



UNIVERZITET SINGIDUNUM

**-MASTER STUDIJSKI PROGRAM-
SAVREMENE INFORMACIONE TEHNOLOGIJE**

Servisi integriteta u računarskim komunikacijama

- master rad -

**Mentor:
Prof. dr. Mladen Veinović**

**Student:
Mladen Kosovac
Br. indeksa: 410379/2015**



Singidunum

Beograd, 2016

Servisi integriteta u računarskim komunikacijama

Sažetak: U ovom radu razmatraće se različiti servisi oza očuvanje integriteta podataka prilikom prenosa u računarskim mrežama i ostalim komunikacionim sistemima. Opisane su metode kompresije podataka, detekcije i korekcije grešaka, šifrovanja i autentifikacije dokumenata.

Ključne reči: podaci, prenos, integritet, kompresija, detekcija, korekcija, greške, kriptografija, šifrovanje, autentifikacija, računarske mreže

Data integrity services in computer communications

Abstract: In this paper we will discuss about various services for data integrity preservation during transmission in computer networks and other communication systems. Methods for data compression, detection and correction, ciphering and authentication of files are described in this work.

Keywords: data transmission, integrity, compression, detection, correction, errors, cryptography, ciphering, authentication, computer networks, data

SADRŽAJ

1. Metodologija	1
1.1 Uvodne napomene	1
1.2 Predmet istraživanja	1
1.3 Hipoteza.....	1
1.4 Ciljevi istraživanja.....	1
1.5 Naučne metode	2
2. Kompresija podataka	3
2.1 Algoritmi kompresije bez gubitaka	4
2.2 Frekventno zavisni kodovi – <i>Hoffman</i> -ov kod	5
2.3 <i>Run-Length</i> kodiranje	6
2.4 <i>Lempel-Ziv</i> kompresija	7
2.5 Relativno kodiranje	7
2.6 Kompresija slika	7
2.7 <i>JPEG</i> kompresija	8
2.8 <i>MPEG</i> kompresija	10
2.9 <i>MP3</i> kompresija.....	12
3. Detekcija i korekcija grešaka	14
3.1 Detekcija i korekcija grešaka na sloju veze računarskih mreža	14
3.2 Čeksume	17
3.3 Detekcija grešaka pomoću ciklične provere redundantnosti	17
3.4 Implementacija <i>CRC</i>	19
3.5 <i>Hamming</i> -ovi kodovi i korekcija grešaka.....	21
4. Simetrični šifarski sistemi	23
4.1 Algoritmi za šifrovanje	24
4.2 Cezarovo (<i>Ceaser</i>) šifrovanje	25
4.3 Polialfabetско šifrovanje	26
4.4 Šifrovanje premeštanjem	27
4.5 Bitsko šifrovanje.....	28

4.6	<i>DES</i> i trostruki <i>DES</i>	29
4.7	<i>AES</i> - <i>Rijndael</i> algoritam	32
4.8	<i>Clipperchip</i> i <i>Skipjack</i> algoritam	33
4.9	Integritet u simetričnim sistemima	34
5.	Asimetrični šifarski sistemi.....	35
5.1	<i>Shamir</i> -ov metod razmene ključa	35
5.2	<i>Diffie-Hellman</i> metod razmene ključa.....	36
5.3	<i>RSA</i> algoritam	37
5.4	Digitalni potpis i digitalni sertifikati	37
5.5	Program <i>PGP</i> (<i>Pretty Good Privacy</i>).....	39
5.6	Digitalni sažetak poruke - <i>hash</i> funkcije	40
5.7	Integritet u asimetričnim sistemima	42
6.	Zaključak.....	44
	Literatura	45

1. Metodologija

1.1 Uvodne napomene

U ovom istraživanju obuhvaćene su metodologije očuvanja integriteta podataka pri prenosu. To podrazumeva analizu postupaka kompresije podataka, detekcije i korekcije grešaka, šifrovanja i autentifikacije dokumenata.

Sa sve većim uplivom kompjutera i Interneta u svakodnevni život i poslovne procese, javlja se i sve veća potreba za većim protokom podataka ka svakom korisniku. Uporedo se javlja i sve veća potreba za sigurnim transferom poverljivih i manje poverljivih podataka. Pri svakom prenosu podataka pa tako i u računarskim mrežama podaci su izloženi raznim uticajima. Pored toga što su izloženi smetnjama koji su uzrok samog medijuma kroz koji se prenose izloženi su često i napadima od strane zlonamernih korisnika. Cilj napada može biti dolazak do informacija ili izmena sadržaja i zloupotreba poruka koje se šalju. Metodi zaštite odnosno dostupni servisi integriteta podataka su obrađeni u ovom radu.

1.2 Predmet istraživanja

Predmet ovog istraživanja su servisi za zaštitu integriteta podataka. Sa tim ciljem opisane su različite metode kompresije podataka, detekcije i korekcije grešaka, šifrovanja i autentifikacije dokumenata. Obuhvaćene su različite raspoložive tehnike za rešenje pomenutih problema.

1.3 Hipoteza

Opšta hipoteza: očuvanje integriteta podataka je proces kojim se onemogućavaju izmene podataka pri prenosu. Mogućnost uočavanja izmena pri prenosu nazivamo detekcija grešaka dok proces detektovanja neželjenih izmena nazivamo autentifikacija.

Posebna/radna hipoteza: Na današnjem nivou razvoja računarskih telekomunikacionih sistema razvili su se različiti servisi za zaštitu integriteta podataka u zavisnosti od potreba korisnika. Integritet podataka je od posebnog značaja sa aspekta potreba današnjih korisnika Interneta.

1.4 Ciljevi istraživanja

Naučni cilj: Istraživanje i definisanje problema zaštite integriteta podataka u računarskim mrežama i prezentacija potencijalnih rešenja.

Društveni ciljevi: Obezbeđivanje adekvatnog informisanja svih uključenih sa osnovnim konceptima servisa za zaštitu integriteta podataka u računarskim mrežama.

1.5 Naučne metode

Pri izradi ovog rada korišćene su osnovne naučne metode deskripcije, indukcije, dedukcije, analize, sinteze i generalizacije.

Metoda deskripcije je korišćenja pri objašnjavanju osnovnih pojmova i sistema funkcionisanja.

U teorijskim segmentima su primenjene metode indukcije, dedukcije i analize, a u izvodjenju zaključaka metode sinteze i generalizacije.

2. Kompresija podataka

Sa stanovišta integriteta podataka od posebnog značaja su različiti vidovi kompresije podataka. Da bi se dobro razumeli servisi integriteta podataka neophodno je imati osnovna znanja o algoritmima kompresije pa će oni biti opsiani u ovom delu rada. Ne postoji praktično ni jedan vid računarskih komunikacija u kojima se ne koristi makar neki vid kompresije podataka.

Kompresija podataka je proces kojim se smanjuje veličina podataka, odnosno postupak redukovanja broja bitova, pri kojem se zadržava značenje. Prvobitna namena kompresovanja je bila smanjenje prostora koji će podaci zauzeti na medijumu na koji se skladište (hard disk, kompakt disk, traka itd.). Sa napretkom tehnologije, povećanjem kapaciteta diskova i značajnim padom njihove cene, potreba za kompresovanjem u cilju uštede prostora je u izvesnoj meri prevaziđena, međutim za potrebe brzog prenosa podataka u današnjim komunikacionim sistemima kompresija je i dalje neophodna (npr. kompresija video materijala kod digitalne televizije, radija i sl.). Ključni pojam svake kompresije je algoritam koji precizno definiše kako se podaci kompresuju i kasnije dekompresuju. Odnos veličina kompresovanog i originalnog fajla se naziva **stepen kompresije** ili kompresioni odnos. Algoritmi (procedure) kompresije se mogu podeliti na: **algoritme koje ne čine gubitke informacija i algoritme koje čine gubitke informacija**.

Osnovna odlika algoritama kompresije kod kojih ne dolazi do gubitaka (*lossless*) je da se nakon dekompresije dobija fajl koji je potpuno veran polaznom dokumentu. Danas postoji više ovakvih algoritama kompresije, a najčešće su u upotrebi Hofmanov (*Hoffman*) kod i takozvani *LZW* (skraćeno od *Lempel-Ziv-Welch*) metod. Ovi algoritmi su bazirani na mogućnosti da stringove znakova koji se češće pojavljuju, kodiraju sa manje bitova. Ovi algoritmi se koriste najčešće za kompresovanje tekstualnih datoteka, izvršnih datoteka i rede multimedijalnih sadržaja, a ponekad i samo radi arhiviranja bez kompresije.

Međutim, ukoliko smo prinuđeni da vršimo prenos digitalizovanog zvuka, slika ili pokretnih slika odnosno filma i televizije, kompresioni odnos koji nam pružaju algoritmi bez gubitaka nije dovoljan. Čak i ukoliko samo arhiviramo sadržaj na disk veličina ovih fajlova je po pravilu takva da ćemo brzo ostati bez slobodnog prostora. Kako bi se rešio ovaj problem pribegava se metodama kompresije koje dozvoljavaju gubitak informacija pri kompresiji. Drugim rečima nakon dekompresije nećemo dobiti fajl koji je veran originalnom, polaznom izvoru podataka. Zbog svoje prirode kompresija koja dozvoljava gubitke nije pogodna za važne podatke i izvršne fajlove ali je idealna za multimedijalne primene jer malim gubicima postiže velike kompresione odnose. Gubici koji nastaju nisu naročito primetni, jer se u krajnjoj liniji interpretiraju od strane naših čula koja imaju brojne nesavršenosti. Nesavršenosti naših čula sluha i vida nam u ovom slučaju pogoduju i koristimo ih u svrhu smanjenja veličine ovih sadržaja odnosno boljem iskorišćenju komunikacionog kanala.

2.1 Algoritmi kompresije bez gubitaka

Prilikom primene algoritama kompresije kod kojih se ne čine gubici informacija dobićemo fajl koji će nakon dekompresije biti potpuno veran polaznom dokumentu, dakle u pitanju je reverzibilan proces. Značajan korak u razvoju ovakvih vrsta algoritama kompresije, je čuveni program *ZIP* koji je se koristio za komprimovanje podataka na disketama i diskovima. U vremenu u kojem se pojavio program prenos sa jednog računara na drugi je često išao disketama malog kapaciteta a računari su se najviše koristili za obradu tekstualne vrste podataka za čije komprimovanje je prvobitno i namenjen. Istorijski gledano *ZIP* program je najduže u primeni pa se i danas vrlo često koristi za smanjivanje veličine podataka i datoteka tekstualnog tipa ili izvršnih fajlova, gde ne sme da dođe ni do kakvog gubljenja bitova. Kao alternativa *ZIP* programu nalazimo u današnjoj primeni programe kao što su *ARJ*, *RAR*, *7z* itd. Poznatiji format koji kompresuje audio sadržaj bez gubitaka relativno uspešno zove se *FLAC*. Svi ovi programi za komprimovanje podržavaju različite modove algoritama za komprimovanje koje biramo prema željama i potrebama tako što se opredeljujemo za veću ili manju brzinu pakovanja i raspakivanja i veći ili manji kompresioni odnos. Algoritam koji koriste programi *ZIP* i *RAR* za kompresovanje datoteka je tzv. *Deflate* algoritam koji je derivat ranije pomenutog *LZW* algoritma.

Jednostavan primer algoritma kompresije možemo da pokažemo na sledećem primeru. Zamislamo jedan tekstualni dokument koji u sebi sadrži isključivo karaktere malih slova alfabeta. Pretpostavimo da su slova zapamćena kao standardni osmobiitni *ASCII* kodovi, u tom slučaju broj prenetih bitova računamo kao $8 \cdot n$, gde sa n označavamo koliko znakova sadrži dokument. Pošto podaci koje šaljemo sadrže isključivo mala slova, možemo da krenemo logikom da ne upotrebljavamo kompletan osmobiitni niz za svako slovo. Sva mala slova alfabeta možemo da napišemo pomoći petobiitnog koda (na slici 1) koji će prijemna aplikacija prevesti (dekompresovati) u originalni *ASCII*.

Slovo	Kod
a	00000
b	00001
c	00010
d	00011
.	.
.	.
.	.
x	10111
y	11000
z	11001

sl. 1

Važan pojam u algoritmima kompresije je i tzv. **redundansa** (engl. redundancy – upotreba više reči nego što je potrebno da bi se opisao pojam, u ovom slučaju deo koda koji se ponavlja za svako slovo i koji nije potreban da bi se prenela informacija) u originalnim podacima. Dobri algoritmi kompresije eliminišu redundansu odnosno svode je na minimum.

