	1

Tabela 4.1: Tabela vrednosti zadovoljavajuće valuacije v.

Glava 5

Eksperimentalni rezultati

U ovoj glavi će biti predstavljeni rezultati koji su dobijeni u toku testiranja mehanizma koji je opisan u prethodnoj glavi, kao i zapažanja do kojih se došlo. Kompletan mehanizam je razvijan i delom testiran pod operativnim sistemom Linux na PC (1.8GHz, 512MB RAM). Međutim, sva testiranja čiji rezultati su prikazani u ovoj glavi su obavljena na klasteru IBM Cluster 1350, na Matematičkom institutu u Beogradu. Na oba računara, za rešavanje konstruisanih instanci problema SAT je korišćen rešavač `Picosat` verzija 632.

U prvom poglavlju, *Korenovanje broja*, je, na primeru korenovanja, upoređen uniformni pristup sa grubom silom. U drugom poglavlju su opisani rezultati primene razvijenog mehanizma na heš funkciju MD5. Takođe, dobijeni rezultati su upoređeni sa rezultatima prikazanim u radu [JJ05]. U trećem poglavlju su predstavljeni rezultati dobijeni pri pokušaju razbijanja algoritma CMEA. Poslednje poglavlje sadrži opis dva različita napada na algoritam DES, kao i dobijene rezultate. Jedan od tih napada je identičan napadu primenjenom u [MM00], pa su upoređeni dobijeni rezultati sa rezultatima predstavljenim u ovom radu.

Pre primene na heš funkciju MD5 i kriptografske algoritme CMEA i DES, uniformni pristup je prvo primenjen na jednu trivijalnu šifru: šifrat se od otvorenog teksta dobija cikličnim pomeranjem za određen broj mesta u neku od dve strane. Ova šifra je poslužila u ranoj fazi razvoja mehanizma za njegovo testiranje. Stoga ovde nisu dati nikakvi rezultati vezani za primenu uniformnog pristupa u razbijanju ove šifre, ali je u dodatku detaljnije prikazan rad kompletnog mehanizma u toku njenog razbijanja.

5.1 Korenovanje broja

Zbog NP-kompletnosti problema SAT, postavlja se pitanje: *Da li će pristup opisan u poglavlju 3.3 dati bolje rezultate od pristupa zasnovanog na gruboj sili*[1] *?* Pre svega zbog jednostavnosti, poređenje je najpre urađeno na jednom jednostavnom primeru: *Naći prirodan broj čiji je kvadrat dat.*

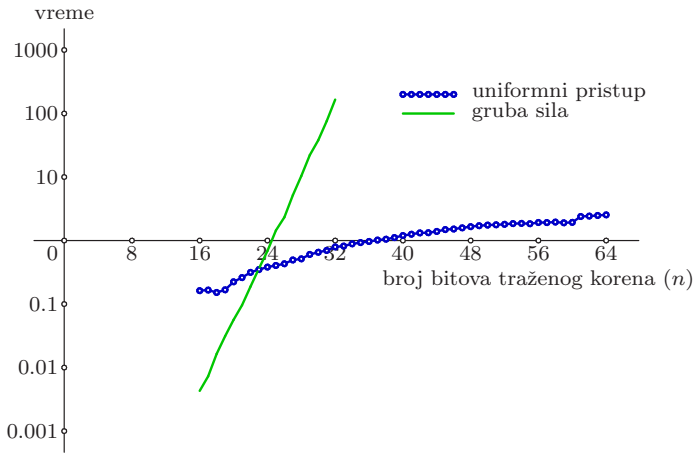
Grubom silom je ovaj problem rešavan linearnom pretragom prostora mogućih korena datog broja. Prostor pretrage je određen brojem čiji je koren potrebno

[1] Gruba sila (eng. *Brute force*) je metod rešavanja problema, za čije rešenje važi da pripada nekom poznatom skupu vrednosti, koji se sastoji od, u najgorem slučaju, sekvencijalnog prolaska kroz skup vrednosti u cilju nalaženja traženog rešenja.

naći. Tačnije, prostor pretrage je $[0, 2^{\frac{n}{2}} - 1]$ gde je n minimalni broj bitova potreban za zapis datog broja u binarnom obliku. Na primer, ako je dat broj 49 (110001 binarno, 6 bitova), njegov koren je 7 (111 binarno, $\frac{6}{2} = 3$ bita).

Za brojeve iz intervala $[0, 2^{32}-1]$, gruba sila je nalazila koren brže od pristupa zasnovanog na uniformnom pristupu. Ovo je posledica malog prostora pretrage (potrebno je svaki element intervala $[0, 2^{16} - 1]$ kvadrirati i uporediti sa datim brojem — brojem čiji se koren traži) i mogućnosti da se sve potrebne aritmetičke operacije realizuju bez upotrebe pomoćnih biblioteka. Međutim, interesantnije je kakav je odnos ova dva pristupa za nenegativne cele brojeve koji ne pripadaju ovom intervalu. Kako je bez upotrebe dodatnih alata bilo nemoguće (na PC-u na kojem je razvijan mehanizam) raditi sa brojevima koji izlaze iz ovog opsega, u tu svrhu je upotrebljena, veoma efikasna, biblioteka GMP (eng. *GNU Multiple Precision*) [GMP].

Za potrebe testiranja, za svako $n \in [16, 32]$ je generisano 100 slučajnih brojeva iz intervala $[0, 2^n - 1]$ (n je broj bitova koji se koriste za zapis datog broja u binarnom obliku) i izračunati su njihovi kvadrati (koji se kodiraju sa $2n$ bitova). Prosečna vremena potrebna za izračunavanje korena datog broja, primenom grube sile i razvijenog uniformnog mehanizma, prikazana su na slici 5.1.



Slika 5.1: Poređenje grube sile i uniformnog pristupa na problemu korenovanja. Vreme je skalirano logaritamski i dato je u sekundama.

Kao što je i očekivano, svaki put kada se broj bitova (kojima se kodira koren datog broja) poveća za jedan, prosečno vreme koje je potrebno da se grubom silom nađe koren datog broja se poveća dvostruko (zbog toga što se prostor pretrage poveća dvostruko). Zbog toga je testiranje grube sile urađeno samo za interval $[16, 32]$. S druge strane, pristup zasnovan na uniformnom pristupu je zahtevao mnogo manje vremena, pa su testiranja urađena za interval $[16, 64]$. Na osnovu ovoga bi se moglo naslutiti da je uniformni pristup efikasniji u odnosu na grubu silu, ali nije lako formalnim dokazom potvrditi to tvrđenje.

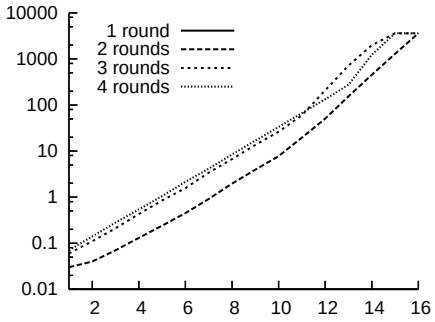
Problem korenovanja, u formi u kojoj je ovde predstavljen, bi se, naravno, mogao rešavati i mnogo efikasnije. S obzirom da se zna prostor pretrage i kako je on sortiran, traženje korena datog broja bi se, umesto linearnom, moglo obaviti binarnom pretragom. Međutim, kako je domen primene binarne pretrage veoma

sužen (ne može se primeniti u pretraživanju prostora svih mogućih poruka ako je data heš vrednost tražene poruke), njeno poređenje sa uniformnim pristupom nema mnogo smisla.

5.2 Heš funkcija MD5

Kako je u okviru ovog rada rekonstruisan mehanizam opisan u radu [JJ05], nakon završetka rekonstrukcije, a pre primene na kriptografske algoritme CMEA i DES, bilo je potrebno ispitati kvalitet konstruisanog mehanizma. Kako je uniformni pristup već primenjen na heš funkciju MD5, a rezultati prikazani u radu [JJ05], kvalitet je proveren razbijanjem upravo te heš funkcije i poređenjem dobijenih rezultata sa rezultatima prikazanim u pomenutom radu.

