Primjena kriptografskih primitiva sa simetričnim ključem u praksi

Živković, Nikola

Sveučilište Josipa Jurja Strossmayera u Osijeku

Fakultet primijenjene matematike i informatike

Sveučilišni diplomski studij matematike
modul: Matematika i računarstvo

Primjena kriptografskih primitiva sa simetričnim ključem u praksi

Diplomski rad

Mentor:
prof.dr.sc. Ivan Matić

Student:
Nikola Živković

Osijek, 2024.

Sadržaj

1 | Uvod

Kroz iskustvo rada u jednoj stranoj kompaniji, gdje sam radeći na razvoju software za telekom kompanije zamijetio kako se jako puno pozornosti posvećuje tajnosti podataka i na svim e2e (eng. end to end) testovima svi govore da je jedan od glavnih prioriteta zaštititi podatke koji putuju između aplikacija. Kako sam većinski dio svog radnog vijeka razvijao aplikacije za 5G mrežu, taj uvjet, da podaci moraju biti sigurni, još je više dobio na važnosti s obzirom na arhitekturu 5G mreža. Stoga sam odlučio za kraj svog studiranja i lakšeg shvaćanja budućih izazova na poslu malo bolje proučiti kriptografiju i njezinu primjenu u praksi.

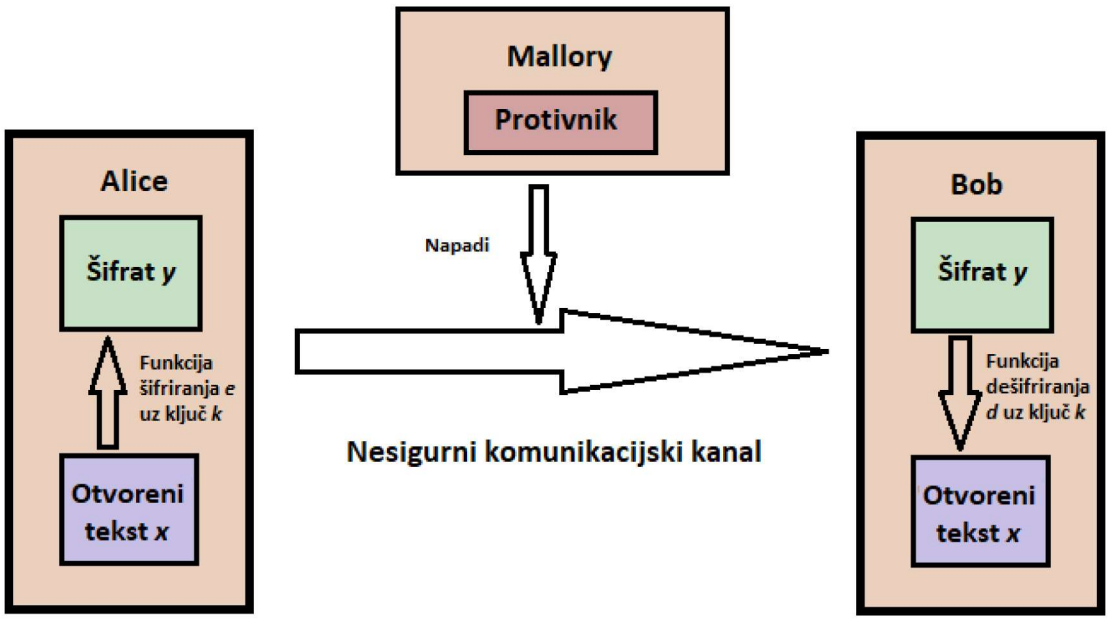
U ovome radu nećemo se bazirati na dokazivanju sigurnosti kriptografskih algoritama teoretskim putem, nećemo učiti kako dizajnirati nove kriptografske primitive što zahtjeva dosta predznanja i vremena te bi samim time i prvo poglavlje u kojemu smo definirali osnovne pojmove trebali znatno proširiti. Umjesto toga temeljiti ćemo ga na heurističkim i učinkovitim konstrukcijama primitiva koji su široko primjenjivi u praksi. Treba naglasiti kako sigurnost teoretski možemo opravdati samo kod nekih od njih, dok za gotovo sve vrijedi da je sigurnost dokazana na način da su izdržali godine proučavanja znastvenika i različitih pokušaja kriptoanalize. Stoga, razumnom je vjerovati kako su primitivi koje ćemo spomenuti u ovom radu sigurni, jedan od primjera tome je i da ne postoji temeljna razlika između pretpostavki da je faktorizacija teška i da je AES pseudoslučajna permutacija, a znamo koliko je problem faktorizacije zapravo težak. U ovome radu želimo pokazati neke principe dizajniranja jednostavnih algoritama u modernoj kriptografiji, te ćemo proći i kroz neke od najpoznatijih kriptografskih pojmova kao što su protočne šifre i blokovne šifre e ih sažeto opisati bez ulaska u dublje analize.

U poglavlju Osnovni pojmovi definirati ćemo neke kriptografske pojmove i vidjeti neke od podjela kriptosustava, a sve to u svrhu lakšeg razumijevanja s tekstom u nastavku ovoga rada. Poglavlje Protočne šifre će nam biti prvo koje čini bit ovog rada i u njemu ćemo saznati što su to protočne šifre i proći kroz nekoliko vrsta protočnih šifri sa simetričnim ključem kao što su: LFSR, FSR, Trivium i RC4. LFSR (eng. Linear-Feedback Shift Registers) i FSR (eng. Feedback Shift Registers) spadaju istu skupinu šifri koja se zove Registar povratnih pomaka, pa ćemo u ovom poglavlju objasniti po čemu su različite, također ćemo upoznati specifičan slučaj FSR-a koji se zove Trivium. I za kraj ovog poglavlja reći ćemo nešto više o najprimjenjenijoj protočnoj šifri, a to je RC4. Kroz poglavlje Blokovne šifre ćemo se se upoznati s općom definicijom blokovnih šifri, upoznati šifriranje na razina bitova,

te čemu to blokovne šifre moraju sličiti. Naučite ćemo što je to Shannonova paradigma zbunjenosti i difuzije, čemu služi efekt lavine, te proći kroz neke primjer simetričnih blokovnih šifri kao što su supstitucijsko-permutacijska mreža, AES i Fiestel mreža. U zadnjem poglavlju bavit ćemo se jednim od najbolje dizajniranih kriptografskih algoritama koji se zove Standard šifriranja podataka, skraćeno DES (eng. The Data Encryption Standard) što je i sam naziv poglavlja. Govorit ćemo o njegovom dizajnu, opisat ćemo primjere napada na krnji DES algoritam s jednom, dvije ili tri runde šifriranja, a zbog usporedbe treba naglasiti kako DES se u stvarnosti sastoji od 16 rundi širiranja. Dotaknut ćemo se njegove sigurnosti, u kojem je trenutku postao pouzdan za šifriranje, te zbog čega, a za kraj poglavlja ćemo naučiti nešto o algoritmima koji su nastali kao potomci DES algoritma.

2 | Osnovni pojmovi

Da bi bilo što lakše razumjeti ovaj rad, za početak ćemo definirati što je to kriptografija, kriptosustav, kriptografski primitivi te napraviti neke osnovne podjele u kriptografiji.



Slika 2.1: Osnovna ideja sigurne komunikacije

Kriptografija je znanstvena disciplina koja koristi matematičke metode kako bi omogućila pošiljaocu slanje kriptiranih poruku prema primaocu, u prisustvu protivnika, tako da ih samo on može dekriptirati. Kriptografija dolazi od starogrčkih riječi koje su romanizirane kryptós što znači skriven ili tajan i graphein što znači napisati, također treba biti pažljiv i ne miješati ju s kriptologijom koja je znanstvena grana i obuhvaća kriptografiju i kriptoanalizu. Kako bi definirali kriptosustav trebamo znati da se on sastoji od kriptografskog algoritma, te skupa svih mogućih otvorenih tekstova, šifrata i ključeva, a koristiti ćemo sljedeću definiciju:

Definicija 1. ([2], *Potpoglavlje 1.1.*) *Kažemo da je uređena petorka* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *kriptosustav ako vrijede sljedeće tvrdnje:*

(1) $\mathcal{P}$ *je konačan skup svih mogućih osnovnih elemenata otvorenog teksta;*

(2) $\mathcal{C}$ *je konačan skup svih mogućih osnovnih elemenata šifrata;*

(3) *$\mathcal{K}$ je konačan skup svih mogućih ključeva;*

(4) *$\mathcal{E}$ je skup svih funkcija šifriranja;*

(5) *$\mathcal{D}$ je skup svih funkcija dešifriranja;*

(6) *Za bilo koji ključ $K \in \mathcal{K}$ postoji funkcija šifriranja $e_K \in \mathcal{E}$ i odgovarajuća funkcija dešifriranja $d_K \in \mathcal{D}$;*

(7) *Za bilo koji otvoreni tekst $x \in \mathcal{P}$ vrijedi $e_K(x) = y$, $y \in \mathcal{C}$;*

(8) *Za x i y iz točke (7) vrijedi da je $d_K(y) = x$.*

Uočimo kako je u ovoj definiciji najvažnije svojstvo $d_K(e_K(x)) = x$ zato što iz njega slijedi svojstvo injektivnosti funkcije e_K, u suprotnom ako bi za otvorene tekstove x_1 i x_2 vrijedilo

$$e_K(x_1) = e_K(x_2) = y$$

tada primalac ne bi znao u koji otvoreni tekst od ova dva bi trebao dešifrirati šifrat y. Kriptografski primitivi su jednostavni algoritmi s kriptografskim svojstvima i služe kako bi obavili jednostavan precizno definiran zadatak te tvore složenije kriptografske alate. Podjelu kriptosustava najčešće radimo prema sljedećim kriterijima

1. Prema tipu operacija koje se koriste prilikom šifriranja dijelimo ih na one koje koriste
   (a) supstitucijske šifre
      - kod ove vrste se prema unaprijed utvrđenoj transformaciji svaki element otvorenog teksta (bit, slovo, grupa bitova ili slova) zamjenjuje s nekim drugim elementom, ovisno o broju transformacija mogu biti monoalfabetske i polialfabetske.

   (b) transpozicijske šifre
      - elementi otvorenog teksta se permutiraju.

   (c) kombinacija prethodne dvije vrste šifri.

2. Prema načinu na koji obrađuju otvoreni tekst, te ih dijelimo na one koji koriste
   (a) blokovne šifre
      - koristeći isti ključ blokovi elemenata otvorenog teksta obrađuje se jedan po jedan.

   (b) protočne šifre
      - koristeći paralelno generirani niz ključeva jedan po jedan element otvorenog teksta se obrađuje.

3. Prema tajnosti ključa dijelimo ih na one koji koriste
   (a) simetrični ključ

- poznavajući ključ za šifriranje može se izračunati ključ za dešifriranje i obratno, ovi su ključevi najčešće identični pa je sigurnost ovih kriptosustava zasnovana na tajnosti ključa, te se zbog toga također zovu i kriptosustavi s tajnim ključem.

(b) javni ili asimetrični ključ

- iz samog naziva je jasno kako je ključ koji se koristi za šifriranje javan te stoga bilo tko može pomoću njega šifrirati poruku, međutim za dešifriranje poruke je potreban odgovarajući ključ i stoga poruku može dešifrirati samo osoba koja ga posjeduje zato što izračunavanje ključa za dešifriranje nije moguće u nekom razumnom roku.

3 | Protočne šifre

Općenito, protočne šifre mogu biti i sa simetričnim i s javnim ključem. Međutim, prema samom naslovu ovoga rada je jasno kako ćemo se u ovom poglavlju baviti isključivo protočnim šiframa sa simetričnim ili tajnim ključem. U skladu s prethodno navedenim dati ćemo općenitu definiciju protočne šifre, no kako bismo ju mogli uvesti najprije moramo definirati pojam tok ključeva.

Definicija 2. ([1], *Potpoglavlje 2.2*) *Neka je $\mathcal{K}$ prostor ključeva za skup svih funkcija šifriranja. Niz znakova $e_1e_2e_3\ldots \in \mathcal{K}$ zove se tok ključeva.*

Definicija 3. ([1], *Potpoglavlje 2.2*) *Neka je $\mathcal{A}$ alfabet veličine n simbola i neka je e_K jednostavna funkcija šifriranja, $K \in \mathcal{K}$ je neki tok ključeva. Označimo s m niz znakova iz alfabeta $\mathcal{A}$ $m_1m_2m_3\ldots$ i on će nam predstavljati otvoreni tekst. Protočne šifre uzmu m i pretvore ga u šifrat $c = c_1c_2c_3\ldots$ gdje je $c_i = e_K(m_i)$. Ako s d_K označimo inverz od e_K onda s $d_K(c_i) = m_i$ možemo dešifrirati šifrat.*

Kako se ne bi puno zadržavali na teoriji, ovdje nećemo ulaziti i u definiciju alfabeta koju se može pronaći u [1], str. 8. U praksi se na protočne šifre gleda kao na nešto čime se instancira algoritam za generiranje pseudoslučajnih vrijednosti, a prilagođen je na način da se izlaz algoritma dobiva postupno i na zahtjev (više o ovom algoritmu se može pronaći u [4], str. 62) i najčešće se definiraju pomoću dva deterministična algoritma *Init* i *GetBits*.