2.2 Frekventno zavisni kodovi – Hofmanov (*Hoffman*) kod

Frekventno zavisni kodovi su zasnovani na činjenici da se neki karakteri pojavljuju češće nego drugi (npr. samoglasnici), te se na osnovu učestalosti pojavljivanja nekih karaktera, tekst kodira tako što se koriste kodovi različite dužine. Za karaktere koji se najčešće pojavljuju se koriste najkraći kodovi (sa manje bitova).

Najbolji primer frekventno zavisnog koda je Hofmanov (*Hoffman*) kod. Da bismo pokazali kako on funkcioniše, zadržaćemo se na primeru od 5 znakova. Princip se može proširiti i na ceo alfabet ili bilo koji set znakova. Na slici 2 su prikazana slova alfabeta a, b, c, d i e, i frekvencija njihovog pojavljivanja u dokumentu data u tabeli:

Slovo	Frekvencija [%]
a	25
b	15
c	10
d	20
e	30

sl. 2

Na osnovu učestalosti dodeljuju se kodovi:

Slovo	Kod
a	01
b	110
c	111
d	10
e	00

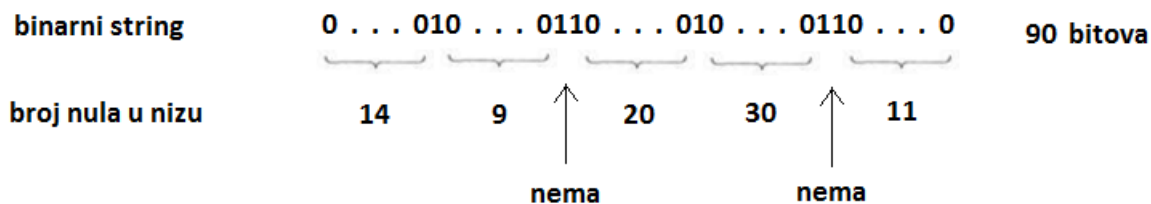
sl. 3

Pošto dužina kodova nije definisana, kodovi mogu biti različite dužine. Problem koji se odmah po logici stvari nameće je određivanje početka i kraja svakog znaka obzirom da koristimo razlilite bitske dužine. Rešenje ovog problema je u samoj biti Hofmanovih kodova a to je korišćenje takozvanog *no-prefix* svojstva. Svojstvo je implementirano na taj način da isključuje mogućnost da kod jednog znaka može da se koristi kao prefiks drugog koda. Kao što je navedeno u primeru na slici 3, Hofmanov metod predviđa kod **01** za znak **a**, dok je za sva ostala slova u upotrebi isključena mogućnost da započinu sa tim kodom.

2.3 Run-length kodiranje

U realnim sistemima računarskih komunikacija retko se sreću samo tekstualni fajlovi odnosno nizova znakova. Prenos podataka često sadrži multimedijalne sadržaje, slike, video materijale, zvukove pa i izvršne fajlove (binarne, programe prevedene na mašinski jezik računara). Ovakve sadržaje nismo u mogućnosti da efikasno komprimujemo Hofmanovom metodom nego nam je neophodna opštija tehnika kojom bi bili u stanju da komprimujemo proizvoljne (naizgled nasumične) nizove bitova. Jedno od rešenja ovakvog univerzalnijeg načina kompresije je takozvani *run-length* algoritam. U osnovi upotrebljavamo jedan jasan princip a to je da vršimo analizu kompletnog niza bitova i pokušavamo da pronađemo duže sekvence nula ili jedinica. Na mestima gde pronađemo adekvatnu sekvencu poslaćemo broj karaktera a ne sve bitove koje ih karakterišu. Što je broj ovakvih mesta veći i sekvence duže ušteda će biti veća.

Jedan od načina za implementaciju *run-length* kodiranja je takav da ćemo dužine nizova od interesa slati kao cele brojeve u adekvatnoj bitskoj interpretaciji koja je određene dužine. Kada na prijemnoj strani primalac analizira primljene bitove i njihove dužine u stanju je da regeneriše izvornu poruku tako što će po diktatu primljene poruke umetnuti odgovarajuće bitove između nizova. Na slici 4 je prikazan niz bitova pre kodiranja a na slici 5 nakon kodiranja.



Sl. 4

binarni nizovi fiksne dužine	1110	1001	0000	1111	0101	1111	1111	0000	0000	1011
decimalni niz	14	9	0	15	5	15	15	0	0	11

Sl. 5

Moguće je da se analizom fajla utvrdi da se određena kombinacija bitova kao celina ponavlja pa se to može iskoristiti kao dodatna ušteda i pojednostavljenje. Kada imamo takvu situaciju poslaćemo znak od interesa u kombinaciji sa njegovom dužinom. U sledećem primeru vidimo da niz znakova:

QQQQQQPPMMMMMMMMMMMMMMMMZZZZZZZZZZZZKXXXXXXXXXX

možemo da pojednostavimo tako što ćemo preneti broj ponavljanja praćen sa znakom koji je od interesa:

6, Q, 2, P, 15, M, 11, Z, 1, K, 8, X.

2.4 Lempel-Ziv kompresija

Metod *Lempel-Ziv* kompresije je sličan *Run-Length* kodiranju. Metod je tako zamišljen i implementiran da vrši pretragu niza znakova koji se češće pojavljuju i zapisuje ih u fajl na jednom mestu dodeljujući im jedinstveni kod. Sa ovom referencom svako pojavljivanje u fajlu se može zameniti kodom koji je dodeljen polaznom nizu znakova. Ovakav pristup uštedi je poznat iz sistema koji upravljaju radom baza podataka (*DBMS*) - podatak se smešta samo jednom i indeksira se na osnovu čega mu se kasnije pristupa gde god da je potreban. Na sistemima *UNIX* tipa i sličnim postoje ovakve komande za kompresovanje datoteka, kod modema kao standard za kompresovanje gde je nazvan *V.42 bis* kao i kod čuvenog formata za kompresovanje slika *GIF* (*Graphics Interchange Format*).

2.5 Relativno kodiranje

Relativno (diferencijalno) kodiranje je našlo svoju adekvatnu primenu kod prenosa pokretnih slika (video). Sa stanovišta količine informacija pokretne slike mogu da budu vrlo kompleksne za razliku od datoteka koje su tekstualnog tipa. Ne samo da zasebne slike traže dosta prostora za skladištenje nego je i potrební broj slika u jedinica vremena relativno velik, počevši od 25 slika po sekundi kod običnog videa pa do nekoliko stotina kada su u pitanju specifične primene. Međutim, po pravilu video materijal sadrži u sebi scene, odnosno nizove više sukcesivnih slika kod kojih se naredna slika ne razlikuje potpuno od prethodne nego je na neki način samo malo izmenjena. Ovo se može pametno iskoristiti tako što nećemo svaku sliku tretirati zasebno i time bespotrebno uvećavati potrebe za protokom nego ćemo pažnju da obratimo na to koliko se naredna slika razlikuje od prethodne i u čemu je ta razlika. Ovakav pristup daje jako dobre rezultate kada su razlika između susednih slika male. Kompletan postupak možemo ugrubo opisati tako da u prvom koraku predajna strana šalje kompletnu prvu sliku koja se takva dočekuje na prijemnoj strani. Predajna strana analizira drugu sliku, poredi je sa prvom, uočava razlika koje se nakon toga kodiraju i šalju po određenom formatu. Prijemna strana nadalje analizom razlika utvđuje sadržaj druge slika koju ekstrapolira iz prve i koristi. Postupak se koristi nadalje na sličan način.

2.6 Kompresija slika

Računarska grafika, odnosno slike u računaru su ustvari jednostvane strukture koje su suštinski pravougaone mreže nastale od većeg broja nedeljivih kvadratića odnosno tačaka čije je osnovno svojstvo da imaju jednu boju. Te kvadratiće ili tačke nazivamo **pikseli** (*pixel - picture element*). Što je više ovakvih elemenata slike spakovno na manjoj površini to je manje uočljiva kvadratsta odnosno zrnasta struktura date slike. Kao što je ranije pomenuto u određenom odnosu broja piksela i dimenzija displeja naše nesavršeno čulo vida prestaje da uočava ove digitalne deformacije i ima utisak da vidi čistu sliku. Broj piksela po površini je određen rezolucijom odnosno razlučivošću. Što je rezolucija veća smatramo da je prikazana slika kvalitetnija. Svaka tehnologija koja se koristila za prikaz slika počevši od prvih televizora do današnjih displeja u pametnim telefonima koristi isti princip. Tri osnovna izvora svetlosti se smeštaju na jako malu površinu tako da naše oko nije u stanju da ih vidi kao zasebne izvore već kao mešavinu osnovnih boja, crvene, zelene i plave (*RGB - Red Green Blue*). Svaki ekran monitora (nezavisno od tipa da li je sa katodnom cevi - *CRT*, tečnim kristalom - *LCD* ili gasnom plazmom - *GPD*) koristi ovakav

pristup za dobijanje slike u boji. Monitorski elektronski sklop osvetljava osnovne boje svake tačke određenim intenzitetom u zavisnosti od signala koji dobija. Promenom intenziteta osnovnih boja svake tačke moguće je podešavati konačnu boju svake tačke koja se emituje posmatraču. Sa stanovišta podataka i informacija svaki piksel mora da ima svoj triplet brojeva koji čuva vrednosti osvetljavanja svakog izvora svetlosti kako bi lako mogli da vršimo mešanje boja. U praksi je dovoljno da odvojimo po jedan bajt za svaku osnovnu boju. Pošto imamo 8 bitova, možemo da intenzitet svake boje skaliramo na 256 vrednosti, od 0 do 255, ukupno 16.777.216 nijansi. Ovaj broj je sasvim dovoljan da ni jedno ljudsko oko nije u stanju da primeti nedostatak neke nijanse.

Uzmimo kao primer kvalitet današnje televizije u *HD* rezoluciji (1920x1080). Računica pokazuje da ova slika zahteva $1920 \times 1080 \times 24 = 49.766.400$ bitova. Kada ovo pomnožimo sa brojem slika u jedinici vremena koje idu od 25 do nekoliko stotina bitska brzina koja nam je potrebna za prenos je reda gigabita. Ni današnji najsavremeniji sistemi nisu u stanju da pruže ovakve usluge prenosa pa je neophodna primenjivati kompresiju.

2.7 JPEG kompresija

Verovatno najpoznatiji i najčešće upotrebljavani metod kompresije koji čini gubitke informacija je *JPEG* (skraćeno od *Joint Photographics Experts Group*). Ovaj standard kompresije je isključivo namenjen smanjenju veličine statičnih računarskih slika a na prvom mestu digitalizovanih fotografija kod kojih daje najbolje rezultate. Sam *JPEG* algoritam u sebi obuhvata četiri osnovna postupka komprimovanja.

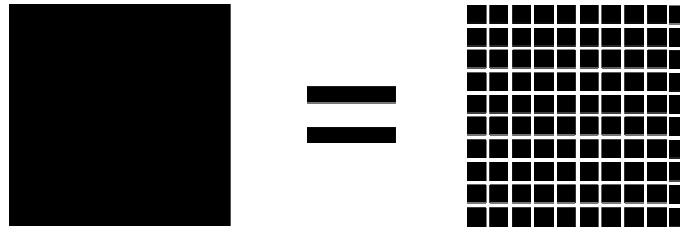
Osnovni postupak koji je ključan za ovu vrstu kompresije je korišćenje takozvane diskretne kosinusne transformacije (*DCT*). Takođe *JPEG* koristi opet nesavršenosti našeg čula vida da se slabije snalazi sa razlikama u nijansama, nego u intenzitetu. Sa korisničke strane predviđen je izbor stepena kompresije odnosno veličine konačnog fajla u odnosu na deformacije slike koje će nastupiti primenom *JPEG* kompresije. Iako je u osnovi algoritam sa gubicima *JPEG* standard uključuje i postupak kompresije koji ne čini gubitke koji se oslanja na prediktivno kodiranje.