Primenjen je isti napad kao u [JJ05], tj. za zadatu dužinu polazne poruke i njenu heš vrednost, pokušana je rekonstrukcija polazne poruke. Za svako $n \in [1, 20]$, konstruisano je 100 slučajnih poruka dužine n (u radu [JJ05] $n \in [1, 16]$, a slučajno je birano 50 poruka) i izračunate su odgovarajuće heš vrednosti. Zatim je pretpostavljeno da su bitovi polazne poruke nepoznati, da bi se na kraju pokušalo sa rekonstrukcijom vrednosti tih bitova na osnovu poznate heš vrednosti. Slika 5.2 je preuzeta iz rada [JJ05] i prikazuje rezultate dobijene



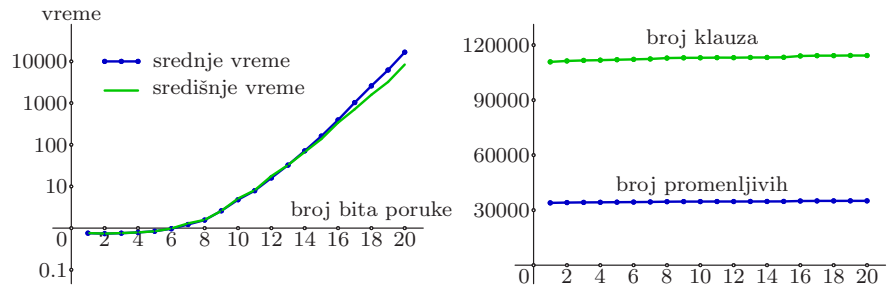
Slika 5.2: Rezultati prikazani u radu [JJ05]. Vremenska osa je logaritamska. Vreme je dato u sekundama.

u toku razbijanja heš funkcije MD5 na opisan način (kao i njenih oslabljenih verzija, dobijenih smanjivanjem broja rundi). Ovi rezultati su dobijeni na Linux 2.60GHz Pentium 4 radnoj stanici, a za rešavanje dobijenih problema SAT je korišćen `zChaff` rešavač.

Grafik zavisnosti vremena potrebnog za razbijanje heš funkcije MD5 od dužine polazne poruke za reimplementirani mehanizam je prikazan na slici 5.3a. Na grafiku su prikazana samo vremena potrebna za rešavanje dobijene iskazne formule. Osim na rešavanje iskazne formule, izvesno vreme se troši i na njenu konstrukciju. Svi podaci dobijeni u toku testiranja su prikazani u tabeli 5.1. Značenje kolona u tabeli je sledeće:

n — dužina ulazne poruke,

T_m — srednje vreme (u sekundama) potrebno za konstrukciju iskazne formule, na osnovu poznate heš vrednosti,



Slika 5.3: (a) Srednje i središnje vreme potrebno za nalaženje bitova polaznih poruka dužine $n \in [1, 20]$ na osnovu poznate heš vrednosti. Vremenska osa je logaritamska, a vreme je dato u sekundama. (b) Broj promenljivih i broj klauza u formulama koje su dobijene za različite dužine polazne poruke.

n	T_m	L	N	T_s	T'_s
1	3.4321	110847	33967	0.757	0.76
2	5.4438	111354	34128	0.7368	0.74
3	4.9334	111649	34225	0.7564	0.76
4	4.9443	111787	34269	0.7887	0.79
5	4.9658	112030	34346	0.8413	0.84
6	4.9752	112220	34406	0.9655	0.98
7	4.9867	112446	34477	1.2228	1.3
8	5.0379	112929	34626	1.5604	1.58
9	5.0477	113067	34670	2.5984	2.6
10	5.0586	113082	34677	4.8035	5.13
11	5.0586	113165	34706	7.9033	8.13
12	5.0686	113142	34700	15.9886	17.74
13	5.0704	113246	34734	32.5143	32.66
14	5.0681	113263	34742	71.3168	67.08
15	5.0798	113342	34769	161.1047	137.29
16	5.112	114069	34989	393.5213	338.91
17	5.1434	114232	35040	1028.931	712.67
18	5.1444	114250	35049	2586.3589	1583.53
19	5.1758	114331	35077	6198.8953	3209.61
20	5.1652	114310	35072	16624.5041	8388.9

Tabela 5.1: Rezultati dobijeni razbijanjem heš funkcije MD5 uniformnim pristupom.

L — broj klauza u iskaznoj formuli,

N — broj promenljivih u iskaznoj formuli,

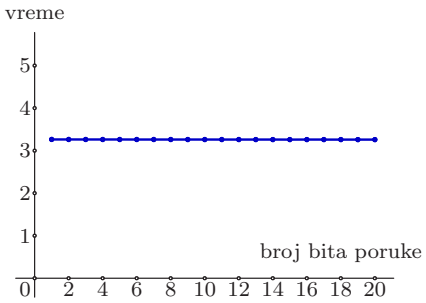
T_s — srednje vreme (u sekundama) potrebno za nalaženje polazne poruke,

T'_s — središnje vreme[2] (u sekundama) potrebno za nalaženje polazne poruke.

Na osnovu rezultata predstavljenih u radu [JJ05] (prikazanih na slici 5.2), polazna poruka dužine 16 bitova je rekonstruisana na osnovu poznate heš vrednosti za nekoliko hiljada sekundi. Rekonstruisani mehanizam je isti problem rešio znatno brže, za oko 400s. Kraće vreme je najverovatnije posledica više faktora, među kojima su brži računari korišćeni za testiranje, efikasniji SAT rešavači i razlike u odnosu na prethodnu implementaciju.

[2]Središnje vreme predstavlja medijatrisu sortiranog niza svih vremena.

Ono što je u radu [JJ05] rečeno za iskazne formule dobijene na osnovu heš funkcije MD5 i poznate heš vrednosti, pokazalo se kao tačno i u ovom radu (slika 5.3b). Naime, nasuprot eksponencijalnog rasta vremena potrebnog za rešavanje dobijene formule (kada dužina ulazne poruke raste), broj promenljivih i klauza u formulama raste kao za neki mali linearni faktor. Shodno tome, približno je konstantan odnos broja klauza i broja promenljivih i iznosi oko 3.26, što može biti relevantno u kontekstu proučavanja fazne promene u problemu SAT i relativne težine generisanih formula. Tačnije, moglo bi se postaviti pitanje da li instance problema SAT, koje se dobijaju na ovaj način, pripadaju klasi teških instanci problema SAT. Međutim, na ovo pitanje se ne može dati odgovor pošto dobijene formule ne zadovoljavaju obrazac nekih klasa slučajno generisanih instanci problema SAT (promenljive u klauzama nemaju uniformnu raspodelu). Kada bi, ipak, promenljive u generisanim formulama imale uniformnu raspodelu i kada bi te formule, zaista, pripadale klasi najtežih instanci problema SAT (među formulama sa istom raspodelom dužina klauza), tada bi tačka fazne promene za dobijeni model problema SAT bila oko 3.26. Eksperimentalno je utvrđeno da dobijene formule sadrže oko 40% klauza dužine 2 i oko 60% klauza dužine 3 (za $M = 1, 2, \ldots 128$)[3]. Za takvu raspodelu dužina klauza, tačka fazne promene se u [GW94] aproksimira sa 1.8, a u [A3K01] vrednošću između 2.1 i 2.4. Kako se vidi, obe ove aproksimacije su manje od vrednosti koja je dobijena u ovoj tezi. Potraga za razlogom ovog nepoklapanja bi mogla biti tema daljeg rada, a jedan od zaključaka bi mogao da bude da se kodiranjem heš funkcije MD5 ne dobijaju najteže instance, tj. da se njenom modifikacijom može dobiti kvalitetnija heš funkcija. S druge strane, možda je nepoklapanje, jednostavno, posledica toga što klauze i promenljive u klauzama nemaju uniformnu raspodelu. Grafik zavisnosti odnosa broja klauza i broja promenljivih (L/N) u odnosu na dužinu ulazne poruke je prikazan na slici 5.4.



Slika 5.4: Grafik zavisnosti odnosa broja klauza i broja promenljivih u odnosu na dužinu ulazne poruke.

Kao što je već rečeno, broj klauza i promenljivih veoma sporo raste, sa rastom dužine polazne poruke. Gotovo da su konstantne vrednosti broja klauza i promenljivih. Zbog velike kompleksnosti kompletnog mehanizma, teško je objasniti taj spori rast. Neformalno objašnjenje bi moglo da bude: kako se pri konstrukciji iskaznih formula koristi ista heš funkcija, a samo se menja dužina polazne poruke i šifrat, skup podformula dobijene formule (koji se koristi u konstrukciji Cajcin definicione normalne forme, pogledati poglavlje 4.5) ostaje

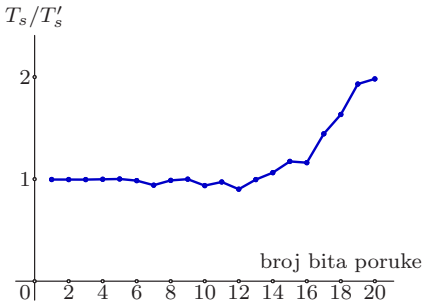
[3] Ovo se poklapa sa rezultatima prikazanim u [JJ05].

u velikoj meri isti; jedine razlike se dobijaju povećanjem ulazne poruke — čime se povećava broj promenljivih, kao i za različite šifrate — čime se u poslednjem koraku različite formule negiraju (pogledati poglavlje 3.1.1), pa se time, takođe, menja skup podformula. Ali sve te razlike su suviše male da bi značajnije uticale na konačan broj klauza i promenljivih.