Algoritam 1 ([4], Algorithm 3.16) Općeniti algoritam za protočne šifre

Input: ključ K i inicijalni vektor V
Output: $y_1, y_2, y_3, \ldots$

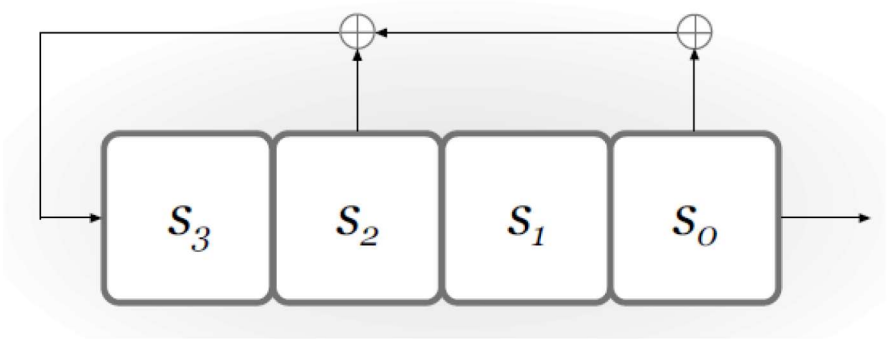
1: $st_0 \leftarrow Init(K, V)$
2: $st_0 \leftarrow Init(K, V)$
3: $i \leftarrow 1$
4: **while** true **do**
5: $\quad (y_i, st_i) \leftarrow GetBits(st_{i-1})$
6: $\quad i \leftarrow i + 1$

return: $y_1, y_2, y_3, \ldots$

Dakle, očigledno je da glavni zahtjev koji protočne šifre moraju zadovoljiti taj da se ponašaju kao pseudoslučajni generator što dovodi do zaključka da bi ponekada trebale zadovoljiti i više sigurnosne zahtjeve, što je moguće u slučaju kada bi bile konstruirane pomoću blok šifri (veličina bloka koji se obrađuje je 1), ali glavna prednost ovakvih konstrukcija je što se lagano mogu koristiti na računalnim sklopovima s malim resursima.

3.1 Registri povratnih pomaka

Registri povratnih pomaka (eng. Feedback Shift Registers, skraćeno FSR) su jedni u nizu algoritama kojima se nastoji implementirati protočna šifra i u ovom potpoglavlju ćemo proći kroz dvije vrste, linearni i nelinearni. Linearni registri povratnih pomaka (eng. Linear-Feedback Shift Registers, skraćeno LFSR) su učestaliji zbog svoje velike pogodnosti za implementaciju na nekom računalnom sklopu i jako dobrih statističkih svojstava koje ima vrijednost koju vraćaju, ali su jako ranjivi na napade te ćemo dakako u nastavku i pokazati jednu vrstu napada.



Slika 3.1: Linearni registar povratnih pomaka stupnja 4 ([4], Figure 6.1)

Linearni registar povratnih pomaka se sastoje od n registara u koje je pohranjen po jedan bit te su označeni s lijeva na desno s $s_{n-1}, s_{n-2}, \ldots, s_0$ i povratne petlje koju čine n povratnih koeficijenata označenih s $c_{n-1}, c_{n-2}, \ldots, c_0$, broj n se još zove i stupanj LFSRa. U nekoj iteraciji t stanje LFSRa je niz bitova koji se nalaze u registrima i označavamo ga s st_t, pa na početku imamo $t = 0$ i to zovemo nulto stanje te ga označavamo s st_0, taj početni niz bitova prema algoritmu 1 možemo dobiti pomoću nekog *Init* algoritma. LFSR se u svakoj iteraciji ažurira na način da se svi registri pomaknu za jedno mjesto u desno, bit koji je bio u registru s oznakom s_0 je bit koji algoritam vraća na izlazu, a novi bit u registru s oznakom s_{n-1} se izračuna pomoću operacije "isključivo ili" (XOR) na nekom podnizu bitova iz prethodne iteracije. Dakle, ako u iteraciji t imamo stanje

$$st_t = (s_{n-1}^{(t)}, s_{n-2}^{(t)}, \ldots, s_0^{(t)}),$$

onda u iteraciji $t+1$ stanje će biti

$$st_{t+1} = (s_{n-1}^{(t+1)}, s_{n-2}^{(t+1)}, \ldots, s_0^{(t+1)}),$$

gdje je

$$\begin{aligned} s_i^{(t+1)} &:= s_i^{(t)}, \qquad i = 0, \ldots, n-2 \\ s_{n-1}^{(t+1)} &:= \bigoplus_{i=0}^{n-1} c_i s_i^{(t)}. \end{aligned} \tag{3.1}$$

Ako pogledamo algoritam 1 možemo lako zaključiti da su definicije iz (3.1) zapravo dijelovi iz *GetBits* algoritma za LFRS, koji će u ovome slučaju imati još

jedan ulazni parametar, a to je ključ K, sam algoritma pomoću pseudojezika može se vidjeti u nastavku.

Algoritam 2 ([4], Potpoglavlje 6.1)GetBits za LFRS

Input: ključ K, st_t
Output: (y_{t+1}, st_{t+1})

1: $y_{t+1} \leftarrow s_0^{(t)}$
2: **for** $i \leftarrow 0,\ldots, n-2$ **do**
3: $\quad s_i^{(t+1)} \leftarrow s_i^{(t)}$
4: $s_{n-1}^{(t+1)} \leftarrow c_0 s_0^{(t)} \oplus c_1 s_1^{(t)} \oplus \ldots \oplus c_{n-1} s_{n-1}^{(t)}$
5: $st_{(t+1)} \leftarrow s_{n-1}^{(t+1)}, s_{n-2}^{(t+1)}, \ldots, s_0^{(t+1)}$

return: (y_{t+1}, st_{t+1})

Nakon što smo sve ovo definirali prvo što se nameće samo po sebi je pitanje što je tu ključ, očigledno da je to povratna petlja $(K = c_{n-1}, c_{n-2}, \ldots, c_0)$, također sve se više nameće pitanje može li se izlaz algoritma 1, tj. bitovi $y_i, i = 0, \ldots$ definirati općenito za neki i. Za neki generirani inicijalni vektor V te poznavajući ključ K možemo dobiti nulto stanje st_0 pa se vrlo jednostavno može doći do sljedećih tvrdnji

$$\begin{aligned} y_i &:= s_{i-1}^{(0)}, && i = 1, \ldots, n \\ y_i &:= \bigoplus_{j=0}^{n-1} c_j y_{i-n+j-1}, && i > 1. \end{aligned} \tag{3.2}$$

Primjer 1. *Neka je ključ $K = (1,0,1,0)$, te neka smo pomoću nekog Init algoritma dobili nulto stanje $st_0 = (0,0,1,1)$. Koristeći definicije iz (3.1) i (3.2) možemo izračunati izlaze algoritma 2, a rezultati se mogu vidjeti u tablici ispod.*

iteracija	stanje	izlazna vrijednost
0	(0, 0, 1, 1)	
1	(1, 0, 0, 1)	1
2	(1, 1, 0, 0)	1
3	(1, 1, 1, 0)	0
4	(0, 1, 1, 1)	0
5	(1, 0, 1, 1)	1
6	(0, 1, 0, 1)	1
7	(0, 0, 1, 0)	1
8	(1, 0, 0, 1)	0

Tablica 3.1: Rezultati

Rezultate iz gornje tablice smo dobili pomoću programskog koda koji se može vidjeti ispod, također u njemu se može vidjeti i implementacija algoritma 2 u programskom jeziku Python (verzija 3.7.7).

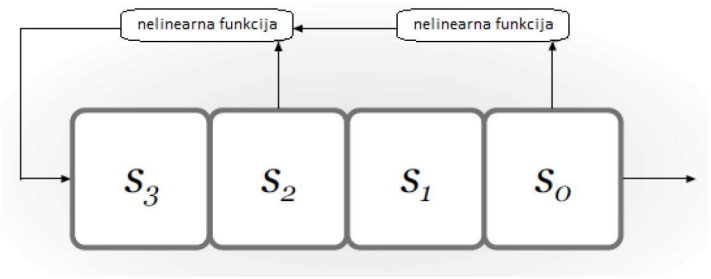
```
def getBits(key, state):
    degree = len(state)
    y = state[degree - 1]
    st = []
    for i in range(degree - 1):
        st.append(state[i])
    # bitwise  AND operacija moze zamijenit mnozenje na skupu {0,1}
    tempVar = [key[i] & state[i] for i in range(degree)]
    tempVar2 = 0
    for s in tempVar:
        tempVar2 ^= s
    st.insert(0, tempVar2)
    return (y, tuple(st))

if __name__ == "__main__" :
    st = (0,0,1,1) # inicijalno stanje
    key = (1,0,1,0) # kljuc/povratni koeficijenti
    print("i = %s, y = %s, state = %s" %(0, '', st))
    for i in range(17):
        (y, st) = getBits(key, st)
        print("i = %s, y = %s, state = %s" %(i+1, y, st))
```

Programski kod 3.1: Implementacija algoritma 2 i simulacija primjera 1

Ako se malo prisjetimo teorije iz kombinatorike dolazimo do zaključka da LFSR s alfabetom koji se sastoji od 0 i 1 može imati najviše 2^n stanja, gdje je n stupanj LFRSa. Kako stanje u kojem svi registri u sebi imaju pohranjen bit 0 nije baš dobro iz razloga što će onda stanje LFSRu zauvijek ostati isto, možemo reći da može imati najviše $2^n - 1$ stanja koja nisu sastavljena od samo 0. Dakle, sa sigurnošću možemo tvrditi da će nakon najviše $2^n - 2$ ponoviti nulto stanje, a broj iteracija do ponavljanja inicijalnog stanja ovisi isključivo o povratnim koeficijentima (ključu). Iako LFSR s maksimalnim brojem iteracija ima dobra statistička svojstva očigledno da mu je sigurnost jako slaba, pogotovo u doba današnjih računala. Međutim probijanje šifre na takav način i dalje može biti vrlo sporo i dugotrajano, jer se ipak radi o eksponencijalnom broju kombinacija, te ćemo sada pokazati jedan brži i jednostavniji način kako napadač može otkriti ključ, dovoljno mu je samo da dobije informaciju o bitovima koji izlaze iz registra. Ako pogledamo definicije iz (3.2) jasno je da je prvi n bitova koji "ispadnu" iz registra zapravo bitovi iz nultog stanja, a idućih n bitova su oni koje smo dobili izračunom. Dakle, napadaču je dovoljno prisluškivati samo prvih $2n$ bitova koji "ispadnu" iz registra i pomoću toga možemo kreirati n jednadžbi s n nepoznanice gdje su nam nepoznanice povratni koeficijenti (ključ) $c_0, c_1, \ldots, c_{n-1}$, te dobijemo sustav linearnih jednadžbi (3.3) koji se može riješiti korištenjem linearne algebre i teorije brojeva. Više o teoriji za LRFS i kako doći do rješenje sustava (3.3) se može naći u [3] potpoglavljima 1.1.7 i 1.2.5.

$$\begin{aligned} c_0 y_1 \oplus c_1 y_2 \oplus \ldots \oplus c_{n-1} y_n &= y_{n+1} \\ c_0 y_2 \oplus c_1 y_3 \oplus \ldots \oplus c_{n-1} y_{n+1} &= y_{n+2} \\ &\vdots \\ c_0 y_n \oplus c_1 y_{n+1} \oplus \ldots \oplus c_{n-1} y_{2n-1} &= y_{2n}. \end{aligned} \tag{3.3}$$

Kada smo uvidjeli koliko slabu sigurnost imaju linearni registri povratnih pomaka postavlja se pitanje može li se nešto učiniti da dobijemo registre povratnih pomaka s većim stupnjem sigurnosti. Kako smo već u uvodnim riječima ovog potpoglavlja naglasili da se registri povratnih pomaka dijele na dvije skupine, jasno je da ćemo stupanj sigurnosti probati povećati na način da ćemo bit u prvom registru s lijeve strane izračunavati pomoću neke nelinearne funkcije, kako je prikazano na slici 3.2.



Slika 3.2: Nelinearni registar povratnih pomaka stupnja 4

S obzirom da im performanse nisu baš tako dobre i primjena im nije toliko rašireni, nemaju svoju posebnu podgrupu pa ih se jednostavno svrstava u općenitu grupu registara povratnih pomaka, te koriste i skraćenicu te grupe FSR. Nelinearne registre povratnih pomaka možemo definirati na vrlo sličan način kao i linearne, imaju n registara označenih na isti način s lijeva na desno s $s_{n-1}, s_{n-2}, \ldots, s_0$, nadalje imaju povratnu petlju koju isto čine povratni koeficijenti, broj n je i u ovom slučaju stupanj FSRa. Stanje bitova u registrima u nekom trenutku i ovdje označavamo s st_t, nulto stanje s st_0 i također ga možemo dobiti pomoću nekog *Init* algoritma, gotovo da je i ažuriranje bitova u iteracijama isto. Prvih $n-1$ bitova se kao i kod LRFSa pomiče u desno za jedno mjesto, vrijednost koja se vraća je ista, tj. bit na poziciji s_0, jedina razlika su načini generiranja novog bita koji dolazi na prvo mjesto s lijeva i to je ono u čemu u biti i jest razlika između linearnih registara povratnih pomaka i nelinearnih registara povratnih pomaka. Sada ako uzmemo situaciju koju smo imali i kod LFSR, u trenutku t imamo stanje st_t onda u $t+1$-oj iteraciji ćemo imati

$$st_{t+1} = (s_{n-1}^{(t+1)}, s_{n-2}^{(t+1)}, \ldots, s_0^{(t+1)}),$$

gdje je za neku nelinearnu funkciju g

$$\begin{aligned} s_i^{(t+1)} &:= s_i^{(t)}, \qquad i = 0, \ldots, n-2 \\ s_{n-1}^{(t+1)} &:= g(s_0^{(t)}, s_1^{(t)}, \ldots, s_{n-1}^{(t)}). \end{aligned} \tag{3.4}$$

Već smo spomenuli činjenice zašto nelinearni registri povratnih pomaka nemaju raširenu uporabu, drugi način kako poboljšati sigurnost FSRa je da se uvede nelinearnost u izlaznu vrijednosti algoritma 1. To možemo učiniti na neki od sljedećih načina:

- *GetBites* algoritam pri svakoj iteraciji za y ne vraća samo bit koji je ispao iz registra, nego vrati vrijednost koja se dobije pomoću neke nelinearne funkcije koja za parametre uzima neke ili sve bitove iz stanja koje je na ulazu u *GetBites* algoritam.
- imamo više LFSRa koji rade paralelno i y vrijednosti, koje dobijemo pomoću svakog od *GetBites* algoritama u svakoj iteraciji, iskoristimo kako bi na neki nelinearan način generirali jedan izlazni bit y_{all}, ovdje pojedini LFSRovi ne moraju imati isti stupanj, štoviše poželjno je da imaju različiti.