Primena *JPEG* kompresije se vrši u tri koraka kao što je prikazano na slici 6. Prvi korak je diskretna kosinusna transformacija (*DCT* – discrete cosine transform), drugi je kvantizacija a treći je korak kodiranja.



sl. 6

Prvi korak odnosno *DCT* podrazumeva deljenje kompletne slike u nizove manjih sekcija veličine 8x8 piksela, kao što je ilustrovano na slici 7. Ukoliko bi komprimovali sliku veličine 1600x1200 piksela, struktura bi sadržala 200 sekcija po *x* osi i 150 sekcija po *y* osi odnosno ukupno 30000. Kod slika koje su predstavljene u nijansama sive (*engl. greyscale*) ili kako se to često nepravilno kaže crno-belih slika svaki piksel se predstavlja kao 8-bitni broj dok u slučaju slika u boji to važi za svaku osnovnu boju.



sl. 7

Svaku sekciju dobijenu na ovaj način možemo da tretiramo kao dvodimenzionalnu matricu oblika 8x8 (osam redova i osam kolona). Svaka od ovih matrica u sebi sadrži elemente koji su kao što je ranije rečeno jednobajtni celi brojevi. Ovaj niz vrednosti je ulazni podatak za primenu diskretne kosinusne transformacije. U osnovi, diskretna kosinusna transformacija je funkcija koja i kao ulazni podatak i kao izlazni podatak ima šablon brojeva dimenzija 8x8. Posmatrano kroz matematičku formalnost, ako sa $\mathbf{P}$ označimo niz vrednosti piksela, odnosno sa $\mathbf{P}[x][y]$ vrednost u redu x i koloni y , diskretna kosinusna transformacija definiše novi niz $\mathbf{T}$, gde $[i]$ i $[j]$ predstavljaju vrednost u i -tom redu i u j -toj koloni prema formuli:

$$T[i][j] = 0.25C(i)C(j) \sum_{x=0}^7 \sum_{y=0}^7 P[x][y] \cos\left(\frac{(2x+1)i\pi}{16}\right) \cos\left(\frac{(2y+1)j\pi}{16}\right)$$

Kao rezultat operacije dobijamo dvodimenzionalni niz $\mathbf{T}$ koji sadrži skup vrednosti koje nazivamo prostornim učestalostima - frekvencijama (*spatial frequency*). Tumačenje ovih vrednosti frekvencija možemo da posmatramo kao broj promena vrednosti piksela u funkciji pozicije piksela u sekciji u odnosu na vrednost prvu $T[0][0]$ vrednost. Ova vrednost ili tzv. DC koeficijent predstavlja prosek vrednosti niza $\mathbf{P}$ (kada su i i j jednaki 0, kosinusi su 1). Sve stale vrednosti u $\mathbf{T}$ nazivamo AC koeficijentima. Naravno postoji i inverzna formula sa kojom možemo da se vratimo sa prostornih frekvencija na vrednosti piksela.

$$P[i][j] = 0.25C(i)C(j) \sum_{x=0}^7 \sum_{y=0}^7 T[x][y] \cos\left(\frac{(2x+1)i\pi}{16}\right) \cos\left(\frac{(2y+1)j\pi}{16}\right)$$

U koraku kodiranja osnovni postupak je linearizacija dvodimenzionalnih podataka kvantifikovanih vrednosti i komprimovanje podataka radi skladištenja ili prenosa. Linearizacija bi mogla da se izvede tako što se prenosi jedan red u jednom trenutku.

Kao celoviti postupak, *JPEG* algoritam komprimovanja slika nije jednostavna procedura. Kompresioni odnos sirove i komprimovane slika će zavisiti od samog uzorka odnosno polazne slike i od niza koji se koristi za kvantizaciju. Ovaj niz nije strogo propisan, pa on može da varira u zavistnosti od aplikacije. U praktičnoj primeni *JPEG* algoritam može da ostvari stepene kompresije koji se kreću od 5:1 do čak 50:1 pa i više od toga ukoliko prihvatimo malo veće deformacije koji mogu da postanu i uočljivije.

2.8 MPEG kompresija

Pokretna slika je ustvari ništa drugo nego brzo prikazivanje nepokretnih slika. Računar i računarske komunikacije se jednako primenjuju kako na statične slike tako i na pokretne. Prethodno opisani algoritam *JPEG* ima na neki način svoju varijantu koja je prilagođena komprimovanjem pokretnih slika. Taj algoritam predviđen za rad sa videom se naziva **MPEG** skraćeno od *Moving Pictures Expert Group*. Razvoj *MPEG-a* je išao u više pravaca i rezultat je skup više standarda:

- *MPEG-1* razvijen za potreba tzv. *videoCD* standarda koji je pokušao da uklopi jedan sat video materijala na veličinu kompakt diska (~700 MB) što bi po kvalitetu bilo adekvatno tada prisutnim analognim *VHS* video uređajima. U manjem obimu se koristio i kod satelitske televizije.
- *MPEG-2* je i istorijski gledano naslednik *MPEG-1* standarda i korišćen je za nešto naprednije aplikacije. Generalni okvir je bio uklapanje dva sata video materijala na veličinu jednostranog jednoslojnog DVD diska (~4 GB). Koristio se za satelitsku i kablovsku televiziju visoke definicije (*HDTV*).
- *MPEG-4* je u početku koncipiran kao pogodan standard za prenos video preko komunikacionih kanala sa manjim protokom. Primenjivan je u sistemima za videofoniju i video konferencije a kasnije je prilagođen i drugim primenama pa je zamenio svoje prethodnike.

Kada video postmatramo kao ono što i jeste a to je niz nepokretnih slika koje se smenjuju brzo u vremenu prirodno je da nam padne na pamet da primenimo *JPEG* komprimovanje za svaku pojedinačnu sliku kako bi smanjili veličinu kompletne video sekvence. Iako je ovo potpuno istinito i potpuno primenjivo rezultati koje bi postigli ne bi bili zadovoljavajući, jednostavno stepen kompresije nije dovoljan da bi ušteda bila onolika koliku bi voleli da vidimo. Upravo kod *MPEG* standarda primenjujemo varijantu relativnog kodiranja odnosno uočavamo pojavu tzv. privremene redundantnosti. Osnovna ideja je takva da ćemo prvo poslati prvu sliku (frejm), i iskodirati sve naredne frejmove tako što ćemo izračunavati razliku sa prvim frejmom (ovim postizemo najveću uštedu), koje ćemo iskompresovati i preneti. Kada se podaci dočekaju na prijemu moguće je izračunavanjima i primenom prvog frejma rekonstruisati ostale frejmove.

Ovakav pristup daje jako dobre rezultate ukoliko su svi susedni frejmovi u izvesnoj meri slični što u praksi odgovara jednoj sceni snimljenog video materijala. Pošto se video materijal obično sastoji od više scena potrebno je predvideti i takve pojave. Zbog ovoga *MPEG* standard predviđa tri vrste frejmova:

- **I frejm (unutrašnji frejm)** je osnovni frejm koji sadrži kompletnu sliku kompresovanu ranije pomenutim *JPEG* algoritmom.
- **P frejm (predviđeni frejm)** je frejm nalik prethodnom ali ih razlikujemo po tome što ga ne kompresujemo kao celovitu sliku već ga kodiramo izračunavanjem razlika sa prethodnim frejmom.
- **B frejm (bidirekcionni frejm)** je frejm koji se potpuno izračunava i dobija interpolacijom između prethodnog i narednog frejma.

Između dva I frejma "uglavljeno" je više B frejmova i jedan P frejm. Generalno gledano broj B frejmova može da bude različit, ali je uobičajeno da se koristi po jedan P frejm između dve grupe B frejmova. P frejm suštinski nije ništa drugo nego razlika između

prethodnog I frejma i B frejmova koji su određeni interpolacijom najbližih I i P frejmova. Recimo da su prva dva B frejma nastala interpolacijom prvog I frejma i P frejma dok su poslednja dva B frejma interpolirana od poslednjeg I frejma i P frejma. P frejm ćemo dakle poslati pre prva dva B frejma, a drugi I frejm ćemo poslati pre poslednja dva B frejma. P frejm i I frejm se najčešće smeštaju u bafer dok se B frejmovi dekodiraju na samom mestu prikazivanja.

Kod *MPEG* standarda metoda kojom se kodiraju P frejmovi se naziva *motion-compensated prediction* (grubo možemo prevesti kao predikcija kompenzovanja promena). Zamišljena je da deli sliku na makroblokovе koji imaju po 256 piksela (16 po horizontali i 16 po vertikali). Pretpostavka je da dati piksel ima jednu vrednost za osvetljenje i dve za boju, pa makroblokovі mogu da se posmatraju kao tri matrice dimenzije 16x16. Kako bi se dobilo na brzini algoritma i postigle dodatne uštede nizovi boje se dodatno smanjuju na dimenzije od 8x8.

Kao što je ranije rečeno B frejmovi se moraju izračunati na samom mestu prikaza video tako da se dekodiranje vrši tako što se makroblokovі izračunavaju interpolacijom iz podudarnih makroblokovа. Interpolacija je zgodan način da predvidimo vrednost koju nemamo ukoliko imamo susedne vrednosti, i kao metod je proizašla iz numeričke analize.

2.9 MP3 kompresija

MP3 je protokol za kompresovanje audio zapisa. Nekompresovani audio fajlovi su veliki, i zbog toga problematični i za skladištenje i za brzi prenos. Sirovi zvučni fajlovi u računaru najčešće koriste takozvanu *PCM* (*Pulse Code Modulation*) tehniku za reprezentaciju zvukova u digitalnom obliku. Problem ove tehnike je prostor koji ovakvi fajlovi zauzimaju. Za kvalitet zvuka koji je standardan danas i minimalan a to su šesnaestobitni semplovi pri frekvenciji uzorkovanja od 44100Hz (teoretski odziv do 22050Hz po Nikvistovoj teoremi) jedna sekunda *PCM*-kodiranog zvuka zahteva $16 \times 44.100 = 705.600$ bitova odnosno dvostruko više ako je u pitanju stereo signal, što je oko 1,4Mb. Dakle, za svaki minut 84 Mb, odnosno 10MB za svaki munut, ugrubo. Još veći problem je prenos. Da bi se prenosio i reprodukovao u realnom vremenu, potrebna je brzina od 1,4 Mb/s. Smanjenje veličine ovih fajlova klasičnim metodama je moguće jedino ukoliko redukujemo broj kanala (npr. umesto stereo zvuka koristimo mono), smanjimo broj mogućih vrednosti svakog sempla (npr. sa šesnaestobitnih pređemo na osmobitne sempleve) ili smanjimo frekvenciju uzorkovanja (npr. sa 44100Hz na 22050Hz). Svi ovi pokušaji da se na ovakav način smanji veličina fajla bi umanjili preciznost digitalizacije polaznog analognog signala, i dobili bi manje kvalitetan zvuk koji bi se lako primetio i čak bio i neprijatan za slušanje. Iz ovih razloga se kod zvučnih fajlova pribegava kompresiji koja je manje konvencionalna i koristi opet ranije pomenute nesavršenosti naših čula, u konkretnom slučaju čula sluha.

Metode za kompresiju zvuka koriste dva bitna svojstva čovekovog čula sluha, nemogućnost da razlikuje visinu tonova bliskih frekvencija, naročito kada su u pitanju frekvencije bliske granicama čujnog opsega i tzv. "čujno maskiranje" (*auditory masking*). Čujno maskiranje je takva pojava kod koje nismo u stanju da čujemo slabije signale po amplitudi određene frekvencije zato što je na bliskim frekvencijama prisutan signal znatno veće amplitude. Kompresija ovog tipa vrši analizu pri kojoj se ovakvi zvukovi koje nismo u stanju da čujemo adekvatno kodiraju koristeći manje informacija ili se izopštavaju potpuno iz procesa.

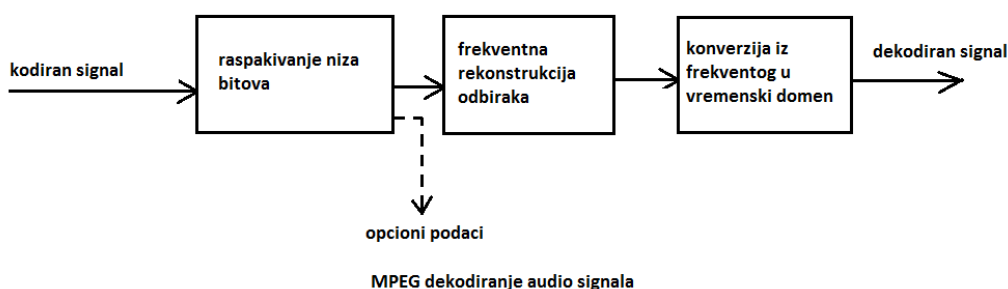
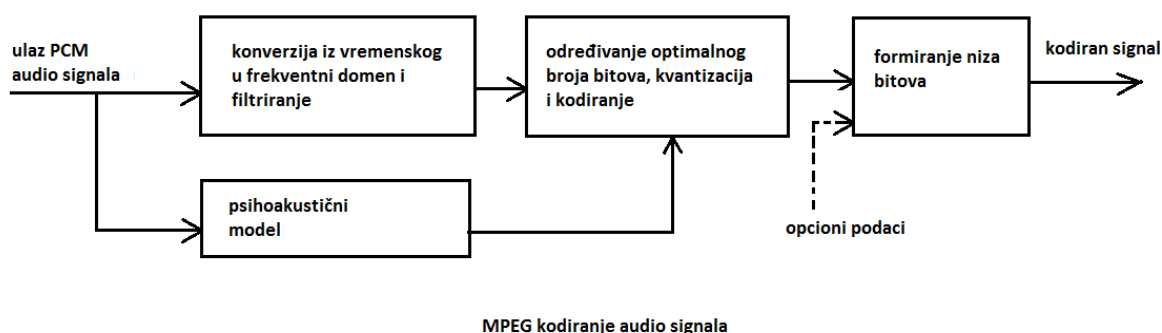
MP3 kao projekat je vezan zapravo za *MPEG* standard koji se od svog početka bavio uporedo i video i audio metodama kompresije. Zapravo sam termin je skraćenica od *MPEG-1 Layer-3* i prihvaćen je danas skoro kao sinonim za digitalni oblik zvuka odnosno muzike. Ekstenzija fajla *.mp3* nedvosmisleno ukazuje na to da je u pitanju fajl sa zvučnim zapisom. Standard koji poznajemo kao *MP3* opisuje operacije dekodiranja, bitske formate i na koji način se kodiraju audio sadržaji. Osnove ovih metoda vuku korene sa *Franhoffer* instituta, na kojem su prvobitno bili razvijani i definisani kroz više patenata. Rezultat pokušaja da se na ovaj način redukuje veličina audio zapisa je takav da pri eksperimentima *hi-fi* stručnjaci nisu uspeali da primete razlike pri kompresiji koja je išla u razmerama od 5:1 do čak 10:1. *MP3* je svojom pojavom dao trajno rešenje audio kompresije.