Osim srednjeg vremena, koje se dobija kao aritmetička sredina postignutih vremena, potrebnog za rešavanje dobijene iskazne formule (koje je prikazano na slici 5.3a), izračunata su i središnja vremena. Odnosi srednjeg i središnjeg vremena, potrebnog za rešavanje formula koje se dobijaju za različite dužine polazne poruke, dati su u tabeli 5.2 i prikazani na slici 5.5. Može se primetiti da

1	2	3	4	5	6	7	8	9	10
0.9961	0.9957	0.9953	0.9984	1.0015	0.9852	0.9406	0.9876	0.9994	0.9364
11	12	13	14	15	16	17	18	19	20
0.9721	0.9013	0.9955	1.0632	1.1735	1.1611	1.4438	1.6333	1.9314	1.9817

Tabela 5.2: Odnosi srednjeg i središnjeg vremena potrebnih za rešavanje formula koje se dobijaju za poruke dužine $n \in [1, 20]$.



Slika 5.5: Grafik odnosa srednjeg i središnjeg vremena potrebnih za rešavanje formula koje se dobijaju za poruke dužine $n \in [1, 20]$.

je od $n = 14$ ovaj odnos veći od 1 i da počinje rast ovog odnosa, što znači da, od pomenute dužine polazne poruke, središnje vreme postaje sve manje u odnosu na srednje vreme rešavanja formula koje se dobijaju za odgovarajuće dužine ulazne poruke. Povećavanje razlike između srednjeg i središnjeg vremena pokazuje da u analizi više nije dovoljno razmatrati samo jedno od ova dva vremena, već oba. Središnje vreme daje vrednost oko koje su *razbacana* dobijena vremena (polovina dobijenih vremena je manja od središnjeg vremena, a polovina je veća). Odnos srednjeg (T_s) i središnjeg (T'_s) vremena govori o raspodeli dobijenih vremena. Tačnije, razlikuju se tri slučaja:

- ako je $T_s < T'_s$, to znači da su neka od dobijenih vremena mnogo manja od vrednosti središnjeg vremena i da su ona, neformalno govoreći, *povukla* za sobom srednje vreme, tj. zbog njih je srednje vreme manje od središnjeg,
- ako je $T_s = T'_s$, to znači da su dobijena vremena uniformno *razbacana* oko vrednosti središnjeg vremena,

- ako je $T_s > T'_s$, to znači da su neka od dobijenih vremena mnogo veća od vrednosti središnjeg vremena i da su ona, slično prvom slučaju, *povukla* za sobom srednje vreme (u ovom slučaju naviše).

5.3 Algoritam CMEA

Kako je u pitanju kriptografski algoritam sa ključem (videti 2.1.1), pri razbijanju algoritma CMEA je primenjen drugačiji napad od onog koji je primenjen na heš funkciju MD5. Pretpostavljeno je da su poznati specifikacija algoritma i jedan par otvorenog teksta i odgovarajućeg šifrata, dok je za odgovarajući ključ pretpostavljeno da je nepoznat. Zatim je pokušana rekonstrukcija vrednosti nepoznatih bitova ključa.

Primećeno je da kod algoritma CMEA kreiranje iskaznih formula traje znatno duže nego kod heš funkcije MD5. Pored toga, primećeno je da ako se ne pretpostavi poznavanje dovoljnog broja parova otvorenih tekstova i odgovarajućih šifrata, u opštem slučaju, može biti pronađeno više različitih ključeva. Odavde sledi da se sa svakim novim parom otvorenog teksta i odgovarajućeg šifrata, vreme potrebno za kreiranje iskazne formule povećava. Zbog toga je bilo neophodno da se broj parova smanji (da bi se mogli obaviti testovi).

Kako su na samom početku bitovi ključa zamenjeni iskaznim promenljavam, uz poznavanje bitova otvorenog teksta, kao izlaz iz kriptografskog algoritma (u konkretnom slučaju algoritma CMEA) dobija se niz iskaznih formula koje zavise samo od promenljivih koje predstavljaju bitove ključa. Na kraju je još potrebno da se te formule izjednače sa poznatim bitovima šifrata. U ovom trenutku je dobijeno n jednačina sa m nepoznatih (gde je n broj bitova šifrata, a m je broj bitova ključa). Sistem jednačina, koji se na opisani način dobija, se razlikuje od sistema jednačina koji se mogu sresti u linearnoj algebri. Stoga se teoreme iz linearne algebre, koje govore o jedinstvenosti rešenja sistema jednačina, ne mogu primeniti na dobijeni sistem. Radi jednostavnosti, pretpostavljeno je da je potrebno da važe uslovi kao kod linearnih sistema. Ti uslovi su da jednačine budu međusobno nezavisne (ovaj uslov nije primenljiv, pa je zanemaren) i da je broj jednačina jednak broju nepoznatih, tj. da je $n = m$. U konkretnom slučaju, kod algoritma CMEA, kako je dužina ključa 64, potrebno je znati 8 parova otvorenih tekstova i odgovarajućih šifrata (jer je svaki otvoreni tekst i šifrat dužine 8 bitova). Ovo je veoma neprecizna ocena, što će se videti i iz rezultata.

Na samom početku testiranja nije se moglo pretpostaviti kakvi su izgledi da se uniformnim pristupom razbije kompletan algoritam CMEA. Zbog toga je pokušano razbijanje, u nekom smislu, oslabljene verzije ovog algoritma. Kako se u algoritmu CMEA koristi 64-bitni ključ (videti 2.1.1), koji se razmatra kao 8 ključeva dužine 8 bitova, oslabljena verzija je dobijena tako što je pretpostavljeno da je izvestan broj ključeva poznat, a da su ostali nepoznati. Tačnije, pretpostavljeno je da je, redom, $1, 2, \ldots, 7, 8$ ključeva nepoznato. U tabeli 5.3 su prikazani rezultati dobijeni u toku testiranja. Značenje kolona u tabeli je sledeće:

n — broj nepoznatih ključeva u oslabljenoj verziji algoritma CMEA,

T_m — srednje vreme (u sekundama) potrebno za konstrukciju iskazne formule, na osnovu poznatih osam parova otvorenih tekstova i odgovarajućih šifrata,

n	T_m	L	N	T_f	T_t	T_a	S_d	M_d	N_{nd}	M_{nd}
1	70.5206	172646	57434	2.59	2.59	2.59	0.00	0	0	0
2	69.9901	173357	57645	27.40	39.29	55.65	1.00	1	2	2
3	69.8361	173567	57686	58.25	86.61	118.64	1.00	1	2	2
4	68.8449	174356	57917	86.54	237.10	365.93	3.00	3	24	4
5	76.264	174790	58033	17363	-	-	-	-	-	-

Tabela 5.3: Rezultati dobijeni pri pokušaju razbijanja oslabljenih verzija algoritma CMEA u kojima je nepoznato $1, 2, 3, 4$ i 5 ključeva.

L — broj klauza u iskaznoj formuli,

N — broj promenljivih u iskaznoj formuli,

T_f — srednje vreme (u sekundama) potrebno za nalaženje prvog ključa,

T_t — srednje vreme (u sekundama) potrebno za nalaženje traženog ključa,

T_a — srednje vreme (u sekundama) potrebno za nalaženje svih ključeva,

S_d — srednji broj duplikata (ključeva),

M_d — maksimalan broj duplikata (ključeva),

N_{nd} — ukupan broj ne-duplikata (ključeva),

M_{nd} — maksimalan broj ne-duplikata (ključeva).

Test čiji su rezultati prikazani u tabeli 5.3 je obavljen tako što je prvo konstruisano 100 slučajnih 64-bitnih ključeva, a za svaki ključ je konstruisano 8 slučajnih otvorenih tekstova i izračunati su odgovarajući šifrati. Potom je za $n \in \{1, 2, 3, 4\}$ pretpostavljeno da je n ključeva nepoznato, a da su ostali poznati. Za svako takvo n i za svaki od 100 ključeva, na osnovu 8 parova otvorenih tekstova i odgovarajućih šifrata je konstruisana iskazna formula. U tabeli su, u koloni označenoj sa T_m, prikazana vremena potrebna za ovu konstrukciju. S obzirom da je iz ranijih testova primećeno da će se pojavljivati duplikati[4], SAT rešavač je pozivan u petlji sve dok ima zadovoljavajućih valuacija (iskazna formula je u svakom prolasku modifikovana na odgovarajući način — dodavana je klauza koja isključuje upravo pronađeno rešenje). Za svaki ključ, koji je dobijen na ovaj način, generisano je 1000 slučajnih otvorenih tekstova i za svaki od tih 1000 otvorenih tekstova je ispitano da li ih originalni (koji je na samom početku slučajno generisan) i novodobijeni (koji je određen zadovoljavajućom valuacijom koja je pronađena pomoću SAT rešavača) ključ šifruju istim šifratom. Kako šifra CMEA spada u grupu *slabijih* šifara, ova provera je urađena zbog mogućnosti pojave *poklapanja* dva različita ključa na 1000 slučajno generisanih otvorenih tekstova i odgovarajućih šifrata. Ako je došlo do poklapanja, takav ključ je proglašen *duplikatom*, a u suprotnom *ne-duplikatom*.

