Novi kriptografski algoritam baziran na Feistel strukturi

Teodora Perić
Jagodina, Srbija
perict.teodora@gmail.com

Žarko Stanisavljević
Katedra za računarsku tehniku i informatiku
Elektrotehnički fakultet, Univerzitet u Beogradu
Beograd, Srbija
zarko.stanisavljevic@etf.bg.ac.rs
ORCID: 0000-0003-0272-5139

Adrian Milaković
Katedra za računarsku tehniku i informatiku
Elektrotehnički fakultet, Univerzitet u Beogradu
Beograd, Srbija
adrian.milakovic@etf.bg.ac.rs
ORCID: 0000-0002-3005-3352

Apstrakt **- Ovaj rad daje predlog novog kriptografskog simetričnog blok algoritma zasnovanog na Feistel strukturi, sa ciljem da poboljša razumevanje funkcionisanja algoritama baziranih na Feistel strukturi, kao i važnosti odabira operacija, parametara i konstanti prilikom dizajna novog algoritma. Identifikacija ključnih karakteristika algoritma obavljena je kroz detaljnu analizu postojećih algoritama zasnovanih na Feistel strukturi. Pored dizajna algoritma, u radu je opisan i implementirani edukativni vizuelni simulator koji olakšava učenje algoritma prikazujući proces šifrovanja i dešifrovanja proizvoljne poruke korak po korak, čime značajno doprinosi ne samo razumevanju implementiranog algoritma, već i same Feistel strukture na kojoj je algoritam zasnovan. Sproveden je i niz testova koji procenjuju efikasnost, bezbednost i praktičnost osmišljenog algoritma, koristeći poznate metode koje se primenjuju u analizi novih algoritama.**

Ključne reči—Vizuelni simulator, dizajn algoritma, Feistel struktura, napadi grubom silom, efekat lavine.

I. UVOD

U periodu razvoja digitalne komunikacije koja obuhvata razmenu poverljivih i privatnih podataka putem interneta, od krucijalnog značaja je zaštititi osetljive informacije. Danas se u tu svrhu koriste različite kriptografske tehnike kako bi se podaci od značaja transformisali u nečitljiv oblik, obezbeđujući njihovu sigurnost u prenosu. U domenu moderne kriptografije, Feistel struktura predstavlja važnu šemu na kojoj se zasnivaju neki od najpoznatijih simetričnih blok algoritama.

Prilikom prvog kontakta sa oblašću kriptografije, veliki broj postojećih algoritama i šema mogu delovati veoma kompleksno. Razumevanje strukture algoritama i značaja mnogobrojnih operacija iz kojih se algoritam sastoji može biti podjednako izazovno. Cilj ovog rada je višestruk: razjasniti teorijske osnove algoritama baziranih na Feistel strukturi [1], opisati proces dizajna i dati predlog novog algoritma koji bi se koristio u edukativne svrhe korišćenjem implementiranog vizuelnog simulatora i proceniti sigurnost i performanse algoritma koristeći definisan skup testova.

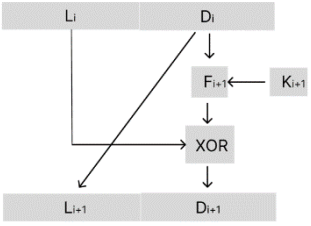
U drugom poglavlju biće opisan princip rada algoritama baziranih na Feistel strukturi. Razumevanje šeme i rada poznatih algoritama zasnovanih na ovoj šemi je od izuzetnog značaja za uočavanje ključnih karakteristika koje su važne prilikom dizajna novog algoritma. Treće poglavlje prikazaće proces dizajna i implementacije novog algoritma, stavljajući akcenat na obrazloženje izbora operacija, parametara i konstanti iz kojih se sastoji algoritam. Imajući u vidu da su vizuelni simulatori jedni od najvažnijih alata za razumevanje algoritama, treće poglavlje daće i kratak pregled implementacije i prikaz načina korišćenja realizovanog simulatora. Četvrto poglavlje prikazaće rezultate evaluacije algoritma, testirajući njegovu sigurnost i otpornost na poznate napade, kao i ispoljavanje poželjnih svojstava simetričnih blok algoritama kao što je efekat lavine. U poslednjem poglavlju ovog rada dat je zaključak.

Ovaj rad je finansiran od strane Ministarstva nauke, tehnološkog razvoja i inovacija kroz ugovor broj:451-03-65/2024-03/200103.

II. FEISTEL STRUKTURA SIMETRIČNIH BLOK ALGORITAMA

U kriptografiji, Feistel struktura [1] je poznata šema koja se koristi u konstrukciji simetričnih blok algoritama. Veliki broj danas korišćenih algoritama koristi ovu šemu, zbog čega je vrlo važno razumeti način po kojem ona funkcioniše, pre učenja samih algoritama. U ovom poglavlju će biti opisane osnovne karakteristike simetričnih blok algoritama zasnovanih na Feistel strukturi.