Jedna od poznatijih šifri nastala na tezama koje smo upravo nabrojali u ove dvije točke je Trivium (vidi sliku 3.3, kreirana od strane Christophe De Cannière i Bart Preneel, a nastala je kao rješenje koje je dano za natječaj za računalne sklopove s ograničenim resursima na eSTREAM natjecanju organiziranom od strane European efforts organizacije 2008. godine. Trivium se sastoji od 3 spojena nelinearna registra povratnih pomaka koje ćemo radi jednostavnosti daljnjih opisa označiti s A, B i C, svaki od njih je redom stupnja 93, 84 i 111, a stanje st_t u trenutku t se sastoji od 288 bitova što je ustvari stanje ova 3 FSRa zajedno. Pri svakoj iteraciji *GetBits* algoritam pojedinog FSRa za y ne vraća isključivo bit na poziciju s_0^t nego se njegova vrijednost izračuna na sljedeći način:

$$y_i = s_0^{(t)} \oplus s_j^{(t)}, \qquad \text{gdje je } s_j^{(t)} \text{ neki od preostalih bitova iz registra.}$$

Novi bit (tj. bit $s_{n-1}^{(t+1)}$) u pojedinom registru računa se pomoću jednog bita iz toga registra i određenog broja bitova iz nekog od preostala dva registra, a funkcija pomoću koje se računa je nelinearna. Nulto stanje u pojedinom FSRu se ne dobiva izravno pomoću *Init* algoritma, nego nam on služi samo kako bi popunio registre FSRova A, B i C pomoću bitova iz ključa i inicijalnog vektora, koji se oba sastoje od po 80 bitova, na sljedeći način:

- kako je ključ isto što i povratna petlja 80 bitova ključa ćemo označiti s $c_0, \ldots c_{79}$, registar FSRa A popunjava se na način:

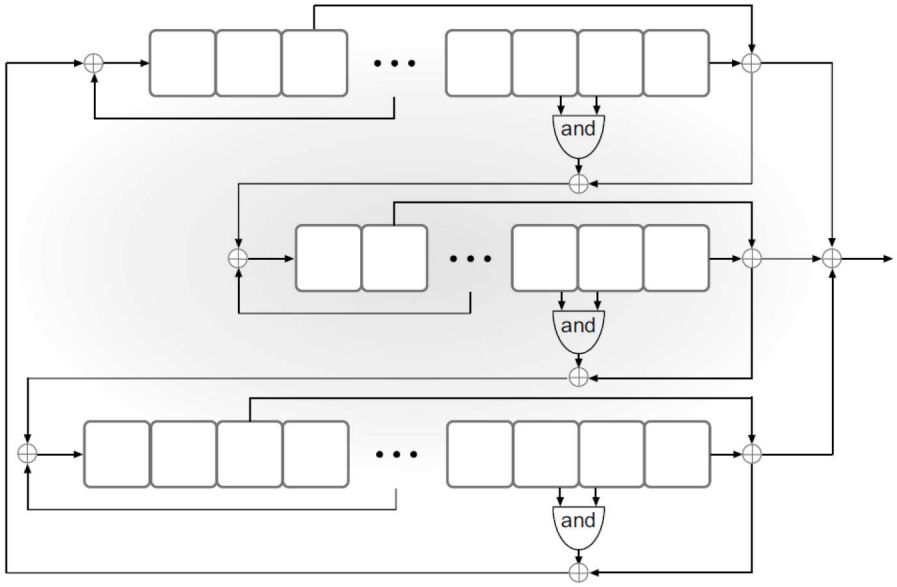
$$s_{14+i} = c_i, \qquad j = 0, 1, \ldots, 79$$

- 80 bitova inicijalnoga vektora ćemo označiti s $v_0, \ldots, v_{79}$, registar FSRa B popunjava se na način:

$$s_{4+i} = v_i, \qquad j = 0, 1, \ldots, 79$$

- u registru FSRa C imati ćemo $s_0 = 1$, $s_1 = 1$ i $s_2 = 1$
- svi ostali registri, koji do sada nisu popunjeni, u sva tri FSRa postavljaju se na bit 0.

Kada se popunu registri u FSRovima A, B i C s *Init* algoritmom onda se pokrene "krnji" *GetBites* algoritam (krnji u smislu da nam nije potrebno spremati vrijednost koju vraćaju FSRovi u svakoj iteraciji, nego samo trenutno stanje) $4 \cdot 288$ puta i zadnje dobiveno stanje bude uzeto za nulto stanje.



Slika 3.3: Trivium ([4], Figure 6.2)

3.2 RC4

RC4 je jedna od najpoznatijih protočnih šifri koja je do nedavno imala primjenu u stvarnome svijetu, čak što više koristila ju je gotovo pa cijela planeta zahvaljujući tome što je korištena kao dio sustava za enkripciju u jako poznatim mrežnim slojevima kao što su TLS i SSL, koji su pružali sigurnost u mrežnim sustavima (npr. WEP protokolu koji je definiran kao sigurnosni protokol za podatke koji idu preko Wi-Fia, ili u WPA protoklu koji je nasljednik WEPa) i web preglednicima (npr. Chrom, Firefox, Internet Explorer, Edge itd.). Osmišljena je od strane djelatnika MIT-a Rona Rivesta , sada već daleke 1987. godine, ime joj je zapravo skraćenica od riječi Ronova šifra (eng. Ron's Cipher) i u početku je bila zaštićena od javnosti kao privatno vlasništvo tvrtke RSA Data Security, Inc. Međutim u listopadu 1994. godine dolazi do afere u kojoj je procurio izvorni kod RC4 algoritma tako što je poslan svim korisnicima iz Cypherpunks mail list, koji su ubrzo iskoristili situaciju i podijelili kod s gotovo svim korisnicima interneta po cijelome svijetu. Već tada je bila uzdrmana sigurnost algoritma, ali se on nastavlja i dalje koristiti kao bitna komponenta u svijetu interneta i to sve do sječnja 2016. godine kada s novim verzijama protokola i mrežnih slojeva izlazi iz uporabe (i dalje je u uporabi tamo gdje su ostale stare verzije). Za razliku od LFSRa koji je dobar ako je implementiran na asembleru, RC4 je puno brži u svijetu softwarea što je danas najčešći slučaj kada se imeplementira neki novi internetski produkt. Stanje RC4 je niz S koji se sastoji od 256 bajtova i kojima je u svakom trenutku spremljena neka od permutacija elemenata iz skupa $0, \ldots, 255$, te dodatna dva bajta $i, j \in 0, \ldots, 255$. U ovom potpoglavlju nije bio cilj teorijski opisivati ovaj algoritam, te načine i mogućnosti napada na njega, više o tome možete pronaći u [4], potpoglavlje 6.1.4. Za kraj ovoga poglavlja pokazati ćemo još kako izgledaju *Init* i *GetBites* algoritam za RC4.

Algoritam 3 ([4], Algorithm 6.1) *Init* algoritam za RC4

Input: 16 bajtni ključ k
Output: Inicijalno stanje $(S,\ i,\ j)$

1: **for** $i \leftarrow 0, \ldots, 255$ **do**
2: $\quad S[i] \leftarrow i$
3: $\quad k[i] \leftarrow k[i \bmod 256]$
4: $j \leftarrow 0$
5: **for** $i \leftarrow 0, \ldots, 255$ **do**
6: $\quad j \leftarrow j + S[i] + k[i]$ ▷ zabrajanje se radi modulo 256
7: $\quad$ **switch** $S[i]$ i $S[j]$
8: $i \leftarrow 0$
9: $j \leftarrow 0$

return: $(S,\ i,\ j)$

Algoritam 4 ([4], Algorithm 6.2) *GetBites* algoritam za RC4

Input: Trenutno stanje $(S,\ i,\ j)$
Output: Bajt y, ažurirano stanje $(S,\ i,\ j)$
(Napomena: sva zabrajanje se rade modulo 256)

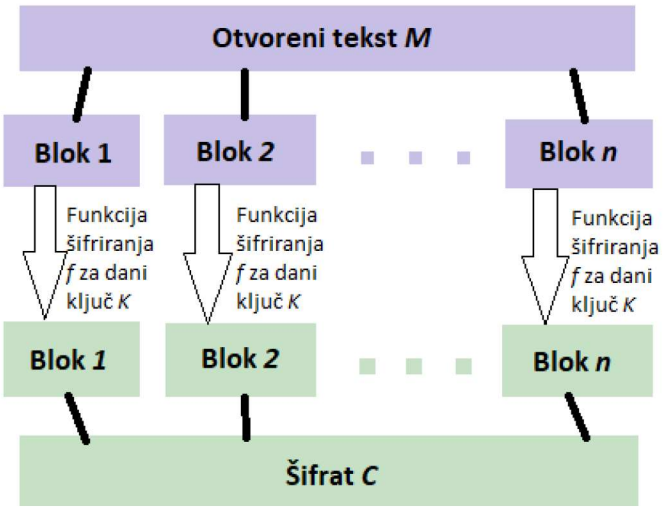
1: $i \leftarrow i + 1$ ▷ zabrajanje se radi modulo 256
2: $j \leftarrow j + S[j]$ ▷ zabrajanje se radi modulo 256
3: **switch** $S[i]$ i $S[j]$
4: $t \leftarrow S[i] + S[j]$ ▷ zabrajanje se radi modulo 256
5: $y \leftarrow S[t]$

return: $(S,\ i,\ j),\ y$

4 | Blokovne šifre

U prošlom poglavlju smo se upoznali s šifriranjem otvorenog teksta znak po znak, a u ovom ćemo upoznati način šifriranja koji obrađuje otvoreni tekst u blokovima koji su točno određene veličine. Postavlja se pitanje, što u slučaju kada je veličina bloka jednaka 1? Je li to poseban slučaj blokovnih šifri koje onda postaju protočne šifre zato što bi se i u ovom slučaju svaki element obrađivao posebno. Odgovor je, dakako, negativan, tj. blokovne šifre s veličinom bloka jednakom 1 nisu protočne šifre nego se one onda zovu supstitucijske šifre i kao takve se ne koriste u stvarnom svijetu, štoviše blokovi veličine manje od 64 bita nisu primjenjivi u praksi zbog svoje jako niske sigurnosti. Blokovne šifre mogu biti s tajnim i s javnim ključem kao što je bio slučaj i s protočnim šiframa gdje smo već jednom istaknuli kako će se u ovome radu obrađivati isključivo šifre s tajnim (simetričnim) ključem. Općenito, kada pričamo o algoritmima za šifriranje sa simetričnim ključem većina njih spada u vrstu blokovnih šifri, stoga je dana sljedeća definicija za sve blokovne šifre općenito.

Definicija 4. ([11], *Potpoglavlje 3.1*) *Neka je $\mathcal{A}$ alfabet veličine q. Funkcija $F : \mathcal{A}^t \times \mathcal{K} \to \mathcal{A}^t$ je n-bitna blokovna šifra takva da za neki ključ $K = k_1, k_2, \ldots, k_l \in \mathcal{K}$ elemente otvorenog teksta $M = m_1, m_2, \ldots, m_t$, $m_i \in \mathcal{A}$, $i = 1, \ldots t$ podijeli u blokove veličine n znakova i šifrira blok po blok u šifrat $C = c_1, c_2, \ldots, c_t$, $c_i \in \mathcal{A}$, $i = 1, \ldots, t$. Ako s $D : \mathcal{A}^t \times \mathcal{K} \to \mathcal{A}^t$ označimo inverz funkcije F onda pomoću D možemo dešifrirati šifrat C u otvoreni tekst M.*



Slika 4.1: Blokovna šifra

Kako je cilj ovog poglavlja opisati blokovne šifre koje se koriste u praksi, za alfabet se ne može uzeti bilo što. Znamo kako se sva naša računala pokreću na električnu energiju pomoću koje računalni sklopovi obrađuje programske kodove, ma koliko oni zahtjevni bili. To odrađuju tako što razlikuju dva stanja, ima struje (označit ćemo ga s 1) i nema struje (označite ćemo ga s 0), u tom slučaju alfabet $\mathcal{A}$ će imati samo dva elementa $\{0,1\}$, pa prema tome i elementi ključa mogu imati za vrijednost samo 0 ili 1. Kako je ključ $K \in \mathcal{K}$ vrijednost koja bude jednom određena za cijeli proces šifriranje nekog otvorenog teksta, tada pomoću definicije 4 definiramo funkciju jedne varijable $F_K : \mathcal{A}^t \to \mathcal{A}^t$, a tako definirana funkcija gdje je $\mathcal{A} = \{0,1\}$ je permutacija i veličina bloka joj je t. Blokovne šifre se dizajniraju kako bi bile takve da ih je gotovo nemoguće razlikovati od jakih pseudoslučajnih permutacija (definiciju jakih pseudoslučajnih permutacija možete pronaći u [4] potpoglavlje 3.5.1, str. 77). Takav način dizajniranja omogućuje dokazivanje sigurnosti za kriptosustave koji su građeni od blokovnih šifri i jasno ističe ispunjenje nužnih uvjeta blokovnih šifri, a ove tvrdnje su potkrijepljene s dokazima iz prakse. Na natječaju za prijedlog kako konstruirati AES algoritma pod točkom u kojoj su opisani uvjeti koje treba ispunjavati algoritam, između ostalog, navedeni su sigurnost algoritma i to da se izlazna vrijednost ne može razlikovati od slučajne permutacije. Blokovne šifre se smatraju dobro dizajniranima ako najbolji poznati napad ima vremensku složenost približno jednaku vremenskoj složenosti pronalaska ključa metodom grube sile, današnje moderne blokovne šifre su dizajnirane tako da zadovoljavaju i više sigurnosne standarde. Najjednostavniji algoritam blokovnog šifriranja uzima u obzir sve moguće permutacije otvorenog teksta kojima se dani otvoreni tekst može zamijeniti, a da bi se sve moguće permutacije otvorenog teksta mogle prikazati treba $n \cdot 2^n$ bitova što i nije baš efikasno za šifre s većim blokovima, naprimjer moderne blokovne šifre imaju blokove veličine 128 bita ili više. Stoga, najveći je izazov bio izraditi što efikasniji algoritam blokovne šifre koji se i dalje ponaša kao slučajna permutacija, tj. da za dva otvorena teksta na ulazu koja se razlikuju u samo jednom bitu dobijemo dva neovisna rezultata. Kako bi dobili takav algoritam poslužit će nam paradigma zbunjenosti i difuzije koju je osmislio Claude Elwood Shannon, američki matematičar, kriptograf i inžinjer elektrotehnike, kojeg su zvali ocem teorije informacija.