Jedan od pristupa koji je u osnovi ovog algoritma je primena eksperimentalne činjenice koja nam govori da naše čulo sluha frekvencijski selektivno doživljava preciznost zvukova. Najoštrije čujemo zvukove koji su u okviru tzv. govornog kanala koji je definisan od 300Hz do 3400Hz. Ispod i iznad ovih vrednosti nas je evolucija lišila preciznosti slušanja. Shodno tome kompletan spektar zvukova koje smo u stanju da čujemo možemo podeliti na nekoliko opsega. Naše čulo sluha ograničeno može da vrši distinkciju frekvencija, pa će odnos signala i šuma različitih frekvencija zavisiti samo od prisutnosti signala u opsegu tih frekvencija. Kod audio kompresije, ovakvu osobinu koristimo kako bi transformisali audio signal iz vremenskog u frekventni domen koristeći brzu Furijerovu transformaciju (*FFT*). Kada jednom pređemo u domen frekvencija podelićemo rezultat u

više podopsega koji su u skladu sa ranije spomenutim kritičnim opsezima na kraju ćemo kvantifikovati svaki od podopsega na osnovu prisutnosti kvantifikovanog šuma koji pripada datom opsegu. Kako bi odredili optimum kompresije, u svakom opsegu bi trebali koristiti pri kvantifikovanju samo neophodan broj nivoa kako bi kvantifikovani šum bio maskiran.

Svakako je osnovni korak u MP3 komprimovanju različito kodiranje od podopsega do podopsega (*engl. subband coding*). Za tu funkciju nam je potreban psihoakustični model i banka sa filterima. Banka sa filterima (*engl. filter bank*) je zapravo zbirka filtera koji polazni signal dele na same podopsege tako što izbegavaju preklapanje graničnih frekvencija opsega.

Osnovna ideja koja nas dalje vodi do komprimovanog zvučnog fajla je da ćemo formirati nizove bitova za svaki zvučni podopseg. Međutim, oni neće biti kodirani na isti način zbog ranije pomenutog zvučnog maskiranja. Ukoliko su prisutni glasniji signali na podopsegu korist ćemo veću rezoluciju kako bi kvantifikovali njihove amplitude. Slično, ukoliko primetimo slabije signale u okolnim opsezima koje maskiraju glasni signali korist ćemo slabiju rezoluciju a samim tim i manje bitova. Takođe ukoliko neke frekvencije nedostaju iz posmatranog podopsega nemamo potrebu da ih uopšte kodiramo pa se time dodatno redukuje potreban broj bitova. Ovakvim postupkom kompresije smo postigli značajno umanjeње polaznog fajla sa narušavanjem kvaliteta koji nije značajan. Sa prijemne strane dekodiranje se vrši kao obrnuta operacija, raspoakovaće se niz kodiranih bitova iz kojih će se rekonstruisati potrebne vrednosti svakog podopsega i izvršiti prelazak iz frekventnog domena nazad u vremenski domen. Principijelna šema MP3 kompresije i dekompresije je data na slici 8.



sl. 8

3. Detekcija i korekcija grešaka

Pri prenosu podataka na daljinu, često mogu da se pojave greške u zapisu, zbog raznih uticaja na signale kojima se podaci šalju. Jaka sunčeva svetlost, električni udari, mehanički kvarovi, padovi ili pikovi napona napajanja, su samo neki od spoljašnjih uticaja - prouzrokovaca grešaka u podacima. Ukoliko smo u mogućnosti da detektujemo promene nastale u toku prenosa kažemo da postoji mehanizam **detekcije grešaka**. Ukoliko se greška detektuje, prijemni uređaj može da obavesti predajni. Javiće pojavljivanje grešaka i tražiti da se poruka pošalje ponovo. Međutim, to često nije praktično, ili nije ni izvodljivo, npr. kod *real-time* aplikacija, pri komunikaciji sa uređajima sa kojima je komunikacija spora (npr. svemirke sonde), a često su okolnosti takve da možemo da očekujemo da će se greška uvek pojaviti. Zato je bolje rešenje detekcija sa ispravljanjem odnosno **korekcijom greške**.

Uopšteno gledajući svi metodi za detekciju grešaka podrazumevaju da ćemo slati još dodatih bitova koji će imati različite vrednosti u zavisnosti od samog sadržaja koji se šalje. Ako se podaci promene, vrednosti dodatnih bitova neće odgovarati novim podacima.

Prvi implementirani ali i još uvek postupak koji se jako često koristi je tzv. proveravanje parnosti (*engl. parity check*). Ovaj metod podrazumeva brojanje bitova koji imaju vrednost 1, koji utiče na vrednost bita koji se dodaje na kraj kako bi ukupan broj setovanih bitova bio paran ili neparan.

3.1 Detekcija i korekcija grešaka na sloju veze računarskih mreža

Prema OSI referentnom modelu računarskih mreža, prikazanom na slici 9, sloj veze se nalazi iznad fizičkog sloja. U računarskim mrežama fizički sloj obavlja posao prenosa i prijema podataka, dok sloj veze, koji je hijerarhijski iznad fizičkog sloja, mora da obezbedi sve uslove da fizički sloj radi sa ispravnim podacima. Sa stanovišta detekcije i korekcije grešaka možemo da kažemo da je sloj veze najznačajniji.

Na sloju veze se mogu razrešiti mnogi problemi do kojih može doći prilikom komunikacije u računarskim mrežama. Moguće je da nekoliko čvorova pokušaju da izvrše predaju podataka preko iste linije ili se može dogoditi da se usled smetnji na medijumu pojedini bitovi pogrešno prenesu. Svakako svaki čvor bi trebalo da zna da li je primio sve što je trebalo i da li su primljeni podaci apsolutno tačni.

7. SLOJ APLIKACIJE
6. SLOJ PREZENTACIJE
5. SLOJ SESIJE
4. TRANSPORTNI SLOJ
3. SLOJ MREŽE
2. SLOJ VEZE
1. FIZIČKI SLOJ

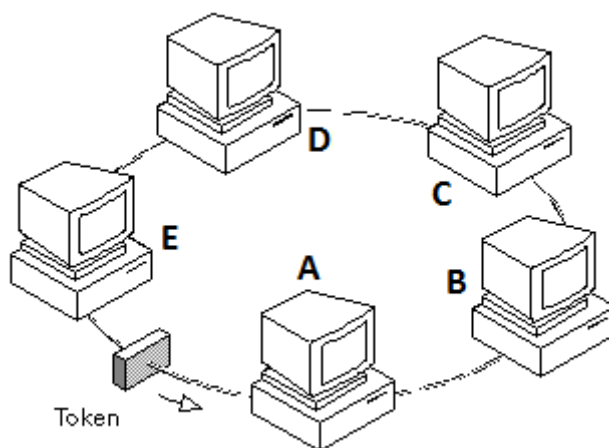
sl. 9

Pojavu u kojoj nekoliko čvorova mreže u istom trenutku pokušavaju da prenesu podatke preko zajedničkog medijuma nazivamo *nadmetanjem* odnosno *kolizijom*. Neke mreže sa topologijom magistrale nadmetanje rešavaju tzv. detekcijom kolizije. Čvorovi se kod ovakve metode ne sprečavaju da iniciraju prenos ali se posebnim kolima za "oslušivanje" detektuje kolizija a neuspešan prenos se kasnije ponovo ponavlja. Ukoliko se detektuje zauzetost medijuma, uređaj će obustaviti započeti prenos. Kada kolo prepozna da su aktivnosti na magistrali nestale, uređaj ponovo inicira prenos. Moguće je ipak da više uređaja "oslušuju" magistralu i prepoznaju da nema nikakvih aktivnosti pa da u istom trenutku započnu prenos podataka. U takvoj situaciji opet dolazi do pojave kolizije. Ovakva metoda izbegavanja kolizije na zajedničkoj magistrali se zove **CMSA/CD** (engl. *Carrier Sense Multiple Access with Collision Detection*). Sa ovakvim postupkom se nije moguće osloboditi pojave kolizije i potpuno ga rešiti ali ga je moguće svesti na najmanju moguću meru.

Jedan od načina za razrešavanje kolizije na zajedničkoj magistrali je tzv. **prosleđivanje tokena**. Token predstavlja jedinstvenu sekvencu bitova koja se emituje od čvora do čvora kroz mrežu. Ukoliko neki čvor treba da krene sa emitovanjem podataka radi prenosa, neophodno je da prvo sačeka prijem tokena, koji će postaviti na kraj poruke. Ukoliko se token trenutno koristi čvor će promeniti kontrolne bitove u tokenu. Poruka se zajedno sa tokenom šalje do njenog odredišta ali zavi od implementacije protokola da li će prijemna strana uzeti token i da iskoristiti ga za emitovanje svojih podataka, ili će pak proslediti token do narednog čvora.

Na slici 10 predstavljena je mreža sa topologijom prstena kod kojih se često koristi metod prosleđivanja tokena. Sam token će kružiti kroz mrežu u obliku prstena u smeru okretanja kazaljke na satu. Na slici samo čvor označen sa **E** može da inicira prenos jer je token u datom trenutku kod njega. Osobina tokena je da jedinstven, pa ne može da dođe do nadmetanja. Nedostaci ovakvog postupka se ogledaju u tome da token može da se izgubi, može da se duplira a može i da ostane stalno u istom čvoru.

Naravno ideja korišćenja tokena nikako nije samo ograničena na primenu u mrežama sa prstenastom topologijom. Moguće ju je iskoristiti i u drugim mrežnim topologijama ali bi čvorove mreže morali da numerišemo. U tom slučaju token će kružiti prema brojevima čvorova. Svakako je numerisanje čvorova najjednostavnije u linearnim mrežama, i kod ranije spomenutih prstenastih topologija.



sl. 10

Kao što je ranije rečeno razni faktori mogu da utiču na to da fizički sloj ne prenose nizove bitova preko mreže onako kako bi trebalo. Zbog toga je sloj veze dužan da detektuje pojavu greški i da eventualno primeni predviđene algoritme za korekciju nastalih

grešaka. Postupkom detektovanja grešaka sloj veze će na prijemnoj strani utvrditi da li je došlo do greške. Ukoliko je došlo do greške, zahtevaće se ponovljeno slanje podataka. Ukoliko je predviđen mehanizam korekcije grešaka sloj veze će primeniti propisane algoritme kako bi ispravili nastala oštećenja.



sl. 11

Vrlo jednostavan način da detektujemo greške je korišćenje tzv. *bita parnosti* (engl. *parity bit*), dodatnog bita koji je pridružen emitovanom nizu bitova (slika 11). Suština ove metode je da se prebrojava broj jedinica u sekvenci. Tako razlikujemo parnu parnost (engl. *even parity*) i neparnu parnost (*odd parity*). Parna osigurava paran broj jedinica (ako uračunamo i sam bit parnosti) pa ako kompletan niz ima neparan broj bitova čija je vrednost 1, bit parnosti će biti 1 a ako je kompletan niz ima paran broj, bit parnosti će biti 0. Na sličan način je definisana i neparna parnost. Slika 12 prikazuje primere 8-bitne sekvence bitova sa bitom parnosti (7 bitova podataka + 1 bit parnosti) uz primenu obe varijante parnosti (parne i neparne).

sekvenca od 7 bitova	bitovi sa vrednošću "1"	sekvenca od 8 bitova sa bitom parnosti	
		parna	neparna
0000000	0	00000000	00000001
1010001	3	10100011	10100010
1101001	4	11010010	11010011
1111111	7	11111111	11111110

sl. 12

Zajedno sa nizom bitova podataka biće prenet i sam bit parnosti a prijemna strana će izvršiti proveru parnosti, slika 13. Ukoliko se detektuje da je broj bitova koji nose vrednost 1 neparan, u prenosu se dogodila greška. Nevolja metode primene parnosti je u tome što postoji verovatnoća da neke greške prođu neprimećene. Postoji verovatnoća da je došlo do takve greške u prenosu da su se promenile vrednosti dva bita. Ako su se promenile vrednosti dva bita sa 0 na 1 onda je i dalje broj jedinica paran. Ovo nam govori da parnost kao tehnika može da primeti samo jednostruke, a ne i dvostruke greške. Ukoliko želimo da ulovimo višestruke greške moramo primeniti komplikovanije metode.



sl. 13

3.2 Čeksume

Detekcija greške pomoću čeksume (*checksum*) se sastoji u sledećem: svi bitovi u nekom zapisu se podele u grupe od po 32 bita, i svakoj se dodeli celobrojna vrednost. Na kraju se podacima doda još 32 bita koji predstavljaju sumu (*checksum*) ovih celih brojeva. Prijemni uređaj sabira celobrojne vrednosti 32-bitnih grupa i upoređuje sa čeksumom. Ukoliko se računica razlikuje, detektovaćemo grešku.