Kolona u tabeli označena sa S_d, predstavlja prosečan broj duplikata (u smislu uvedenom u prethodnom pasusu) ključa. Sledeća kolona, M_d, označava maksimalan broj duplikata koji je u toku eksperimenta odgovarao jednom ključu, dok kolona M_{nd} označava isto to samo za ne-duplikate. Sa N_{nd} je označena

[4] Duplikatom nekog ključa K, zovemo ključ koji se razlikuje od K, a proizvoljni otvoreni tekst šifruje istim šifratom kao i ključ K.

kolona koja sadrži ukupan broj ne-duplikata (za svih 100 ključeva). Odavde se primećuje da je broj ne-duplikata mnogo manji od broja duplikata.

Kada je pretpostavljeno da je nepoznato pet ključeva, a da su tri poznata, kao i da je poznato osam parova otvorenih tekstova i odgovarajućih šifrata, nepoznati deo ključa je pronađen za oko 17363s, tj. manje od 5 sati (ovo je vreme traženja prvog ključa). Ovi rezultati su dobijeni na osnovu samo 10 urađenih test primera. Ovaj skok, u odnosu na slučaj kada je pretpostavljeno da su četiri ključa nepoznata, se može tumačiti povećanjem kvaliteta algoritma CMEA, kada se povećava dužina ključa[5]. Za kompletan algoritam CMEA, tj. kada je pretpostavljeno da je nepoznat ceo ključ, generisanu iskaznu formulu SAT rešavač nije rešio za nedelju dana.

Kao što je ranije rečeno, na početku rada na ovoj tezi je postojala želja da se pre pokušaja razbijanja algoritma DES uniformnim pristupom razbije neki jednostavniji kriptografski algoritam, kao što je algoritam CMEA. Međutim, kako su rezultati bili neočekivano loši (jer je pretpostavljeno je da je algoritam CMEA dovoljno jednostavan da se može razbiti uniformnim pristupom), na algoritam CMEA je, takođe, primenjena i gruba sila, da bi se uporedili rezultati. Rezultati dobijeni primenom grube sile na oslabljene verzije algoritma CMEA su prikazani na slici 5.4. Značenje kolona u ovoj tabeli je isto kao i u tabeli 5.3. Što se tiče

n	T_f	T_t	T_a
1	0	0	0
2	0.0026	0.0057	0.0087
3	0.7351	2.2338	2.2338
4	188.4787	572.9641	572.9641

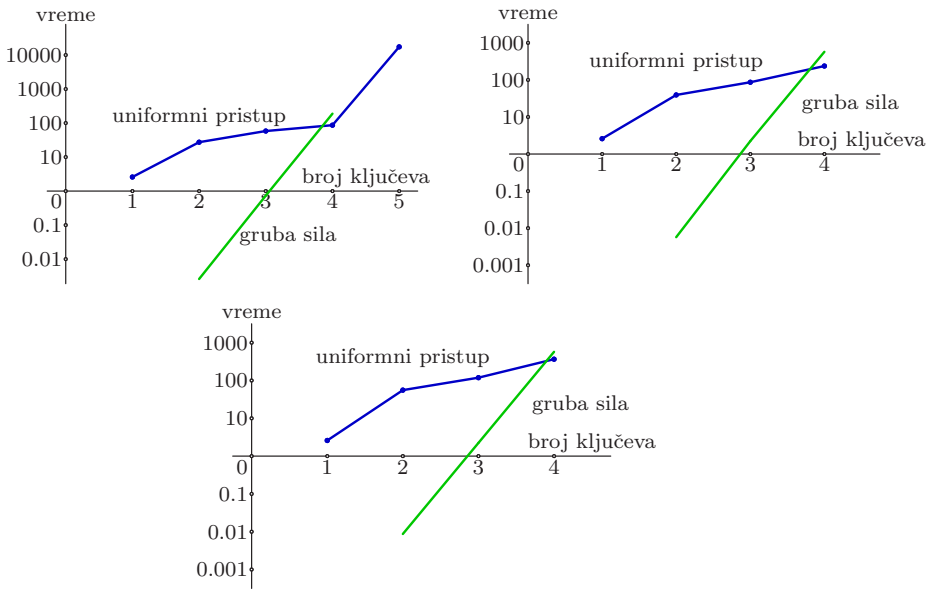
Tabela 5.4: Rezultati dobijeni pri pokušaju razbijanja grubom silom oslabljenih verzija algoritma CMEA u kojima je nepoznato $1, 2, 3$ i 4 ključa.

broja duplikata i ne-duplikata, njihov broj je, naravno, isti kao i kod uniformnog pristupa. Na slici 5.6 su prikazani grafici zavisnosti vremena potrebnog za razbijanje oslabljene verzije algoritma CMEA grubom silom i uniformnim pristupom u zavisnosti od broja nepoznatih ključeva. Jedino u slučaju kada se tražio prvi ključ (videti sliku 5.6(a)), prikazano je i vreme potrebno za rešavanje iskazne formule dobijene na osnovu oslabljene verzije algoritma CMEA kod koje je pretpostavljeno da je nepoznato 5 ključeva. Treba napomenuti da su ti rezultati dobijeni na osnovu nepotpunih testova (kada je pretpostavljeno da su nepoznata 1, 2, 3 ili 4 ključa, eksperiment je rađen 100 puta, dok je za 5 ključeva rađen samo 10 puta).

Kao što je i očekivano (zbog malog prostora pretrage), grubom silom su postignuti mnogo bolji rezultati nego uniformnim pristupom za oslabljene verzije algoritma CMEA u kojima je nepoznato $1, 2$ i 3 ključa. Međutim, za oslabljenu verziju algoritma CMEA u kojoj su nepoznata 4 ključa bolje rezultate je postigao uniformni pristup.

Na osnovu navedenog, može se zaključiti da je za mali broj nepoznatih ključeva mali i prostor pretrage koji se pretražuje kod grube sile, pa je ona

[5]U ovom slučaju se kvalitet algoritma smanjuje tako što se smanjuje, uslovno rečeno, dužina ključa. U samom algoritmu CMEA dužina ključa ostaje 64 bita, ali se u novouvedenom kontekstu, ključem smatra deo ključa koji je nepoznat, dok se poznati deo razmatra kao neka konstanta koja figuriše u algoritmu.



Slika 5.6: Grafici zavisnosti vremena potrebnog za razbijanje oslabljene verzije algoritma CMEA grubom silom i unifomnim pristupom u zavisnosti od broja nepoznatih bitova ključa: (a) vreme potrebno za nalaženje prvog ključa, (b) vreme potrebno za nalaženje traženog ključa, (c) vreme potrebno za nalaženje svih ključeva. Vremenska osa je logaritamska, a vreme je dato u sekundama.

mnogo efikasnija od uniformnog pristupa. S druge strane, kada se prostor pretrage poveća (za 4 nepoznata ključa prostor pretrage je $[0, 2^{32}-1]$) u konkretnom slučaju, za 4 nepoznata ključa, uniformni pristup je postigao bolje rezultate. Međutim, bez detaljnijih testova ne može se zaključiti da to uvek važi.

5.4 Algoritam DES

Kako je algoritam DES kriptografski algoritam sa ključem, pretpostavljeno je, kao i kod algoritma CMEA, da je poznata specifikacija algoritma kao i jedan par otvorenog teksta i odgovarajućeg šifrata. Za ključ je pretpostavljeno da je nepoznat, a potom je pokušana rekonstrukcija vrednosti bitova ključa. Kako je na samom početku i očekivano, a kasnije eksperimentalno pokazano, kompletan DES je bilo nemoguće razbiti uniformnim pristupom. Zbog toga je pokušano razbijanje, u nekom smislu, oslabljenih verzija algoritma DES. Algoritam DES je oslabljen na dva različita načina. Kako se algoritam DES sastoji od 16 rundi (videti 2.1.2), prvi način se sastojao u variranju broja rundi. Ovako oslabljen algoritam DES je razbijan i u okviru rada [MM00], pa je bilo moguće uporediti rezultate dobijene korišćenjem uniformnog pristupa sa rezultatima datim u pomenutom radu. Drugi način se sastojao u varijaciji broja nepoznatih bitova ključa.

