UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Kristina Miletić

KRIPTOANALIZA ALGORITMA A5/1

master rad

Beograd, 2023.

Mentor:

dr Miodrag Živković, redovni profesor u penziji
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

prof. dr Saša Malkov, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

prof. dr Mladen Nikolić, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane:

Rezime: U ovom radu opisana je kriptoanaliza algoritma A5/1. Algoritam A5/1 se koristi za šifrovanje, odnosno dešifrovanje podataka koji se prenose komunikacijom u GSM-mreži. Najpre su u uvodnom delu opisane osnovne karakteristike GSM-mreže, kao i osnovni pojmovi kriptologije. U poglavljima 2 i 3 izloženi su arhitektura i načini kojim se obezbeđuju privatnost i sigurnost u GSM-mreži. Detaljan način funkcionisanja generatora niza ključa A5/1 koji se koristi u procesima šifrovanja i dešifrovanja opisan je u poglavlju 4. Nakon toga su u poglavlju 5 izloženi neki kriptoanalitički napadi na algoritam A5/1. Realizacija jednog od navedenih napada opisana je detaljno u poglavlju 6. Prikazan je i primer izvršavanja programa koji generiše niz bitova ključa, programa koji izvršava napad na algoritam A5/1 i programa koji određuje početno stanje registara. Takođe, izloženi su dobijeni rezultati testiranja programa.

Ključne reči: protočna šifra, simetrični šifarski sistem, A5/1, kriptoanaliza, GSM, 2G, generator ključa, pomerački registar sa linearnom povratnom spregom

Sadržaj

1 Uvod

Krajem 1980-ih, Evropski institut za telekomunikacione standarde - **ETSI** (engl. European Telecommunications Standards Institute), koji se bavi razvojem globalno primenljivih standarda za informacione i komunikacione tehnologije, uključujući fiksne, mobilne, radio i internet tehnologije, razvio je standard **GSM** (engl. Global System for Mobile Communications). GSM opisuje protokole za drugu generaciju (2G) digitalne mobilne tehnologije, zbog čega se i sam još naziva digitalnom tehnologijom. Iako prvobitno razvijen kao evropski digitalni komunikacioni standard koji bi omogućio korisnicima da neograničeno koriste svoje mobilne uređaje širom Evrope, vrlo brzo je postao standard koji se i danas koristi širom sveta [1].

GSM je prva digitalna tehnologija koja je ozbiljno razmatrala bezbednosne pretnje, za razliku od prethodnih koje praktično nisu bile zaštićene, a sve češće su bile predmet kriminalnih aktivnosti kao što su kloniranje telefona, prisluškivanje i krađa mobilnih poziva. Neki od načina kojim je GSM obezbedio zaštitu jesu uvođenje SIM kartice i dodeljivanje privremenog identifikacionog broja pretplatnika (TMSI), o čemu će biti više reči u poglavlju 3. Sem toga, privatnost telefonskih razgovora zaštićena je algoritmom **A5**. A5 se može nazvati klasom algoritama koja obuhvata nekoliko varijanti od kojih se za zaštitu privatnosti u 2G mreži koriste algoritmi A5/0, A5/1 i A5/2, koji su detaljnije opisani u tački 3.3. Iako su algoritmi čuvani u tajnosti, njihovi dizajni su otkriveni 1999. godine korišćenjem obrnutog inženjeringa, što ih je učinilo podložnijim napadima [2, 3, 4, 5].

Šifarski sistem (ili samo sistem) je par koji se sastoji od algoritma za šifrovanje i algoritma za dešifrovanje. **Ključ** je parametar od kog zavise ovi algoritmi i kojim se bira konkretna šifarska transformacija u okviru izabranog sistema. U ovom radu razmatra se **simetrični (šifarski) sistem**. On podrazumeva upotrebu istog tajnog ključa za šifrovanje i dešifrovanje. Strane čija komunikacija se šifruje moraju unapred da se dogovore koji ključ će koristiti. Simetrični šifarski sistem koji transformiše tekst simbol po simbol, odnosno najčešće bit po bit, naziva se **protočna šifra**. Algoritam A5/1 je jedan ovakav šifarski sistem.

Tekst koji se šifruje se zove **otvoreni tekst** (engl. plaintext), dok se šifrovani tekst zove **šifrat** (engl. ciphertext). Transformacija otvorenog teksta u šifrat naziva se **šifrovanje** (engl. encryption), dok je **dešifrovanje** (engl. decryption) inverzna

transformacija šifrata u otvoreni tekst. Šifrat se dobija primenom operacije XOR na otvoreni tekst i niz ključa, bit po bit, dok se otvoreni tekst dobija primenom iste operacije na šifrat i niz ključa.

Kriptografija je nauka koja se bavi razvojem i stvaranjem matematičkih algoritama koji se koriste za šifrovanje i dešifrovanje poruka. **Kriptoanaliza** je proces kojim se pokušava transformacija šifrata u otvoreni tekst, bez poznavanja ključa kojim je učinjeno šifrovanje. Takođe, kriptoanaliza se bavi proučavanjem i razbijanjem kriptografskih algoritama. Pronalaženjem slabe tačke u kriptografskom algoritmu može se poboljšati sam algoritam i kreirati bezbedniji šifarski sistem. Da bi se utvrdile slabe tačke kriptografskog sistema, potrebno je napasti sistem. Ovi napadi se nazivaju kriptoanalitički napadi.

Kriptologija je nauka koja proučava metode šifrovanja, odnosno analize šifrata. Kriptografija i kriptoanaliza zajedno čine kriptologiju [6, 7].

2 Arhitektura mreže GSM

Arhitekturu mreže GSM čine (videti sliku 1):

- Mobilna stanica (engl. Mobile Station)
- Podsistem baznih stanica (engl. Base Station Subsystem)
- Mrežni komutatorski podsistem (engl. Network Switching Subsystem)

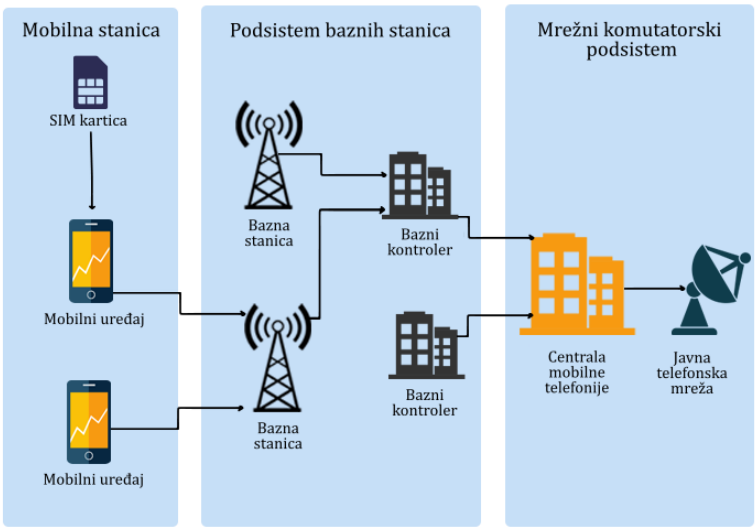
Mobilna stanica predstavlja bilo koji uređaj, a najčešće mobilni telefon. U svim sistemima mobilne telefonije geografsko područje se deli na ćelije (engl. cells), zbog čega se mobilni telefoni ponekad nazivaju celularni telefoni, a mobilna mreža celularna (engl. cellular network). U centru ćelije se nalazi bazna stanica ili bazni primopredajnik (engl. Base Transceiver Station – BTS) sa kojom, bežičnim putem, komuniciraju mobilne stanice. Svaka bazna stanica povezana je sa svojim kontrolerom (engl. Base Station Controller – BSC) koji upravlja radio-resursima ćelije i zajedno čine podsistem baznih stanica.

Mrežni komutatorski podsistem obavlja funkcije komutiranja i upravljanja komunikacije između mobilnih telefona i javne telefonske mreže. On se nalazi u vlasništvu operatera mobilne telefonije. Kontroleri baznih stanica komuniciraju sa centralom mobilne telefonije (engl. Mobile Switching Center – MSC), koja je deo mrežnog komutatorskog podsistema, a koja usmerava pozive i po potrebi se povezuje sa javnom telefonskom mrežom.

Centrala mobilne telefonije održava sledeće baze podataka:

- Registar matičnih pretplatnika (engl. Home Location Register – HLR) – baza podataka koja sadrži informacije o svakom pretplatniku mobilnog telefona u GSM-mreži, zajedno sa njegovom trenutnom lokacijom. Kada osoba kupi pretplatu kod mobilnog operatera u vidu SIM (engl. Subscriber Identity Module) kartice, sve informacije o njegovoj pretplati se upisuju u registar matičnih pretplatnika.
- Registar gostujućih pretplatnika (engl. Visitor Location Register – VLR) – baza podataka koja sadrži privremene informacije o pretplatnicima koji se trenutno nalaze u ćelijama kojim centrala mobilne telefonije upravlja, kako bi im pružila usluge.

- Autentifikacioni centar (engl. Authentication Center – AuC) – zaštićena baza podataka koja čuva kopije tajnih ključeva koji se nalaze u SIM karticama pretplatnika, koji se koriste za autentifikaciju i šifrovanje radio kanala prilikom povezivanja na mrežu i tokom komunikacije kroz nju.
- Registar identiteta uređaja (engl. Equipment Identity Register – EIR) – baza podataka koja sadrži listu validnih mobilnih uređaja koji se identifikuju svojim IMEI (engl. International Mobile Equipment Identity) brojem, kao i uređaje koji su zabranjeni na mreži ili se nadziru. Ova baza podataka omogućuje praćenje ukradenih ili neispravnih mobilnih telefona [8, 12].



Slika 1: Arhitektura GSM-mreže

3 Sigurnost i privatnost u okviru mreže GSM

Bežična komunikacija koja se odvija u GSM-mreži je manje sigurna od ožičene komunikacije, pa je stoga podložnija upadima i prisluškivanjima na mreži. U cilju zaštite privatnosti pretplatnika, u GSM-mrežu je ugrađeno nekoliko bezbednosnih funkcija. To su:

- Autentifikacija registrovanog pretplatnika
- Obezbeđenje sigurnosti prenosa podataka šifrovanjem
- Zaštita pretplatnikovog identiteta
- Nefunkcionalnost mobilnog telefona bez SIM kartice
- Nedozvoljena upotreba duplikata SIM kartice
- Bezbedno skladišten tajni ključ pretplatnika

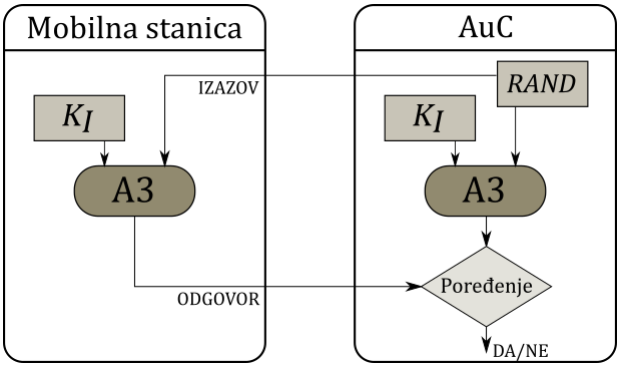
U nastavku se razmatra obezbeđenje sigurnosti podataka šifrovanjem, zbog svoje važnosti u ovom radu.

Sigurnost prenosa podataka šifrovanjem u GSM-mreži obezbeđena je korišćenjem tri algoritma: A3 (za autentifikaciju), A8 (za generisanje ključa) i A5 (za šifrovanje podataka). Algoritmi A3 i A8 su ugrađeni u SIM karticu.

3.1 Algoritam A3 - Autentifikacija

Procesom autentifikacije se proverava ispravnost pretplatnikove SIM kartice, a zatim se odlučuje da li je mobilnoj stanici dozvoljen pristup mreži. Mreža (tačnije centrala mobilne telefonije, MSC) autentifikuje pretplatnika korišćenjem metode izazov-odgovor (engl. challenge-response).