Ulaz u algoritam predstavlja blok veličine *2W* (paran broj bita) i ključ veličine *K* bita. Ulazni blok se deli na levu polovinu L_0 i desnu polovinu D_0. Obe polovine bloka prolaze kroz *n* rundi obrade. Sve runde imaju istu stukturu. Na kraju svake runde generišu se leva i desna izlazna vrednost, koje predstavljaju ulaz u sledeću rundu obrade. Na slici 1. prikazan je način generisanja izlaznih vrednosti za rundu *i+1*. Numeracija vrednosti *i* je u opsegu od [0, n-1]. Vrednosti L_i, D_i i K_{i+1} jesu leva vrednost generisana u rundi *i*, desna vrednost generisana u rundi *i* i ključ runde *i+1*, respektivno. Ove vrednosti predstavljaju ulaz u rundu *i+1*. Leva vrednost runde *i+1*, u oznaci L_{i+1}, jeste jednaka desnoj vrednosti runde *i*, u oznaci D_i. Desna vrednost runde *i+1*, u oznaci D_{i+1} jednaka je rezultatu XOR operacije koja se izvršava nad levom vrednošću runde *i*, u oznaci L_i, i rezultatu funkcije F_{i+1}. Funkcija označena sa F_{i+1} jeste Feistel funkcija koja se primenjuje nad desnom polovinom ulaza, u oznaci D_i, ukjučujući kombinaciju sa ključem runde K_{i+1}. Nakon *n* rundi, leva i desna polovina bloka menjaju mesta [1].



Sl. 1. Runda Feistel strukture

Proces enkripcije algoritama zasnovanih na Feistel strukturi uključuje podelu originalne poruke tj. ulaznog bloka na levu i desnu polovinu, prolazak kroz *n* rundi obrade i zamenu polovina nakon poslednje runde. Na ovaj način dobija se šifrovana poruka. Proces dekripcije se oslanja na isti algoritam, samo se ključevi rundi koriste u obrnutom redosledu. Ovo poželjno svojstvo Feistel strukture omogućava da se ne implementira zasebno algoritam za dešifrovanje, već se zbog samih osobina strukture i zamene leve i desne polovine nakon poslednje runde, može iskoristiti algoritam za šifrovanje, pri čemu se vodi računa o redosledu korišćenja ključeva rundi koji mora biti obrnut od redosleda njihovog korišćenja pri šifrovanju. Ključevi rundi se generišu iz osnovnog ključa *K* i njihov broj jeste jednak broju rundi *n* [2].

Kriptografska sigurnost algoritama zasnovanih na Feistel strukturi se temelji na ideji da se kombinovanjem jednostavnijih algoritama za šifrovanje postiže jači kriptografski rezultat nego u slučaju primene samo jednog složenog algoritma. Primena većeg broja jednostavnijih algoritama ostvarena je postojanjem rundi obrade. Svaka runda se može predstaviti kao zaseban algoritam za šifrovanje. Implementacija takvog algoritma podrazumeva primenu više jednostavnih operacija. Ovakve operacije odabrane i raspoređene na odgovarajući način treba da proizvode visok stepen konfuzije i difuzije [3].

Da bi algoritam mogao biti svrstan među algoritme zasnovane na Feistel strukturi, on mora posedovati osovne karaketistike ove strukture navedene na početku poglavlja. Ono što ga može izdvojiti i razlikovati od ostalih algoritama iz ove grupe jeste broj bita ulaznog bloka podataka, veličina ključa *K*, način generisanja ključeva rundi, izbor i implementacija operacija unutar Feistel funkcije *F* i eventualne primene dodatnih operacija pre ulaska u prvu rundu kao i nakon poslednje runde.

III. Dizajn i implementacija novog algoritma zasnovanog na Feistel strukturi

U ovom poglavlju rada biće detaljnije predstavljene operacije dizajniranog algoritma zajedno sa svim parametrima koji su usvojeni pri njegovoj implementaciji. Nakon toga, ukratko će biti objašnjena i sama implementacija algoritma. Kao pomoć u razumevanju algoritama zasnovanih na Feistel strukturi, kroz celo poglavlje biće prikazan izgled implementiranog simulatora koji demonstrira način rada dizajniranog algoritma.

A. Dizajn algoritma

Za veličinu ulaznog bloka odabrano je 128 bita. Ovaj ulazni blok prvo prolazi kroz operaciju *Inicijalne permutacija*. Ideja za

Sl. 2. Matrični prikaz

Sl. 3. Operacija kružnog šiftovanja reda udesno 0,1,2,3

postojanjem *Inicijalne permutacije*, pre nego što se krene u prvu rundu obrade, jeste preuzeta iz implementacije DES (*Data Encryption Standard*) [4] algoritma. U implementaciji novog algoritma, *Inicijalna permutacija* je predstavljena tako da svojim podoperacijama ne samo da doprinosi difuziji, kao kod DES algoritma, već i ostvarenju konfuzije čime se dodatno pojačava sigurnost algoritma. Kod *Inicijalne permutacije*, ulazni blok je potrebno predstaviti matrično 4x4, gde je svako polje matrice jedan bajt bloka. Primer takve predstave dat je na slici 2. Ulazni blok od 128 bita se predstavlja heksidecimalno. Ako se vrednost posmatra s leva na desno i sa *k* označimo poziciju bajta, onda se *k*-ti bajt smešta u matricu tako što se red određuje kao celobrojni deo rezultata *k/4*, a kolona kao rezultat operacije *k mod 4*.