3.3 Detekcija grešaka pomoću ciklične provere redundantnosti

Ciklična provera redundantnosti (*CRC - cyclic redundancy check*), tretira sekvencu bitova kao koeficijente polinoma. Datu sekvencu bitova možemo predstaviti polinomom na sledeći način:

U opštem slučaju, ako imamo iz bitova $b_{n-1}b_{n-2}b_{n-3}...b_2b_1b_0$ taj niz možemo da predstavimo polinomom oblika $b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + ... + b_2x^2 + b_1x^1 + b_0x^0$, gde svako b može da bude 0 ili 1, tačnije uzima vrednost bita na datom mestu u nizu.

Praktično, ako imamo niz 110100110, možemo da ga predstavimo polinomom:

$$1 \cdot x^8 + 1 \cdot x^7 + 0 \cdot x^6 + 1 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + x^2 + 1 \cdot x + 0 \cdot x^0$$

odnosno:

$$x^8 + x^7 + x^5 + x^2 + x$$

CRC metod se sastoji od sledećih koraka:

1. Za datu sekvencu bitova dodaćemo određeni broj nula sa kraja i nazvećemo ga B . Sa $B(x)$ označavamo polinom koji je nastao iz niza B .
2. $B(x)$ ćemo podeliti sa nekim unapred definisanim polinomom $G(x)$ (generator polinoma) i utvrdićemo ostatak pri deljenju $R(x)$.
3. Definišemo $T(x) = B(x) - R(x)$. $T(x)/G(x)$ generiše ostatak 0 pa oduzimanje možemo izvesti zamenom ranije dodavanih nula nizom koji bi odgovarao polinomu $R(x)$.
4. Prenosimo T , niz bitova koji odgovara polinomu $T(x)$.
5. Sa T označavamo sekvencu bitova koju primalac dobija a sa $T(x)$ pridruženi polinom. Primalac će podeliti $T(x)$ sa $G(x)$. Ukoliko je ostatak 0, primalac će ustanoviti da je $T = T$ i da nije došlo do grešaka. Inače zaključuje da se pojavila greška i šalje zahtev za iniciranje ponovljenog prenosa.

Ovakvo deljenje polinoma je nalik klasičnom deljenju polinoma koje nam je poznato iz algebre, sa tom razlikom što koristimo **aritmetiku po modulu 2**. I sabiranje i oduzimanje po modulu 2 su identični operaciji **isključivo ILI (XOR)**.

Principi *CRC* provere možemo da vidimo kroz sledeći primer. Recimo da želimo da pošaljemo bitsku sekvencu 1101011, a kao generator polinoma koristimo polinom $G(x) = x^4 + x^3 + 1$. Pratimo sledeće korake:

1. Dodajemo bitove sa nulama sa kraja sekvence. Pošto je koristimo polinom četvrtog stepena i broj dodatih nula će biti toliki. Sada sekvenca izgleda 11010110000.

2. Delimo $B(x)$ sa $G(x)$. Dobijamo ostatak $R(x) = x^3 + x$, odnosno njegov bitski ekvivalent 1010.

3. Definišemo $T(x) = B(x) - R(x)$. Oduzimanje se izračunava tako što se oduzimaju bitovi svakog polinoma. U konkretnom slučaju:

$$\begin{array}{r}
 \mathbf{11010110000} \quad \text{niz bitova B} \\
 \mathbf{-1010} \quad \text{niz bitova R} \\
 \hline
 \mathbf{11010100110} \quad \text{niz bitova T}
 \end{array}$$

4. Ukoliko se na prijemnoj strani T prihvati bez grešaka, deljenje sa generatorom polinoma $G(x)$ daje ostatak 0. Ukoliko bi se T oštetiо prilikom samog prenosa, recimo da su se središnji bitovi promenili na nulu pa je na prijemu detektovano 11000001010, prijemna strana će izvršiti deljenje sa $G(x)$ pa će ostatak biti različit od nule. Prijemna strana na osnovu ovog rezultata konstatuje da je došlo do grešaka u prenosu.

Postavlja se pitanje koliko je CRC efikasna, naročito za navalne greške. Ako šaljemo neki niz $T(x)$ i on pretrpi promene, te promene možemo da predstavimo novim nizom $E(x)$, a niz koji stiže na prijemni uređaj kao $T(x) + E(x)$. Ako sada podelimo novi niz za generatorom polinoma $G(x)$, dobićemo

$$\frac{T(x) + E(x)}{G(x)} = \frac{T(x)}{G(x)} + \frac{E(x)}{G(x)}$$

prvi član nam neće dati ostatak, međutim problem je ako je $E(x)$ takvo da nam i drugi član ne da ostatak, tada greška neće biti detektovana. Dobrim odabirom generatora polinoma $G(x)$ možemo značajno da smanjimo mogućnost nedetektovanja greške. Ovo su neki od principa kojih se treba držati pri izboru $G(x)$:

- Ukoliko x nije faktor $G(x)$, uspećemo da detektujemo sve nastale greške čija je dužina manja ili jednaka stepenu $G(x)$.
- Ukoliko je $x+1$ faktor $G(x)$, uspećemo da detektujemo svaku grešku koja je prouzrokovala izmenu neparnog broja bitova.
- Ukoliko je nastala greška dužine veće od stepena polinoma $G(x)$, verovatnoća da greška ne bude detektovana je $\frac{1}{2^{r-1}}$, sa r je označen stepen polinoma $G(x)$.

Ovo su neki od standardnih generatora polinoma koji se koriste u praksi:

$$\text{CRC-12: } x^{12} + x^{11} + x^3 + x^2 + x + 1$$

$$\text{CRC-16: } x^{16} + x^{15} + x^2 + 1$$

$$\text{CRC-CCITT: } x^{16} + x^{12} + x^5 + 1$$

$$\text{CRC-32: } x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

3.4 Implementacija CRC

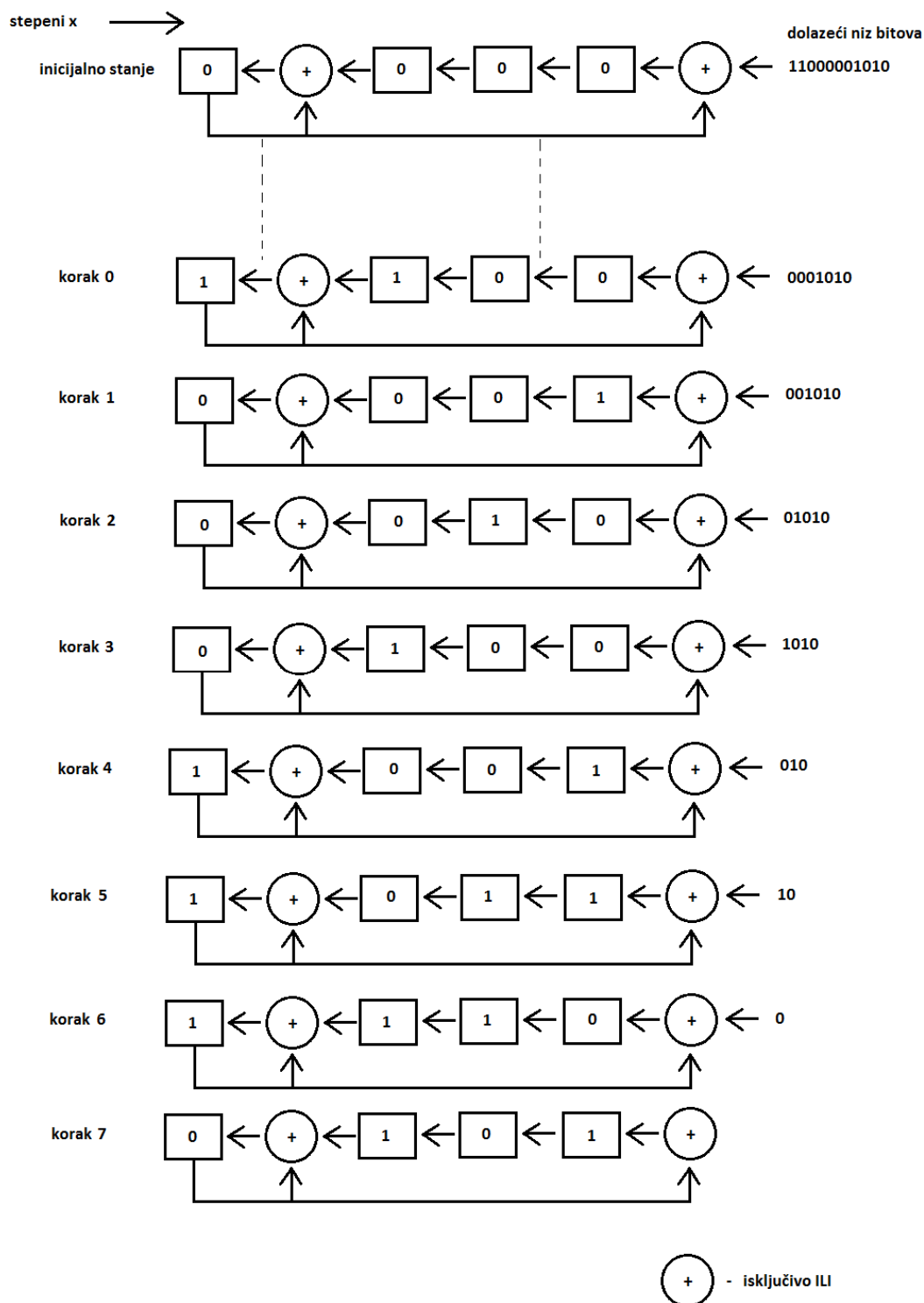
CRC je u teoriji veoma efikasna, međutim u praksi postoje dva značajna problema. Prvi je što se dodavanjem nula na nizove bitova povećava ukupna broj bitova, odnosno povećava se veličina podataka, a drugi je što se povećava broj operacija pri obradi podataka, zbog dodavanja deljenja polinoma za svaki pristigli niz bitova na prijemnom uređaju.

U praksi se CRC implementira tako da se proizvode različita kola prema izabranom generatoru polinoma $G(x)$. Obzirom da smo videla da su u upotrebi standardom definisani polinomi, proizvodnja kola se obavlja tipski i masovno. Kola u sebi imaju pomeračke registre i hardversku operaciju XOR (*isključivo ILI*) koja se izvodi prema sledećim pravilima:

- $G(x) = b_r x^r + b_{r-1} x^{r-1} + b_{r-2} x^{r-2} + b_{r-3} x^{r-3} + \dots + b_2 x^2 + b_1 x^1 + b_0 x^0$ gde je b 0 ili 1, a r stepen polinoma $G(x)$. Broj pozicija bitova u registru je r . Prva pozicija sa desna odgovara članu b_0 , dok je prva sa leva $b_r x^r$
- XOR kolo se nalazi sa desne strane bilo koje pozicije kojoj je dodeljena vrednost $b=1$.
- Bitska sekvenca se uvodi u registar jedan po jedan, a počinje se sa krajnje desne pozicije.
- Kada se uvodi novi bit, svi bitovi prisutni u registru se pomeraju ulevo za po jednu poziciju. Bitovi se kao operandi propuštaju kroz XOR kolo.
- Bit sa krajnje leve pozicije prolazi kroz svako XOR kolo, formirajući drugi član svake XOR operacije.