Najpre se 128-bitni slučajan broj RAND (engl. random number) bežičnim putem šalje mobilnoj stanici, a zatim prosleđuje SIM kartici (videti sliku 2). Na osnovu broja RAND i tajnog ključa SIM kartice K_I, algoritam A3 računa odgovor SRES (engl. Signed Response) koji se bežičnim putem vraća centrali mobilne telefonije. Autentifikacioni centar upoređuje svoj SRES (izračunat na identičan način



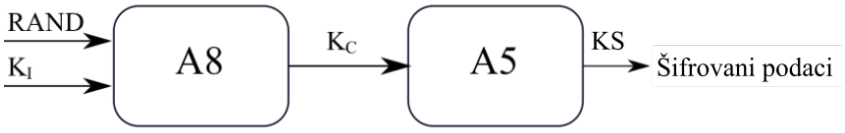
Slika 2: Autentifikacija u GSM-mreži

algoritmom A3) sa dobijenim SRES odgovorom koji je dobio od mobilne stanice. Ako se ta dva odgovora poklapaju, autentifikacija je uspešna i pretplatnik se pridružuje mreži.

Kada novi GSM pretplatnik prvi put uključi svoj telefon, njegov internacionalni identifikacioni broj pretplatnika (engl. International Mobile Subscriber Identity – IMSI), koji se nalazi u SIM kartici, se šalje autentifikacionom centru mreže. Nakon toga mu se dodeljuje privremeni identifikacioni broj pretplatnika (engl. Temporary Mobile Subscriber Identity – TMSI). Posle dodele TMSI, IMSI se retko šalje, izuzev ako je apsolutno neophodno. Ovo obezbeđuje zaštitu od potencijalnog prisluškivanja i identifikovanja pretplatnika na osnovu njegovog IMSI. Pretplatnik nastavlja da koristi isti TMSI sve dok ne promeni svoju lokaciju, nakon čega mu se dodeljuje novi. Mobilna stanica koristi TMSI da se prijavi na mrežu ili tokom iniciranja poziva. Slično, mreža koristi TMSI da komunicira sa mobilnom stanicom. Registar gostujućih pretplatnika (VLR) vrši dodeljivanje, administraciju i ažuriranje TMSI. Kada je telefon isključen, mobilna stanica čuva TMSI na SIM kartici da obezbedi njegovu dostupnost kada se ponovo uključi.

3.2 Algoritam A8 - Generisanje ključa

Kada se izvrši autentifikacija korisnika, broj RAND (primljen od strane MSC) zajedno sa tajnim ključem K_I na SIM kartici se prosleđuje algoritmu A8 za generisanje ključa za šifrovanje K_C (videti sliku 3). Dobijeni ključ se zatim zajedno sa 22-bitnim javnim brojem okvira koristi u algoritmu A5 za generisanje ključa KS za šifrovanje i dešifrovanje podataka.



Slika 3: Šifrovanje podataka pomoću ključa

3.3 Klasa algoritama A5 - Šifrovanje podataka

Algoritam klase A5 je implementiran u mobilnom telefonu, kako bi šifrovao i dešifrovao podatke pred slanje, odnosno posle prijema. Trenutno postoji nekoliko verzija algoritma A5 koji se koriste u GSM-mreži, a to su:

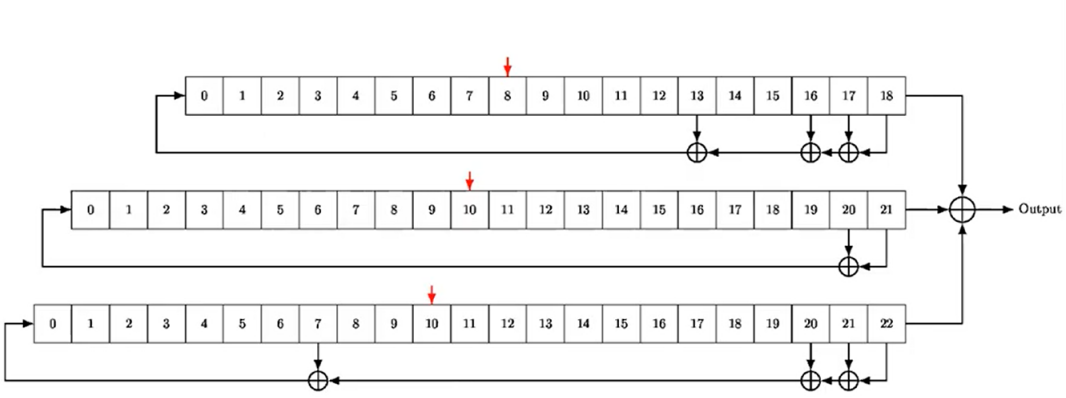
- Algoritam A5/0 koji ne šifruje podatke. Koristi se u zemljama pod sankcijom Ujedinjenih nacija i nekim zemljama trećeg sveta.
- Algoritam A5/1 predstavlja najjaču verziju algoritma i široko se koristi u Evropi i Americi.
- Algoritam A5/2 [9] je slabija verzija algoritma koji se koristio u Aziji.
- Algoritmi A5/3 [10] i A5/4 [11] koji se koriste u 3G mobilnim tehnologijama. Zasnovani su na blokovskoj šifri KASUMI.

Sva tri algoritma su protočne šifre, koje su potpuno određene svojim generatorom niza ključa. Pri tome je A5/0 trivijalni generator, koji na izlazu daje niz nula [12, 13].

4 Generator niza ključa A5/1

A5/1 je protočna šifra koja koristi tri pomeračka registra sa linearnom povratnom spregom R_1, R_2 i R_3 (engl. LFSR - Linear Feedback Shift Register) dužina 19, 22 i 23 bita, redom. Struktura algoritma data je na slici 4. Ćelije svakog registra su numerisane sleva udesno sa $0, ..., n-1$, gde je n dužina pomeračkog registra.

Taktovanje registra podrazumeva pomeranje sadržaja registra za jednu poziciju udesno, pri čemu se u oslobođenu ćeliju sa indeksom nula upisuje bit koji se dobija tako što se XOR-uju vrednosti bitova sa povratnih grana tog registra. Bit u ćeliji sa indeksom $n-1$ je izlaz registra u tom taktu i on se odbacuje. Za registar R_1 se za povratnu spregu uzimaju bitovi na pozicijama 13, 16, 17 i 18, za registar R_2 na pozicijama 20 i 21, a za registar R_3 na pozicijama 7, 20, 21 i 22.



Slika 4: Struktura algoritma A5/1

Taktovanje se vrši koristeći većinsku (engl. majority) funkciju. Ulazne vrednosti za ovu funkciju su bitovi registara $R_1[8], R_2[10]$ i $R_3[10]$. U daljem tekstu ovi bitovi se zovu taktujući bitovi. Izlazna vrednost funkcije je vrednost koju ima većina bitova, tj.

$$\text{majority}(a, b, c) = a \cdot b \oplus a \cdot c \oplus b \cdot c$$

Taktuju se samo oni registri čiji bit se poklapa sa vrednošću izlaznog bita m većinske funkcije, odnosno: registar R_1 se taktuje ako i samo ako se $R_1[8]$ poklapa sa m, registar R_2 se taktuje ako i samo ako se $R_2[10]$ poklapa sa m i registar R_3 se taktuje ako i samo ako se $R_3[10]$ poklapa sa m. Uvek se taktuju najmanje dva od tri registra (videti tabelu 1).

$R_1[8]$	$R_2[10]$	$R_3[10]$	majority($R_1[8]$, $R_2[10]$, $R_3[10]$)	R_1	R_2	R_3
0	0	0	0	✓	✓	✓
0	0	1	0	✓	✓	–
0	1	0	0	✓	–	✓
0	1	1	1	–	✓	✓
1	0	0	0	–	✓	✓
1	0	1	1	✓	–	✓
1	1	0	1	✓	✓	–
1	1	1	1	✓	✓	✓

Tabela 1: Taktovanje u algoritmu A5/1

Ulazne vrednosti algoritma A5/1 su 64-bitni ključ K (engl. key) i javni 22-bitni broj okvira F (engl. frame number). Generator A5/1 niza bitova ključa za šifrovanje radi na sledeći način. Prvo se izvršava *faza inicijalizacije*. Na početku ove faze se vrednosti u svim ćelijama registara postave na nulu. Potom se vrši postavljanje ključa (engl. key setup), gde se sva tri registra taktuju bez upotrebe većinske funkcije (u svakom koraku se taktuju sva tri registra). Na poziciju sa indeksom nula se upisuje bit dobijen primenom XOR operacije na prethodnu vrednost tog bita i tekući bit ključa. Ovakvo taktovanje se ponavlja 64 puta. Zatim se na sličan način vrši postavljanje inicijalizacionog vektora (engl. initialization vector setup), pri čemu se umesto bitova ključa XOR-uju bitovi okvira F i taktovanje se ponavlja 22 puta. Neka je S^I stanje pomeračkih registara na kraju ove faze. Opisani postupak dat je sledećim pseudokodom:

1. $R_1 = R_2 = R_3 = 0$;
2. for $i = 0$ to 63
   - taktovatiRegistre();
   - $R_1[0] = R_1[0] \oplus K[i]$;
   - $R_2[0] = R_2[0] \oplus K[i]$;
   - $R_3[0] = R_3[0] \oplus K[i]$;
3. for $i = 0$ to 21
   - taktovatiRegistre();
   - $R_1[0] = R_1[0] \oplus F[i]$;
   - $R_2[0] = R_2[0] \oplus F[i]$;

- $R_3[0] = R_3[0] \oplus F[i]$;

Funkcija taktovatiRegistre() vrši taktovanje registara R_1, R_2 i R_3, bez korišćenja većinske funkcije.

Sledeća faza je *faza praznog hoda* u kojoj se generator taktuje 100 puta, ali se izlazni bitovi odbacuju. Taktovanje je, kao i u fazi koja sledi, kontrolisano većinskom funkcijom. Neka je S^W stanje pomeračkih registara na kraju ove faze.

Nakon ove faze, generator proizvodi prvi izlazni bit, što je tek u 101. koraku nakon inicijalizacione faze. U svakom taktu se dobija po jedan izlazni bit primenom operacije XOR na najznačajnije bitove registara, tj.

$$R_1^t[18] \oplus R_2^t[21] \oplus R_3^t[22] = KS[t],$$

gde su $R_1^t[18]$, $R_2^t[21]$ i $R_3^t[22]$ najznačajniji bitovi datih registara u trenutku t, $t = \overline{0, 227}$, a $KS[t]$ predstavlja t–ti bit niza bitova ključa KS. Generator ključa A5/1 se taktuje 228 puta i tako proizvodi 228-bitnu izlaznu vrednost KS.

Izlazna vrednost od 228 bitova se deli na dva dela. Prvih 114 bitova se koriste za šifrovanje podataka u odlaznom saobraćaju, od telefona ka mreži, a ostalih 114 bitova za dešifrovanje podataka u dolaznom saobraćaju, od mreže ka telefonu [5, 14, 15].

Tokom razgovora, strane koje vrše komunikaciju razmenjuju podatke u okvirima dužine 228 bitova (po 114 bitova u oba smera) na svakih 4,6 milisekundi. Svaki okvir se šifruje primenom operacije XOR na dobijene bitove niza ključa i bitove poruke. Dešifrovanje se vrši na sličan način - primenom operacije XOR na bitove niza ključa i bitove šifrata [16].

5 Napadi na algoritam A5/1

Kriptoanalitički napadi se, na osnovu informacija sa kojim se raspolaže, mogu podeliti na:

- Napad sa poznavanjem samo šifrata - napadač poseduje samo šifrat, ali ne i otvoreni tekst. U ovom slučaju napadač može znati algoritam šifrovanja, ali ne i ključ kojim se šifrovanje vrši.
- Napad sa poznatim otvorenim tekstom - napadač poseduje neke parove (šifrat, otvoreni tekst).
- Napad sa poznavanjem izabranog otvorenog teksta - pretpostavlja se da napadač može da po želji izabere otvoreni tekst i šifruje ga [7].

U ovom radu razmatraju se napadi na algoritam A5/1 sa poznatim otvorenim tekstom. U tački 5.1 je opisan napad KZ, a u tački 5.2 je opisan napad GNR, realizovan u poglavlju 6.