Paradigma zbunjenosti i difuzije (eng. confusion-diffusion paradigm) sastoji se od dva dijela. Prvi dio je uvođenje svojstva zbunjenosti u permutaciju F veličine bloka n, $n \in \mathbb{N}$, a svojstvo zbunjenosti se dobije tako što se F konstruira pomoću (pseudo)slučajnih permutacija f_i s blokovima veličine znatno manje od n. Kako bi uvođenje svojstva zbunjenosti u permutaciju F bilo lakše shvatiti dan je sljedeći jednostavan primjer.

Primjer 2. *Neka je definirana permutacija F_K s veličinom bloka* 128 *bita i ključem K takvim da ćemo dobiti* 16 *permutacija s blokom veličine* 8 *bita, označenih redom s $f_1, \ldots, f_{16}$. Dani ulazni otvoreni tekst x podijelit ćemo na blokove od po* 8 *bitova ($x = x_1, \ldots, x_{16}$), a permutacija F_k se izračuna na sljedeći način:*

$$F_K(x) = f_1(x_1) \parallel \ldots \parallel f_{16}(x_{16}). \tag{4.1}$$

Već spomenuti najjednostavniji blokovni algoritam kako bi izračunao $F_k(x)$ bez svojstva zbunjivanja zapisat će sve moguće permutacije otvorenog teksta x, što znači da nam za blok veličine 128 bitova treba $128 \cdot 2^{128}$ bitova prostora u memoriji računala. S druge strane, da bi zapisao sve moguće permutacije pojedinog x_i taj isti algoritam treba $8 \cdot 2^8$ bitova, kada se to pomnoži s 16 permutacija dobijemo $16 \cdot 8 \cdot 2^8$, što je malo više od 3 kilobajta. Dakle, svojstvom zbunjenosti dobio se efikasniji algoritam, ali je očito da F_K definiran kao u primjeru 2 nije pseudoslučajan. Na primjer, ako se uzme otvoreni tekst $x = x_1, x_2, \ldots, x_{128}$, tada će permutacija biti dana s $F_K(x)$, ako se u x promijeni bit na nekom od mjesta s indeksom od 1 do 8 prema (4.1) to će dovesti do promjene u nekom od samo prvih 8 bitova u permutaciji $F_K(x)$. Što znači da promjenama u ulaznim vrijednostima znamo točno koje ćemo promjene izazvati u izlaznim vrijednostima. Kako bi se postiglo da F_K bude pseudoslučajna permutacija poslužiti će drugi dio paradigme, koji se zove difuzija. Difuzija koristi transpozicijsku šifru kako bi još jednom permutirala $F_K(x)$, te na taj način omogući da promjena jednog bita ulazne vrijednosti može utjecati na bilo koji bit izlazne vrijednosti. Svojstva zbunjenosti i difuzije zajedno nazivamo runda šifriranja, a kako bi se dobila što je moguća veća razina sigurnosti u blok šiframa runde šifriranja ponavljaju se više puta.

4.1 Supstitucijsko-permutacijska mreža

Prva vrsta simetričnih blokovnih šifri koju ćemo upoznati je supstitucijsko-permutacijska mreža (eng. substitution-permutation networks, skraćeno SPN) koja se zasniva na spomenutoj paradigmi zbunjenosti i konfuzije. Za razliku od slučaja u primjeru 2 gdje smo svojstvo zbunjenosti primijenili na način da smo permutaciju f_i odabrali iz skupa svih mogućih permutacija njezine ulazne vrijednosti x_i, u ovome slučaju spomenute permutacije f_i biti će definirane na točno određeni način. Dakle, svaki f_i je točno definirana permutacija S, te za dani ključ k definiramo f_i na sljedeći način:

$$f_i(x_i) = S(k \oplus x_i), \qquad \text{za neki ulaz } x_i. \tag{4.2}$$

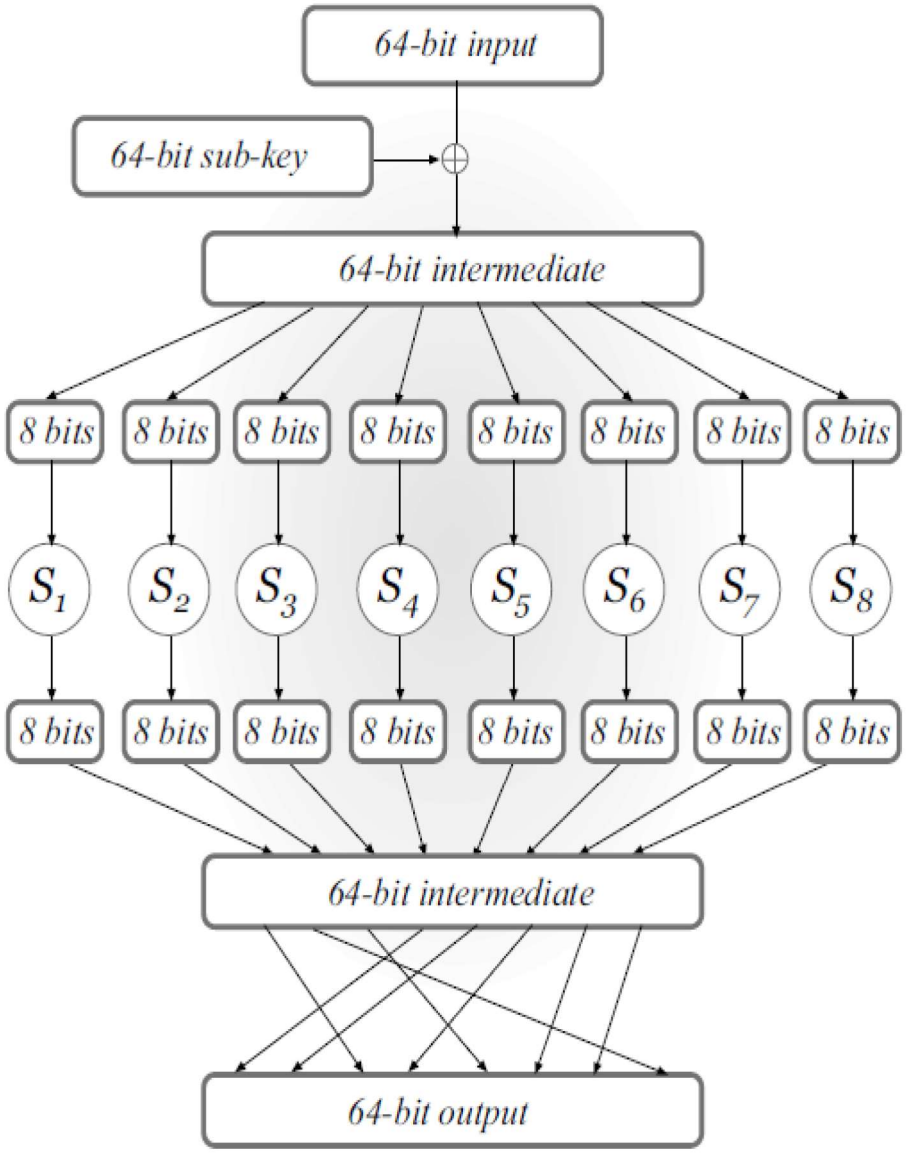
Permutacija S se popularno zove još i S-kutija. Potpuna definicija i svi kriteriji konstruiranja funkcije S su ono što se čuva u tajnosti od napadača i ono što čini sigurnost blokovnih šifri. Jedna iteracija SPNa sastoji se od runde šifriranja modificirane S-kutijama i dodatnog koraka kojeg zovemo miješanje s ključem (eng. key mixing) koji se događa prije same runde šifriranja, pa je algoritam jedne iteracije SPNa definiran na sljedeći način.

Algoritam 5 ([4], Potpoglavlje 6.2.1) Algoritam jedne iteracije SPNa

Input: podključ k_i, ulazna vrijednost x
Output: izlazna vrijednost y

1: $x \leftarrow x \oplus k_i$
2: $S_k(x) \leftarrow S_1(x_1) \parallel \ldots \parallel S_n(x_n)$, za $n \in \mathbb{N}$, $x = x_1 \parallel \ldots \parallel x_n$
3: $y \leftarrow \pi(S_k(x))$, za danu permutaciju π

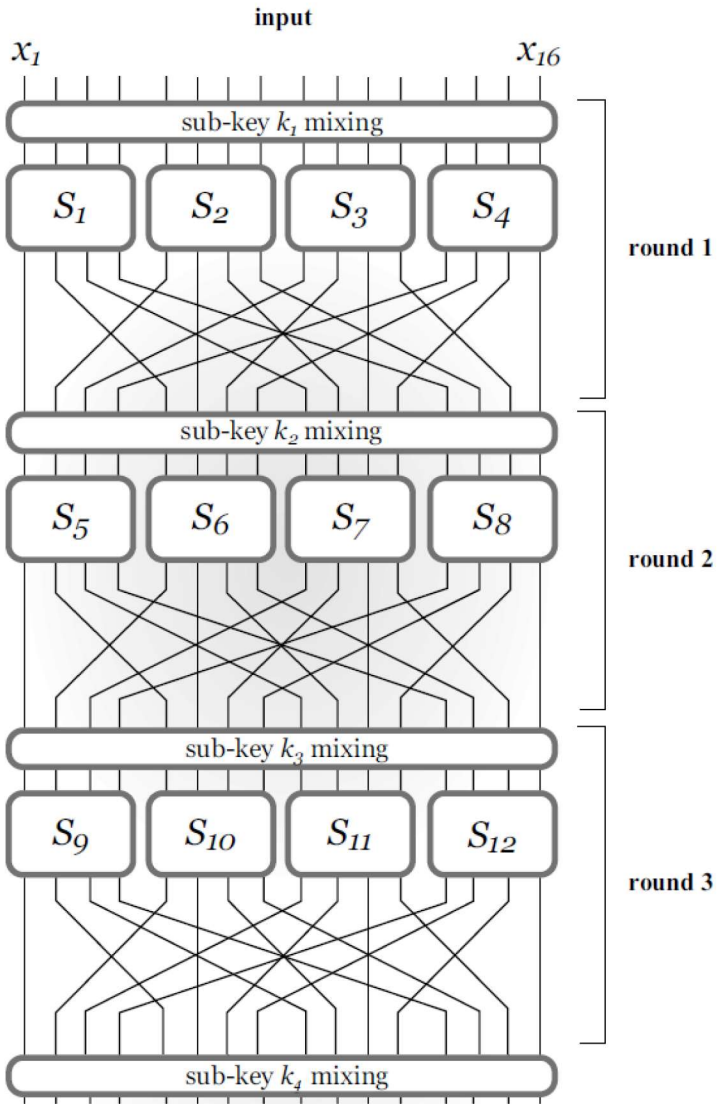
return: y



Slika 4.2: Iteracija u supstitucijsko-permutacijskoj mreži ([4], Figure 6.3)

Zbog lakšeg razumijevanja algoritma 5 na prethodnoj slici možemo vidjeti prikaz jedne iteracije SPNa s ulaznom vrijednošću x veličine 64 bita, podključem k_i veličine također 64 bita i 8 S-kutija kojima je veličina ulazne i izlazne vrijednosti 8 bita. Dakle, SPN je konstruiran od više takvih iteracija i završnog koraka koji je zapravo još jedno miješanje ključem.

Ključ koji se koristi u svakoj od iteracija i završnom koraku je ustvari podključ ključa K blokovne šifre SPN. Ključa K nazivamo još i glavni ključ, a prema zadanim pravilima za generiranje podključeva iz glavnog ključa izvodimo podključeve k_i za svaku pojedinu iteraciju i završni korak, pa slijedi da jedan SPN s n iteracija ima $n + 1$ generiranih podključeva. Prikaz SPNa s 3 iteracije, gdje se u svakoj iteraciji koriste 4 S-kutije s ulaznim i izlaznim vrijednostima veličina 4 bita, završnom permutacijom i 4 izvedena podključa prikazan je na slici 4.3.