1) Inicijalna permutacija

U nastavku su dati detalji operacija iz kojih se sastoji *Inicijalna permutacija*:

1. Rotacija viših 4 bita sa nižih 4 bita unutar svakog bajta. Operacija se naziva *4bit zamena*. Npr. primenom ove operacije nad heksidecimalnom vrednošću 9C dobija se rezultat C9. Ova operacija rezultuje nekom vrstom zamene te utiče na ostvarivanje konfuzije u okviru *Inicijalne permutacije*.

2. Kružno rotiranje udesno po redovima tako što se bajtovi u prvom redu pomere za 0 mesta udesno, u drugom redu za 1 mesto, trećem redu za 2 a u četvrtom redu za 3 mesta. Operacija se naziva *rotacija reda 0,1,2,3*. Primer ove operacije prikazan je na slici 3. Matrica levo jeste vrednost pre operacije, matrica desno je vrednost nakon operacije.

3. Kružno rotiranje nadole po kolonama i to tako što se bajtovi u prvoj koloni pomere za 3 mesta nadole, u drugoj koloni za 2 mesta, trećoj koloni za 1 a u četvrtoj koloni za 0 mesta. Operacija se naziva *rotacija kolone 3,2,1,0*. Primer ove operacije prikazan je na slici 4. Matrica levo predstavlja vrednost pre operacije, matrica desno je vrednost nakon operacije.

Operacijama *rotacija reda 0,1,2,3* i *rotacija kolone 3,2,1,0* ostvaruje se neki vid permutacije te one utiču na postojanje difuzije u operaciji *Inicijalne permutacije*.

2) Iteracije

Nakon *Inicijalne permutacije* blok od 128 bita se deli na levu i desnu polovinu, svaka po 64 bita i kao takav ulazi u prvu rundu obrade. Za broj rundi uzet je broj *n=16*, kao kod većine algoritama zasnovanih na Feistel strukturi.

Sl. 4. Operacija kružnog šiftovanja kolone nadole 3,2,1,0

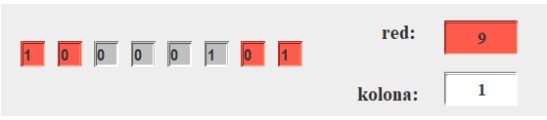
Generalno broj rundi ne bi trebalo da bude manji od 3, jer se tada ne garantuje dobra kriptografska sigurnost [1].

Leva i desna polovina bloka se na kraju svake runde formiraju na način opisan u prethodnom poglavlju, karakterističan za algoritme sa Feistel strukturom. Ono što nije naznačeno u opisu strukture jesu operacije Feistel funkcija *F* jer njihov izbor i implementacija zavisi od konkretnog algoritma za šifrovanje. U implementaciji algoritma koji je predmet ovog rada, Feistel funkcija se sastoji iz četiri operacije.

Prva operacija ima naziv *permutacija i proširenje 128b*. Ulazna vrednost Feistel funkcije, te i ove operacije, za posmatranu rundu *i+1* jeste blok D_i. Blok D_i se rotira udesno kružnom rotacijom za 17 bita, ista ulazna vrednost D_i se rotira udesno kružnom rotacijom za 11 bita, zatim se ove dve vrednosti nadovezuju jedna za drugom, čime nastaje vrednost veličine 128 bita. Ova vrednost se zatim kružno pomera udesno za 13 bita. Pomeranje za broj bita koji je neparan dovodi do permutacije unutar vrednosti takve da nijedan bit ne zadrži svoje prvobitno mesto i da novonastala vrednost bude prihvatljiva kao dovoljno siguran rezultat permutacije. Ideja za ovakvu implementaciju permutacije, bez postojanja tabela sa informacijama koji bit treba smestiti na koje mesto, preuzeta je iz GOST 28147-89 (Magma) algoritma [5]. Nadovezivanje vrednosti kako bi nastala vrednost od 128 bita je neophodno zbog XOR operacije sa ključem runde, koji je veličine 128 bita.

Druga operacija ima naziv *S box zamena*. Ova operacija ostvaruje supstituciju korišćenjem tzv. *S box*-ova, a primenjuje se nad rezultatom prethodne operacije. Postoje dve *S box* tabele, *S1* i *S2*, a konkretan *S box* koji će se koristiti zavisi od vrednosti tekuće runde. Ukoliko je broj runde neparan koristi se *S1 box*, a ukoliko je broj runde paran *S2 box*. Zamena se vrši na nivou bajtova. *S box* je predstavljen kao matrica dimenzije 16x16, svako polje veličine jedan bajt. Da bi se izvršila zamena potrebno je bajt ulaza predstaviti binarno. Vrednost reda tabele određuju biti sa pozicija 1, 2, 7 i 8 (numeracija kreće od jedinice, a pozicija bitova se određuje s leva na desno). Vrednost kolone u tabeli određuju biti na pozicijama 3, 4, 5, i 6. Primer indeksiranja reda i kolone *S box* tabele je dat na slici 5. Na slici 5. je prikazana binarna predstava heksidecimalne vrednosti 85. Markirani biti predstavljaju bite sa pozicija 1, 2, 7 i 8, respektivno.