Pretpostavlja se da je moguće presretanje šifrovanog okvira, kao i da se poseduju odgovarajući otvoreni tekst i javni broj okvira F. Prva pretpostavka je uobičajena u kriptoanalizi i u ovom radu neće biti objašnjeno kako se vrši presretanje podataka. S druge strane, ova pretpostavka je opravdana na osnovu sledećeg. Signalni i govorni podaci se prenose u okvirima. Signalni podaci imaju specifičan format i stoga, ako je moguće presresti šifrovani okvir sa pomenutim podacima, sa određenom verovatnoćom se može saznati njegov sadržaj, tj. otvoreni tekst. U opisanim napadima potreban je samo mali broj okvira (obično samo jedan), dok se u drugim napadima pretpostavlja da se zna nekoliko sekundi otvorenog teksta [16].

5.1 Napad KZ

Napad KZ ([16], čiji su autori J. Keller i B. Seitz) se sastoji od dve faze: determinističke faze, u kojoj se generišu kandidati za stanje S^W i naknadne obrade, u kojoj se proverava konzistentnost dobijenih kandidata.

U petlji se prolazi kroz sva moguća stanja registara R_1 i R_2. Za svaku pretpostavku pokušava se određivanje stanja registra R_3 na sledeći način. Neka je

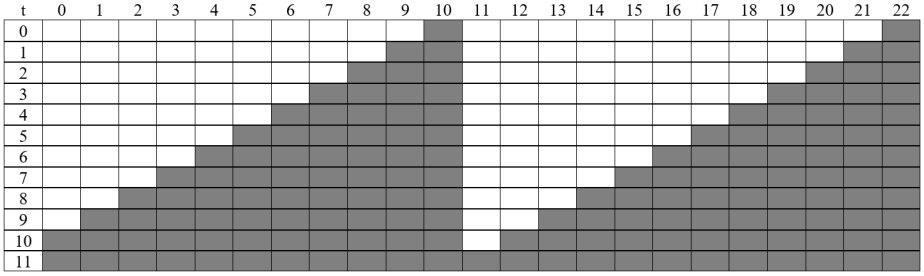
$R_i^{(t)}[n]$ n-ti bit registra R_i u trenutku t. Trenutak $t = 0$ je odmah nakon faze praznog hoda generatora niza ključa A5/1 i uvećava se za jedan nakon svakog takta. Prvo se izračuna prvi najznačajniji bit registra R_3, tj. $R_3^{(0)}[22]$, na osnovu $R_1^{(0)}[18]$, $R_2^{(0)}[21]$ i prvog bita poznatog niza ključa KS, odnosno $R_3^{(0)}[22] = R_1^{(0)}[18] \oplus R_2^{(0)}[21] \oplus KS[0]$. Potom se ispituju taktujući bitovi registara R_1 i R_2, tj. $R_1^{(0)}[8]$ i $R_2^{(0)}[10]$, i pogađa se prvi taktujući bit registra R_3, $R_3^{(0)}[10]$.

- Ako $R_1^{(0)}[8]$ i $R_2^{(0)}[10]$ nisu jednaki, R_3 se u svakom slučaju taktuje, pa se moraju proveriti obe mogućnosti za $R_3^{(0)}[10]$.
- Ako su $R_1^{(0)}[8]$ i $R_2^{(0)}[10]$ jednaki, onda se ova dva registra sigurno taktuju.
  - Ako je $R_3^{(0)}[10]$ izabran tako da je $R_3^{(0)}[10] = R_1^{(0)}[8]$, onda su taktujući bitovi sva tri registra jednaki pa se sva tri registra taktuju.
  - Ako je $R_3^{(0)}[10]$ izabran tako da ima vrednost različitu od taktujućih bitova registara R_1 i R_2, tj. $R_3^{(0)}[10] \neq R_1^{(0)}[8]$, tada se R_3 ne taktuje i njegov najznačajniji bit u sledećem taktu ostaje isti. U jednoj polovini slučajeva (u proseku) izlazni bit generisan od najznačajnijih bitova sva tri registra ($R_1^{(1)}[18] = R_1^{(0)}[17]$, $R_2^{(1)}[21] = R_2^{(0)}[20]$, $R_3^{(1)}[22] = R_3^{(0)}[22]$) se ne poklapa sa sledećim bitom poznatog niza ključa $KS[1]$, što dovodi do kontradikcije. Odavde sledi da se taktujući bit registra R_3 mora odabrati tako da se R_3 taktuje zajedno sa R_1 i R_2, tj. mora biti jednak taktujućim bitovima registara R_1 i R_2, kako bi novi najznačajniji bit registra R_3 mogao da se izračuna. Ovim se redukuje broj mogućnosti za registar R_3 sa dve na jednu, pa je potrebno proveriti $(\frac{3}{2})^{11} \approx 85$ mogućnosti.

Nakon izračunavanja prvog najznačajnijeg bita registra R_3, proces računanja sledećeg najznačajnijeg bita i pogađanja taktujućeg bita se ponavlja sve dok se registar $R_3^{(0)}$ u potpunosti ne odredi, pri čemu se poslednji bit $R_3^{(0)}[11]$ računa u 12. taktu. U slučaju dobijanja kontradikcije tokom pogađanja $R_3^{(0)}[10]$, nije potrebno nastavljati opisani postupak.

Sadržaj registra R_3 se može podeliti na dve polovine: prva polovina sadrži bitove na pozicijama od 0 do 10, a druga polovina bitove na pozicijama od 11 do 22. Na osnovu izloženog postupka određivanja registra R_3 u napadu KZ, u svakoj od polovina bitovi se određuju unazad, tj. od pozicija veće ka pozicijama manje težine. U svakom taktu t se izračuna jedan najznačajniji bit i pogodi jedan taktujući bit registra R_3. Nakon prvog takta, u registru $R_3^{(0)}$ su poznati 22. i 10.

bit; nakon drugog takta 22, 21, 10. i 9. bit; nakon trećeg takta 22, 21, 20, 10, 9. i 8. bit, itd. U 12 taktu još treba da se izračuna poslednji nepoznati bit, 11. bit. Nakon 12. takta u registru R_3 su poznati svi bitovi (videti sliku 5). Dakle, registar R_3 u trenutku $t = 0$ je u potpunosti određen nakon 12 taktova.



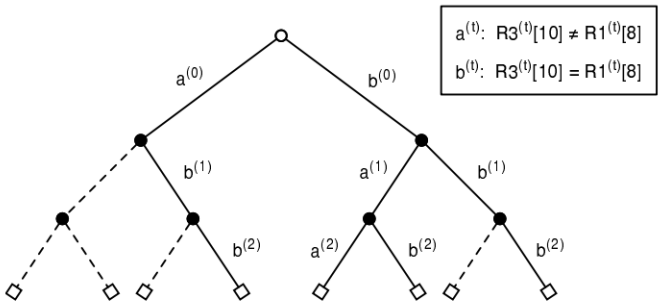
Slika 5: Ćelije obojene sivom bojom predstavljaju pozicije određenih bitova registra $R_3^{(0)}$ u svakom taktu t. U svakom taktu se računa sledeći najznačajniji bit i pogađa sledeći taktujući bit registra R_3.

U drugoj fazi, koja sledi odmah nakon determinističke faze, generator ključa A5/1 se izvršava sa generisanim kandidatom za stanje S^W i dobijeni niz bitova ključa se upoređuje sa bitovima poznatog niza ključa KS. Ako dođe do nesaglasnosti, odbacuje se pretpostavljeno stanje registara R_1 i R_2. U suprotnom je pronađeno jedno moguće stanje registara nakon faze praznog hoda [14].

5.2 Napad GNR

Napad GNR ([14], čiji su autori T. Gendrullis, M. Novotnỳ i A. Rupp) predstavlja modifikaciju napada KZ. U ovom napadu se pretpostavlja da su poznata najmanje 64 uzastopna bita niza ključa za šifrovanje, odnosno dešifrovanje (od maksimalnih 228 bitova, koji se dobijaju generatorom niza ključa A5/1). U slučaju kada su taktujući bitovi registara R_1 i R_2 jednaki, ako pretpostavka $R_3^{(t)}[10] \neq R_1^{(t)}[8]$ o taktujućem bitu registra R_3 ne dovodi do kontradikcije, nastavlja se sa izvršavanjem rekurzivnog poziva, iako se registar R_3 ne taktuje. Mogući kandidat za $R_3^{(t)}[10]$ se odbacuje samo u slučaju kontradikcije pri proveri jednakosti bita poznatog niza ključa KS sa generisanim bitom niza ključa dobijenim primenom operacije XOR na najznačajnije bitove registara R_1, R_2 i R_3, pri čemu je za registar R_3 to poslednji izračunat najznačajniji bit.

Napad GNR koristi binarno stablo odlučivanja visine 11 (videti sliku 6) u okviru kojeg se na svakom nivou uvode dve pretpostavke o taktujućem bitu registra R_3.



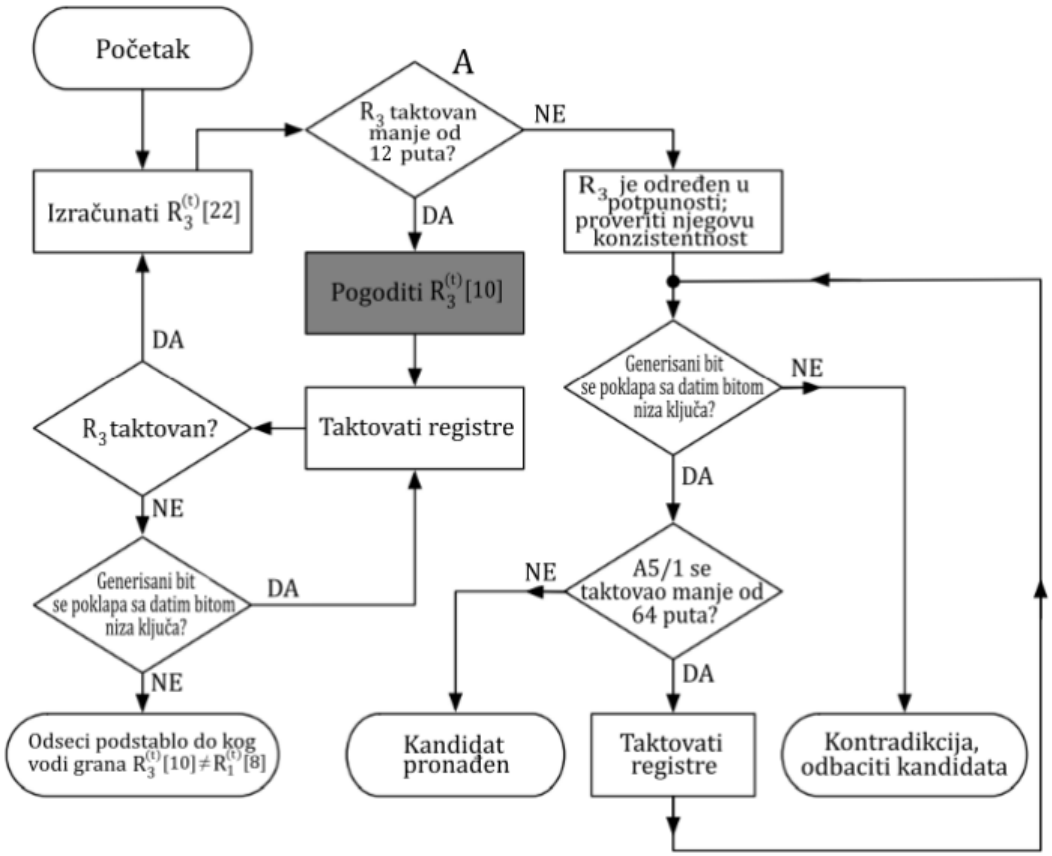
Slika 6: Primer binarnog stabla odlučivanja do dubine 3

Od svakog čvora polaze dve grane: leva grana kojoj odgovara mogućnost $R_3^{(t)}[10] \neq R_1^{(t)}[8]$ i desna grana kojoj odgovara mogućnost $R_3^{(t)}[10] = R_1^{(t)}[8]$. Prilikom prelaska po desnoj grani registar R_3 se uvek taktuje, jer ako je $R_3^{(t)}[10] = R_1^{(t)}[8]$ onda registri R_1 i R_3 imaju isti taktujući bit i oni se sigurno taktuju. Prilikom prelaska po levoj grani taktovanje registra R_3 zavisi od jednakosti taktujućih bitova registara R_1 i R_2. Ako oni nisu jednaki, registar R_3 se taktuje, jer njegov taktujući bit ima vrednost taktujućeg bita registra R_2, pa se taktuju ova dva registra. U suprotnom se taktuju samo registri R_1 i R_2.