Variranje broja rundi. Svi kvalitetniji kriptografski algoritmi, pa tako i algoritam DES, teže ka tome da se umanji mogućnost da se proizvoljan otvoreni

tekst sa dva različita ključa šifruje istim šifratom. Međutim, do ovoga može vrlo često da dođe u slučaju oslabljene verzije DES algoritma, dobijene smanjivanjem broja rundi, jer se na taj način umanjuje kvalitet kriptografskog algoritma. Na samom početku je pokušano sa razbijanjem oslabljene verzije DES algoritma koja se sastoji samo od prve runde, a pretpostavljeno je da je poznat samo jedan par otvorenog teksta i odgovarajućeg šifrata. Najpre je, na način opisan u glavi 4, napisan program koji na osnovu zadate specifikacije DES algoritma i zadatog para (otvoreni tekst, šifrat — oba dužine 64 bitova) konstruiše iskaznu formulu, a potom poziva SAT rešavač da reši dobijenu formulu. Zatim je taj program izmenjen tako da se SAT rešavač poziva u petlji i tako da se u svakom prolasku kroz petlju pronalazi nov ključ (koji zadovoljava zadati uslov), a iskazna formula se ažurira na odgovarajući način. Nakon pokretanja, program se nije zaustavio nakon nekoliko minuta. Drugim rečima, od mogućih 2^{56} ključeva, veliki broj ključeva slika proizvoljni otvoreni tekst u odgovarajući šifrat. Potom je pokušano sa povećavanjem broja poznatih parova otvorenih tekstova i odgovarajućih šifrata (2, 4 i 8 parova). Za svaki pojedinačni par je kreirana iskazna formula a potom je napravljena njihova konjunkcija. Ishod je bio isti kao i u slučaju kada je poznat bio samo jedan par. Zbog toga su u nastavku dati rezultati dobijeni za oslabljenu verziju DES algoritma koja se sastoji od 2, 3 i 4 runde. Za oslabljenu verziju algoritma DES koja se sastoji od prvih 5 rundi su konstruisane dve iskazne formule na osnovu dva para slučajno odabranih otvorenih tekstova i odgovarajućih šifrata. SAT rešavač je jednu od njih rešio za 9683 minuta, odnosno nešto manje od nedelju dana. Drugu formulu SAT rešavač je rešio za oko dve nedelje. Zbog ovakve zahtevnosti nisu vršeni detaljniji eksperimenti za verziju algoritma koja se sastoji od pet rundi.

Za bilo koju od oslabljenih verzija, testovi su obavljeni na isti način. Konstruisano je 100 slučajnih 56-bitnih ključeva. Za svaki ključ su konstruisana četiri različita slučajna 64-bitna otvorena teksta i izračunati su odgovarajući šifrati na osnovu tekuće oslabljene verzije. Potom je za svaku oslabljenu verziju (2, 3 i 4 runde), za 1, 2 i 4 para otvorenih tekstova i odgovarajućih šifrata, rekonstruisan ključ, uz traženje duplikata. U tabeli 5.5 su dati rezultati dobijeni u toku razbijanja oslabljenih verzija algoritma DES dobijenih variranjem broja rundi. Treba napomenuti da je srednje vreme, koje je potrebno za rešavanje

R	P	T_m	L	N	T_s	ND
2	1	0.3561	15847	5289	0.2446	2.10
	2	0.9185	28234	9342	0.3014	0.13
	4	2.9500	51785	17035	0.4188	0.00
3	1	0.6222	22085	7357	1.0635	0.03
	2	1.7405	40719	13481	2.1688	0.00
	4	6.0098	76756	25316	35.2298	0.00
4	1	0.9454	28330	9428	773.7066	0.00
	2	2.7807	53212	17625	1154.7448	0.00
	4	9.9006	101746	33603	1076.2673	0.00
5	1	1.285	34570	11498	$\approx$ 876000	-

Tabela 5.5: Rezultati dobijeni pri pokušaju razbijanja oslabljenih verzija algoritma DES koje se sastoje od 2, 3, 4 i 5 rundi.

oslabljene verzije algoritma DES, koju čini prvih pet rundi, dobijeno eksperimentom koji je urađen samo dva puta, dok je u svim ostalim slučajevima eksperiment rađen 100 puta. Za razliku od algoritma CMEA, kod algoritma

DES ne postoje duplikati, ali ima ne-duplikata. Međutim, kako se broj poznatih parova otvorenih tekstova i odgovarajućih šifrata povećava, broj ne-duplikata se smanjuje (kolona *ND* u tabeli 5.5). Ovo je posledica kvaliteta algoritma DES. Značenje kolona u tabeli je sledeće:

R — broj rundi koje čine oslabljenu verziju DES algoritma,

P — broj parova otvorenih tekstova i odgovarajućih šifrata za koje je pretpostavljeno da su poznati,

T_m — srednje vreme (u sekundama) potrebno za konstrukciju iskazne formule,

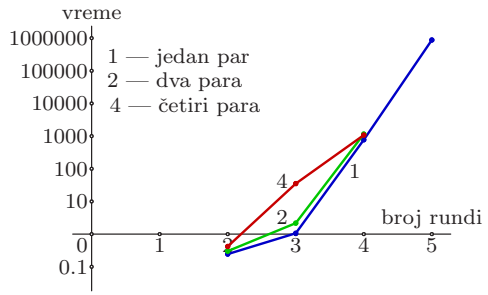
L — broj klauza u iskaznoj formuli,

N — broj promenljivih u iskaznoj formuli,

T_s — srednje vreme (u sekundama) potrebno SAT rešavaču za rešavanje iskazne formule,

ND — srednji broj ne-duplikata (ključeva).

Na slici 5.7 su prikazani grafici zavisnosti vremena potrebnih za razbijanje oslabljenih verzija algoritma DES. Prikazana su tri grafika od kojih svaki odgo-



Slika 5.7: Grafici zavisnosti vremena potrebnih za razbijanje oslabljenih verzija algoritma DES u zavisnosti od broja rundi. Vremenska osa je logaritamska, a vreme je dato u sekundama.

vara određenom broju parova otvorenih tekstova i odgovarajućih šifrata za koje je pretpostavljeno da su poznati.

Kao prvo, treba konstatovati da je, u odnosu na rezultate prikazane u radu [MM00], napravljen napredak. U pomenutom radu su dati rezultati dobijeni u toku razbijanja oslabljene verzije DES algoritma koja se sastoji od 1, 2 i 3 runde. Kao što se vidi iz tabele 5.5, uniformnim pristupom je uspešno razbijeno prvih pet rundi DES algoritma. Međutim, pogrešno bi bilo ceo uspeh pripisati uniformnom pristupu. Velikim delom su za ovaj uspeh zaslužni i mnogo brži računari (u radu [MM00] su dati rezultati dobijeni korišćenjem SUN UltraSparcs i Pentium II 300 MHz, oba sa 64 MB RAM) koji su korišćeni u toku testiranja, kao i trenutno najbrži SAT rešavači.

Kao drugo, iz tabele 5.5 se može uočiti kako kvalitet (u smislu koji je uveden u ovom poglavlju) oslabljenog algoritma DES raste (smanjuje se broj duplikata), kako mu se dodaju runde. Takođe, treba napomenuti da je kod oslabljenog algoritma DES koji se sastoji od prve dve runde, a kada se pretpostavi da je poznat

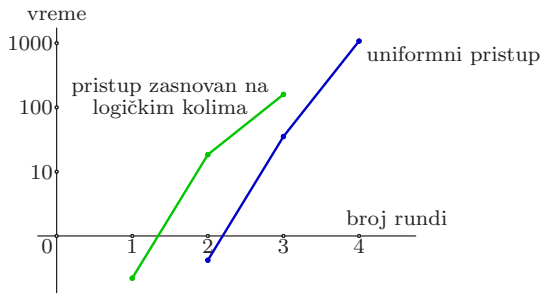
samo jedan par otvorenog teksta i odgovarajućeg šifrata, ranije opisani program u pojedinim slučajevima nalazio i do 24 duplikata ključeva.

Poređenje sa rezultatima iz rada [MM00]. Rezultati prikazani u tabeli 5.6 su preuzeti iz rada [MM00] i predstavljaju najbolja[6] vremena dobijena razbijanjem oslabljenih verzija algoritma DES.

R	P	T_s
1	1	0.02
	2	0.11
	4	0.22
	8	0.45
2	1	32.43
	2	111.15
	4	18.39
	8	0.81
3	1	$\geq$ 1h
	2	976.28
	4	159.13
	8	75.02

Tabela 5.6: Rezultati preuzeti iz rada [MM00]. Kolona R označava broj rundi, kolona P broj poznatih parova otvorenih tekstova i odgovarajućih šifrata, a kolona T_s vreme potrebno SAT rešavaču da reši iskaznu formulu.

Kao što se vidi, u tabeli su, pored ostalog, dati i rezultati dobijeni razbijanjem samo jedne runde algoritma DES, što nije urađeno u ovoj tezi (detaljno je obrazložen razlog). U radu [MM00] su ovi problemi prevaziđeni tako što je tražena prva zadovoljavajuća valuacija (bez obzira da li ona odgovara traženom ključu ili ne). Kako u ovom radu nije pretpostavljano da je poznato više od 4 para otvorenih tekstova i odgovarajućih šifrata, na slici 5.8 su prikazani grafici zavisnosti vremena prikazanih u tabeli 5.6 i tabeli 5.5.