Slika 4.3: Supstitucijsko-permutacija mreža

Broj iteracija, dizajn S-kutija, način na koji se radi difuzija i pravilo za generiranje podključeva su svojstva kojima se može poboljšavati sigurnost samog SPNa. Dakako, prva pomisao bi bila da je SPN sigurniji što ima veći broj iteracija što i jest točno, ali to ne znači da broj iteracija može ići u beskonačnost zato što treba uzeti u obzir mogućnosti nekog računalnog sustava na kojemu se algoritma izvodi, te na vrijeme izvršavanja tog algoritma. Stoga, treba biti vrlo pažljiv s konstruiranjem navedenih komponenti SPNa, zato što je glavni cilj postići takvu šifru da promjena jednog bita ulazne vrijednosti ima jednake mogućnosti utjecaja na sve bitove u izlaznoj vrijednosti. Za postizanje tog cilja koristi se jedan od poznatijih principa za dizajn S-kutija i transpozicijske šifre u svojstvu difuzije, a nazivamo ga efekt lavine. Kako bi se moglo reći da neki SPN koristi efekt lavine moraju vrijediti sljedeća tri svojstva:

- promjena jednog bita ulazne vrijednosti S-kutije utječe na minimalno dva bita njezine izlazne vrijednosti

- difuzija mora osigurati da bitovi izlazne vrijednosti jedne S-kutije u iteraciji i budu rasprostranjeni u ulazne vrijednosti više S-kutija u iteraciji $i + 1$.
- u SPNu mora postojati dovoljan broj iteracija

Za dani ključ K lako je dešifrirati SPN, dovoljno je samo pokazati kako se napravi inverz zadnjeg koraka SPNa i dešifrira jedna iteracija, dok ostale iteracije onda idu analogno. Ako dodamo efekt lavine i u funkciju dešifriranja, što znači da promjena jednog bita u izlaznoj vrijednosti S-kutije dovodi do promjene u najmanje dva bita ulazne vrijednosti, otežati ćemo dešfriranje ali ćemo zato znatno povećati sigurnost šifre. Primjere napada na SPNove s manjim brojem iteracija možete pronaći u [4], potpoglavlje 6.2.1, str. 209-211.

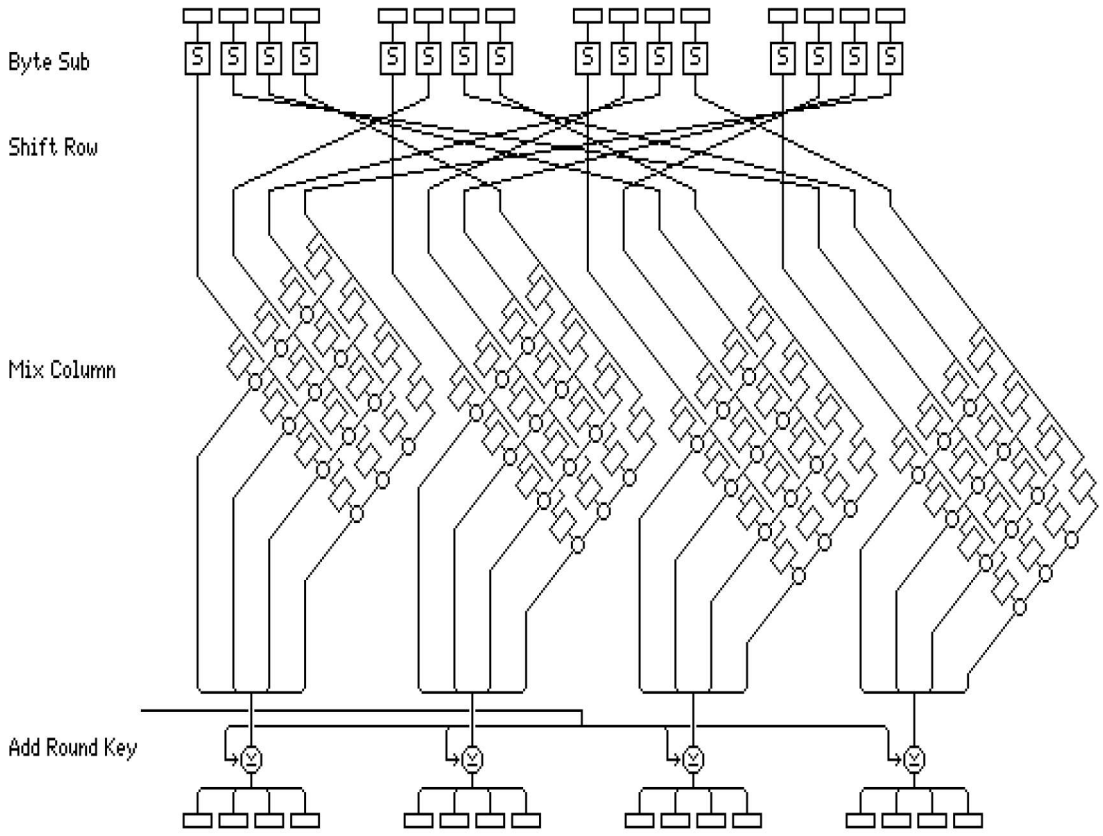
4.2 AES

Danas najpoznatija blokovna šifra koja ima strukturu sličnu supstitucijsko-permutacijskoj mreži je Rijndael algoritam kojega su dizajnirali belgijski kriptografi Vincent Rijmen i Joan Daemen, inspirirani svojim Squere algoritmom koji je bio pravi SPN s 8 iteracija. Ovaj algoritam poznat je još i pod nazivom Advanced Encryption Standard (skrećano AES) koji je dobio pobjedivši na natječaju, koji je objavljen 1997. godine od strane nacionalnog instituta za standarde i tehnologiju Sjedinjenih Američkih Država (eng. United States National Institute of Standards and Technology, skraćeno NIST). Natječaj je otvoren kako bi se pronašao algoritam koji će zamijeniti DES (o DESu će biti rečeno nešto više u idućem poglavlju), a pobjednik je biran među 15 algoritama koji su zadovoljili kriterije natječaja. Sam postupak natječaja trajao je gotovo 4 godine i bio je organiziran u 2 kruga. U prvome krugu su bile dvije radionice nakon kojih je odlučeno koji su to pet najboljih algoritama, u posljednjem krugu odabranih pet finalista je podvrgnuto dodatnoj analizi, a pobjednik je proglašen u listopadu 2000. godine. Prema izvještaju NISTa svih 5 finalista bili su izvrsni algoritmi, ali u korist Rijandeal algoritma je odlučeno zbog njegove fleksibilnosti, efikasnosti i performansi koje je pokazao na različitim vrstama računalnog sklopovlja. Proces izbora najboljeg algoritma je bio zanimljiv i zbog činjenice da su na kriptoanalizi konkurentskih algoritama i pronalaženju napada za iste mogli raditi i sami timovi čiji su algoritmi bili kandidati. Dakle, može se vjerovati u visoku sigurnost AESa s obzirom da je prošao razne analize i pokušaje pronalaženja učinkovitog napada ne samo od strane komisije iz NISTa i javnosti, nego i samih timova koji su imali svoje algoritme kao kandidate.

AES je 128-bitna blokovna šifra kojoj broj iteracija ovisi o veličini ključa, ključ može biti veličine 128, 192 ili 256 bita, pa je ovisno o tome broj iteracija redom 10, 12 i 14. Veličina ključa, osim što utječe na broj iteracija, utječe i na pravilo za generiranje podključeva. Zanimljiva je činjenica da sve operacije u AESu se izvodi na razini bajtova, u ovome radu nećemo ulaziti u dublju analizu što leži iza ove činjenice. Specifičnost AES je ta da ima ulaznu vrijednost, izlaznu vrijednost i trenutno stanje iste veličine sve od po 128 bit. Dodatno, za trenutno stanje specifično

je još da je ono u svakom trenutku procesa šifriranja spremljeno u matrici 4×4 na način da se bajtovima prvo popunjavaju stupci ($4 \cdot 4 = 16$, $16 \cdot 8 = 128$, 1 bajt $= 8$ bitova). Također, i glavni ključ AESa se sprema u matricu s 4 retka i brojem stupaca ovisnim o veličini samoga glavnog ključa, a broj stupaca može biti 4, 6 i 8 za veličinu ključa redom 128, 192 ili 256, popunjavanje matrice se izvodi na isti način kao i kod trenutnog stanja, prvo se popunjaju stupci. U potpoglavlju 4.1 rečeno je kako algoritmi koji spadaju u grupu SPN algoritama u koju spada i sam AES za proces šifriranja trebaju ($broj\ iteracija\ + 1$) podključeva. Sama rutina za generiranja svih podključeva iz glavnog ključa zove se proširenje ključa (eng. KeyExpansion), više detalja o njoj možete pronaći na [13] potpoglavlje 5.2. Osim KeyExpansion rutine AES se dodatno sastoji od još 4 glavne rutine, koje sljedećom redoslijedom zapravo i čine jednu iteraciju algoritma

- SubBytes
- ShiftRows
- MixColumns
- AddRoundKey.



Slika 4.4: Jedna iteracija AESa,
([8] potpoglavlje The Advanced Encryption Standard, str. 449)

Sve su iteracije jednake osim posljednje. Posljednja iteracija nema rutinu MixColumns i za potpuno definiranje AESa, koji je prikazan algoritmom 6, nedostaje nam još inicijalno pozivanje rutine AddRoundKey prije generiranja svih podključeva. Više o rutinama u algoritmu možete potražiti u [13]. Ako zamislimo procedure ShiftRows i MixColumns kao jednu proceduru tada bi ona mogla predstavljati difuziju koju smo spomenuli u paradigmi zbunjenosti i difuzije, a SubBytes bi bio dio koji donosi svojstvo zbunjenosti. Dakle, vrlo lako može se uočiti da vrijedi prva rečenica iz ovoga potpoglavlja, a to je da AES ima sličnu strukturu kao SPN. Ne možemo reći da je ista zbog svojstva difuzije koje je kod AESa puno složenije. Za kraj ćemo nešto reći i o sigurnosti AESa. Dosta je toga već rečeno i pri samom opisu procesa izbora AES, no treba napomenuti da do danas nije pronađena dobra metoda za napad koja će biti značajno bolja od klasične metode pokušaja otkrivanja ključa.

Algoritam 6 ([5], Potpoglavlje 5.9)AES

Input: ulazna vrijednost x, glavni ključ K, broj iteracija m
Output: izlazna vrijednost y

$k_1, \ldots, k_{m+1} \leftarrow KeyExpansion(K, n)$ ▷ n broj stupaca matrice glavnog ključa
$x \leftarrow AddRoundKey(x, k_1)$
for $i \leftarrow 1,\ldots,m-1$ **do**
    $x \leftarrow SubBytes(x)$
    $x \leftarrow ShiftRows(x)$
    $x \leftarrow MixColumns(x)$
    $x \leftarrow AddRoundKey(x, k_{i+1})$
$x \leftarrow SubBytes(x)$
$x \leftarrow ShiftRows(x)$
$y \leftarrow AddRoundKey(x, k_{n+1})$

return: y

4.3 Feistel mreža

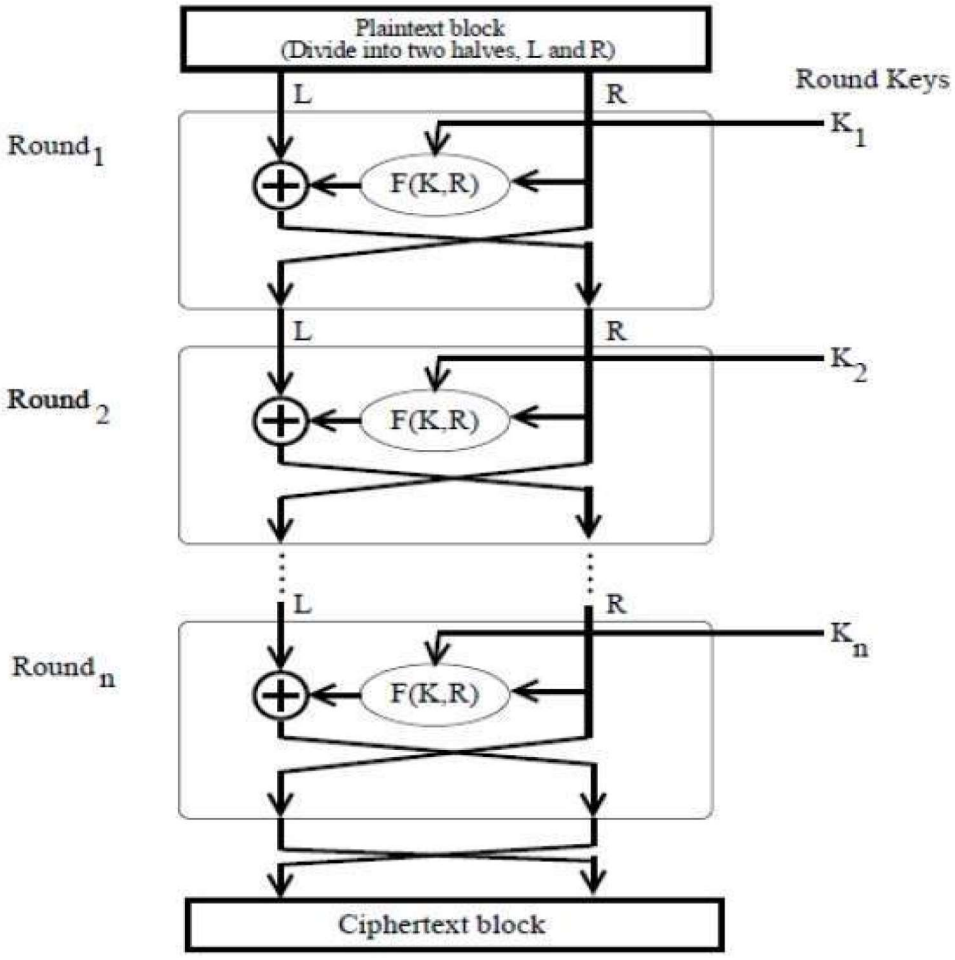
Još jedna blokovna šifra koja služi kao baza za dizajn mnogih drugih blokovnih šifri je Fiestel mreža (eng. Fiestel network), iako sama baš i nije specifična blokovna šifra. Osmislio ju je njemačko-američki kriptograf Horst Feistel kao zaposlenik IBM-a, a šifra je u početku bila napravljena u komercijalne svrhe kao dio Lucifer šifre. Jedna od najpoznatijih blokovnih šifri koje su izgrađene prema Feistel mreži je DES, o kojem će više biti rečeno u idućem poglavlju. Za razliku od SPNa i funkcija u S-kutijama Feistel mreže nude drugačiji pristup izgradnje blokovnih šifri samim time što funkcija f_i, koja se koristi kod izračunavanja novog desnog dijela bloka ne mora biti inverzna, štoviše može biti bilo kakvog oblika. Na taj način Feistel mreže omogućavaju izgradnju šifri koje se mogu dešifrirati pomoću komponenti koje nisu invertibilne, što je dobar znak za sigurnost same šifre zato što smo dobili strukturirani sustav iz nestrukturiranih komponenti. Kao što je bio slučaj i kod SPNa, Feistel mreže se izvode u više iteracija i podključ svake iteracije je izveden iz glavnog ključa šifre. Također se zahtijeva da ulazni blok bude