Napad se, kao i u slučaju napada KZ, sastoji od dve faze (videti sliku 7). Pre samog početka se primenom XOR operacije na najznačajnije bitove registara R_1 i R_2 i prvi bit niza ključa KS izračuna najznačajniji bit registra R_3.
Prva faza se može podeliti na dva dela.

- U prvom delu se računa najznačajniji i pogađa taktujući bit registra R_3 u trenutku t. Zatim se odgovarajući registri taktuju, uz korišćenje većinske funkcije. Treba naglasiti da se registar R_3 ovde ne taktuje stvarno, jer su njegovi bitovi nepoznati, već se samo beleži da li on treba ili ne da bude taktovan u tekućem koraku.

- U drugom delu se proverava da li se registar R_3 taktovao i, ako jeste, opisani postupak iz prvog dela se ponavlja. U suprotnom se vrši provera jednakosti sledećeg bita niza ključa KS i bita dobijenog primenom operacije XOR na najznačajnije bitove registara R_1, R_2 i R_3, pri čemu su najznačajniji bitovi registara R_1 i R_2 iz tekućeg takta, dok je kod registra R_3 to poslednji izračunat najznačajniji bit. Ukoliko jednakost nije ispunjena, vrši se odsecanje levog podstabla tekućeg čvora u stablu odlučivanja, jer odabir taktujućeg bita registra R_3 koji odgovara levoj grani tekućeg čvora dovodi do kontra-



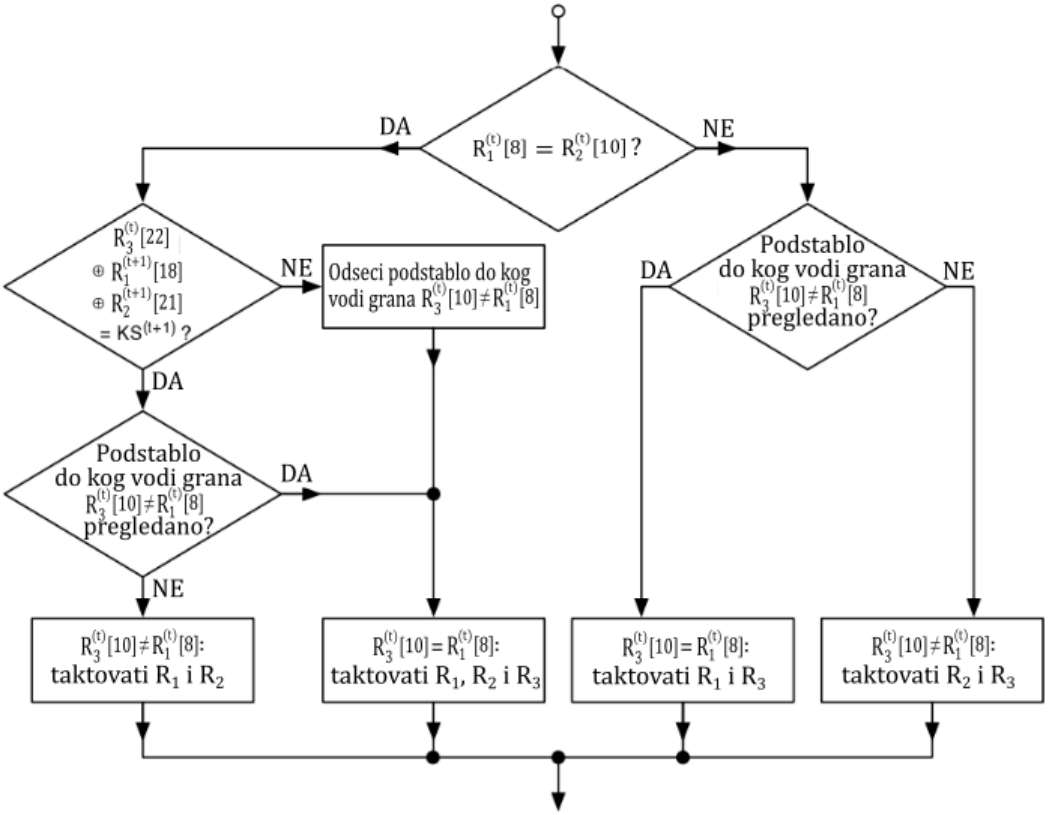
Slika 7: Dijagram toka napada GNR. Prva faza napada počinje od bloka „Početak" i traje sve dok je ispunjen uslov u tački A. Druga faza obuhvata čitav deo naredbi od negativnog odgovora na uslov u tački A. Naredba u pravougaoniku obojenom sivom bojom ukazuje na algoritam pogađanja taktujućeg bita registra R_3 koji je prikazan na slici 8.

dikcije. U suprotnom se odgovarajući registri ponovo taktuju, nakon čega se ponavlja drugi deo ove faze, sve dok se registar R_3 ne taktuje.

Druga faza napada je identična drugoj fazi napada KZ.

Pogađanje taktujućeg bita registra R_3 se vrši tako što se najpre upoređuju taktujući bitovi registara R_1 i R_2 (videti sliku 8).

- U slučaju da nisu jednaki, proverava se da li je levo podstablo iz tekućeg čvora u stablu odlučivanja pregledano. Ako jeste, taktujući bit registra R_3 se postavlja na vrednost taktujućeg bita registra R_1, odnosno $R_3^{(t)}[10] = R_1^{(t)}[8]$.



Slika 8: Detaljan postupak pogađanja taktujućeg bita registra R_3

U stablu odlučivanja se bira desna grana iz tekućeg čvora, a potom se registri R_1 i R_3 taktuju. U suprotnom, taktujući bit registra R_3 se postavlja na vrednost taktujućeg bita registra R_2, odnosno $R_3^{(t)}[10] = R_2^{(t)}[10]$. U stablu odlučivanja se bira leva grana iz tekućeg čvora, a potom se registri R_2 i R_3 taktuju.

- U slučaju da su taktujući bitovi registara R_1 i R_2 jednaki, najpre se proverava jednakost
$$R_3^{(t)}[22] \oplus R_1^{(t+1)}[18] \oplus R_2^{(t+1)}[21] = KS^{(t+1)}.$$
  - Ukoliko je jednakost tačna, proverava se, kao u prethodnom slučaju, da li je levo podstablo iz tekućeg čvora stabla odlučivanja pregledano. Ako jeste, taktujući bit registra R_3 se postavlja na vrednost koju imaju taktujući bitovi registara R_1 i R_2. U stablu odlučivanja se bira desna grana iz tekućeg čvora, a potom se taktuju sva tri registra. U suprotnom,

taktujući bit registra R_3 se postavlja na suprotnu vrednost od one koju imaju taktujući bitovi registara R_1 i R_2. U stablu odlučivanja se bira leva grana iz tekućeg čvora, a potom se taktuju samo registri R_1 i R_2.

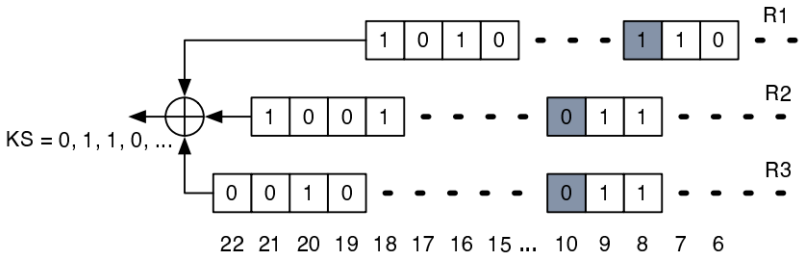
– Ukoliko jednakost nije tačna, vrši se odsecanje levog podstabla trenutnog čvora stabla odlučivanja, odnosno netaktovanje registra R_3 dovodi do kontradikcije i u tom slučaju se taktujući bit registra R_3 mora postaviti na vrednost koju imaju taktujući bitovi registara R_1 i R_2. U stablu odlučivanja se bira desna grana iz tekućeg čvora, a potom se taktuju sva tri registra.

U nastavku je prikazano prvih nekoliko koraka dobijanja jednog kandidata za registar R_3. Primer prati slika 9, na kojoj su prikazana prva četiri bita poznatog niza ključa KS, po četiri najznačajnija bita i po tri taktujuća bita registara R_1, R_2 i R_3. Algoritam se izvršava na sledeći način:

1. Računa se $R_3^{(0)}[22] = R_1^{(0)}[18] \oplus R_2^{(0)}[21] \oplus KS[0] = 0$.
2. $R_1^{(0)}[8] \neq R_2^{(0)}[10]$: Prvo se bira $R_3^{(0)}[10] = 0 \neq R_1^{(0)}[8]$ i taktuju se registri R_2 i R_3.
3. Računa se $R_3^{(1)}[22] = R_3^{(0)}[21] = R_1^{(0)}[18] \oplus R_2^{(0)}[20] \oplus KS[1] = 0$.
4. $R_1^{(0)}[8] = R_2^{(0)}[9]$: Ako se registar R_3 ne bi taktovao, došlo bi do kontradikcije, jer $R_1^{(0)}[17] \oplus R_2^{(0)}[19] \oplus R_3^{(0)}[21] \neq KS[2]$, pa se odbacuje mogućnost $R_3^{(1)}[10] = 0 = R_3^{(0)}[9] \neq R_1^{(1)}[8]$. Stoga se bira $R_3^{(1)}[10] = 1 = R_3^{(0)}[9] = R_1^{(0)}[8]$ i taktuju se sva tri registra.
5. Računa se $R_3^{(2)}[22] = R_3^{(0)}[20] = R_1^{(0)}[17] \oplus R_2^{(0)}[19] \oplus KS[2] = 1$.
6. ...

Odgovarajuće binarno stablo odlučivanja je prikazano na slici 6. U gornjem primeru se najpre bira grana $a^{(0)} = R_3^{(0)}[10] = 0 \neq R_1^{(0)}[8]$ iz korena stabla, a potom obacuje mogućnost $a^{(1)} = R_3^{(1)}[10] = 0 \neq R_1^{(0)}[8]$ na visini 1 itd.

Primetimo da ovaj napad određuje samo kandidate za stanje registara S^W, tj. stanje nakon faze praznog hoda. Početno stanje ostaje nepoznato. Realizacija ovog napada izložena u poglavlju 6, sem određivanja kandidata za stanje registara S^W, obuhvata i određivanje početnog stanja (ili više njih) za svako dobijeno



Slika 9: Primer određivanja kandidata za registar R_3

stanje S^W. Program u kom je realizovano određivanje početnog stanja se zove initial_state_reversion i opisan je u tački 6.5. Ovaj program vrši „premotavanje" unazad za 100 koraka stanja registara S^W koristeći stablo pretrage u čijem korenu se nalazi pomenuto stanje. Rezultujuća stanja, tj. moguća početna stanja registara S^I, se nalaze na visini $h = 100$ stabla, tj. u njegovim listovima.

6 Realizacija napada na algoritam A5/1

U ovom poglavlju je opisana realizacija napada GNR sa poznatim delom niza bitova ključa koji je izložen u tački 5.2. Tačnije, poznato je prvih n bitova niza ključa, gde je $n \in \{18, 32, 40, 48, 56, 64\}$. Napad je, za razliku od opisanog, implementiran i za skraćene verzije registara čiji su zbirovi dužina n iz pomenutog skupa.

Za razvoj programa korišćen je programski jezik Python (verzija 3.9.0).