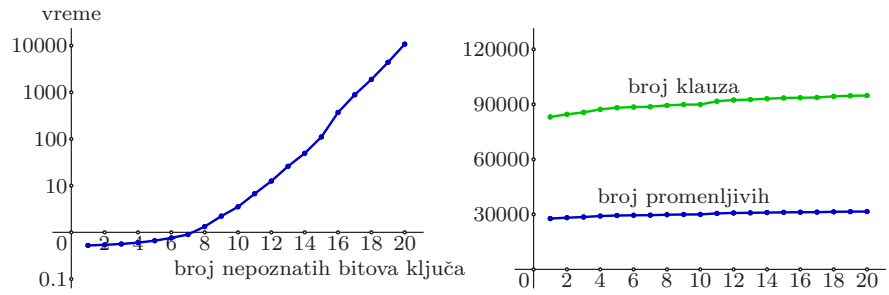
Slika 5.8: Poređenje uniformnog pistupa i pristupa zasnovanog na logičkim kolima na razbijanju oslabljenih verzija algoritma DES koje se dobijaju variranjem broja rundi. Vremenska osa je logaritamska, a vreme je dato u sekundama.

Poređenje sa grubom silom. U slučaju variranja broja rundi, dužina ključa ostaje 56 bitova, pa bi u ovom slučaju prostor pretrage bio dovoljno veliki ($[0, 2^{56} - 1]$)

[6] Za rešavanje iskaznih formula su u ovom radu korišćena tri različita SAT rešavača. Ovde su prikazana najbolja vremena koja su postignuta jednim od tih rešavača.

da primena grube sile izgubi smisao (bez obzira na broj rundi). S druge strane, za uniformni pristup 56 nepoznatih bitova ključa nije predstavljao veliki problem. U prilog ovome govore rezultati prikazani u tabeli 5.5. Ovde se vidi da su prve tri runde algoritma DES razbijene za relativno kratko vreme. Problem je nastao povećavanjem broja rundi što je rezultovalo kompleksnijim iskaznim formulama. Kompleksnije formule rezultuju duže vreme rešavanja što se, takođe, može videti iz tabele 5.5. Naime, za razbijanje prve četiri runde algoritma DES je potrebno oko 1000s, dok je za prvih pet rundi potrebno više od nedelju dana (ovo je rezultat dobijen samo na osnovu dva testa; za opštiji rezultat bi trebalo obaviti obimnija testiranja).

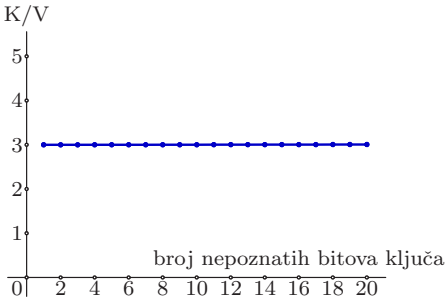
Variranje broja nepoznatih bitova ključa. Drugi način oslabljivanja algoritma DES je smanjivanje dužine ključa. Kako algoritam DES zahteva da ključ bude dužine 56, njegovo smanjivanje je urađeno tako što je pretpostavljeno da su neki bitovi ključa već poznati. Za potrebe testiranja, prvo je generisano 100 slučajnih 56-bitnih ključeva. Potom je za svaki ključ generisan slučajan 64-bitni otvoreni tekst i izračunat je odgovarajući šifrat. Ovim se dobija 100 trojki koje se sastoje od otvorenog teksta, ključa i odgovarajućeg šifrata. Test je urađen na sledeći način: za svako $n \in [1, 20]$ i za svaku od 100 trojki otvorenog teksta, ključa i odgovarajućeg šifrata, pretpostavljeno je da je prvih n bitova ključa nepoznato, a da su ostali bitovi poznati i konstruisana je iskazna formula, koja je potom rešena i na taj način su rekonstruisane vrednosti nepoznatih bitova ključa. Dobijena vremena su prikazana na slici 5.9a. Sa slike se može primetiti



Slika 5.9: (a) Vreme potrebno za nalaženje n ($n \in [1, 20]$) nepoznatih bitova ključa na osnovu poznatog jednog para otvorenog teksta i odgovarajućeg šifrata. Vremenska osa je logaritamska, a vreme je dato u sekundama. (b) Broj promenljivih i broj klauza u formulama koje su dobijene za različite brojeve nepoznatih bitova ključa.

da vreme ima sličan (eksponencijalan) rast, kada raste broj nepoznatih bitova ključa, kao u slučaju heš funkcije MD5 (slika 5.9a).

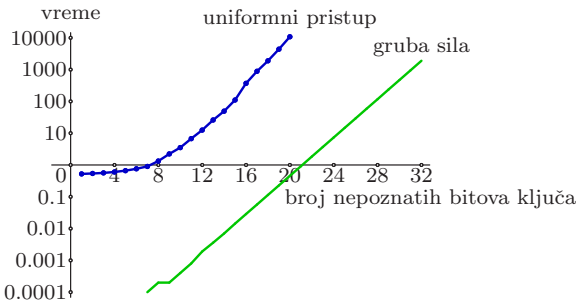
Takođe, kao i u slučaju heš funkcije MD5, broj promenljivih i broj klauza, nasuprot eksponencijalnom rastu vremena potrebnog za pronalaženje nepoznatih bitova ključa, raste kao za neki mali linearni faktor (slika 5.9b). Odnos ove dve veličine (broja klausa i broja promenljivih) je dat na slici 5.10. Ovaj odnos ne odstupa mnogo od vrednosti 3, kada broj nepoznatih bitova raste. Ono što se ne vidi jasno sa grafika je da vrednost ovog odnosa nije monoton (niti raste niti opada monotono). Ova nemonotonost govori o nepostojanju nekog vidljivog šablona po kojem se ponašaju dobijene formule. Zbog nje, pronalaženjem tačne vrednosti tačke fazne promene za klasu problema SAT kojoj pripadaju problemi



Slika 5.10: Odnos broja klauza (K) i broja promenljivih (V) u iskaznim formulama dobijenih u toku razbijanja oslabljenih verzija algoritma DES koje se dobijaju variranjem broja nepoznatih bitova ključa.

koji se ovim putem dobijaju se ne bi moglo zaključiti da li se sa povećanjem broja nepoznatih bitova ključa dobijaju teže ili lakše formule.

Dužina ključa u algoritmu DES je 56 bita. Na slici 5.9a su prikazani rezultati postignuti pri razbijanju oslabljene verzije algoritma DES dobijene pri pretpostavci da je od 56 bita ključa nepoznato n bita, gde $n \in [1, 20]$. Kako je 20 mali broj, u smislu da je u tom slučaju prostor pretrage ($[0, 2^{20} - 1]$) relativno mali, na oslabljene verzije algoritma DES je primenjena gruba sila. Poređenje rezultata dobijenih uniformnim pristupom i grubom silom su prikazani na slici 5.11. Kao što se i vidi, gruba sila je postigla znatno bolje rezultate od uni-



Slika 5.11: Poređenje grube sile i uniformnog pristupa na razbijanju oslabljenih verzija algoritma DES koje se dobijaju variranjem broja nepoznatih bitova ključa. Vremenska osa je logaritamska, a vreme je dato u sekundama.

formnog pristupa. Uniformni pristup je 20 nepoznatih bitova ključa našao za nešto više od 10000s, dok je ista stvar grubom silom urađena za manje od 1s. Ovo, zajedno sa rezultatima prikazanim na slici 5.6, nam daje jednu smelu pretpostavku: da uniformni pristup u odnosu na grubu silu daje bolje rezultate kod slabih (na primer, CMEA) algoritama nego kod jakih algoritam (na primer, DES). Formalnije, ovo znači da algoritam DES, pre svega zbog svog kvaliteta, generiše mnogo teže instance problema SAT nego neki jednostavniji kriptografski algoritmi, kao što je algoritam CMEA. Ovo bi, verovatno, bilo još izraženije kada bi se uniformni pristup primenio u razbijanju kvalitetnijih kriptografskih algoritama od algoritma DES. Odavde sledi da bi se uniformni pristup, možda,

mogao upotrebljavati pri poređenju kriptografskih algoritama u smislu njihovog kvaliteta.

Kao što je ranije napomenuto, za rešavanje problema SAT, koji su konstruisani u toku rada na ovoj tezi, korišćen je rešavač `Picosat` bez dodatnih parametara. Ovaj rešavač je korišćen zbog toga što je postigao najbolja vremena na malom skupu test formula. Drugim rečima, nisu obavljena opsežnija testiranja da bi se utvrdilo koji SAT rešavač izabrati za korišćenje. To navodi na pomisao da bi se možda postigla bolja vremena ako bi se koristio neki drugi rešavač, eventualno, uz korišćenje dodatnih pogodnih parametara.