Na slici su prikazani registar i kola isključivo ILI za polinom $G(x) = x^4 + x^3 + 1$. U nultom koraku prvi bit dolazećeg niza pomeren je na kraj sa leve strane u registar. U prvom koraku se krajnji levi bit propušta kroz svako XOR kolo i sve se pomera ulevo za po jedan korak (slika 14).

CRC se kao metod detekcije grešaka jako puno i široko primenjuje. Prednosti ove metode je da se može relativno lako i efikasno implementirati i hardverski proizvesti zbog postojanja standardnih generatora polinoma. Što se tiče efikasnosti vreme koje se potroši na detekciju proporcionalno je dužini sekvence bitova.



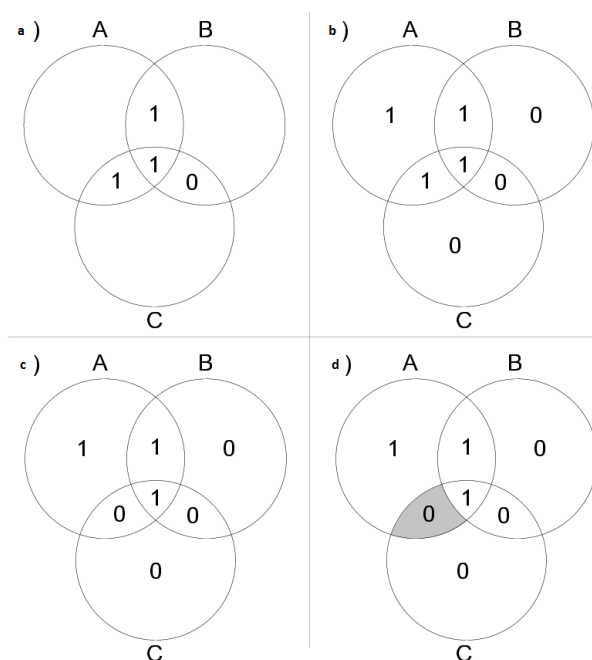
sl. 14

3.5 Hamming-ovi kodovi i korekcija grešaka

Hamming-ovi kodovi koji se mogu koristiti za korekciju odnosno ispravljanje nastalih grešaka su u suštini bitovi parnosti postavljeni tako da su na pozicijama na kojima se mogu iskoristiti za popravljjanje oštećenih bitova. Ukoliko bi došlo do jedne greške, nad jednim bitom, njegovo mesto će uticati na jedinstvenu kombinaciju kada se bude proveravala parnost. Na ovaj način smo detektovali grešku i utvrdili na kom je mestu nastala. Sa ovim saznanjem greška se može sa lakoćom i ispraviti pa ne mora da se traži ponovni prenos. Ovakva tehnika je na određeni način primenjena kod pristupa memoriji odnosno prilikom prenošenja bajtova između *RAM*-a i registara procesora.

Ovakve tehnike koje predviđaju korekciju grešaka generalno gledano moramo da platimo povećanim brojem bitova i komplikovanijem procesu prenosa pa nisu opravdane da se koriste u svim primenama pogotovo ako greške nastaju relativno retko. Tada je isplativije ići na jednostavniji postupak i ponoviti prenos podataka kod kojih je došlo do greške. Naravno ovakav pristup uvodi neko malo kašnjenje kod protoka, ali ono može biti dovoljno malo da je prihvatljivo i u ukupnom procesu zanemarljivo. Naravno tehnika je apsolutno primenjiva i neophodna u nekim tzv. *real-time* sistemima kod kojih nismo u mogućnosti da prištimo nikakva kašnjenja i gde se svaki bit mora isporučiti u tačno određeno vreme. Takođe u nekim sistemima kod kojih prenos traje dugo a količina informacija relativno mala, kao što je recimo prenos podataka između planeta od sonde do baze, ponavljanje prenosa je praktično i neizvodljivo pa je uključivanje procesa korekcije grešaka nužno, ne predstavlja nikakav luksuz nego potrebu.

Primer detekcije i korekcije grešaka *Hamming*-ovom metodom možemo ilustrovati na primeru *Hamming*-ovog *SEC* koda koji je principijelno, šematski prikazan slikom 15.



sl. 15

Na slici je prikazan primer primene *SEC* koda na reč dužine 4 bita. U polje koje je presek bar dva kruga upiše se po jedna vrednost bita. U preostala polja se upisuje 0 ili 1 tako da se očuva parnost. Na slici 10a prikazani su bitovi podataka, na slici 10b prikazani su bitovi podataka sa bitovima parnosti, na slici 10c je prikazana promena bita usled greške i konačno na slici 10d lokalizovana greška koja može lako da se ispravi.

Postoje i drugi *Hamming*-ovi kodovi kao što je na primer *DED* koji u suštini predstavlja isti postupak sa tom razlikom da je dodat još jedan bit ukupne parnosti što dodatno utiče na poboljšanje algoritma detekcije i korekcije grešaka ovom metodom.

4. Simetrični šifarski sistemi

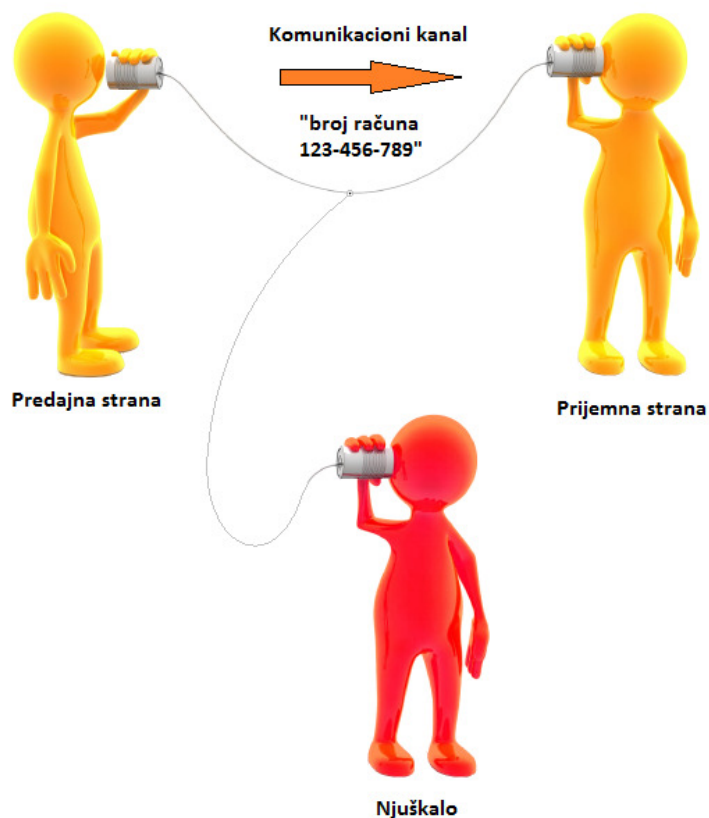
Metodi za detekciju i korekciju grešaka štite integritet podataka u smislu da sprečavaju prijem netačnih podataka. Međutim, pored problema netačnog prijema postoji i problem neautorizovanog prijema. Ako pogledamo prisutnost komunikacionih sistema u savremenom svetu možemo da vidimo da ovakvi sistemi ne samo da moraju da budu pouzdani i sigurni nego i bezbedni. Ne samo ako su upitanju preduzeća, banke i transfer novca već i u brojnim drugim sistemima. Gledano prostom logikom mogli bi da onemogućimo pristup svima osim izabranim autorizovanim osobama. Problem ovakvog pristupa je u tome što je komunikacioni kanal u svakom trenutku izložen potencijalnim neželjenim slušaocima bilo da je u pitanju bežični prenos (zemaljski ili satelitski) ili kablovski. Svaka komunikacija se na određeni način može presresti.

Zbog ovih problema uobičajeni pristup za osiguravanje bezbednosti je takav da se u principu uopšte i ne vodi računa o neautorizovanom prijemu podataka, s obzirom da on realno i ne može da se spreči. Umesto da se brinu o potencijalnom osluškivanju, sistemi se tako koncipiraju da šifruju poruke pre slanja tako da su nerazumljive svakom neautorizovanom presretaču. Tipičan primer ovakvih sistema je npr. kablovska ili satelitska televizija. Ukoliko pokušamo da pratimo program koji nije plaćen niko nas u tome neće sprečiti ali praktično nećemo biti u mogućnosti da ga pratimo jer je šifrovan odnosno *skremblovan*. Kada se usluga plati obično se dobijaju kodovi ili uređaj koji vrši dešifrovanje na prijemnoj strani.

U ovom i narednom delu rada biće razmatrane osnovne tehnike koje osiguravaju bezbedno rukovanje informacijama, odnosno metode za šifrovanje i dešifrovanje osetljivih podataka kao i načine za autentifikaciju i neporecivost poruka koje se šalju. U modernim računarskim komunikacijama važno je imati mehanizme zaštite kako od neautorizovanog čitanja tako i od neautorizovanog pisanja. Generalno sisteme za šifrovanje možemo podeliti na *simetrične* i *asimetrične*. **Simetrični šifarski sistemi** podrazumevaju da se dešifrovanje vrši istim ključem koji se koristio za šifrovanje ili ključem koji je moguće izvesti iz njega. Istorijski gledano nastali su pre asimetričnih i ovde će biti opisani prvo.

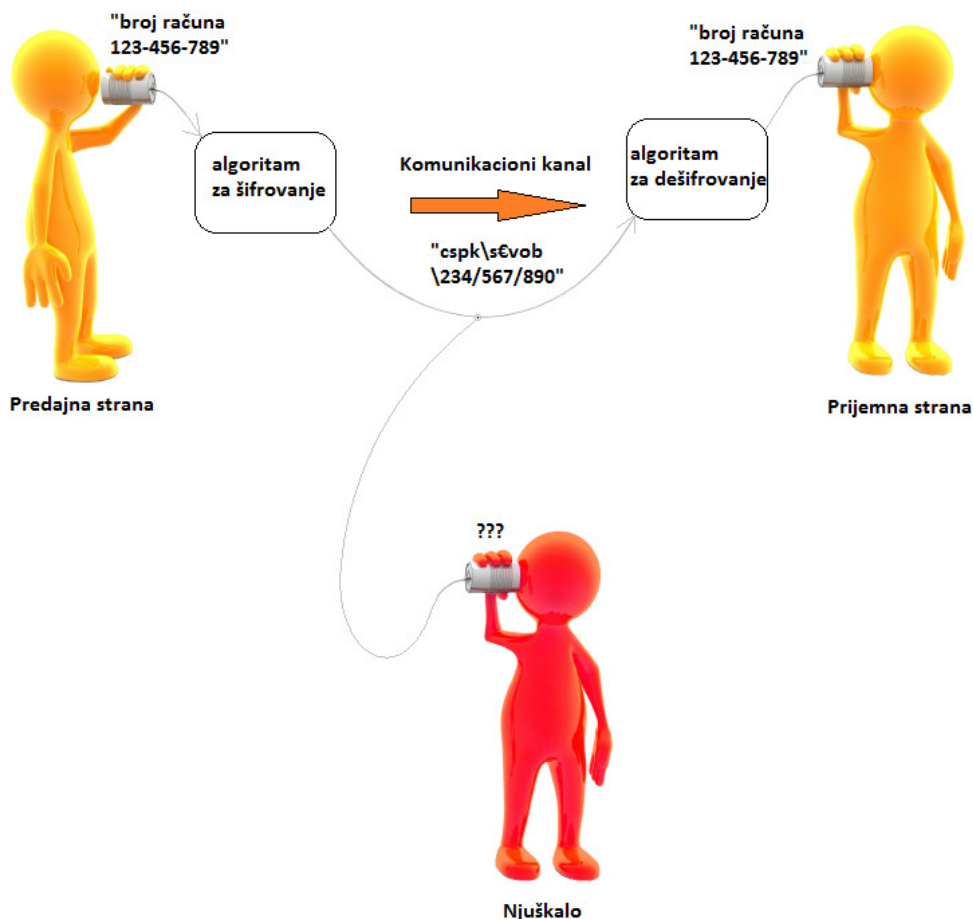
4.1 Algoritmi za šifrovanje

Uopšteno gledano svaki proces komunikacije može da se pojednostavljeno predstavi kao sistem sa prijemnom stranom, predajnom stranom, komunikacionim kanalom i trećom stranom koja neautorizovano presreće podatke koji se šalju (slika 16). Često ni prijemna a ni predajna strana neće znati da neko treći ima pristup njihovim podacima. Iako sam pristup podacima ne mora uvek da bude štetan, zloupotreba tih podataka svakako može. Dok strane u komunikaciji shvate da su im podaci kompromitovani i zloupotrebjeni već je kasno i šteta je već načinjena. Da bi se mogućnost zloupotrebe isključila odnosno svela na minimum pristupa se postupcima šifrovanja i dešifrovanja podataka odnosno algoritmima.