6.1 Organizacija programa

Program se sastoji iz tri dela: a51_keystream_generator, a51_attack i initial_state_reversion. Program a51_keystream_generator generiše niz bitova ključa koji se koristi za šifrovanje otvorenog teksta, odnosno dešifrovanje šifrata. Izlaz ovog programa je dobijeni niz bitova ključa skraćen na prvih n bitova, gde je $n \in \{18, 32, 40, 48, 56, 64\}$ zbir dužina registara. Drugi program, a51_attack, izvršava napad na A5/1. Ulaz ovog programa je n bitova niza ključa (dobijenih prvim programom), dok je izlazna vrednost skup kandidata za stanje registara S^W. Treći program, initial_state_reversion, određuje početna stanja registara S^I od stanja S^W dobijenih drugim programom.

Pored ova tri glavna programa, postoji i jedan pomoćni, a51_utils.

6.2 Program a51_utils

Program a51_utils je pomoćni program koji sadrži funkcije i globalne promenljive koje koriste programi a51_keystream_generator, a51_attack i initial_state_reversion. U nastavku je opisan njegov sadržaj.

Globalne promenljive

Globalne promenljive u ovom programu su promenljive koje sadrže odgovarajuće dužine i bitne pozicije bitova registara. To su:

- **LEN_R1, LEN_R2, LEN_R3** - dužine registara R_1, R_2 i R_3;

- **LEN_REG_TOTAL** - zbir dužina registara R_1, R_2 i R_3;
- **R1_TAP1, R1_TAP2, R1_TAP3, R1_TAP4** - pozicije bitova povratne sprege registra R_1;
- **R2_TAP1, R2_TAP2** - pozicije bitova povratne sprege registra R_2;
- **R3_TAP1, R3_TAP2, R3_TAP3, R3_TAP4** - pozicije bitova povratne sprege registra R_3;
- **R1_CLOCKING_BIT, R2_CLOCKING_BIT, R3_CLOCKING_BIT** - pozicije taktujućih bitova registara R_1, R_2 i R_3.

Registar se u programima čuva kao lista celih brojeva čiji elementi odgovaraju pojedinačnim bitovima registra. Najniža pozicija registra je skroz levo, tj. 2^0, dok je najviša pozicija skroz desno, tj. 2^{n-1}, gde je n dužina registra. Pri taktovanju, registar se pomera udesno, tj. od bitova najniže ka bitovima najviše težine, odnosno pozicije.

U implementacijama programa se radi efikasnijeg rada programa još koriste skupovi, rečnici i generatori. Generatori su posebna vrsta funkcija u Pythonu koje vraćaju „lenji" iterator. Generatori su korisni kada želimo da proizvedemo veliki niz vrednosti, ali ne želimo da ih sve uskladištimo u memoriju odjednom.

Funkcija get_taps

Različitim dužinama registara odgovaraju različite povratne sprege, kao i taktujući bitovi. Funkcija get_taps na osnovu zbira dužina registara određuje njihove povratne sprege, taktujuće bitove i dužine registara tako što upoređuje prosleđeni parametar sa globalnom promenljivom LEN_REG_TOTAL. Tako određeni parametri se upisuju u liste: r1_taps, r2_taps, r3_taps, clocking_bits i registers_lengths. Ako je n zbir dužina registara, odgovarajuće dužine, povratne sprege i taktujući bitovi registara su dati u tabeli 2 [17]. Pomenuti podaci za zbir dužina registara od 64 bita su dati u poglavlju 4. Prethodno opisane globalne promenljive se popunjavaju vrednostima iz lista dobijenih pozivom ove funkcije.

Funkcija get_majority

Funkcija get_majority implementira većinsku funkciju. Njeni argumenti su vrednosti tri bita koji se sabiraju. Ako zbir premašuje jedinicu, povratna vrednost funkcije je 1, što znači da većina bitova ima vrednost 1. Inače, većina bitova

n	dužine reg. R_1, R_2, R_3	taktujući bitovi registara R_1, R_2, R_3	povratna sprega registra R_1	povratna sprega registra R_2	povratna sprega registra R_3
18	5, 6, 7	1, 2, 2	0, 1, 2, 4	0, 5	0, 1, 3, 6
32	9, 10, 13	3, 4, 5	2, 4, 5, 8	2, 9	2, 4, 7, 12
40	10, 11, 19	4, 4, 8	1, 2, 7, 9	1, 10	2, 8, 9, 18
48	11, 18, 19	4, 8, 8	0, 7, 9, 10	6, 17	2, 8, 9, 18
56	17, 18, 21	7, 8, 9	3, 11, 15, 16	6, 17	2, 3, 8, 20

Tabela 2: Dužine, taktujući bitovi i povratne sprege za kraće verzije registara

ima vrednost 0, pa se ta vrednost vraća kao povratna vrednost funkcije. Na primer, ako su taktujući bitovi registara 1, 1 i 0 (ili permutacija ovih vrednosti, ili sve tri jedinice), zbir je veći od jedan, pa funkcija get_majority vraća vrednost 1, dok za taktujuće bitove 1, 0 i 0 (slično za permutaciju ovih vrednosti i sve tri nule), zbir je manji ili jednak jedan pa funkcija vraća vrednost 0.

Funkcija right_shift_reg

Funkcija koja pomera sadržaj prosleđenog registra za jednu poziciju udesno, tj. od bitova manje ka bitovima veće težine. U oslobođenu ćeliju sa indeksom nula se upisuje prosleđena vrednost sa povratne sprege tog registra. Bit u ćeliji sa indeksom $n-1$, gde je n dužina registra, se odbacuje. Ova funkcija se koristi pri taktovanju registra.

Funkcija irregular_clocking

Funkcija koja taktuje prosleđene registre korišćenjem većinske funkcije. Najpre se funkcijom get_majority odredi vrednost koju ima većina taktujućih bitova registara - većinski bit. Potom se taktujući bitovi svakog registra upoređuju sa većinskim bitom. Taktuju se oni registri čiji se taktujući bit poklapa sa većinskim bitom. Taktovanje se izvršava tako što se izračuna vrednost povratne sprege, koja se zatim prosleđuje funkciji right_shift_reg, zajedno sa registrom koji treba taktovati.

Program sadrži još dve funkcije: **list_to_string** i **string_to_list**. Pomoćna funkcija list_to_string listu prevodi u nisku tako što elemente liste prevede u tip string a zatim nadovezuje na praznu nisku. Pomoćna funkcija string_to_list vrši

obrnuto prevođenje, tj. nisku prevodi u listu tako što svaki karakter niske prevede u celobrojni tip, a zatim ga dodaje u praznu listu. Ove dve funkcije se koriste za prevođenje sadržaja registra iz liste u nisku i obratno (razlog je što u Pythonu lista ne može biti element skupa).

6.3 Program a51_keystream_generator

U ovom programu je realizovano generisanje niza bitova ključa za šifrovanje, odnosno dešifrovanje. On sadrži funkcije koje su opisane u nastavku.

Funkcija initialization_phase

Funkcija initialization_phase implementira inicijalizacionu fazu generatora ključa A5/1, koja je opisana u poglavlju 4. Najpre se sadržaji svih registara postave na nulu. Potom se, na osnovu prosleđenog ključa i 22-bitne vrednosti F, vrši postavljanje ključa, a onda i postavljanje inicijalizacionog vektora, kao što je opisano na strani 12. Na kraju se sadržaji registara spajaju u početno stanje, koje je povratna vrednost ove funkcije.

Funkcija get_keystream

Funkcija koja generiše 228 bitova niza ključa za šifrovanje, odnosno dešifrovanje. Prvo se za prosleđene argumente, ključ i 22-bitnu vrednost F, poziva funkcija initialization_phase, koja određuje početno stanje registara. Iz dobijenog početnog stanja se određuju sadržaji registara R_1, R_2 i R_3. Potom se izvršava faza praznog hoda u kojoj se, uz korišćenje većinske funkcije, registri taktuju 100 puta, nakon čega dospevaju u stanje S^W. Zatim sledi taktovanje registara 228 puta. U svakom koraku se primenom operacije XOR na najznačajnije bitove registara dobija po jedan bit niza ključa, koji se smešta u listu keystream. Dobijena lista, tj. niz bitova ključa je povratna vrednost ove funkcije.

Pored navedenih, program sadrži još dve funkcije koje se koriste za unos ključa i 22-bitnog broja okvira F - **user_input_key** i **user_input_frame_number**. Pomoću njih se proverava da li su navedene vrednosti ispravno unete. Proveravaju se njihove dužine, kao i da li sadrže samo vrednosti 0 i 1. Ove dve funkcije se pozivaju u glavnoj (main) funkciji kada se pokrene program a51_keystream_generator.

Nakon unošenja validnih traženih vrednosti ključa i broja okvira F, program poziva funkciju get_keystream. Dobijeni rezultat, tj. 228-bitni niz ključa, se ispisuje na standardni izlaz u obliku skraćenom na prvih n bitova, gde je n dužina ključa (key), odnosno zbir dužina registara. Skraćeni oblik dobijenog niza ključa je ulazna vrednost za program a51_attack.

6.4 Program a51_attack

U programu a51_attack je realizovan napad GNR na algoritam A5/1. Njegov ulaz je n bitova niza ključa, gde je $n \in \{18, 32, 40, 48, 56, 64\}$ zbir dužina registara. Izlazna vrednost programa je skup validnih kandidata za stanje registara S^W.

U nastavku su opisane funkcije koje ovaj program sadrži.

Funkcija fill_with_zeroes

Funkcija vrši dodavanje nula levo od sadržaja prosleđenog registra, tj. na njegove pozicije manjih težina. Registar ovde predstavlja listu koja sadrži registar R_2 nadovezan na sadržaj registra R_1. Rezultujući registar sa dodatim nulama ima dužinu zbira dužina registara R_1 i R_2. Ovo je pomoćna funkcija koju koristi funkcija all_r1_r2 (opisana sledeća), kako bi sve mogućnosti registara R_1 i R_2 bile iste dužine. Na primer: u slučaju zbira dužina svih registara 18 bitova, registar [1, 1] se proširuje do registra [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1] dužine 11, što je zbir dužina registara R_1 i R_2 u ovom slučaju.

Funkcija all_r1_r2

Funkcija koja određuje sva moguća stanja registara R_1 i R_2. Stanja se dodaju u generator. Elementi rezultujućeg generatora su niske koje sadrže registar R_2 nadovezan na sadržaj registra R_1. Obuhvaćene su sve mogućnosti iz intervala $[0, 2^{p+q})$, gde su p i q, redom, dužine registara R_1 i R_2. Na primer, u slučaju kada je zbir dužina svih registara 18, prvi element skupa je 00000000000, drugi element je 00000000001, treći element je 00000000010, i tako se redom obuhvate sve mogućnosti iz intervala $[0, 2^{11})$.

Funkcija clock_registers

Funkcija koja vrši taktovanje registara na sličan način kao funkcija irregular_clocking, opisana u 6.2. Razlika između ove dve funkcije je u tome što funkcija clock_registers taktuje registre R_1 i/ili R_2, dok se registar R_3 ne taktuje, jer njegov sadržaj nije u potpunosti određen. Stoga se, kao povratna vrednost, samo prosleđuje informacija da li u tekućem pozivu ove funkcije registar R_3 treba ili ne treba taktovati. Ukoliko registar R_3 treba da se taktuje, vraća se 1, a inače se vraća 0.

Funkcija check_state_candidate

Funkcija koja implementira drugu fazu napada [14], odnosno proverava konzistentnost određenog kandidata za stanje registara S^W. Najpre se iz prosleđenog stanja dobiju sadržaji registara R_1, R_2 i R_3. U petlji se, primenom operacije XOR na najznačajnije bitove registara, generiše bit niza ključa i upoređuje sa bitom datog niza ključa. Poznati niz ključa se prosleđuje ovoj funkciji kao argument. Ukoliko se generisani i dati bit ne poklapaju, proces se završava i funkcija vraća vrednost 0, što označava da prosleđeno stanje registara nije validno. U suprotnom se taktuju odgovarajući registri, a potom se izvršava sledeći korak petlje. Ukoliko se poklope svi bitovi, funkcija vraća vrednost 1, tj. prosleđeno stanje registara je validno. Ova funkcija je deo generatora niza ključa A5/1 u kojem se dobijaju bitovi niza ključa, pri čemu se umesto 228 bitova generiše i proverava najviše n bitova, gde je n dužina datog niza ključa.