Glava 6

Zaključci i dalji rad

U radu [JJ05] je predstavljen jedan uniforman pristup kodiranju kriptografskih algoritama formulama iskazne logike. U tom radu je pristup primenjen na heš funkcije MD4 i MD5, a u zaključku je navedena pretpostavka da se isti pristup može primeniti i na kriptografske algoritme. Ova teza predstavlja prirodni nastavak pomenutog rada. Osim rezultata primene ovog pristupa na neke kriptografske algoritme, ova teza sadrži i detaljniji opis njegove realizacije (od onog koji je dat u [JJ05]), kao i kratak primer koji detaljno prikazuje kako funkcioniše kompletan mehanizam razvijen u toku rada na ovoj tezi.

Do sada ovaj pristup nije bio primenjen na kriptografske algoritme CMEA i DES. U ovoj tezi je uniformni pristup prvo primenjeno na heš funkciju MD5, kako bi se uporedili rezultati sa rezultatima datim u [JJ05]. Kao što je i očekivano, dobijeni su bolji rezultati, pre svega zbog primene kvalitetnijih SAT rešavača, bržeg hardvera, a možda i zbog efikasnije implementacije. Nakon toga je isti pristup uspešno primenjen i na kriptografske algoritme CMEA i DES. Univerzalnost ovog pristupa je demonstrirana i rešavanjem problema korenovanja (*Naći broj, čiji je kvadrat dat*). Poređenjem rezultata, dobijenih primenom uniformnog pristupa i grube sile na problem korenovanja, dobijen je pozitivan odgovor na pitanje: *Da li se ovim pristupom postiže bilo kakvo ubrzanje u odnosu na grubu silu?*

Korišćeni pristup je zasnovan na osobini objektno-orijentisanih programskih jezika — preopterećivanju operatora. Za razvoj kompletnog mehanizma bilo je potrebno definisati nove tipove podataka koji su implementirani u vidu klasa. Primena razvijenog mehanizma u kriptoanalizi konkretnog kriptografskog algoritma je prilično jednostavna, što se može videti na slici 4.9.

Konstruisanje formule iskazne logike na osnovu zadate implementacije kriptografskog algoritma, pomoću razvijenog mehanizma, traje do nekoliko minuta. U nastavku rada bi moglo da se pokuša smanjivanje ovog vremena. Međutim, ovo ne predstavlja najveći problem jer se u kriptoanalizi veći deo vremena troši na rešavanje dobijene formule. Ovo je posledica, pre svega, same složenosti kriptografskih algoritama koja indukuje konstrukciju teških instanci problema SAT. Jedan način da se pokuša sa prevazilaženjem opisanih problema je da se ispitaju ostali pristupi za dobijanje KNF date formule, ali to je, bez mnogo uspeha, već pokušano u radu [JJ05]. Drugi način bi bio modifikacija postojećih realizacija SAT rešavača. Kako su gotovo svi SAT rešavači *otvorenog kôda* (eng. *open source*), njihovim prilagođavanjem problemima koji se dobijaju pri kodiranju

kriptografskih algoritama iskaznim formulama, bi se, verovatno, postigao izvestan napredak.

Osnovni problem kod opisanog univerzalnog pristupa je dobijanje teških instanci problema SAT, od kojih se, mnogi, ne mogu rešiti u razumnom vremenskom intervalu. Uzrok tome je pre svega kompleksnost kriptografskih algoritama. Osim toga, problem je u tome što se koriste postojeće implementacije algoritama. U opštem slučaju, jednom algoritmu odgovara više implementacija koje se međusobno razlikuju. Upravo zbog toga, bilo bi interesantno proveriti da li se kodiranjem neefikasnijih implementacija dobijaju teže formule. Ako bi se ispostavilo da je odgovor na ovo pitanje potvrdan, izvesno poboljšanje bi se moglo postići konstrukcijom što efikasnije implementacije konkretnog kriptografskog algoritma.

Pomenuti problem opisanog univerzalnog pristupa — dobijanje teških instanci problema SAT, može se smatrati i dobrom stranom. Naime, opisani pristup omogućava jednostavno generisanje i zadovoljivih i nezadovoljivih instanci problema SAT, praktično proizvoljne složenosti. Zato opisana metodologija može da se koristi za generisanje kvalitetnih test primera, što je veoma važno za poređenje i razvoj SAT rešavača.

I pored mogućih poboljšanja, uniformni pristup verovatno ne može da ima praktičnu primenu u kriptoanalizi. Svi problemi sa ovim pristupom su zapravo posledice njegove prevelike opštosti. Upravo ona je razlog nekonkurentnosti uniformnog pristupa u odnosu na specifična rešenja. S druge strane, zahvaljujući toj opštosti, moguće je da postoje praktične primene opisanog uniformnog pristupa u nekim drugim oblastima.

Nakon pojave rada [MZ], pretežno zbog dobrih rezultata koji su u njemu izloženi, objavljeno je nekoliko predloga teza (jedna od njih je i [JCV]) koje su zasnovane na ideji predstavljenoj u radu [MZ]. Zbog tih rezultata bi bilo interesantno implementirati napad predstavljen u tom radu, a potom povezati tu implementaciju sa mehanizmom opisanim u ovoj tezi. U radu [MZ] je jedan deo postojećeg kriptoanalitičkog napada kodiran formulama iskazne logike. Ako bi se u postojećim napadima na još neke kriptografske algoritme otkrila slična mogućnost (tj. mogućnost da se neki delovi napada kodiraju formulama iskazne logike), tada bi se mehanizam razvijan u toku rada na ovoj tezi, možda mogao upotrebiti u kodiranju delova tih napada iskaznim formulama.

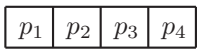
Dodatak A

Primer

U ovom dodatku će biti detaljnije prikazan rad kompletnog mehanizma u toku razbijanja jedne trivijalne šifre: *šifrat se od otvorenog teksta dobija cikličnim pomeranjem bitova za dve pozicije u levu stranu.*

Zbog jednostavnosti, pretpostavimo da radimo sa podacima koji se kodiraju sa 4 bita (tj. pretpostavimo da osnovni konstruktor klase `Word` konstruiše reč dužine 4). Program, po uzoru na onaj sa slike 4.11, koji služi za razbijanje ove šifre je prikazan na slici A.1.

Inicijalizacija. Linije 3 i 4 deklarišu dve promenljive tipa `Word`. Nakon deklaracije, ovi objekti nisu definisani pa je potrebno, pre njihovog korišćenja, to uraditi. Način definisanja objekta tipa `Word` zavisi od toga šta taj objekat treba da predstavlja. Jedan način definisanja je prikazan linijom 7, kojom se konstruiše niz objekata tipa `FormulaVar` a pokazivači na njih se smeštaju u privatni član `bitArray` objekta `plaintext` (pogledati deo deklaracije klase `Word` na slici 4.1). Dakle, nakon linije 7, objekat `plaintext` bi mogao da bude predstavljen na sledeći način[1]:



(A.1)

Drugi način da se definiše objekat tipa `Word` je na osnovu zadatog neoznačenog broja. Na primer, ako u kôdu postoji linija:

```
Word w = 10;
```

tada će privatni član, `bitArray`, objekta `w` predstavljati niz pokazivača[2] `T`, `NT`, `T` i `NT` (jer je 4-bitni zapis broja 10: 1010).

[1] Kao što je rečeno, promenljiva `plaintext` sadrži niz pokazivača na objekte koji predstavljaju formule. Međutim, da bi se uprostila slika, ovde su sa p_i označene same iskazne promenljive a ne pokazivači na njih.

[2] Sa `T` (`NT`) je ovde označen pokazivač na objekat klase `FormulaT` (`FormulaNT`) koja opisuje logičku konstantu *tačno* (*netačno*).

```
    /* Broj bitova kojima kodiramo otvoreni tekst i sifrat. */
 1  #define LEN 4

 2  main() {
 3          Word plaintextW,  /* Otvoreni tekst. */
 4               ciphertextW. /* Sifrat. */

            /*
             * Datoteka u koju ce biti upisana Cajcin definiciona
             * forma dobijene formule.
             */
 5          ofstream outf("formula.cnf");

            /* Poznata vrednost sifrata. */
 6          unsigned char ciphertextUC = 6;

            /* Inicijlizujemo otvoreni tekst (kao niz iskaznih promenljivih). */
 7          plaintext.init();

            /*
             * Sifrujemo otvoreni tekst tako sto ga ciklicno pomeramo
             * za dve pozicije u levo.
             */
 8          ciphertext = (plaintext << 2) | (plaintext >> (LEN - 2));

            /*
             * Izjednacavamo dobijene formule sa ucitanom vrednoscu sifrata
             * tako sto negiramo odgovarajuce formule (one koje odgovaraju
             * 0-bitovima u ucitanom sifratu).
             */
 9          ciphertext.negCorrespondingTo(ciphertextUC);

10          vector<vector<int> > conj;

            /*
             * Dodeljuju se jedinstveni identifikatori svim do ovog trenutka
             * napravljenim formulama.
             */
11          FormulaFactory::Instance()->SetIds();

            /* Kreira se Cajcin definiciona forma. */
12          FormulaFactory::Instance()->to_cnf(e_n, conj);

            /*
             * Dobijena Cajcin definiciona forma se zapisuje u datoteku
             * u DIMACS formatu.
             */
13          outf << "p cnf " << varCount << " " << conj.size() << endl;

14          vector<vector<int> >::iterator b_c = conj.begin(), e_c = conj.end();
15          for (vector<vector<int> >::iterator iter_c = b_c; iter_c != e_c; iter_c++) {
16                  vector<int>::iterator b_d = iter_c->begin(), e_d = iter_c->end();
17                  for (vector<int>::iterator iter_d = b_d; iter_d != e_d; iter_d++)
18                          outf << *iter_d << " ";
19                  outf << "0" << endl;
20          }
21  }
```

Slika A.1: Kôd programa koji razbija trivijalnu šifru.