Ono što treba da važi za svaki *S box* jeste da nijedan izlazni bit *S box*-a ne bi trebalo da bude previše blizu linearnoj funkciji ulaznih bitova. To znači da ako izaberemo bilo koju poziciju izlaznog bita i bilo koji podskup od osam pozicija ulaznih bitova, verovatnoća za koju ovaj izlazni bit odgovara XOR-u ovih ulaznih bitova ne bi trebalo da bude blizu 0 ili 1. Ovo je važno za obezbeđivanje sigurnosti *S box*-a u kriptografskim sistemima, sprečavajući jednostavne linearne veze između ulaza i izlaza. Jasno je da je generisanje *S box*-a dosta zahtevna operacija koja podrazumeva razumevanje matematičkih aspekata kako bi se gore navedena pravila ispoštovala. Dodatno, dobijeni *S box* prolazi kroz iscrpne analize kako bi se utvrdilo da je kriptografski dobijen jak efekat. Zato se ovaj rad oslanja na način generisanja *S box*-a korišćen kod AES algoritma.



Sl. 5. Indeksiranje reda i kolone S box-a

Generisanje *S box*-ova u AES (*Advanced Encryption Standard*) [6] algoritmu uključuje specifičan matematički proces. Način generisanja jednog bajta *S box*-a je opisan formulom (1).

$$Y = Ax \oplus c \bmod M \qquad (1)$$

Y je rezultat ove operacije i predstavlja jedan ulaz u *S box* tabelu. Red *S box* tabele u koji se upisuje *Y* označimo sa *r* a kolonu sa *k*. *A* je afina matrica definisana u specifikaciji AES algoritma. *C* je afina konstanta koja iznosi 63h, isto zadata specifikacijom AES algoritma. *M* je nesvodljivi polinom (x^8 + x^4 + x^3 + x + 1) koji se koristi za modulo operacije u Galoa polju GF(2^8). Vrednost ovog polinoma se označava kao 11Bh. Neka je x^{-1} heksidecimalna vrednost dobijena kao *(r<<4) | k*. Vrednost se mapira na svoj multiplikativni inverz u Galoa polju. To znači da se za svaku vrednost x^{-1} traži takva vrednost *x* da važi izraz (2).

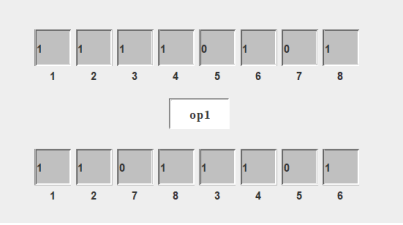
$$x*x^{-1} \equiv 1 \bmod M \qquad (2)$$

Ovaj korak se izvodi kako bi se obezbedilo da svaka vrednost x^{-1} ima jedinstveno određenu vrednost *x*. *Y* predstavlja bajt koji se upisuje u *S box* tabelu u redu *r* i koloni *k*. Da bismo popunili celu *S box* tabelu, potrebno je da se predstave sve moguće kombinacije reda i kolona. Red i kolona pojedinačno mogu uzimati vrednosti u opsegu od 0 do 15 [7].

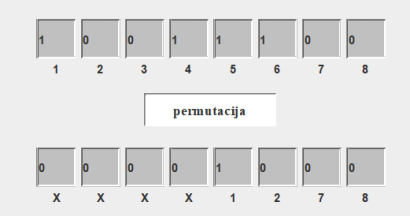
Usvojeno je da *S box* AES algoritma odgovara *S1 box*-u. Za generisanje *S2 box*-a iskorišćen je jedan od testiranih *S box*-ova od strane NIST-a (*National Institute of Standards and Technology*) koji su dali dobre rezultate. Razlika između *S1 box*-a i *S2 box*-a jeste u nesvodljivom polinomu koji se koristi u GF (2^8) i afinoj konstanti. Za *S2 box* je usvojeno da je vrednost nesvodljivog polinoma 11Dh tj. x^8 + x^4 + x^3 + x^2 + 1 polinomijalno. Vrednost afine konstante za generisanje *S2 box*-a jeste 0Ah [8].

Treća operacija je operacija kombinovanja sa ključem runde. U osnovi ove operacije jeste sabiranje dve 128 bitne vrednosti po modulu 128. Naziv operacije je *sabiranje mod128*. Vrednosti koje se sabiraju jesu rezultat prethodne operacije *S box zamena* i ključ tekuće runde. Većina kriptografkih algoritama na ovom mestu koristi XOR operaciju. XOR operacija je reverzibilna operacija. Sabiranje sa odgovarajućim ključem je pogodnije za korišćenje od XOR operacije, jer je teže vratiti originalne podatke iz rezultata sabiranja. XOR jeste operacija koja je brža u odnosu na operaciju sabiranja, ali operacija sabiranja daje bolju sigurnost kritpografskom algoritmu te je iz tih razloga odabrana. Kriptografski algoritam koji takođe koristi sabiranje sa ključem iteracije umesto XOR operacije jeste GOST 28147-89 (Magma) algoritam [5].