Sl. 16

Algoritmi za šifrovanje su stari koliko i naša moderna civilizacija a vremenom su postajali složeniji i teži za neautorizovano dešifrovanje odnosno *razbijanje* (engl. *cracking*). Prevođenje osnovne poruke na njen nerazumljivi oblik se zove još i šifrovanje ili enkripcija (engl. *encryption*) dok se suprotan proces vraćanja nerazumljive poruke u osnovni oblik zove dešifrovanje ili dekripcija (engl. *decryption*). Često se za osnovni tekst u originalnom obliku koristi reč *plaintext*, a za šifrovani *ciphertext*. Šema komunikacije putem šifrovanih poruka je data na slici 17.



Sl. 17

4.2 Cezarovo (Ceaser) šifrovanje

Jednostavan algoritam (koji je ovde prikazan iz istorijskih razloga) za šifrovanje kod kojeg se svaki karakter menja drugim karakterom se naziva monoalfabetско šifrovanje ili Cezarovo šifrovanje (veruje se da je to jedan od najstarijih algoritama). Koji će se karakter zameniti kojim proizvoljno biramo. Ključ šifrovanja odnosno način biranja zamenskih karaktera može da bude takav da recimo dodamo 1 na *ASCII* kod karaktera koji šifrujemo. Na taj način npr. poruka ABCDE postaje BCDEF. Na prijemnoj strani je tako dovoljno oduzeti 1 od *ASCII* koda svakog šifrovanog karaktera i dobijamo originalnu poruku. Za ovakav algoritam kažemo da ima identičan ključ za šifrovanje i dešifrovanje.

Algoritam je jednostavan za definisanje i formalno proizvodi nečitljive poruke potencijalnom uljezu ali ipak retko nalazi primenu u ozbiljnijim primenama. Poruka se može relativno lako razbiti odnosno dešifrovati iako ne znamo koji je ključ budući da algoritam samo prikriva često korišćene karaktere ili kombinacije. Ako poznajemo jezik na kome je poruka napisana možemo po procentu pojavljivanja određenih znakova, logikom i pogađanjem da dešifrujemo poruku bez ključa.

4.3 Polialfabetsko šifrovanje

Polialfabetsko šifrovanje predstavlja poboljšani način da se utiče na mogućnost ponavljanja pojedinih karaktera originalne poruke i time oteža razbijanje. Slično monoalfabetskom šifrovanju i ovde se svaki karakter menja drugim karakterom ali se ne koristi uvek isti karakter. Najbolje se polialfabetsko šifrovanje ilustruje primerom.

Neka su nizovi $P[]$ i $C[]$ nizovi karaktera takvi da je u P nizu poruka a u C nizu šifrovana poruka iz P niza. K predstavlja ključ koji je neki ceo broj. Sledeći deo koda vrši šifrovanje:

```
for(int a=0; i<duzinaP; a++)  
    C[a]=P[a]+K+(a mod 3);
```

Ako pretpostavimo da je $K=1$ vrednost izraza će zavisi od *ASCII* koda karaktera pa ukoliko je deljiv sa tri dodaće se 1 ako je deljiv sa tri, dodaće se 2 ako je ostatak pri deljenju sa 3 jednak 1 ili će se dodati 3 ako je ostatak pri deljenju sa 3 jednak 2. Na prvi pogled ovakav algoritam izbegava ponavljanja ali ustvari samo smanjuje ponavljanja na manju meru. Na primer ako se u poruci nađu susedni karakteri THE oni će se šifrovati kao UJH, VKF i WIG. U dužim porukama je onda ipak moguće ustanoviti šablone koji se ponavljaju pa je to ipak dosta velika olakšica onome ko pokušava da razbije šifru.

Kao i monoalfabetsko šifrovanje i polialfabetskog šifrovanje se ne koristi u modernim sistemima. Ovde je prikazano iz istorijskih razloga kao primer izmene jednostavnog algoritma kako bi se on poboljšao i otklonili njegovi nedostaci. Iz ovog primera možemo već da naslutimo da dobar algoritam za šifrovanje nije lako ni osmisliti a ni implementirati u praksi što će se videti u narednim primerima algoritama koji se danas primenjuju i koriste.

4.4 Šifrovanje premeštanjem

Umesto da algoritmom zamenjujemo karaktere drugim karakterima na razne načine možemo da pristupimo problemu i drugačije. Možemo da preuređujemo tekst osnovne poruke odnosno da premeštamo karaktere. Ovo možemo izvesti na taj način što karaktere osnovne poruke smeštamo u matričnu strukturu sa k kolona. Prvih k karaktera poruke se smešta u prvi red niza, drugih k karaktera se smešta u drugi red niza i tako redom. Sledeći korak je utvrđivanje permutacija brojeva od 1 do k koje zapisujemo redom kao $p_1, p_2, \dots, p_k$. Određivanje permutacija može da bude nasumično ili po nekom metodu za koji se opredelimo. Na kraju prenosimo redom karaktere kolone $p_1, p_2, \dots, p_k$.

Princip ovog algoritma se jasno vidi na sledećem primeru. Originalna poruka koju želimo da prenesemo glasi:

EYE OF THE WHITE TIGER

Poruku smeštamo u dvodimenzionalni niz sa pet kolona na način koji je prikazan dijagramom na slici 18.

kolone					
	1	2	3	4	5
	E	Y	E		O
	F		T	H	E
		W	H	I	T
	E		T	I	G
	E	R			

sl. 18

Ukoliko su brojevi kolona npr. uređeni u sekvenci 3, 1, 5, 2, 4 poruka koja se šalje bi izgledala ovako:

ETHT EF EE OETG Y W R HII

Preneta poruka je potpuno drugačija od originalne ali ukoliko znamo broj kolona i redosled prenošenja možemo lako da rekonstruišemo originalnu poruku. Osnovni problem kod šifrovanja premeštanjem je taj što ovaj metod nije dovoljno siguran. Analizom učestalosti ponavljanja karaktera može se zaključiti relativno lako da nije u pitanju zamena karaktera i time bi se već moglo pretpostaviti da je u pitanju ovaj algoritam. Nakon toga bi se moglo pristupiti analizi broja kolona i traženju čestih sekvenci što bi u krajnjoj instanci dovelo do razbijanja šifre.

4.5 Bitsko šifrovanje

Uopšteno gledajući postoje postupci šifrovanja koja se tiču manipulisanjima nad karakterima i postupci koji su na nivou bitova. Jedan od načina da se poruka šifruje na nivou bitova je takav da se odabere niz (*string*) bitova koji predstavlja ključ. Poruka (glavni string) se deli na podstringove bitova dužine jednake ključu. Nakon toga se svaki deo poruke šifruje izvođenjem bit operacije isključivo ILI (*Exclusive OR - XOR*) sa ključem. Ono što je interesantno kod ovakvih metoda je da se dešifrovanje poruke ne vrši posebnim inverznim postupkom nego se vrši ponovo identična *XOR* operacija sa istim ključem. Dakle dva puta primenjena *XOR* operacija sa istim bitovima daje prvobitni niz (slika 19).

	1000 1011 0010 1001	Izvorna poruka
XOR	0001 0010 1100 1010	Ključ
	1001 1001 1110 0011	Šifrovana poruka
XOR	0001 0010 1100 1010	Ključ
	1000 1011 0010 1001	Izvorna poruka

Sl. 19

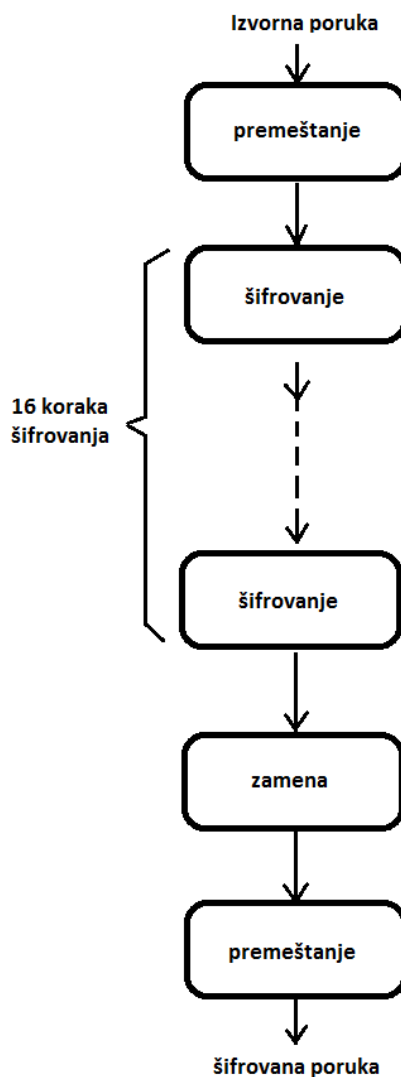
Sigurnost ove metode zavisi delimično od dužine ključa. Ukoliko se koriste kraći ključevi a poruka ima veću dužinu moguće je uočiti neke šablone koji se ponavljaju. Izbor dužih ključeva smanjuje mogućnost da se šifra razbije. U ekstremnim slučajevima koristi se ključ dužine cele poruke. Ukoliko su bitovi birani nasumično i ključ se koristi samo jednom imamo praktično poruku koju je nemoguće neautorizovano dešifrovati. Ovakvi robusni i donekle nepraktični metodi se nazivaju *one-time pads*.

4.6 DES i trostruki DES

Kao što je prethodno rečeno u prikazanim jednostavnijim metodama za šifrovanje, kraći ključevi olakšavaju razbijanje šifri dok duži ključevi smanjuju verovatnoću da do toga dođe. U praksi se ipak koriste metode sa kraćim ključevima (relativno kraćim u odnosu na kompletnu poruku) ali se u njima primenjuju složeniji postupci šifrovanja.

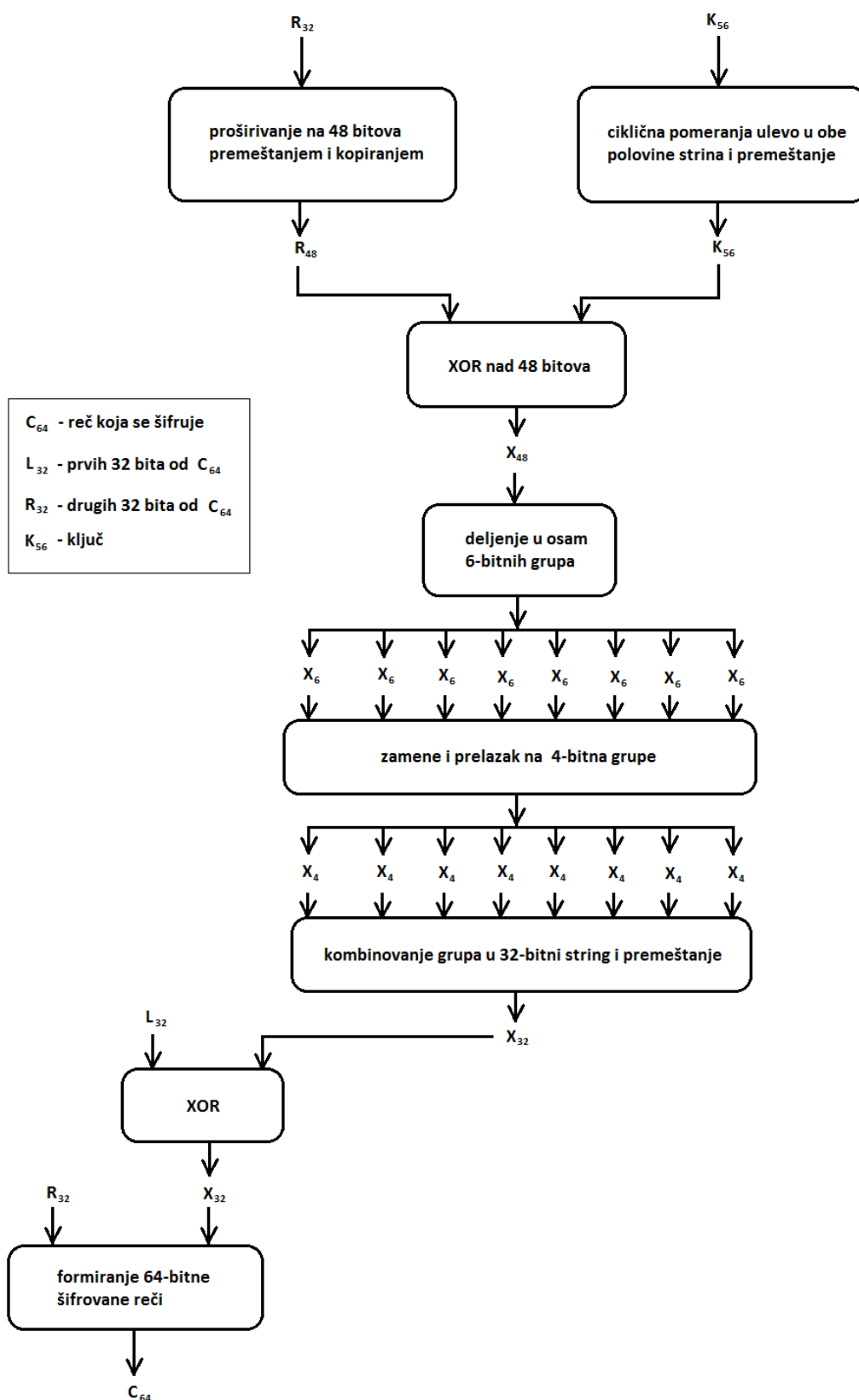
Jedan takav algoritam je nazvan **DES (Data Encryption System)**, i razvijen je od strane firme **IBM** sedamdesetih godina prošlog veka. U početku je korišćen od strane institucija američke vlade za primene koje se ne smatraju poverljivim a kasnije je primenjivan i u privatnom sektoru.