parne veličine, a najčešće je i broj iteracija paran, te su dovoljne barem 3 iteracije kako bi Feistel izgledao kao pseudoslučajna permutacija. Veći broj iteracija osigurava veću sigurnost, ali uzrokuje i pad efikasnosti zato se u ovisnosti od potrebe određuje broj iteracija. Sam algoritam se izvodi na način da se ulazna vrijednost podijeli na dva jednaka dijela (lijevi i desni), a onda se u svakoj iteraciji jednako generiraju novi lijevi i desni dio na sljedeći način:

$$\begin{aligned} L_i &:= R_{i-1} \\ R_i &:= L_{i-1} \oplus f_i(R_{i-1}) \end{aligned} \tag{4.3}$$

i za kraj algoritma lijeva i desna strana se još jednom zamijene i dobijemo šifrat. Lako se može uočiti da se R_i mogu promatrati kao elementi niza čiji su početni elementi lijeva i desna strana ulazne vrijednosti, a svaki novi element se izračunava pomoću prethodna dva. Ako ulaznu vrijednost podijelimo na dva jednaka dijela, te ih označimo s R_{-1} i R_0 vrlo lako se može eliminirati L_{i-1} i L_i iz notacije prikazane u (4.3) te ju prikazati na sljedeći način:

$$R_i := R_{i-2} \oplus f_i(R_{i-1}) \tag{4.4}$$



Slika 4.5: Feistel mreži([6] Figure 2-9)

Grafički prikaz Feistel mreže može se vidjeti na slici 4.5, a sam algoritam zapisan pseudokodom prikazan je u 7. Dešifriranje šifrata dobivenog pomoću Feistel šifre izvodi se istim brojem iteracija kao i samo šifriranje. Koriste se isti potključevi, ali u obrnutom redoslijedu, a stanje svake prethodne iteracije računa se na sljedeći način:

$$\begin{aligned} R_{i-1} &:= L_i \\ L_{i-1} &:= R_i \oplus f_i(R_{i-1}). \end{aligned} \tag{4.5}$$

Algoritam 7 ([4], Potpoglavlje 6.2.2) Feistel mreža s m iteracija

Input: ulazna vrijednost x, glavni ključ K, broj iteracija m
Output: izlazna vrijednost y

1: $k_1, \ldots, k_{m+1} \leftarrow KeyExpansion(K)$
2: $l \leftarrow lenght(x)$
3: $L \leftarrow x[1, l/2]$
4: $R \leftarrow x[l/2+1, l]$
5: **for** $i \leftarrow 1,\ldots,m$ **do**
6: $\quad R' \leftarrow R$
7: $\quad R \leftarrow L \oplus f_i(R, k_i)$
8: $\quad L \leftarrow R'$
9: $y \leftarrow R \parallel L$

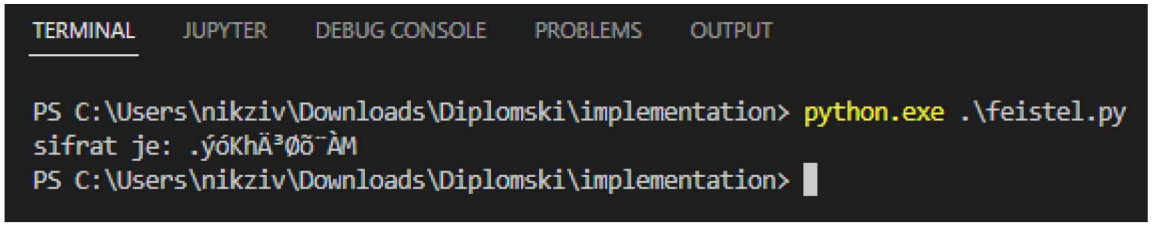
return: y

Jedna od implementacija algoritma 7 u programskom jeziku Python (verzija 3.7.7) može se vidjeti u programskom kodu 4.1. Šifrirana je riječ "kriptografija" ključem "diplomski", broj iteracija u Feistel mreži je 3, a sam šifrat se može vidjeti na slici 4.6. Kako je već naglašeno u ovome radu, za alfabet ćemo uzimati skup $\{0, 1\}$, zbog toga smo također i u implementacije funkcije fiestel, koju možemo vidjeti ispod, u retcima 3 i 6 napravili pretvaranje ulazne vrijednosti i ključa u bitove.

```
1  def feistel(input, key, numberOfRounds):
2      # pretvaranja teksta u bitove
3      inputAsBinary = ''.join(format(ord(i), '08b') for i in input)
4      lengthOfInputAsBinary = len(inputAsBinary)
5      # pretvaranja glavnog kljuca u bitove
6      keyAsBinary = ''.join(format(ord(i), '08b') for i in key)
7      lengthOfKeyAsBinary = len(keyAsBinary)
8      LRlen = lengthOfInputAsBinary // 2
9      lengthOfAllSubkeysBinary = numberOfRounds * LRlen
10     keyIter = int(lengthOfAllSubkeysBinary / lengthOfKeyAsBinary) +
       (lengthOfAllSubkeysBinary % lengthOfKeyAsBinary > 0)
11     # kreirati listu svih podkljuceva pomocu konkatenacije
       stringova
12     k = ''
13     for i in range(keyIter):
14         k = k + keyAsBinary
15     # podijeliti ulaznu vrijednost na dva jednaka dijela
```

```
16      [L, R] = [inputAsBinary[:LRlen],inputAsBinary[LRlen:]]
17      for i in range(numberOfRounds):
18          Rtemp = R
19          f = [int(R[j])^int(k[i*LRlen + j]) for j in range(LRlen)]
20          R = [int(L[j])^int(f[j]) for j in range(LRlen)]
21          L = Rtemp
22      y = R + L
23      y = ''.join(str(y[i]) for i in range(lengthOfInputAsBinary))
24      return ''.join(chr(int(y[i*8:i*8+8],2)) for i in range(
    lengthOfInputAsBinary//8))
25
26  if __name__ == "__main__":
27      print("sifrat je: %s" % feistel("kriptografija", "diplomski",
    3))
```

Programski kod 4.1: Implementacija algoritma 7



Slika 4.6: Ispis na terminalu nakon pokretanja feistel.py skripte

5 | DES

Algoritam Standard šifriranja podataka (eng. The Data Encryption Standard), poznatiji kao DES, spada u skupinu blokovnih šifri, a izdvojili smo ga kao posebno poglavlje zbog njegove povijesne važnosti te jer predstavlja standard pouzdanosti i niz godina je u tome bio jedinstven, te što je bio podvrgnut pomnom ispitivanju od strane kriptografske zajednice kao ni jedan algoritam šifriranja do sada. Razvijen je 1970. godine od strane IBMa u suradnji s Američkom agencijom za sigurnost, poznatom kao NSA (eng. National Security Agency), a 1977. godine je prihvaćen kao Federalni standard za procesuiranje informacija. Usprkos tome što je i nakon toliko godina, u praksi, najbolji napad na DES algoritam iscrpna pretraga za glavnim ključem u skupu svih 2^{56} mogućih ključeva, danas se ipak koristi u svojoj puno snažnijoj verziji poznatoj kao trostruki DES o kojoj ćemo nešto više reći kasnije. Svrha ovog poglavlja nije detaljna analiza algoritma do u najsitnije detalje nego samo površni opisi njegovih glavnih dijelova kako bi smo mogli pokazati i neke primjere napada na DES algoritme (ali samo krnje), te reći nešto i o njegovoj sigurnosti.

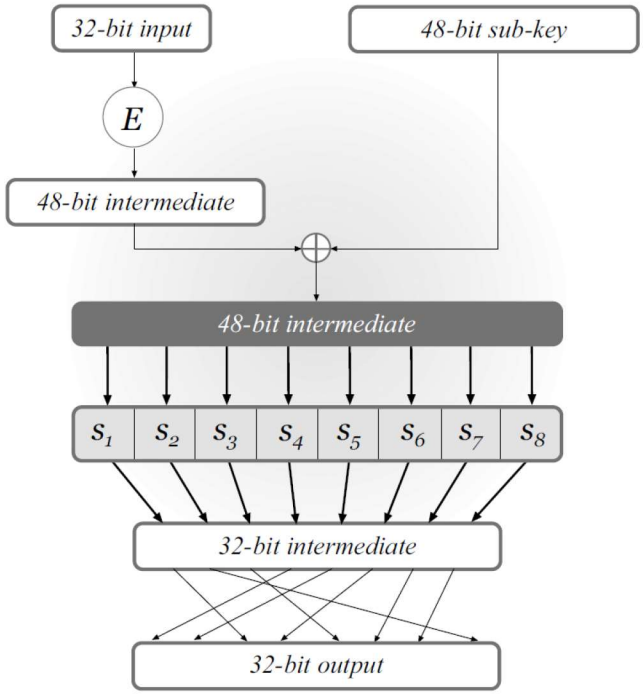
DES je blokovna šifra kojoj je ulazna vrijednost veličine 64 bita, glavni ključ veličine 56 bita, a dizajnirana je način da se ulazni blok permutira početnom permutacijom, nakon čega slijedi 16 iteracija Fiestel mreže i prije nego dobijemo izlaznu vrijednosti svih 64 bita se permutira završnom permutacijom. Dakle, u svakoj iteraciji koristiti ćemo podključ $k_i, i = 1, \ldots, 16$, koji je veličine 48 bita i izveden je iz glavnog ključa, a generira se na način da se glavni ključ inicijalno permutira, a zatim se u svakoj iteraciji podijeli na dva jednaka dijela gdje nam lijevi dio služi za generiranje prva 24 bita, a desni dio za generiranje druga 24 bita podključa k_i. Algoritam svake od iteracija je identičan, te je detaljnije opisan u podpoglavlju 4.3. Razlika DES algoritma u odnosu na Feistel mreže je u funkciji f_i, koja je poznata još i pod nazivom DES mangler funkcija.

DES mangler funkcija je u svojoj osnovi supstitucijsko-permutacijska mreža, a za ulaznu vrijednost uzima polovinu od ulazne vrijednosti DES algoritma što iznosi 32 bita (time dobivamo balansiranu Feistel mrežu) i definirana je na sljedeći način: Ulazni blok se funkcijom proširenja proširuje na veličinu od 48 bita na način da se posljednjih 16 bita udvostručuje, na novo dobivenu vrijednost i na odgovarajući podključ k_i djelujemo operacijom "isključivo ili" te dobijemo međuvrijednost od 48 bita koju podijelimo u 8 grupa po 6 bitova. Svaka grupa je ulazna vrijednost za pojedinu S-kutiju. Svih 8 S-kutija čine jezgru DES mangler funkcije i nisu invertibilne za razliku od S-kutija u supstitucijsko-permutacijskoj mreži. S-

kutije nisu invertibilne zato što im je ulazna vrijednost duljine 6 bitova, a izlazna duljine 4 bita. S-kutije možemo promatrati i kao tablice od 4 retka i 16 stupaca gdje svaka ćelija sadrži podatak veličine 4 bita ($2^6 = 4 \cdot 16$), ulaznu vrijednost možemo promatrati kao indeks za određivanje ćelije iz tablice, a izlazna vrijednost je podatak iz odabrane ćelije. Točna pozicija ćelije se određuje na način da se uzmu prvi i zadnji bit iz ulazne vrijednosti te na taj način odredimo odgovarajući redak, a pomoću 2. do 5. bita odredimo odgovarajući stupac. Od mnogih svojstava koje S-kutija mora zadovoljiti izdvojit ćemo sljedeća tri:

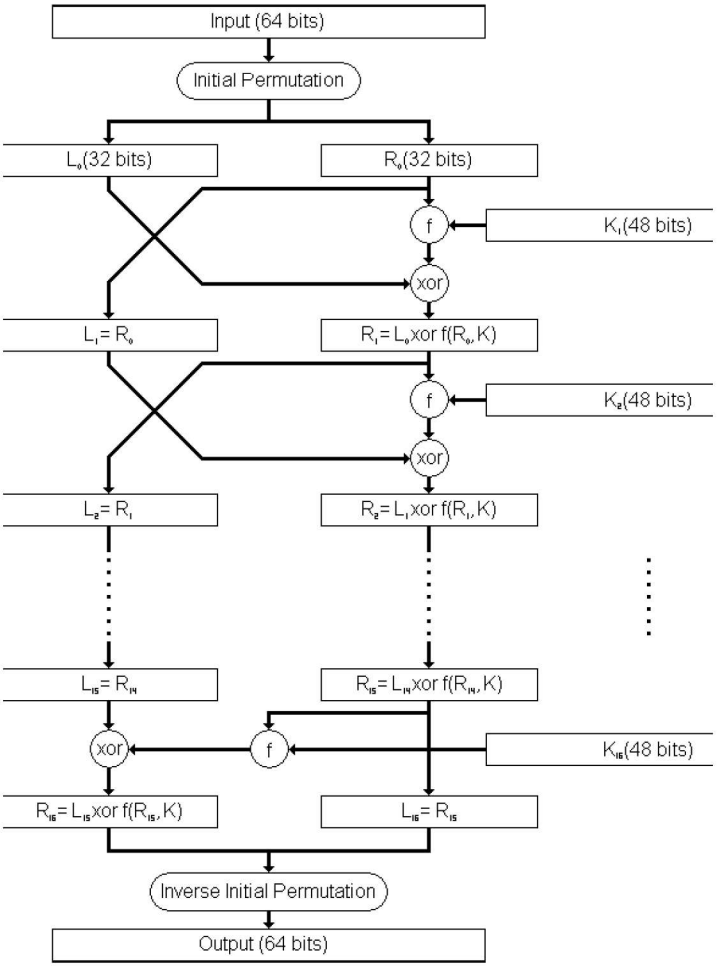
- svaki redak tablice sadrži svaki od 16 mogućih slučajeva niza od 4 bita
- točno 4 ulazne vrijednosti preslikavaju se u svaku od mogućih izlaznih vrijednosti, što slijedi iz prethodnog svojstva
- promjena jednog bita ulazne vrijednosti uvijek mora dovesti do barem dvije promjene u bitovima izlazne vrijednosti.