Poslednja operacija ima naziv *permutacija i sužavanje 64b*. Primenjuje se nad rezultatom operacije *sabiranje mod128*. Najpre se unutar svakog bajta vrši zamena mesta bita.



Sl. 6. Op1 zamena



Sl. 7. Op2 permutacija

Zamena je izvršena tako što se svaki bajt predstavi binarno, a zatim biti unutar bajta menjaju mesta. Redosled zamene bita je prikazan na slici 6. Ova podoperacija se naziva *op1*.

Nad rezultatom ove podoperacije vrši se kružna rotacija udesno za 17 bita. Na kraju, da bi se dobila 64 bitna vrednost primenjuje se podoperacija *op2* koja podrazumeva permutaciju na nivou bajta gde se iz svakog bajta uzimaju samo biti sa odgovarajućih pozicija (1, 2, 7 i 8) i postavljaju se kao niža četiri bita redom. Numeracija bita unutar bajta kreće od jedinice s leva u desno, slika 7. Na slici 7 data je vrednost bajta nakon permutacije, gde biti označeni sa X nemaju nikakvu ulogu u kreiranju kranjeg rezultata operacije *permutacija i sužavanje 64b* te su postavljeni na 0. Ovi biti se odbacuju kako bi se kreirala krajnja vrednost, tj. primenjuje se *supstitucija* u okviru *op2*. Na taj način se od 128 bitne vrednosti dobija 64 bitna vrednost. Vrednost nakon *op2* jeste rezultat Feistel funkcije F_i za rundu *i*.

Izlaz Feistel funkcije F_{i+1} za posmatranu rundu *i+1*, zajedno sa levom polovinom bloka na ulazu runde, u oznaci L_i, ulazi u XOR operaciju, nakon koje se kao rezultat dobija desna polovina bloka za rundu *i+1* u oznaci D_{i+1}. Kako je leva polovina bloka za rundu *i+1*, u oznaci L_{i+1}, jednaka desnoj polovini bloka sa ulaza u rundu *i+1*, u oznaci D_i, u potpunosti je dobijen izlaz runde *i+1* i može se nastaviti sa sledećom rundom.

3) Swap i inverzna inicijalna permutacija

Nakon 16. runde, leva i desna polovina bloka menjaju mesta operacijom *Swap*. Nakon operacije *Swap* sledi *Inverzna inicijalna permutacija*, ujedno i poslednja operacija ovog algoritma za šifrovanje. Pošto implementacija algoritma mora biti takva da se dešifrovanje izvrši sa minimalnim promenama originalnog algoritma za šifrovanje, *Inverzna inicijalna permutacija* je implementirana tako da predstavlja zaista inverz *Inicijalne permutacije*. Ukoliko ova operacija ne predstavlja inverz *Inicijalne permutacije*, dešifrovanjem se ne bi dobila očekivana originalna poruka. U nastavku sledi skup operacija iz kojih se sastoji *Inverzna inicijalna permutacija*.

1. Kružno šiftovanje nagore po kolonama i to tako što se bajtovi u prvoj koloni šiftuju za 3 mesta nagore, u drugoj koloni za 2 mesta, trećoj koloni za 1, a u četvrtoj koloni za 0 mesta. Operacija se naziva *rotacija kolone 3,2,1,0*. Primer ove operacije prikazan je na slici 8. Matrica levo predstavlja vrednost pre operacije, matrica desno je vrednost nakon operacije.

Sl. 8. Operacija rotacije kolona 3, 2, 1, 0

Sl. 9. Operacija rotacije reda 0, 1, 2, 3

2. Kružno rotiranje ulevo po redovima i to tako što se bajtovi u prvom redu rotiraju za 0 mesta ulevo, u drugom redu za 1 mesto, trećem redu za 2, a u četvrtom redu za 3 mesta. Operacija se naziva *rotacija reda 0,1,2,3*. Primer ove operacije prikazan je na slici 9. Matrica levo jeste vrednost pre operacije, matrica desno je vrednost nakon operacije.

3. Rotacija viša 4 bita sa nižih 4 bita unutar svakog bajta. Operacija se naziva *4bit zamena*. Npr. primenom ove operacija nad heksadecimalnom vrednošću 9C dobija se rezultat C9.

Ovim su u potpunosti opisane operacije algoritma za šifrovanje koje manipulišu ulaznim blokom podatka. U daljem delu teksta će detaljno biti objašnjen način generisanja ključeva rundi.