Funkcija get_valid_states

Funkcija get_valid_states određuje skup kandidata za stanje registara S^W, odnosno skup validnih stanja, na osnovu prosleđenog niza bitova ključa. U petlji se najpre praznim skupom deklariše skup r3_candidates koji predstavlja skup svih kandidata za registar R_3 i praznim rečnikom se deklariše valid_states koji predstavlja rečnik čiji ključ je određeno validno stanje registara S^W, a vrednost vreme njegovog pronalaska. Zatim se fiksiraju stanja registara R_1 i R_2 dobijena pozivom funkcije all_r1_r2 (pre početka petlje). Registar R_3 se popuni simbolom *. Nakon toga sledi poziv rekurzivne funkcije determine_r3 (koja je opisana sledeća) za registre R_1, R_2 i R_3, niz bitova ključa, indeks i bita niza ključa KS, indeks b bita registra R_3, broj taktova t registra R_3, skup kandidata r3_candidates, kao i vrednost 0 koja govori o tome da u prvom pozivu funkcije nije došlo do kontradikcije

(detaljnije objašnjeno u opisu pomenute funkcije).

Nakon toga se, na osnovu fiksiranih registara R_1 i R_2 i dobijenih kandidata za registar R_3, u generator state_candidates smeštaju svi kandidati za stanje S^W, tj. nadovezani registri R_1, R_2 i R_3. Zatim se popunjava rečnik valid_states validnim stanjima generatora uz vreme njihovog pronalaska. Validnost stanja se utvrđuje pozivom funkcije check_state_candidate prilikom kreiranja ovog rečnika. Nakon toga se sva validna stanja upisuju u fajl valid_states.txt i ispisuju na standardni izlaz.

Funkcija determine_r3

Funkcija determine_r3 rekurzivno određuje sadržaj registra R_3 za prosleđena fiksirana stanja registara R_1 i R_2. Prosleđeni parametar i je pozicija bita ključa. Parametar b je pozicija bita registra R_3 koji treba da se izračuna sledeći. Parametar t predstavlja broj taktova registra R_3. Parametar r3_candidates je referenca na skup u kome će se čuvati dobijeni kandidati za registar R_3. Parametar k govori o tome da li je prethodnim rekurzivnim pozivom došlo do kontradikcije. Može imati vrednost 0 ili 1. Ako je prosleđena vrednost $k = 1$, izlazi se iz rekurzije. U suprotnom se nastavlja sa izvršavanjem rekurzivnog poziva.

Funkcija razlikuje dva slučaja:

- U prvom slučaju se proverava da li se registar R_3 taktovao $\lfloor \frac{m}{2} \rfloor$ puta, gde je m dužina registra R_3. Tada je potrebno izračunati poslednji bit registra R_3, nakon čega je njegov sadržaj u potpunosti određen i dodaje se u skup results, a potom se izlazi iz rekurzije.

- U drugom slučaju registar R_3 još uvek nije taktovan $\lfloor \frac{m}{2} \rfloor$ puta. Najpre se računa najznačajniji bit registra R_3 na osnovu bita ključa na poziciji i i najznačajnijih bitova registara R_1 i R_2. Vrednost taktujućeg bita registra R_3 se postavlja na 0, a u njegovoj kopiji R_3^1 se postavlja na 1. Zbog uvođenja kopije registra R_3, koriste se pomoćne promenljive (za oba registra R_3 i R_3^1 po jedna): t_1 i t_2 za brojanje taktova registara, i_1 i i_2 za pozicije bitova niza ključa KS i k_1 i k_2 za obaveštavanje da je došlo do kontradikcije. Sledeći postupak se potom izvršava za kopije R_3 i R_3^1:

  - Taktuju se odgovarajući registri funkcijom clock_registers kojoj se, pored registara R_1 i R_2, kao treći argument prosleđuje samo taktujući bit registra R_3 (odnosno R_3^1) (jer se on ovom funkcijom ne taktuje, kao što je već rečeno).

– U slučaju da registar R_3 (odnosno R_3^1) ne treba da bude taktovan, primenom operacije XOR na najznačajnije bitove registara (u registru R_3 (odnosno R_3^1) je to poslednji izračunati najznačajniji bit) generiše se bit niza ključa. Generisani bit se zatim upoređuje sa tekućim bitom poznatog niza ključa. Ako se poklapaju, indeks i (tačnije i_1 ili i_2, u zavisnosti od toga da li se petlja izvršava za registar R_3 ili kopiju R_3^1) niza ključa se uvećava, tj. pomera se na sledeću poziciju, što znači da poslednji izračunat najznačajniji bit registra R_3 (odnosno R_3^1) daje više od jednog bita niza ključa. Uz pomeranje indeksa i, vrši se i taktovanje registara. U slučaju nepoklapanja generisanog bita sa bitom niza ključa, došlo je do kontradikcije i promenljiva k, odnosno k_1 ili k_2, se postavlja na vrednost 1, nakon čega se izlazi iz petlje. Ako se nije desila kontradikcija, opisani postupak računanja, upoređivanja i taktovanja se ponavlja sve dok se ne dobije informacija (povratnom vrednošću funkcije clock_registers) da registar R_3 (odnosno R_3^1) treba da bude taktovan ili dok ne dođe do kontradikcije.

Kada se registar R_3 (odnosno R_3^1) uspešno taktuje, proverava se da li je k, odnosno k_1 ili k_2, različito od 1, tj. da li je došlo do kontradikcije. Ako nije, uvećava se promenljiva t, odnosno t_1 ili t_2, tj. njegov broj taktova. Nakon toga slede rekurzivni pozivi ove funkcije za obe kopije R_3 i R_3^1.

Pored opisanih funkcija, ovaj program sadrži još i funkciju user_input_keystream. Ova funkcija proverava da li je niz bitova ključa ispravno unet, kao i da li je odgovarajuće dužine i da li sadrži samo vrednosti 0 i 1. U programu je sadržana i jedna globalna promenljiva **R3_MAX_CLOCKING** koja sadrži koliko maksimalno puta registar R_3 treba da bude taktovan, tako da su u registru R_3 poznati svi bitovi sem bita na sredini registra. Srednji bit se poslednji računa, kako je opisano u prvom slučaju funkcije determine_r3.

Kada se pokrene program a51_attack, poziva se glavna (main) funkcija. Najpre se pozivom funkcije user_input_keystream unese validan niz bitova ključa, koji se prosleđuje (preveden u listu) funkciji get_valid_states, koja zatim ispisuje sva validna stanja na standardni izlaz.

6.5 Program initial_state_reversion

Program initial_state_reversion izvršava „premotavanje" unazad stanja registara S^W do početnog stanja S^I, tj. taktovanje unazad stanja S^W 100 puta. Taktovanje stanja predstavlja taktovanje registara R_1, R_2 i R_3, jer se stanje dobija od nadovezanih sadržaja ova tri registra. Podsećanja radi, generator niza bitova ključa A5/1 najpre izvršava fazu inicijalizacije nakon koje se registri nalaze u početnom stanju S^I. Zatim se izvršava faza praznog hoda, koja obuhvata taktovanje registara 100 puta, nakon koje se registri nalaze u stanju S^W. Posle toga se generiše 228-bitni niz ključa. U procesu napada GNR opisanog u tački 5.2, polazi se od poznatog dela niza bitova ključa i određuje se stanje registara S^W. Postupak implementiran u ovom programu jeste transformacija (potencijalno nejednoznačna) stanja S^W u početno stanje S^I. Ovaj postupak nije opisan u radu [14].

U nastavku je prikazan sadržaj programa initial_state_reversion.

Klasa Node

Klasa Node predstavlja čvor stabla varijanti taktovanja registara R_1, R_2 i R_3 na svaki mogući način. On sadrži podatak state, koji predstavlja stanje registara i tipa je string, i listu children, koja predstavlja listu čvorova koji polaze od tekućeg čvora, koja je inicijalno prazna dok se u nju ne dodaju novi čvorovi. Uz ovu klasu implementirana je i pomoćna funkcija **newNode** koja pravi novi čvor na osnovu prosleđene vrednosti podatka state.

Funkcija get_irregular_taps

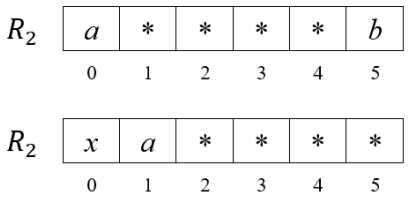
Funkcija get_irregular_taps na osnovu zbira dužina registara određuje njihove povratne sprege za taktovanje unazad. Prosleđeni parametar se upoređuje sa globalnom promenljivom LEN_REG_TOTAL (definisanu u programu a51_utils). Na taj način se, na osnovu tabele 3, određuju povratne sprege registara koje se, zatim, upisuju u liste r1_taps, r2_taps i r3_taps.

Podaci prikazani u tabeli su dobijeni na sledeći način. Posmatrajmo, zbog jednostavnosti povratne sprege, taktovanje registra R_2, u slučaju zbira dužina registara od 18 bitova. Bitovi njegove povratne sprege su na pozicijama 0 i 5. Označimo ih sa $a = R_2[0]$ i $b = R_2[5]$. Tada je rezultujući bit povratne sprege $x = a \oplus b$. Pri pomeranju registra za jednu poziciju udesno, bitovi na pozicijama 0 do 4 se pomeraju u ćelije na pozicijama 1 do 5, pa bit b biva izbačen iz registra. U oslobođenu

n	povratna sprega registra R_1	povratna sprega registra R_2	povratna sprega registra R_3
18	0, 1, 2, 3	0, 1	0, 1, 2, 4
32	0, 3, 5, 6	0, 3	0, 3, 5, 8
40	0, 2, 3, 8	0, 2	0, 3, 9, 10
48	0, 1, 8, 10	0, 7	0, 3, 9, 10
56	0, 4, 12, 16	0, 7	0, 3, 4, 9
64	0, 14, 17, 18	0, 21	0, 8, 21, 22

Tabela 3: Povratne sprege registara za taktovanje unazad

ćeliju na poziciji 0 se upisuje bit x (videti sliku 10).



Slika 10: Pomeranje registra R_2, ako je dužina stanja 18 bitova

Ako se sada dobijeni registar R_2 pomeri ulevo, treba da se dobije registar od kojeg se krenulo. Pomeranjem registra ulevo se bitovi na pozicijama 1 do 5 pomeraju na pozicije 0 do 4, pa bit na poziciji 0 biva izbačen iz registra. U oslobođenu ćeliju se upisuje bit koji se dobija povratnom spregom za taktovanje unazad, što bi trebalo da bude baš bit b. Pomenuta jednakost iz povratne sprege registra R_2

$$a \oplus b = x$$

može se (sleva) XOR-ovati sa a tako da se dobije

$$a \oplus (a \oplus b) = a \oplus x.$$

Operacija XOR je asocijativna, pa je prethodna jednakost ekvivalentna sa

$$(a \oplus a) \oplus b = a \oplus x.$$

Primenivši $a \oplus a = 0$ i $0 \oplus b = b$ na prethodnu jednakost, dobijamo

$$b = a \oplus x, \text{ odnosno } R_2[5] = R_2[0] \oplus R_2[1].$$

Odavde sledi da su bitovi povratne sprege za taktovanje unazad registra R_2 na pozicijama 0 i 1. Na sličan način se mogu dobiti i ostale vrednosti prikazane u tabeli 3.

Funkcija left_shift_reg

Funkcija left_shift_reg pomera sadržaj prosleđenog registra za jednu poziciju ulevo, tj. od bitova veće ka bitovima manje težine. Bit u ćeliji sa indeksom 0 se odbacuje, dok se u oslobođenu ćeliju najznačajnijeg bita upisuje prosleđena vrednost sa povratne sprege tog registra pri taktovanju unazad.