Kao zadnji korak DES mangler funkcije imamo još završnu permutaciju miješanjem koja je dizajnirana na način da će izlazna vrijednost jedne S-kutije u sljedećoj iteraciji imati utjecaja na ulazne vrijednosti 6 različitih S-kutija. DES algoritam pokazuje izuzetno snažan efekt lavine, te već nakon 8 iteracija imamo slučaj utjecaja na svih 64 bita što se dijelom može vidjeti u primjeru 3. Znajući kako DES algoritam ima 16 iteracija te da efekt lavine nastupa vrlo brzo, nakon samo nekoliko iteracija, dolazimo do zaključka kako će DES algoritam za isti ulazni blok davati naizgled proizvoljne izlazne vrijednosti (šifrate). Treba naglasiti kako je u DES algoritmu sve osim glavnog ključa poznato i javno dostupno.



Slika 5.1: DES mangler funkcija, ([4], Figure 6.6)

Primjer 3. *Neka su dana dva ulazna bloka koja se razlikuju u samo jednom bitu* $x = (L_0, R_0)$ *i* $x' = (L'_0, R'_0)$*, gdje je* $R_0 = R'_0$*, a* L_0 *i* L'_0 *se razlikuju u jednom bitu, a radi jednostavnosti primjera pretpostavit ćemo da bitovi koji se razlikuju nikada ne nalaze u dijelu koji se udvostručuje, također radi lakšeg razumijevanja daljnjih koraka možete pogledati sliku 5.2. Nakon prve iteracije blokovi se i dalje razlikuju u samo jednom bitu, ali je sada ta razlika u 32 bita koji se nalaze na desnoj strani blokova, tj. imamo* $L_1 = L'_1$ *i* $R_1 \neq R'_1$*. U drugoj iteraciji blokovi će se razlikovati u 3 bita, razlika u jednom bitu dolazi iz dijela definicije DES algoritma koji glasi* $L_i := R_{i-1}$*. Do razlike u ostala dva bita koja se nalaze u desnom dijelu bloka dolazi zato što se ulazne vrijednosti* (R_1 *i* R'_1) *za DES mangler funkciju razlikuju u 1 bitu, te će se izlazne vrijednosti razlikovati u najmanje dva bita zbog zadnjeg od 3 gore navedena svojstva S-kutije. Završna permutacija miješanjem u DES mangler funkciji u drugoj iteraciji će razlike u dva bita rasporediti na način da u trećoj iteraciji budu u ulaznim vrijednostima dvije različite S-kutije što će dovesti do razlike od 4 bita u desnoj polovini bloka. Dodamo li tome razliku od 2 bita u lijevoj polovini bloka, u trećoj iteraciji dobivamo razliku u 6 bitova. Dakle, nakon 7 iteracija ćemo dobiti da se blokovi* x *i* x' *razlikuju u sva 32 najdesnija bita, a iz toga slijedi da će se u osmoj iteraciji razlikovati u sva 32 najljevija bita.*



Slika 5.2: Grafički prikaz DES algoritma,
https://pclub.in/cryptography/2019/04/15/DES/

Kada govorimo o sigurnosti DES algoritma dovoljno je reći kako je i nakon 30 godina njegovog proučavanja najbolji poznati napad, primjenjiv u praksi, pretraživanje ključa u skupu svih mogućih ključeva, veliku ulogu u tome su imali financijska i vremenska ograničenost. Iako je već pri njegovom odabiru 1970. godine bilo dosta kritika na sigurnost DES algoritma zbog njegove veličine ključa od 56 bita, te kritike su bile samo teorija. Prvi puta u praksi napad grubom silom na DES algoritam pokazan je 1997. godine kada je DESCHALL pojekt za 96 dana koristeći tisuće računala koordiniranih putem interneta riješio prvi u nizu izazova koje je postavila RSA Security. Tada se došlo do izračuna kako bi računalo koje može riješiti taj izazov u jednom danu koštalo oko 20 milijuna dolara. Iste te godine riješen je i drugi u nizu izazova, projektu distributed.net trebalo je 41 dan kako bi ga riješio. Značajan napredak u probijanju DES algoritma desio se 1998. godine kada je za samo 56 sati treći u nizu izazova riješio stroj po imenu Deep Crack čija je vrijednost bila 250 tisuća dolara, a izgrađen je od strane Electronic Frontier Foundation. Kombinacijom Deep Crack stroja i distributed.net projekta postignuto je najkraće poznato vrijeme rješavanja DES izazova, a najsuvremeniji stroj zove se Kutija za probijanje DESa koju je napravio PICO Computing, a koristi 48 FPGA čipova i može pronaći ključ DES algoritma za približno 23 sata. Metode napada koje su bolje od metode grubom silom opisane su samo teorijski, a jedna od takvih je diferencijalna kriptoanaliza koju su razvili Biham i Shamir u ranim 90-im. Njihovoj metodi je potrebno 2^{37} vremena i 2^{47} odabranih otvorenih tekstova što je i predstavljalo najveći problem zato što je u to vrijeme bilo nemoguće da napadač uspije otkriti toliki broj šifrata. Ova metoda je potvrdila kako su S-kutije DES algoritma otporne na diferencijalnu kriptoanalizu. Sredinom 90-ih Matsui je razvio i teorijski opisao još jednu metodu koja može probiti DES algoritam, a zove se linearna kriptoanaliza. Prednosti ove metode su bile što koristi već poznate parove otvorenih tekstova i njihovih pripadajućih šifrata, ali je bilo potrebno 2^{43} takvih parova što je i dalje bilo nemoguće za primjenu u praksi. Algoritmi koji su imali mogućnost manipulacije vremenskom složenošću nekih svojih operacija kao memorijskim prostorom mogli su značajno ubrzati primjenu metode grubom silom. Predobradom podataka bilo je moguće generirati tablice veličine nekoliko terabajta koje će s velikom vjerojatnošću omogućiti probijanje DES algoritma uz pomoć samo jednog otvorenog teksta i njegovog šifrata. Problem kod sigurnosti DES algoritma predstavlja i dužina ulaznog bloka zato što se na njoj temelji sigurnost današnjih algoritama šifriranja koji koriste blokovne šifre. Na osnovu svega ovoga očito je kako za sigunornost DES-a nije problem u njegovoj arhitekturi, nego u duljini njegovog glavnog ključa i ulaznog bloka što je veliko priznanje njegovim tvorcima koji su uspjeli dizajnirati gotovo savršene algoritme šifriranja. Kako je arhitektura DES-a njegova velika prednost to ga čini savršenim gradivnim elementom za njegove nasljednike, kao npr. 3DES o kojem ćemo nešto više reći u potpoglavlju 5.1, i AES o kojem možete pročitati u potpoglavlju 4.2, koji je dizajniran tako da pokrije njegove glavne nedostatke. U primjeru 4 ćemo pokazati tri slučaja napada na krnje DES algoritme, tj. napade na samo prve tri iteracije DES-a kako bi smo pobliže objasnili kako funkcioniraju napadi grubom silom na DES algoritam. Iako 3 iteracije nisu dovoljne kako bi se postigao efekt lavine vidjet ćemo kako već pri opisivanju napada na DES s 3 iteracije stvari postaju

kompleksnije, te će nam biti puno jasnije zašto je bio gotovo neprobojan algoritam šifriranja sve do pojave modernih računala.

Primjer 4. *Za svaki od* 3 *slučaja vrijedi sljedeće. Neka je dan ulazni blok* x *i njegov šifrat* y *takav da je* $y = DES_k(x)$ *za neki ključ* k*. Označimo s* L_0 *lijevu polovinu ulaznog bloka, a s* R_0 *njegovu desnu polovinu, te s* L_i *i* R_i *lijevu i desnu polovinu izlazne vrijednosti svake pojedine iteracije, gdje je* i *označava broj iteracije. Nadalje, neka je s* E *označena funkcija udvostručenja, a s* $f_i(R) = \hat{f}_i(k_i, R)$ *DES mangler funkciju u* i*-toj iteraciji. Želimo dobiti podključeve koji su korišteni u pojedinoj iteraciji, što bi omogućilo rekonstrukciju glavnog ključa.*

1. *Napad na DES s jednom iteracijom*
   U ovom je slučaju $y = (L_1, R_1)$*. Dakle, poznati su nam i* L_0 *i* R_0 *i* L_1 *i* R_1*, a prema DES algoritmu vrijedi sljedeće*

$$\begin{aligned} L_1 &= R_0 \\ R_1 &= L_0 \oplus f_1(R_0). \end{aligned}$$

   Odakle slijedi da je

$$f_1(R_0) = R_1 \oplus L_0.$$

   Primjenom inverza završne permutacije miješanjem dobiti ćemo međuvrijednosti koju čine ulančane izlazne vrijednosti svi 8 *S-kutija. Dakle, prva* 4 *bita su izlazna vrijednost prve S-kutije, druga* 4 *bita su izlazna vrijednost druge S-kutije itd. Prema drugom od* 3 *navedena svojstva S-kutija znamo da svaka od izlaznih vrijednosti ima točno* 4 *moguće ulazne vrijednosti. S druge strane znamo da su ulazne vrijednosti u S-kutiji dobivene tako što se desna polovina od* R_0 *duplicira, te se na tu dobivenu međuvrijednost i podključ* k_1 *djeluje opercijom "isključivo ili". Kako je* R_0 *poznat, a samim time i* $E(R_0)$ *lako možemo izračunati* 4 *moguće vrijednosti za set od* 6 *bita podključa* k_1*. Dakle, broj mogućih ključeva smo smanjili s* 2^{48} *na* $4^{48/6} = 4^8 = 2^{16}$*. Time smo dobili mogućnost da u razumnom vremenu uz dodatan par ulaznog bloka i šifrata možemo probati sve mogućnosti koje podključ može poprimiti.*

2. *Napad na DES s* 2 *iteracije*
   U ovom slučaju $y = (L_2, R_2)$*, a prema DES algoritmu vrijedi sljedeće*

$$\begin{aligned} L_1 &= R_0 \\ R_1 &= L_0 \oplus f_1(R_0) \\ L_2 &= R_1 = L_0 \oplus f_1(R_0) \\ R_2 &= L_1 \oplus f_2(R_1). \end{aligned}$$

   Dakle, osim L_0*,* R_0*,* L_2 *i* R_2 *poznati su nam i* L_1*, te* R_1*, što znači da imamo sve ulazne i izlazne vrijednosti od obje DES mangler funkcije,* f_1 *i* f_2*. Stoga slijedi*

$$\begin{aligned} f_1(R_0) &= R_1 \oplus L_0 = L_2 \oplus L_0 \\ f_2(R_1) &= R_2 \oplus L_1 = R_2 \oplus R_0. \end{aligned}$$

   Na isti način kako smo odredili k_1 *u napadu na DES s jednom iteracijom možemo odraditi i* k_1*, te* k_2 *u ovome slučaju. Dakle, broj mogućih podključeva smanjili smo na samo* $2 \cdot 2^{16}$*, ovakav napad bi bio uspješan čak i da su* k_1 *i* k_2 *potpuno neovisni ključevi iako to nisu.*

3. *Napad na DES s 3 iteracije*
   U ovom slučaju $y = (L_3, R_3)$, a prema DES algoritmu vrijedi sljedeće

$$\begin{aligned} L_1 &= R_0 \\ R_1 &= L_0 \oplus f_1(R_0) \\ L_2 &= R_1 = L_0 \oplus f_1(R_0) \\ R_2 &= L_1 \oplus f_2(R_1) \\ L_3 &= R_2 = L_1 \oplus f_2(R_1) \\ R_3 &= L_2 \oplus f_3(R_2). \end{aligned}$$

   Dakle, osim L_0, R_0, L_3 i R_3 poznati su i L_1, te R_2. Jedine dvije vrijednosti koje nam ostaju nepoznate su L_2 i R_1 i znamo da za njih vrijedi $L_2 = R_1$. Možemo primjetiti da nemamo niti jedan ulaz kao niti jedan izlaz za nijednu DES mangler funkciju, te nam način koji smo otkrili podključeve u prva dva slučaja ovaj puta neće koristiti. Kako bi smo odredili podključeve k_i, za $i = 1, 2, 3$ morale bi se raditi pretpostavke koje nisu baš trivijalne stoga nećemo ulaziti u daljnje analize u ovom radu. Već na ovome slučaju vidimo kako se stvari kompliciraju pri povećavanju broja iteracija u DES algoritmu. Unatoč tome što s 3 iteracije ne možemo postići pseudoslučajnu permutaciju očito je koliko je teško probiti DES algoritam. Dodati ćemo samo još kako vrijeme koje je potrebno za napad otprilike iznosi $2 \cdot 2^{28} + 2^{24} < 2^{30}$ što je i dalje dosta manje od 2^{56}.