4) Generisanje ključeva runde

Ulazni ključ je veličine 256 bita dok su ključevi rundi veličine 128 bita. Ukupno se generiše 16 ključeva rundi. Ključevi rundi se generišu na potpuno identičan način, korišćenjem istih operacija, samo se razlikuju vrednosti koje su na ulazu svake runde. Ključ koji se zadaje na ulazu se deli na dve polovine označene kao *L[0]* tj. leva vrednost ulaza i *L[1]* tj. desna vrednost ulaza, svaka 128 bita. U prvoj rundi, ove vrednosti prolaze kroz četiri operacije:

1. Operacija *S box zamena*. Zamena se vrši na isti način kao i kod Feistel funkcije *F* (način određivanja kolone i reda u odnosu na ulazni bajt koji se menja je isti kao u opisanoj operaciji Feistel funkcije, u pitanju je operacija sa rednim brojem 2.). Razlika je u načinu određivanja koji će se *S box* koristiti. Bajt na parnom mestu za zamenu koristi *S1 box* dok bajt na neparnom mestu koristi *S2 box*. Ovo važi za svaku rundu. Koriste se već navedeni *S box*-ovi. Mesto bajta se određuje s leva na desno a numeracija kreće od jedinice. Operacija se vrši nad levom vrednošću ulaza. Za rundu 1 to je vrednost *L[0]*, za rundu 2 to je vrednost *L[1]*, za rundu i to je vrednost K_{i-2} gde *i* predstavlja broj runde za koju se generiše ključ.

2. Operacija *xor P[i]* gde je *i* vrednost runde. *P* je niz od šesnaest 128b vrednosti. Ove vrednosti su konstante. U pitanju je XOR operacija vrednosti *P[i]* i vrednosti koja se dobija nakon primene prethodne operacije. Konstante *P* niza su generisane kombinovanjem konstanti niza *P Blowfish* algoritma [9]. Ovo je deo dizajna *Blowfish* algoritma gde su svi parametri i konstante pažljivo odabrani kako bi se postigla dobra sigurnost i performanse. Generalno, generisanje dobrih konstanti se zasniva na iscrpnom procesu testiranja rezultata šifrovanja primenom te konstante na konkretnom mestu.

3. Operacija *rotacija udesno 17b* gde se vrši kružna rotacija udesno za 17 bita desne vrednosti ulaza i vrednosti dobijene nakon operacije *xor P[i]*. Ove dve vrednosti se svaka zasebno rotiraju. Što se tiče desne vrednosti ulaza to je zapravo vrednost *L[1]* za prvu rudnu, za rundu *i* vrednost K_{i-1}.

4. Operacija *XOR nakon svih operacija*, podrazumeva XOR vrednosti koje su rezultati kružnih šiftovanja u operaciji iznad.

Na ovaj način u prvoj rundi dobijemo vrednost K_1 tj. ključ runde 1. Za generisanje ključa runde 2 tj. K_2, ulazne vrednosti runde su *L[1]* sada kao leva vrednost i K_1 kao desna. Nadalje, za generisanje ključa runde *i* tj. K_i, leva vrednost ulaza je K_{i-2} a desna vrednost ulaza K_{i-1} gde je *i* vrednost tekuće runde.

B. Implementacija algoritma

Algoritam je implementiran u programskom jeziku Java u integrisanom razvojnom okruženju Eclipse IDE for Java Developers verzije 2023-09. U svrhe vizuelne reprezentacije operacija algoritma, implementiran je i simulator. Implementirane klase su podeljene u četiri paketa.

Klase paketa *operationPanels*, *subOperationDialogs* i *windows* jesu klase koje se koriste za implementaciju simulatora. Grafički elementi i korisnički interfejs koji čine simulator razvijeni su koristeći Java Swing, standardnu biblioteku za izradu GUI-a u Javi. Ovo uključuje JButton za dugmad, JPanel za panele, JFrame za prozore, JDialog za dijaloge i JLabel za labele [10]. Kako bi se olakšao proces razvoja, koristi se WindowBuilder, dodatak za Eclipse, koji pruža vizuelno okruženje za dizajniranje Swing GUI komponenata [10].

Klase koje implementiraju operacije i logiku algoritama nalaze se u zasebnom paketu namenjenom za tu svrhu, paket *algorithms*. Implementacija operacije *Inicijalna permutacija* enkapsulirana je u klasi *InitialPermutation.java*. Implementacija svih šesnaest rundi algoritma je enkapsulirana u klasi *RoundAlgorithm.java*. Implementacija operacije *Swap* je enkapsulirana u klasi *SwapLeftAndRight.java* dok je za operacije *Invezna inicijalna permutacija* implementirana klasa *InversalInitialPermutation.java*. Za generisanje ključeva rundi implementirana je klasa *SubKeyGeneration.java*.

Navedenom organizacijom paketa se ostvaruje bolja modularnost koda, olakšava se razumevanje i održavanje koda, omogućava se ponovna upotrebljivost i dr.

IV. Analiza algoritma

Analiza algoritma za šifrovanje obično zahteva pažljivu procenu različitih aspekata kako bi se osigurala njegova efikasnost, bezbednost i praktičnost. U ovom poglavlju ćemo prikazati neke od metoda koje se primenjuju u analizi algoritma za šifrovanje/dešifrovanje.