Šifrovanje. Linija 8 obavlja šifrovanje. Prvo se kreiraju dva privremena objekta tipa `Word` u koje se, redom, smeštaju promenljiva `plaintext` pomerena (ne ciklično) za dve pozicije u levu, odnosno desnu stranu. Bitskom disjunkcijom dobijenih objekata se nalazi traženi šifrat koji se smešta u promenljivu `ciphertext`. Operatore pomeranja u levu i desnu stranu, kao i operator bitske disjunkcije ima smisla primeniti na objekte tipa `Word` jer su prethodno pre-

opterećeni (pogledati poglavlje 4.2). Nakon ovog koraka, objekat `ciphertext` bi mogao da bude predstavljen na sledeći način:

p_3	p_4	p_1	p_2

Zatim, u liniji 9, pozivom funkcije `negCorrespondingTo()` se *izjednačava* dobijeni šifrat (u promenljivoj `ciphertext`) sa argumentom koji prosleđujemo funkciji. U kôdu na slici A.1, argument ima vrednost 6, što je 0110 binarno, što znači da treba naći negaciju prvog i poslednjeg *bita* objekta `ciphertext`. Posle ovih promena, `ciphertext` će imati sledeći oblik:

$\neg p_3$	p_4	p_1	$\neg p_2$

(A.2)

Konstrukcija Cajcin definicione normalne forme. Kao što je ranije već rečeno, formula čija zadovoljivost treba da bude ispitana se dobija kao konjunkcija formula na koje pokazuju pokazivači koje sadrži promenljiva `ciphertext`. Dakle, tražena formula je

$$\neg p_3 \wedge p_4 \wedge p_1 \wedge \neg p_2.$$

Kao što se vidi, formula je već u KNF obliku tako da je ovde suvišno transformisanje u Cajcin definicionu normalnu formu, ali, s obzirom da je ovo primer koji treba da demonstrira rad kompletnog mehanizma, ovde će to biti urađeno.

Linija 12 sadrži poziv funkcije `to_cnf()` koja konstruiše Cajcin definicionu normalnu formu. Kako ova funkcija radi, već je opisano u poglavlju 4.5, tako da će se sada samo slediti koraci opisani u tom poglavlju.

Međutim, pre ovoga je potrebno pozvati funkciju `SetIds()` koja dodeljuje svim do tog trenutka napravljenim promenljivama jedinstveni identifikator, u vidu neoznačenog celog broja. Sve formule se čuvaju u heš tabeli koju održava objekat klase `FormulaFactory`. U ovom primeru, nakon što je izvršeno prvih 9 linija kôda, kreirano je šest formula koje su prikazane u tabeli A.1 (prikazani su i odgovarajući identifikatori). Cajcin definiciona normalna forma konstruiše se

`var`	p_1	p_2	p_3	p_4	$\neg p_2$	$\neg p_3$
`id`	1	2	3	4	5	6

Tabela A.1: Promenljive i identifikatori koji su im pridruženi.

u dva koraka:

1. Prolazi se kroz niz formula promenljive `ciphertext` A.2 i za svaku formulu u vektor `conj` ubacuje se jedan jednočlani vektor koji sadrži samo identifikator te formule. Posle ovog koraka, vektor `conj` će izgledati ovako:

$$\{\ \{\ 6\ \},\ \{\ 4\ \},\ \{\ 1\ \},\ \{\ 5\ \}\ \}$$

2. Nakon toga, prolazi se kroz heš tabelu i za svaku neatomičnu formulu ubacuje se u vektor `conj` 2, 3 ili 4 vektora koja konstruišu na način koji je opisan u poglavlju 4.5. Posle ovog koraka, vektor `conj` će izgledati ovako:

   $\{ \{ 6 \}, \{ 4 \}, \{ 1 \}, \{ 5 \}, \{ 6, 3 \}, \{ -6, -3\}, \{ 5, 2 \}, \{ -5, -2 \} \}$

Kreiranje datoteke u DIMACS formatu. Nakon što je vektor `conj` popunjen, njegov sadržaj se na opisani način upisuje u datoteku u DIMACS formatu. Ovo rade linije programa 13-20. Biće kreirana datoteka sa imenom `formula.cnf`, a njen sadržaj će biti:

```
p cnf 6 8
6 0
4 0
1 0
5 0
6 3 0
-6 -3 0
5 2 0
-5 -2 0
```

Rešavanje dobijene formule. Formula koja je opisana prethodnom datotekom je prilično jednostavna, pa može biti rešena i bez korišćenja SAT rešavača. Biće korišćene oznake iz tabele A.1, s tim da će biti uvedene promenljive p_5 i p_6 koje će odgovarati identifikatorima 5 i 6, respektivno. Tada se formula, opisana datotekom `formula.cnf` može zapisati na sledeći način:

$$p_6 \wedge p_4 \wedge p_1 \wedge p_5 \wedge (p_6 \vee p_3) \wedge (\neg p_6 \vee \neg p_3) \wedge (p_5 \vee p_2) \wedge (\neg p_5 \vee \neg p_2).$$

Jedina valuacija koja zadovoljava ovu formulu je prikazana tabelom A.2. Na

p	p_1	p_2	p_3	p_4	$\neg p_2$	$\neg p_3$
$v(p)$	1	0	0	1	1	1

Tabela A.2: Tabela vrednosti valuacije v.

početku ovog primera su bitovima promenljive `plaintext` dodeljene promenljive p_i $(i = 1, 2, 3, 4)$ (videti tabelu A.1) Dobijeno je da su bitovi otvorenog teksta 1001. Cikličnim pomeranjem 1001 za dve pozicije ulevo se zaista dobija šifrat 0110.

Bibliografija

[A3K01] D. Achlioptas, L. M. Kirousis, E. Kranakis, D. Krizanc, *Rigorous results for random 2 + p-SAT*, Theoretical Computer Science, 265:109-129, 2001.

[JCV] Jair Cazarin Villanueva, *Cryptanalysis of Hash Functions using an approach based on SAT-solvers*, February 2008.

[GW94] Ian P. Gent, Toby Walsh, *The SAT phase transition*, In Proceedings of ECAI 1994, pages 105-109, 1994.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlisside, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Massachusetts, 1995.

[Jan04] Predrag Janičić, *Matematička logika u računarstvu*, Matematički fakultet, 2004.

[JJ05] Dejan Jovanović, Predrag Janičić, *Logical Analysis of Hash Functions*, Frontiers of Combining Systems (FroCoS), editor Bernhard Gramlich, Lecture Notes in Artificial Intelligence, Volume 3717, Springer-Verlag, 2005.

[Kir07] Aleksandar Kirćanski, *Kriptoanaliza šifre CMEA*, 2007.

[FM05] Filip Marić, *Implementacija shema za ugradnju procedura odlučivanja u dokazivače teorema*, 2005.

[FM08] Filip Marić, *Formalization of SAT Solvers*, submitted to Journal of Automated Reasoning, 2008.

[MM00] Fabio Massacci, Laura Marraro, *Logical Cryptanalysis as a SAT Problem*, Journal of Automated Reasoning, Volume 24 Number 1-2, 2000.

[MZ] Ilya Mironov, Lintao Zhang, *Applications of SAT Solvers to Cryptanalysis of Hash Functions*

[WSK] David Wagner, Bruce Schneier, John Kelsey, *Cryptanalysis of the cellular message encryption algorithm*, Advances in Cryptology - CRYPTO97, volume 1294 of Lecture Notes in Computer Science, pages 526-537. Springer-Verlag, 1997.

[Ziv00] Miodrag Živković, *Algoritmi*, Matematički fakultet, 2000.

[TIA] Telecommunications Industry Association, TR45.0.A, *Common Cryptographic Algorithms*, June 1995, Rev B.

[Pico] `http://fmv.jku.at/picosat`

[GMP] `http://gmplib.org`

[SATComp] `http://www.satcompetitions.com`