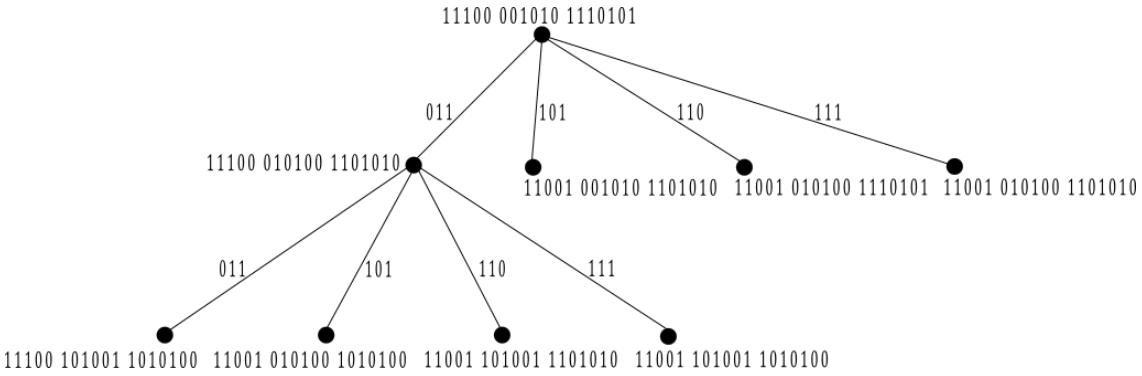
Funkcija get_tree_leaf_nodes

Funkcija get_tree_leaf_nodes konstruiše stablo pretrage, čiji koren je stanje registara S^W (videti sliku 11). Od korena, tj. od svakog čvora u stablu, polaze četiri grane koje odgovaraju mogućnostima taktovanja registara, odnosno opisuju taktovanje unazad stanja u čvoru od kog polaze. Grane su:

- 011, što označava da su taktovani registri R_2 i R_3;
- 101, što označava da su taktovani registri R_1 i R_3;
- 110, što označava da su taktovani registri R_1 i R_2;
- 111, što označava da su taktovana sva tri registra.

Druge mogućnosti ne postoje, jer se u svakom koraku taktuju bar dva registra (što označavaju jedinice u prikazanim trojkama). Svaki čvor stabla sadrži stanje dobijeno taktovanjem unazad stanja u njegovom roditeljskom čvoru, na osnovu grane koja ih povezuje. U listovima se nalaze moguća početna stanja registara S^I, nakon „premotavanja" stanja S^W za 100 taktova unazad. Dakle, stanje u korenu je stanje nakon 100 taktova, tj. 101. stanje, što je upravo S^W. Ukupna visina konstruisanog stabla je 100.

Primetimo da taktovanje unazad opisano granom koja vodi do čvora u koje se smešta novo stanje registara ne mora da se poklapa sa taktovanjem unapred novog stanja. Na slici 11 na visini 1 datog stabla se kod svih osim prvog čvora javlja kontradikcija, jer taktujući bitovi registara stanja koja oni sadrže opisuju taktovanja različita od onih koje opisuju grane koje do njih vode. Konkretno, u drugom čvoru taktujući bitovi su 1, 1 i 0 (njihove pozicije su 1, 7 i 13 u 18-bitnom stanju, odnosno za registar R_1 bit na poziciji 1, za registar R_2 bit na



Slika 11: Primer stabla pretrage do dubine 2, ako je zbir dužina registara 18 bitova

poziciji 2 i za registar R_3 bit na poziciji 2), što označava da registri R_1 i R_2 treba da se taktuju, a ne R_1 i R_3, kako je opisano granom 101. Na osnovu ovoga, funkcija get_tree_leaf_nodes istovremeno redukuje opisano stablo pretrage i time smanjuje broj mogućih početnih stanja koja ostaju u listovima.

Argumenti funkcije get_tree_leaf_nodes su koren stabla, nivo h, odnosno visina stabla na kojoj se dodaju čvorovi u tekućem rekurzivnom pozivu funkcije, parametar match koji predstavlja povratnu vrednost funkcije branch_clocking_comparison iz prethodnog rekurzivnog poziva i referenca na skup leaf_nodes (na početku prazan) u koji se smeštaju listovi stabla, tj. određena početna stanja S^I.

Ovo je rekurzivna funkcija, pri čemu se iz rekurzije izlazi u slučaju kada parametar match ima vrednost $False$ ili kada se dostigne list, odnosno kada se nivo h izjednači sa 100. Vrednost $False$ parametra match označava nepoklapanje taktovanja unazad opisanog granom kojom se došlo do tekućeg čvora sa taktovanjem unapred stanja tekućeg čvora. U tom slučaju se tim čvorom ne nastavlja konstrukcija stabla. Ukoliko je dostignuta visina $h = 100$, odnosno list, njegovo stanje se dodaje u skup leaf_nodes. Ako uslovi za izlazak iz rekurzije nisu ispunjeni, stanje tekućeg čvora se podeli na registre R_1, R_2 i R_3. Zatim se (implicitno) formira grana 011. Najpre se taktuju unazad registri R_2 i R_3. Potom se poziva funkcija branch_clocking_comparison sa argumentima grane stabla u vidu liste [0, 1, 1] i taktujućih bitova svakog od registara R_1, R_2 i R_3. Povratna vrednost funkcije se smešta u promenljivu match koja se kasnije prosleđuje novim rekurzivnim pozivom. Potom se napravi novi čvor od stanja koje se dobija nadovezivanjem sadržaja registara jedan na drugi, pa se doda u listu children tekućeg čvora. Nakon toga se funkcija poziva rekurzivno za napravljeni čvor. Postupak je isti i za ostale tri grane, osim što se različiti registri taktuju unazad i različite grane se prosleđuju

funkciji branch_clocking_comparison.

Funkcija branch_clocking_comparison

Funkcija upoređuje taktovanje unazad stanja opisano granom iz čvora gore-pomenutog stabla sa taktovanjem unapred stanja određenim na osnovu taktujućih bitova registara. Najpre se za prosleđene taktujuće bitove poziva funkcija get_majority kojom se odredi većinski bit. Zatim se formira taktovanje registara u tročlanoj listi koja se inicijalizuje nulama tako što se taktujući bitovi registara upoređuju sa dobijenim većinskim bitom i, u slučaju poklapanja se na odgovarajuću poziciju u listi upisuje vrednost 1. Upisana jedinica označava da registar koji odgovara poziciji jedinice treba da se taktuje. Isto značenje ima i jedinica upisana u prosleđenu granu u obliku liste, kao što je i opisano funkcijom get_tree_leaf_nodes. Ako se novodobijena lista poklapa sa datom granom, funkcija vraća vrednost True, dok u suprotnom vraća vrednost False.

Na primer, ako su taktujući bitovi registara R_1, R_2 i R_3 redom 1, 0 i 0, onda je majority$(1, 0, 0) = 0$. Taktujući bitovi registara R_2 i R_3 se poklapaju sa većinskim bitom i oni treba da se taktuju. Isto je zapisano i granom [0, 1, 1], pa je povratna vrednost ove funkcije True.

Funkcija get_initial_states

Funkcija get_initial_states je glavna funkcija ovog programa (kao main funkcija). Prvo se iz fajla valid_states.txt učitaju validna stanja registara određena programom a51_attack. Zatim se obilazi skup validnih stanja. Od tekućeg stanja se napravi čvor i za njega se poziva funkcija get_tree_leaf_nodes. Ovim pozivom funkcije se u pomoćni skup inicijalizovan praznim skupom se smeštaju listovi stabla, tj. određena početna stanja registara. Na kraju se za svako validno stanje ispisuju dobijena početna stanja registara.

6.6 Upotreba programa

U ovoj tački je prikazano korišćenje programa za generisanje niza ključa za šifrovanje, odnosno dešifrovanje, napada na algoritam A5/1 i određivanje početnog stanja registara.

Na računaru na kojem se pokreće program, mora biti instaliran Python verzije 3.9.0. Uputstva koja slede namenjena su pokretanju programa na operativnom si-

stemu Windows. Sva tri programa, a51_keystream_generator.py, a51_attack.py i initial_state_reversion.py, se pokreću dvoklikom (uz eventualno podešavanje da podrazumevani pokretač bude Python) na odgovarajući program. Program je moguće pokrenuti i na bilo koji drugi način poznat čitaocu (na primer Python IDLE okruženje ili uz pomoć nekog editora). Za izlazak iz programa unosi se tekst *exit*. Takođe, pre pokretanja programa se može podesiti veličina ulaza, odnosno zbir dužina registara R_1, R_2 i R_3, izmenom vrednosti promenljive LEN_REG_TOTAL u a51_utils.py.

6.6.1 Upotreba programa a51_keystream_generator

Pri pokretanju programa, unose se ključ (key), a zatim i 22-bitni broj okvira F. Nakon toga se ispisuje početno stanje registara i niz bitova ključa u obliku skraćenom na prvih $n \in \{18, 32, 40, 48, 56, 64\}$ bitova. Na slici 12 prikazan je primer rada ovog programa za $n = 18$ bitova.

```
Unesite 18-bitni kljuc K: 100010001000100011
Unesite 22-bitni broj okvira F: 0111010000101110100010

Pocetno stanje: 100011110101001100
Stanje nakon faze praznog hoda: 000011100011000011
Prvih 18 bitova niza kljuca: 110000000001100011
```

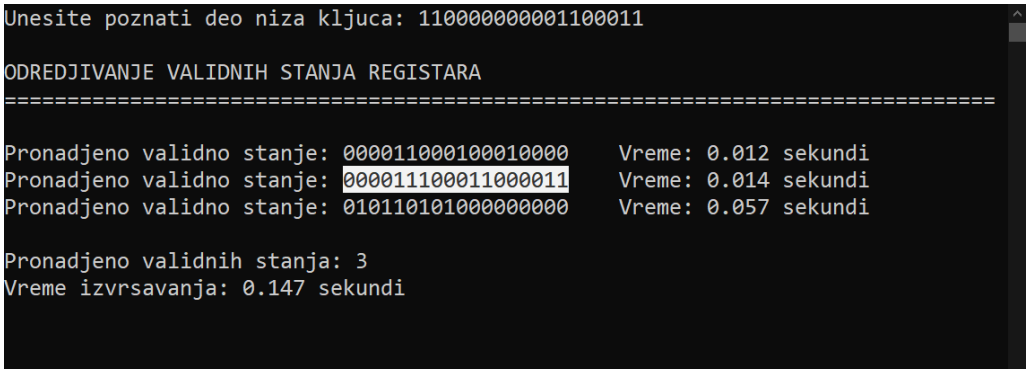
Slika 12: Rad generatora ključa A5/1

6.6.2 Upotreba programa a51_attack

Nakon pokretanja programa, unosi se $n \in \{18, 32, 40, 48, 56, 64\}$ bitova niza ključa. Program ispisuje sva dobijena stanja registara S^W, tj. stanja nakon faze praznog hoda generatora ključa A5/1. Na slici 13 prikazan je primer rada ovog programa za $n = 18$ bitova, koji se nadovezuje na prethodni primer generisanja niza ključa.

6.6.3 Upotreba programa initial_state_reversion

Ovaj program je implementiran tako da nema korisničkog unosa, već se ulazni parametri učitavaju iz fajla valid_states.txt. Ovaj fajl sadrži rezultujuće vrednosti programa a51_attack.py, tj. sva moguća stanja registara S^W. Nakon pokretanja programa initial_state_reversion, za svako stanje iz fajla se ispisuju sva moguća



Slika 13: Program koji izvršava napad na A5/1

početna stanja registara S^I. Ovim programom se mogu eliminisati kandidati za stanje registara S^W ukoliko se njihovim „premotavanjem" unazad ne dobije nijedno početno stanje registara S^I. Na slici 14 prikazan je primer rada ovog programa za $n = 18$ bitova, koji se nadovezuje na prethodni primer napada. U ovom primeru se eliminišu dva moguća stanja registara S^W, a preostalo stanje je stanje dobijeno u primeru na slici 12.