A. Brute force

Brute force je najjednostavniji način za razbijanje šifre. Napad podrazumeva generisanje svih mogućih kombinacija ključa dok se ne pronađe onaj pravi. Broj mogućnosti određen je veličinom ključa u bitima. Za ključ veličine 256 bita broj mogućih kombinacija iznosi 2^256.

TABELA I. Trajanje šifrovanja

Ulaz	*Vreme (s)*
7d464725ad023da2e5b4f5170aebc18d	0.00518611
53ff733b641e6f99d93bae76b42fa67c	0.00502608
7d464725ad023da2e5b4f5170aebc18d	0.00474204

TABELA II. Trajanje dešifrovanja

Ulaz	*Vreme (s)*
dd8cf1b123f37b1651b627c6cf741c2e	0.00526588
985f5421ea8fbb355b96ad64036dcacd	0.00425266
112233445566778899aabbccddeeff00	0.00491807

Brute force napad na 256-bitni ključ u realnom vremenu za sada nije izvodljiv. Čak i kvantno računarstvo trenutno nije dovoljno razvijeno da bi efikasno izvelo brute force napad na 256-bitni ključ [2].

B. Trajanje šifrovanja/dešifrovanja

U programskom jeziku Java, jedan od najpreciznijih načina za merenje vremena izvršenja je korišćenje metode *System.nanoTime()*. Metoda *System.nanoTime()* pruža vremensku tačnost u nanosekundama što je znatno preciznije od *System.currentTimeMillis()*, koja meri vreme u milisekundama. Performanse algoritma su merene na računaru LENOVO 81AX sa operativnim sistemom Microsoft Windows 10 Pro, procesorom Intel Core i5 8th Generation, 12 GB RAM.

U tabeli 1. prikazani su rezultati ovog merenja za algoritam šifrovanja. Kolona *Ulaz* predstavlja ulaznu vrednost koja se šifruje, za vrednost ključa uzeta je vrednost: *896b4855e4802da4ff52bbad9134beea3eca27b4d63f966b8d5cc0b0099e4be1*, a kolona *Rezultat* jeste vreme neophodno algoritmu da izvrši operaciju šifrovanja. Vreme je prikazano u sekundama.

U tabeli 2. prikazani su rezultati merenja za algoritam dešifrovanja. Kolona *Ulaz* predstavlja ulaznu vrednost koja se dešifruje, za vrednost ključa uzeta je vrednost: *896b4855e4802da4ff52bbad9134beea3eca27b4d63f966b8d5cc0b0099e4be1*, a kolona *Rezultat* jeste vreme neophodno algoritmu da izvrši operaciju dešifrovanja. Vreme je prikazano u sekundama.

Vreme navedeno u tabelama obuhvata i trajanje operacija za generisanje ključeva rundi. Kada bi se oduzelo trajanje ovih operacije, samo šifrovanje/dešifrovanje bi otprilike trajalo upola manje od vrednosti iz tabela. Pretpostavimo da šifrujemo 1kB ulaznih podataka što predstavlja 1024 bit-a, tj. 1024/128b = 8 ulaznih blokova (8 blokova znači da se algoritam šifrovanja primenjuje 8 puta kako bi se svaki blok zasebno šifrovao). Kako je vreme šifrovanja jednog bloka približno oko 0.005s, ali polovina tog vremena predstavlja trajanje operacija za generisanje ključeva rundi što treba da se računa samo jednom, a ne za svaki blok posebno, za šifrovanje 8 blokova je potrebno 0.005s+(0.005/2)*7 što iznosi 0.0225s. Koristeći istu logiku, za šifrovanje 1MB (1048576 b tj. 8192 bloka) potrebno je oko 20s, dok je za šifrovanje 1GB (1073741824 b tj. 8388608 blokova) potrebno oko 20000s, što je oko 5,5 sati.

Ne postoji jasan kriterijum kojim je moguće odrediti da li je trajanje ovog algoritma zadovoljavajuće ili ne. Čak ni upoređivanje vremena trajanja algoritma sa algoritmima koji su bili kandidati za javni proces standardizacije pokrenut od strane NIST-a nije od tolikog značaja. Glavni razlog za ovo jeste korišćenje različitih hardverskih i sofrtverskih resursa pri merenju performansi jer je NIST interno evaluirao performanse kandidata na mikrokontrolerima. Ono što treba naglasiti jeste da

u NIST-ovom procesu selekcije, vreme trajanja algoritma nije bila krucijalna tačka izbora [11].

C. Efekat lavine

Efekat lavine [4] je važan koncept u kriptografiji. Značaj efekta lavine jeste u tome što se sprečava mogućnost da napadač predvidi kako promene u ulaznom tekstu ili ključu utiču na rezultat šifrovanja/dešifrovanja. Time su kriptografski algoritmi otporni na različite vrste napada, a jedan od značajnijih jeste diferencijalna kriptoanaliza. Ako se promenom samo jednog bita ulaznih podataka rezultat šifrovanja promeni za barem 50% bita, to ukazuje da algoritam ima jak efekat lavine.