5.1 3DES

S pojavom rutina koje su u praksi mogle izvršiti napad na DES u razumnom vremenu došlo je do potrebe za novim, sigurnijim algoritmom šifriranja koji će ga zamijeniti. Kako smo se i sami mogli upoznati s činjenicom koliko dugo je bio standard pouzdanosti već i sami možemo zaključiti zašto su se tvorci njegovog nasljednika odlučili ići smjerom koji će dovesti do novog algoritma za šifriranje koji se zove trostruki DES (3DES). 3DES je standardiziran 1999. godine, a može ga se i danas pronaći u širokoj primjeni. Duljina glavnog ključa DES algoritma je bio njegov glavni nedostatak te su se prilikom dizajniranja 3DES algoritma vodile mnoge rasprave koji je najbolji smjer, treba li mijenjati strukturu samog DES algoritma ili ga koristiti kao neovisnu instancu oko koje će se graditi novi algoritam. Prijedlozi koji su bili vezani za smjer koji vodi u mijenjanje same strukture su

- ostaviti funkciju f_i takvu kakva je, glavni ključ bi bio duljine 128 bita, te bi bio promijenjen način generiranja podključeva k_i koji bi i dalje bile duljine 48 bita
- promijeniti dizajn S-kutija kako bi podržavale podključ duljine veće od 48 bita.

Bilo koji od ovih prijedloga imao je veliku manu, a to je činjenica da bilo kakva, pa čak i najmanja promjena u dizajnu algoritma šifriranja može dovesti do velikog pada razine pouzdanosti i učiniti ga puno ranjivijim na napad. Pogotovo

kada govorimo o promijeni strukture DES agloritma zato što bi se time izgubila pouzdanost koja je građena na činjenici da je toliko godina ostao otporan na razne vrste napada.

Dakle, jedini ispravan bi bio drugi smjer, a to je da je DES savršen algoritam šifriranja s ključem duljine 56 bita i treba ga koristiti kao neovisnu instancu. U ovom slučaju također imamo nekoliko prijedloga na koji bi se način DES trebao koristiti kao neovisna instanca. Jedan od njih je dvostruko šifriranje nazvano dvostruki DES (2DES) koji će imati glavni ključ veličine 112 bita, a definirano je na sljedeći način:

$$F'_{k_1,k_2} \stackrel{\text{def}}{=} F_{k_2}(F_{k_1}(x)), \text{ gdje je } F \text{ DES algoritam.}$$

S obzirom da je do tada najbolji poznati napad na DES algoritam, izveden u praksi, bio napad pretraživanja glavnog ključa u skupu svih mogućih ključeva kojih je bilo 2^{56} očekivali bi da će najbolji mogući napad na 2DES trebati 2^{112} vremena, ali nažalost to nije bio slučaj. Algoritmu 8, grafički prikazan na slici 5.3, potrebno je $\mathcal{O}(n \cdot 2^n)$ vremena i $\mathcal{O}((n+l) \cdot 2^n)$ prostora kako bio pronašao glavni ključ duljine 2^{2n} algoritma šifriranja. Skup S koji dobijemo kao izlaznu vrijednost algoritma 8 sadrži točno one parove (k_1, k_2) za koje vrijedi

$$F_{k_1}(x) = F_{k_2}^{-1}(y) \iff y = F'_{k_1,k_2}(x), \tag{5.1}$$

a ponavljajući sam algoritam nekoliko puta, te računanjem presjeka svih skupova S nastalih kao izlazna vrijednost algoritma, s velikom vjerojatnošću možemo odrediti pravi par (k_1^*, k_2^*).

Drugi prijedlog je generalizacija smjera koji DES uzima kao neovisnu instancu, a to je trostruko šifriranje, poznat kao trostruki DES (3DES) koji ima dvije varijacije

- šifriranje s 3 različita neovisna pod ključa k_1, k_2 i k_3 definirano na sljedeći način:
$$F''_{k_1,k_2,k_3} \stackrel{\text{def}}{=} F_{k_3}(F_{k_2}^{-1}(F_{k_1}(x))), \text{ gdje je } F \text{ DES algoritam.}$$
- šifriranje s 2 različita neovisna pod ključa k_1 i k_2 definirano na sljedeći način:
$$F''_{k_1,k_2} \stackrel{\text{def}}{=} F_{k_1}(F_{k_2}^{-1}(F_{k_1}(x))), \text{ gdje je } F \text{ DES algoritam.}$$

Možemo uočiti kako je drugi poziv funkcije F zapravo inverz DES agloritma što ne bi trebalo predstavljati problem sigurnosti s obzirom da je F pseudoslučajna permutacija pa je onda i F^{-1}. Ovaj korak je uveden zbog kompatibilnosti unatrag, ako bi smo uzeli da su sva tri ključa k_1, k_2 i k_3 u jednoj varijanti ili oba ključa k_1 i k_2 u drugoj varijanti jednaka onda onda bi i F''_{k_1,k_2,k_3} i F''_{k_1,k_2} zapravo bili DES agloritmi. Prva varijanta je osjetljiva na "meet-in-the-middle" napad baš kao i 2DES zato očekivano vrijeme napada nije 2^{3n}, pa je njezina sigurnost otprilike jednaka kao i sigurnost druge varijante, a vrijeme potrebno za napad je 2^{2n}. Usprkos tome, prva je varijanta postala standardna za upotrebu u 3DES agloritmu zato što kod druge varijante je problem s duljinom glavnog ključa koji iznosi 112 bita, a minimalna preporučena vrijednost u današnje vrijeme iznosi 128 bita.

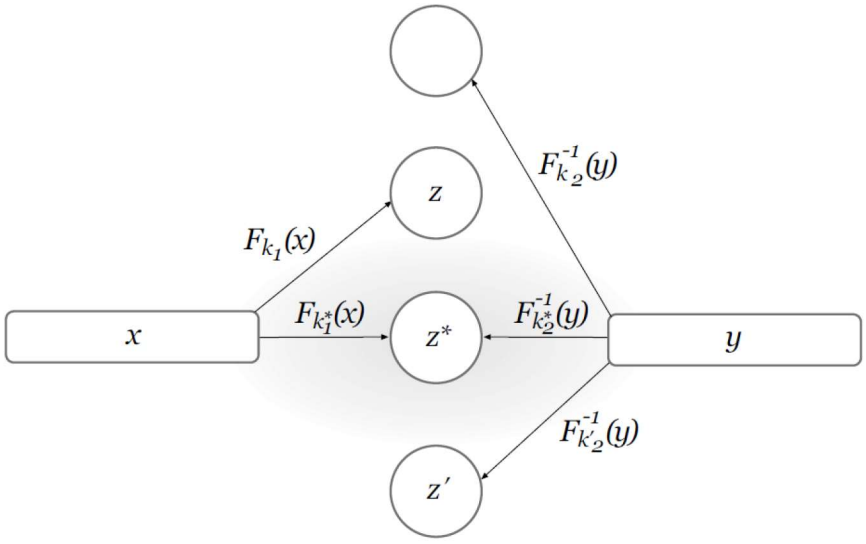
Algoritam 8 ([4], Potpoglavlje 6.2.4) Napad "meet-in-the-middle"

Input: ulazna vrijednost x, šifrat $y = F'_{k_1^*,k_2^*} = F_{k_2^*}(F_{k_1^*}(x))$

Output: Skup S svih parova podključeva k_1^* i k_2^*

1: **for all** $k_1 \in \{0,1\}^n$ **do**
2: **compute** $z = F_{k_1}(x)$
3: **store** (z, k_1) in L
4: **for all** $k_2 \in \{0,1\}^n$ **do**
5: compute $z = F_{k_2}(x)$
6: store (z, k_2) in L'
7: **for all** $(z_1, k_1) \in L$ and $(z_2, k_2) \in L'$ **do**
8: $(z_1, k_1) = (z_2, k_2)$, if $z_1 = z_2$
9:
10: **if** $(z_1, k_1) = (z_2, k_2)$ **then**
11: store (k_1, k_2) in S

return: S



Slika 5.3: Grafički prikaz meet-in-the-middle napada, ([4], Figure 6.7)

Literatura

[1] M. Barakat, C. Eder, T. Hanke, *An Introduction to Cryptography*, Kaiserslautern, 2017.

[2] A. Dujella, M. Maretić, *Kriptografija*, Element, Zagreb, 2007.

[3] J. Hoffstein, J. Pipher, J. H. Silverman, *An Introduction to Mathematical Cryptography, 2nd editino*, Springer, New York, 2014.

[4] J. Katz, Y. Lindell, *Introduction to Modern Crypthography, 2nd edition*, Chapman & Hall/CRC Press, New York, 2015.

[5] G. C. Kessler, *An Overview of Crypthography*, 2022, https://www.garykessler.net/library/crypto.html

[6] M. Maqableh, *Durham E-Theses Analysis and Design Security Primitives Based on Chaotic Systems for eCommerce*, Jordan, 2012.

[7] A. J. Menezes, P. C. Oorschot, S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, 1996.

[8] J. J. G. Savard, A Cryptographic Compendium, 2002.

[9] B. Schneier, *Applied Cryptography, 2nd Edition*, John Wiley & Sons, Inc., New Jersey, 1996.

[10] N. Smart, *Cryptography: An Introduction, 3rd Edition*, Mcgraw-Hill College, Bristol, 2004.

[11] D. R. Stinson, *Cryptography Theory And Practice, 3rd edition*, Chapman & Hall/CRC, New York, 2006.

[12] T. St Denis, Simon Johnson, *Cryptography for Developers*, Syngress Publishing, Inc., Rockland, 2007.

[13] *Specification of the Advanced Encryption Standard*, https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf

Sažetak

Na početku ovoga rada je opisana kriptografija i po čemu se razlikuje od kriptologije, te je definiran kriptosustav. Zatim je opisano što su to kriptografski primitivi i dana je njihova podjela prema nekoliko kriterija. Glavni zahtjev za protočne šifre je da se ponašaju kao pseudoslučajni generatori. Postoji nekoliko načina implementacije protočnih šifri, a to su linearni i nelinearni registri povratnih pomaka. Najpoznatiji protočne šifre su Trivimu i RC4 koji se koristio kao dio sustava za šifriranje u TLS-u i SSL-u. Blokovne šifre ulazni tekst šifriraju blok po blok, smatraju se dobro dizajniranima ako najbolji poznati napad ima vremensku složenost približno jednaku vremenskoj složenosti pronalaska ključa metodom grube sile. Potrebna razina sigurnosti postiže se paradigmom zbunjenosti i konfuzije, te veličina bloka ne smije biti manja od 64 bita. Supstitucijsko-permutacijska mreža je baza za konstruiranje puno poznatijih algoritama kao što su AES i DES. AES je samo naziv kojega je Rijndeal algoritam dobio tako što je pobijedio na natječaju koji je objavljen 1977. godine kako bi se pronašla zamjena za DES algoritam, veličina bloka mu je 128 bita i ima nekoliko verzija ovisno o duljini glavnog ključa. Feistel mreža osim što se u praksi koristila kao dio Lucifer šifre, prema njoj je izgrađen i DES. Za kraj smo opisali DES algoritam koji je najpoznatija blokovna šifra i dugi niz godina je predstavljao standard za sigurnosti. Razvijen je 1970. godine od strane IBM-a u suradnji sa NSA, a kako s vremenom više nije smatran sigurnim naslijedio ga je 3DES koji svoju primjenu u praksi ima i danas.

Ključne riječi

kriptografija, kriptologija, kriptosustav, kriptografski primitivi, protočne šifre, registri povratnih pomaka, trivium, RC4, blokovne šifre, paradigma zbunjenosti i difuzije, supstitucijsko-permutacijska mreža, AES, DES, Feistel mreža, 3DES

Application of Cryptographic Symmetric-Key Primitives in Practice

Summary

At the beginning of this work, cryptography is described and how it differs from cryptology, and a cryptosystem is defined. Then it is described what cryptographic primitives are and their division according to several criteria is given. The main requirement for stream ciphers is that they behave like pseudorandom generators. There are several ways to implement stream ciphers, linear-feedback shift registers and nonlinear-feedback shift register. The most famous stream ciphers are Trivimu and RC4 which was used as part of the encryption system in TLS and SSL. Block ciphers encrypt each block of input separately, they are considered well designed if the best known attack has a time complexity roughly equal to the time complexity of finding the key using the brute-force search, the required level of security is achieved by the confusion-diffusion paradigm and the block size must not be less than 64 bits. The substitution-permutation network is the basis for constructing much more well-known algorithms such as AES and DES. AES is just the name given to the Rijndeal algorithm by winning a competition announced in 1977. to find a replacement for the DES algorithm, its block size is 128 bits and there are several versions depending on the length of the master key. In addition to being used in practice as part of the Lucifer cipher, the Feistel network was also used in DES. Finally, we described the DES algorithm, which is the most famous block cipher and has been the standard for security for many years. It was developed in 1970 by IBM in cooperation with the NSA, and since it was no longer considered secure over time, it was succeeded by 3DES, which is still used in practice today.

Keywords

cryptorgaphy, cryptology, cryptosystem, cryptographic primitives, stream ciphers, feedback shift registers, trivium, RC4, block ciphers, confusion-diffusion paradigm, substitution-permutation networks, AES, DES, feistel networks, 3DES

Životopis

Rođen sam 26. kolovoza 1994. godine u Vinkovcima. Završio sam Osnovnu školu fra Ilije Starčevića u Tolisi, gdje sam prva četiri razreda pohađao u područnom odjelu u Kostrču, te Opću gimnaziju u Školskom centru fra Martin Nedić u Orašju. Nakon završene srednje škole upisujem preddiplomski studij Matematika na Odjelu za matematiku (sadašnji Fakultet primijenjene matematike i informatike) i isti završavam 2017. godine te se upisujem na diplomski studij Matematike, smjer matematika i računarstvo, također na Odjelu za matematiku. Tijekom studija sam sudjelova na Mini Maker Fairu 2018. godine gdje sam predstavio robota sa autonomnim upravljem pod nazivom Mathbot 2. 2019. godine sam se zapslio u tvrtki Enea Software d.o.o na mjestu Software inžinjera, 2022. sam promoviran na mjestu Senior Software inžinjera.