6.7 Dobijeni rezultati

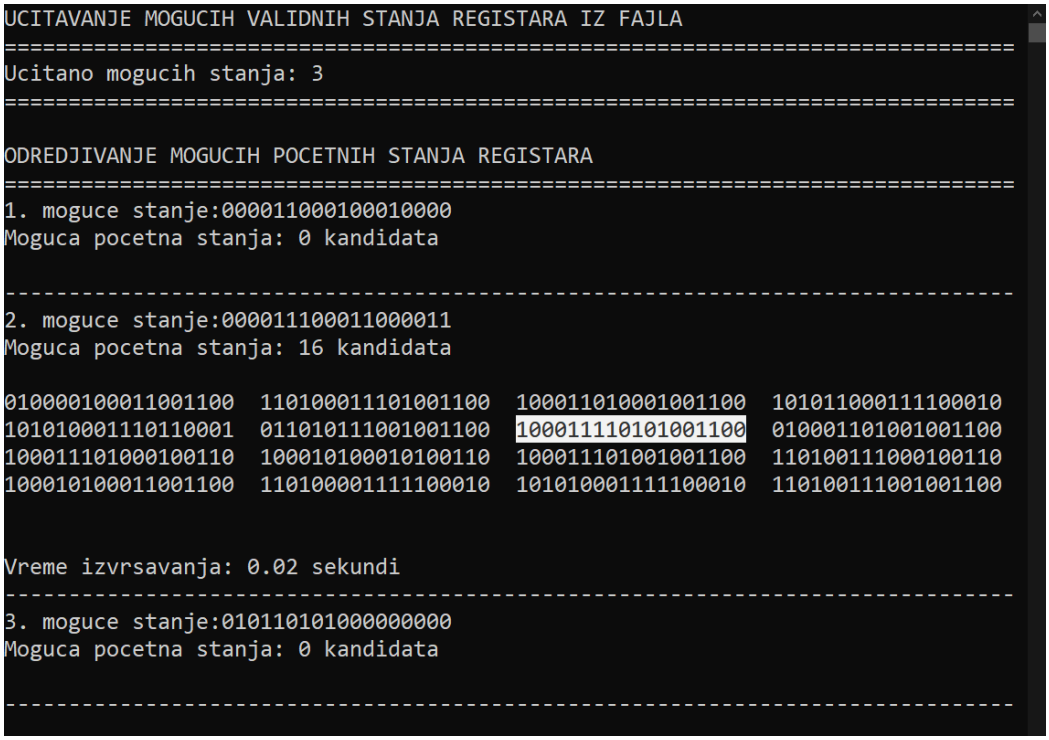
U ovoj tački su prikazani dobijeni rezultati testiranja prethodno opisanih programa.

Programi a51_keystream_generator i initial_state_reversion se izvršavaju za manje od jedne sekunde za bilo koje $n \in \{18, 32, 48, 56, 64\}$, gde je n veličina ulaza, odnosno zbir dužina registara. Vreme izvršavanje programa a51_attack zavisi od veličine ulaza i prikazano je u tabeli 4.

veličina ulaza n	prosečno vreme izvršavanja $t(n)$
18	< 0.2 sekunde
32	3 minuta i 45 sekundi
40	1 sat i 40 minuta

Tabela 4: Vreme izvršavanja programa a51_attack u zavisnosti od veličine ulaza

Za veće ulaze $n \in \{48, 56, 64\}$ izvršavanje programa traje dosta duže. U ovim slučajevima program je testiran za nekoliko fiksiranih vrednosti parova registara

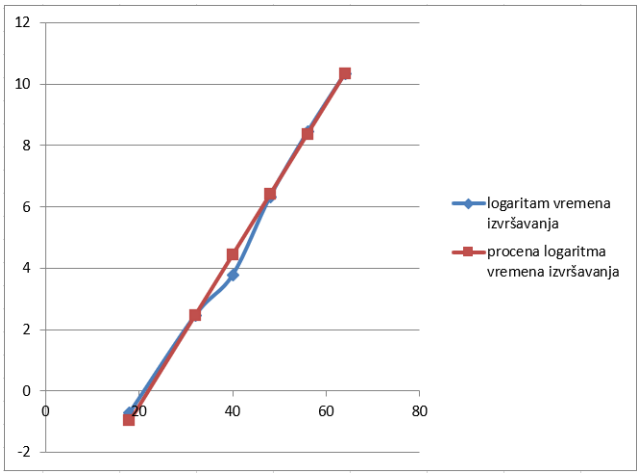


Slika 14: Program koji određuje sva moguća početna stanja registara

R_1 i R_2, umesto za sve moguće, zbog vremena trajanja izvršavanja programa. Dobijene su vrednosti:

- Za $n = 48$ bitova prosečno vreme izvršavanja za jedan par registara R_1 i R_2 iznosi približno 0.004 sekunde, odnosno oko 25 dana za svih 2^{29} (dužina registra R_1 je u ovom slučaju 11, a registra R_2 18) mogućnosti za registre R_1 i R_2;
- Za $n = 56$ bitova prosečno vreme izvršavanja za jedan par registara R_1 i R_2 iznosi približno 0.008 sekundi, odnosno oko 8,5 godina za svih 2^{35} (dužina registra R_1 je u ovom slučaju 17, a registra R_2 18) mogućnosti za registre R_1 i R_2;
- Za $n = 64$ bitova prosečno vreme izvršavanja za jedan par registara R_1 i R_2 iznosi približno 0.01 sekundi, odnosno nekoliko stotina godina za svih 2^{41} (dužina registra R_1 je u ovom slučaju 19, a registra R_2 22) mogućnosti za registre R_1 i R_2.

Na slici 15 je prikazan grafik zavisnosti logaritma vremena izvršavanja programa a51_attack od veličine ulaza n, kao i grafik procene logaritma vremena izvršavanja. Složenost vremena izvršavanja programa je eksponencijalna i odgovara joj funkcija $t(n) = 2.302 \cdot 10^{-6} \cdot 1.777^n$ (1).



Slika 15: Grafik zavisnosti logaritma vremena izvršavanja programa a51_attack i njegove procene od veličine ulaza

Procena je dobijena na sledeći način. Ako je n veličina ulaza, a $t(n)$ vreme izvršavanja programa za n-bitni ulaz, potrebno je odrediti funkciju koja aproksimira funkciju $t(n)$. Sve tačke sa grafika logaritma vremena izvršavanja su oblika $X(n, \log(t(n)))$. Uzete su dve tačke takve tačke $A(n_1, \log(t(n_1))) = A(32, 2.352182518)$ i $B(n_2, \log(t(n_2))) = B(64, 10.34222982)$. Kroz njih se provlači prava čiji je nagib, odnosno koeficijent pravca jednak $k = \frac{\log(t(n_2)) - \log(t(n_1))}{n_2 - n_1} = 0.249688978$. Koristeći jednačinu prave kroz dve tačke dobija se

$$\log(t(n)) = \log(t(n_1)) + k \cdot (n - n_1)$$

odnosno, kada se zamene odgovarajuće vrednosti

$$\log(t(n)) = 2.352182518 + 0.249688978 \cdot (n - 32).$$

Sada se obe strane podignu u izložilac osnove logaritma, pa se dobija

$$t(n) = 10^{2.352182518 + 0.249688978 \cdot (n - 32)}$$

odnosno

$$t(n) = 10^{2.352182518 - 0.249688978 \cdot 32} \cdot 10^{0.249688978 \cdot n}.$$

Sređivanjem izraza se dobija (zaokrugljeno na tri decimale)

$$t(n) = 2.302 \cdot 10^{-6} \cdot 1.777^n$$

što je upravo prikazana jednačina (1).

Što se tiče broja dobijenih rešenja, programi a51_attack i initial_state_reversion daju po nekoliko različitih rešenja u većini slučajeva, među kojima se uvek javi i ono rešenje od koga se krenulo, tj. koje je prethodno određeno programom a51_keystream_generator. Pri tome program initial_state_reversion često eliminiše neka rešenja dobijena programom a51_attack, jer ne daju nijednu mogućnost za početno stanje registara.

U opisanom radu [14] korišćen je uređaj specijalne namene koji se zove COPACOBANA (engl. Cost-Optimized Parallel Code Breaker). To je uređaj visokih performansi optimizovan za probijanje šifri. Sastoji se od 120 rekonfigurabilnih integrisanih kola koja rade paralelno. Zahvaljujući ovom uređaju, potrebno je u proseku 7h (odnosno 14h u najgorem slučaju) da se odrede sva validna stanja registara S^W. Dakle, jasno je da bi ovakav uređaj optimizovao realizovani napad.

7 Zaključak

U ovom radu je opisan i realizovan napad [14] na algoritam A5/1 za koji je dovoljno poznavanje 64 bita niza ključa kako bi se odredilo stanje registara nakon faze praznog hoda, ali i početno stanje registara nakon faze inicijalizacije generatora A5/1. Nažalost, bez posedovanja mašine specijalizovane za ovakav napad, program opisan u 6.4 je u potpunosti testiran samo za kraće verzije registara, tj. za kraće nizove ključa (dužina 18, 32 i 40 bitova). Međutim, programi a51_attack i initial_state_reversion su za svaki uneti niz ključa generisan programom a51_keystream_generator davali skupove rešenja koji su uvek sadržali i ona stanja (početno stanje i stanje nakon faze praznog hoda) prethodno dobijena ovim programom.

U poglavljima 3 i 4 je opisano kako se algoritmom A3 vrši autentifikacija korisnika, algoritmom A8 generiše ključ koji algoritam A5/1 koristi za generisanje ključa za šifrovanje i dešifrovanje, a zatim kako funkcioniše generator ključa A5/1. Kao što je već pomenuto, A5/1 šifruje i dešifruje podatke koji se prenose komunikacijom u 2G mreži i koristi se i danas. Naravno, kao i svaka druga tehnologija, GSM-mreža se kontinuirano razvija i danas se u velikoj meri koriste optimizovanije i sigurnije mreže koje koriste druge sisteme za šifrovanje podataka. Kriptoanaliza algoritma A5/1 opisana u ovom radu, kao i drugi izvedeni napadi na ovaj algoritam, iako su oslabili sigurnost 2G mreže, takođe su i doprineli poboljšanju algoritama za šifrovanje i same bezbednosti komunikacije u mobilnim mrežama današnjice.

Literatura

[1] ETSI Official Website: www.etsi.org

[2] The GSM Association Official Website: www.gsma.com

[3] Barkan, E., Biham, E., Keller, N. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. In: Journal of Cryptology (2008) 21: 392–429

[4] Jensen, O. D., Andersen, K. A. A5 Encryption In GSM. (2017)

[5] Biryukov, A., Shamir, A., Wagner, D. (2001) Real Time Cryptanalysis of A5/1 on a PC. In: Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B. (eds) Fast Software Encryption. FSE 2000. Lecture Notes in Computer Science, vol 1978. Springer

[6] Kessler, G. An Overview of Cryptography. (1998)

[7] Živković, M. Kriptografija (skripta). (2022) Matematički fakultet Univerziteta u Beogradu.

[8] Tanenbaum, A., Wetherall, D. Computer Networks, Fifth Edition (2011), 162 - 166

[9] Hofer, M. Kriptoanaliza algoritma A5/2. Matematički fakultet Univerziteta u Beogradu, 2016.

[10] Stanojević, K. Kriptoanaliza šifre KASUMI. Matematički fakultet Univerziteta u Beogradu, 2022.

[11] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security; Specification of the A5/4 Encryption Algorithms for GSM and ECSD, and the GEA4 Encryption Algorithm for GPRS (Release 9). 3GPP TS 55.226 V9.0.0 (2009-09)

[12] Scourias, J. Overview of GSM: The Global System for Mobile Communications. University of Waterloo, 1996

[13] Srinivas, Suraj. The GSM Standard (An overview of its security). SANS Institute, 2021.

[14] Gendrullis, T., Novotný, M., Rupp, A., A Real-World Attack Breaking A5/1 within Hours, In: Oswald, E., Rohatgi, P. (eds) Cryptographic Hardware and Embedded Systems – CHES 2008. CHES 2008. Lecture Notes in Computer Science, vol 5154. Springer

[15] Aumasson, Jean-Philippe. Serious cryptography. (2018) Section 5: Stream Ciphers.

[16] Keller, J., Seitz, B. (2001) A Hardware-Based Attack on the A5/1 Stream Cipher. ITG FACHBERICHT, 2001 - VDE; 1999

[17] Živković, M. A Table of Primitive Binary Polynomials. Math. Comp. 62 (1994), 385-386