Analiza efekta lavine se zasnivala na posmatranju razlike u izlazima algoritma kada su ulazi ili ključevi različiti za samo jedan bit. Rezultati analize se kreću oko vrednosti od 52%. Iako sama analiza nije podrazumevala testiranje algoritma pomoću SAC (*Strict Avalanche Criterion*) i BIC (*Bit Independence Criterion*) testova kao jednih od zvaničnih testova, analiziranjem dobijenih rezultata pri evaluaciji algoritma za različite ulaze i ključeve ispunjeni su kriterijumi SAC testova. Što se tiče kriterijuma koji se proveravaju BIC testovima, njih je moguće testirati samo standardizovanim metodama [12] što predstavlja jedan od daljih pravaca istraživanja ovog rada.

V. ZAKLJUČAK

U ovom radu su objašnjeni teorijski koncepti na kojima se zasniva sama Feistel struktura i važnost prilikom izbora operacija iz kojih će da se sastoji dizajnirani algoritam. Opisan je detaljan proces dizajna i implementacije novog algoritma, kao i edukativnog simulatora koji na interaktivan način prikazuje funkcionisanje algoritma po koracima. Najzad, prikazani su rezultati testova koji potvrđuju sigurnost i otpornost algoritma na poznate napade, kao i prisustvo poželjnih svojstava algoritama kao što je efekat lavine.

Iako algoritam daje zadovoljavajuće rezultate na poljima sigurnosti, prostor za unapređenje algoritma u polju efikasnosti postoji. Jedan dalji pravac istraživanja zasnivaće se na poboljšanju brzine izvršavanja algoritma optimizacijom matematičkih operacija koje predstavljaju sastavni deo algoritma, kao i njihovom paralelizacijom. Dodatno, ispitivanje efekta lavine može biti ispitano i standardizovanim metodama. Drugi pravac daljeg istraživanja je unapređenje simulatora i prilagođavanje korisničkog interfejsa da odgovara savremenim standardima. Treći pravac daljeg istraživanja je dodavanje mogućnosti šifrovanja većeg broja blokova podataka odjednom inkorporiranjem nekog od poznatih modova funkcionisanja.

REFERENCE/LITERATURA

[1] Feistel, H. “Cryptography and Computer Privacy.” Scientific American, May 1973.

[2] Schneier, B. (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C. John Wiley & Sons, Inc.

[3] Stallings, W. (Poslednje izdanje, npr. 2017). Cryptography and Network Security: Principles and Practice. Prentice Hall.

[4] Smith, J. (2018). Computer Security and Cryptography. O'Reilly Media.

[5] Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of Applied Cryptography. CRC Press. https://doc.lagout.org/security/Handbook_of_Applied_Cryptography-A.Menezes-P.vanOorschot-S.Vanstone-CRC_Press_1996.pdf 23.12.2023.

[6] National Institute of Standards and Technology (NIST). (2001). Announcing the Advanced Encryption Standard (AES) (Federal Information Processing Standards Publication 197). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf 23.12.2023.

[7] Singh, A., Agarwal, P., & Chand, M. (2017). Analysis of Development of Dynamic S-Box Generation. https://www.hrpub.org/download/20171130/CSIT2-13510193.pdf 30.12.2023.

[8] Das, S., Uz Zaman, J.K.M.S., & Ghosh, R. (2013). Generation of AES S-Boxes with various modulus and additive constant polynomials and testing their randomization. In Proceedings of the International Conference on Computational Intelligence: Modeling Techniques and Applications (CIMTA) 2013. https://www.sciencedirect.com/science/article/pii/S2212017313006051?ref=pdf_download&fr=RR-2&rr=83dbd235aaff6e98 24.12.2023.

[9] Schneier, B. (1993). Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). https://www.schneier.com/academic/blowfish/ 25.12.2023.

[10] Eclipse Foundation. (n.d.). Eclipse documentation - Current Release. https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.wb.doc.user%2Fhtml%2Findex.html 30.12.2023.

[11] National Institute of Standards and Technology. (2023). Efficient Implementation of AES-GCM and AES-GCM-SIV on Various Platforms. NIST Interagency/Internal Report (NISTIR) 8454.

[12] Doe, J., "Design and analysis of image encryption based on chaotic system and cellular automata," IEEE Xplore, DOI: [DOI broj], Pristupljeno: 14. mart 2024.

ABSTRACT

This paper proposes a new cryptographic symmetric block algorithm based on the Feistel structure, with the aim of improving the understanding of the functioning of algorithms based on the Feistel structure, as well as the importance of selecting operations, parameters and constants when designing a new algorithm. The key features of the algorithm were identified through a detailed analysis of existing algorithms based on the Feistel structure. In addition to the design of the algorithm, the paper describes and implements an educational visual simulator that facilitates the learning of the algorithm by showing the process of encryption and decryption of an arbitrary message step by step, which significantly contributes not only to the understanding of the implemented algorithm, but also to the Feistel structure on which the algorithm is based. A series of tests evaluating the efficiency, security and practicality of the designed algorithm were also carried out, using known methods that are used in the analysis of new algorithms.

A novel cryptographic algorithm based on the Feistel structure

Teodora Perić, Žarko Stanisavljević, Adrian Milaković