DES algoritam je primer tzv. blokovskog šifrovanja. Izvorna poruka se na početku deli u blokove od po 64 bita. Algoritam koristi 56-bitni ključ (uz još 8 bitova za detekciju grešaka daje ukupno 64 bita) u kombinacijama premeštanja, zamene i primene **XOR** operacija. Ukupno će blok bitova proći kroz 19 koraka sukcesivno (izlaz jednog koraka je ulaz za sledeći) šifrovanja da bi se dobila konačna šifrovana poruka od 64 bita. Osnovni koraci prikazani su na slici 20.



Sl. 20

Nakon prvog koraka premeštanja, sledi 16 koraka šifrovanja. Generalno su ovi koraci vrlo slični, sa tom razlikom što se za svaki korak koristi različit ključ s time da je on izveden iz početnog. Na slici 21 principijelno je prikazan jedan od koraka šifrovanja.



sl. 21

Iako ovde nije detaljno opisan vidi se da je *DES* dosta komplikovan algoritam. *IBM* ga je namerno tako i koncipirao. Ideja je od početka bila da se nizom koraka onemogući njegovo reprodukovanje bez poznavanja ključa za šifrovanje.

Inače, sam *DES* algoritam može da se koristi u više modova. Najpoznatija dva moda su *ECB* (*electronic codebook*) i *CBC* (*cipher block changing*). *ECB* mod šifruje poruku na prethodno opisani način pa ako se isti blok javi više puta u originalnoj poruci generisaće se ista i šifrovana poruka. Na ovaj način je moguće pojavljivanje šablona u šifrovanoj poruci koji bi mogli biti olakšica potencijalnom razbijaču šifre. *CBC* mod sa druge strane onemogućava ovakva ponavljanja. To se radi tako što se prvo nad prvim blokom koji se šifruje primeni XOR operacija sa nekim prethodno odabranim nizom bitova tzv. **vektorom inicijalizacije**. Dalje, kada se šifruje sledeći blok, nad njim se vrši XOR operacija sa prethodno šifrovanim blokom. Dakle svaki šifrovani blok ne zavisi samo od originalne poruke i ključa već i od sadržaja prethodnog bloka. Ovakvim pristupom se izbegavaju šabloni u šifrovanoj poruci koji se javljaju primenom drugih modova.

Uopšteno uzevši *DES* algoritmi nisu laki za razbijanje. Razbijači se služe raznim tehnikama i metodama kako bi razbili šifru nastalu ovim algoritmom. Cilj je uvek da se pronađe originalna poruka i ključ. Jedna od metoda je traženje šablona u šifrovanom tekstu odnosno parova blokova originalnog teksta koji se donekle razlikuju, ovakva metoda se naziva **diferencijalna kriptanaliza**. Naravno postoji i manje elegantne metode kao što je isprobavanje svih mogućih ključeva redom dok se ne otkrije pravi ključ, u žargonu se ovakvi postupci nazivaju **brute force**, brutalnom silom. Što je broj mogućih ključeva veći to je teže razbiti šifru ovakvim metodama.

S obzirom na 56-bitni ključ broj kombinacija je relativno veliki pa je razbijanje *DES* šifara samo donekle uspešno. Međutim, sa performansama današnjih računara ovaj posao je ipak nešto olakšan. Interesantno je da je u početku *DES* algoritam zamišljen da radi sa 128-bitnim ključem ali se kasnije od toga odustalo i svedeno je na postojećih 56 bita. Ipak vremenom se nastojalo da se ovaj algoritam poboljša zbog dinamičnog razvoja primene ovakvih sistema u današnjem vremenu.

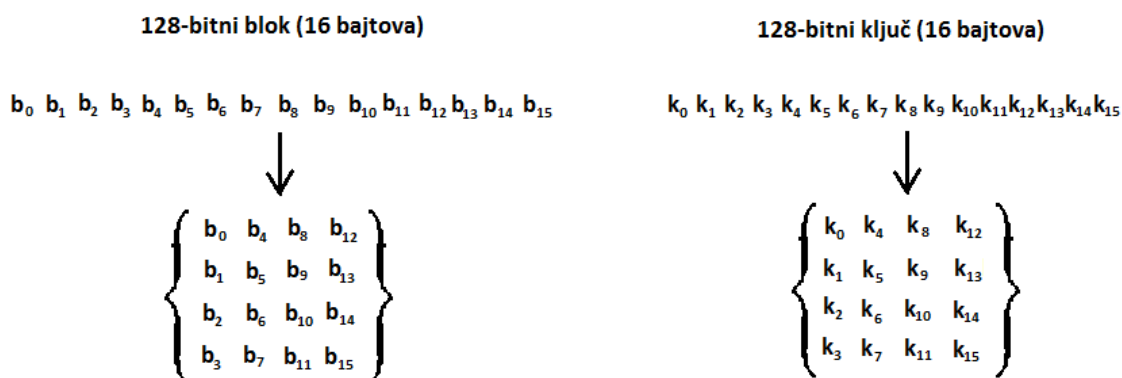
Da bi se postojećem *DES* algoritmu udahnuo novi život, koncipiran je tzv. **trostruki DES**. U osnovi se oslanja na originalni postupak ali kao što mu i samo ime govori podaci se šifruju trostruko. Osnova mu je *ANSI* standard X9.52, a radi tako što se *DES* algoritam primenjuje uzastopno tri puta. Ova varijanta algoritma koristi 168-bitni ključ kao kombinaciju tri 56-bitna ključa. Prednost trostrukog *DES*-a je u tome što se u osnovi naslanja na originalni algoritam, koji se ipak pokazao dovoljno robustan. Zadržana je i kompatibilnost sa originalnim algoritmom a to se postiže ako su sva tri dela ključa istovetni. Dužina ključa od 168-bita bi trebala da bude dovoljna da se otežaju napadi, međutim ovaj algoritam je trpeo mnoge kritike zbog svoje sporosti iako je ponekad žrtvovanje brzine sasvim opravdano zarad povećanja sigurnosti.

4.7 AES - Rijndael algoritam

Kako bi se očuvanje integriteta podataka podiglo na viši nivo, istraživanja su usmerena ka novim rešenjima koja bi otklonila nedostatke postojećeg *DES* sistema. Kao najbolji kandidat za novi standard *AES* (*Advanced Encryption Standard*) usvojen je Rijndael algoritam. Čudan naziv potiče od tvorca samog algoritma *dr Vincent Rijman* i *dr Joan Daeman*. Ovaj metod je kasnije potvrđen kao zvaničan vladin standard.

Slično kao i *DES* i *AES* koristi blokovsko šifrovanje, sa tom razlikom da postoji više varijanti algoritma. Otuda i različite oznake kao AES-128, AES-192, AES-256. Algoritam je zapravo tako zamišljen da može da radi sa različitim dužinama blokova i ključeva, odnosno ukupno devet različitih kombinacija. U najsloženijoj kombinaciji ovaj algoritam ima $1,1 \cdot 10^{77}$ mogućih ključeva. Pošto je na neki način predviđen kao naslednik starijeg *DES*-a algoritam ima najmanje jačinu trostrukog *DES* algoritma, uz tu prednost da se šifrovanje odvija brže.

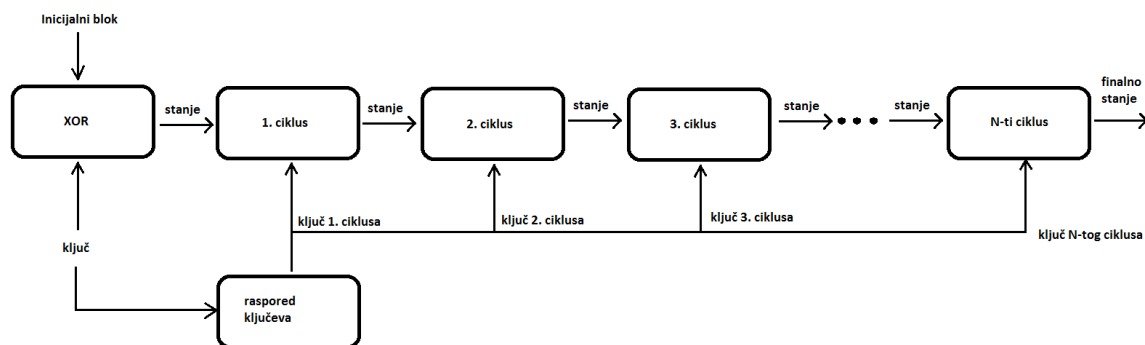
Dizajn Rijndael algoritma je složen i opis detalja bi prevazišao namenu ovog rada, ipak možemo opisati principejlnu osnovne korake. Iako smo rekli da ovaj algoritam koristi ključeve i do 256 bita zbog jednostavnosti uzećemo najjednostavniji slučaj kada su i blok i ključ veličine 128 bita. Algoritam gleda na blokove od 128 bita kao na niz od 16 bajtova koji se organizuju u matrice veličine četiri reda i četiri kolone. Slično se radi i sa ključem za šifrovanje (slika 22).



sl. 22

U prvom koraku Rijndael algoritma izvršava se XOR operacija između polaznog bloka i ključa. Izlazl prvog koraka nazivamo *stanje* koje uvodimo u prvi ciklus šifrovanja (slika 23). Broj ciklusa šifrovanja zavisi od dužine ključa pa tako se može desiti da se šifrjuje u 10, 12 ili 14 ciklusa. Karakteristika svakog ciklusa je da menja ulazno stanje koje je definisano operacijama sa matricama pri čemu se na izlazu opet dobija matrica koja se uvodi u naredni ciklus. Na kraju potrebnog broja ciklusa proces se završava.

Svaki od ciklusa koristi poseban ključ koji menja sadržaj ulaznog stanja. Ključevi se definišu procedurom za proširivanje ključeva. Formiranje ključa se počinje od definisane četiri 4-bajtna reči. Pošto je za svaki ciklus potrebno po četiri preostale reči se rekursivnom formulom izvlače iz prvobitne četiri reči. Krajnje uprošćeno ključevi su u zavisnosti od rednog broja ciklusa formiraju XOR operacijom, cikličnom permutacijom ili zamenom iz prethodno odabrane tabele koja se naziva *Sbox*.



sl. 23

Pošto su izdefinisane sve reči ključa algoritam u osnovi za svaki ciklus izvodi četiri osnovna koraka. Ta četiri osnovna koraka su:

1. **BSB** (*byte substitution*) - zamena bajta, svaki bajt u tekućoj matrici zamenjuje se drugim bajtom iz **Sbox**-a. Koristi se isti **Sbox** kao i kod procedure proširenja ključa.

2. **SR** (*shit row*) - pomeranje reda, bajtovi u svakom redu matrice, osim u prvom, se ciklično pomeraju odnosno rotiraju za određeni broj koraka koji propisuje algoritam u zavisnosti od veličine bloka i ključa.

3. **MC** (*mix column*) - mešanje kolona, najkomplikovaniji korak u kojem se bajtovi matrice interperetiraju kao koeficijenti polinoma nad kojima se vrše kompleksna izračunavanja.

4. **RKA** (*round key addition*) - uvećavanje ključa ciklusa, u ovom koraku se izvršava XOR operacija između tekućeg stanja i ključa trenutnog ciklusa.

Iz ovog grubog opisa može da se vidi da je ovaj algoritam baziran na moćnoj i složenoj matematičkoj teoriji i da je zbog svoje kompleksnosti u dobroj meri bezbedan.

4.8 Clipper Chip i Skipjack algoritam

Devedesetih godina prošlog veka američka administracija je najavila novu bezbednosnu inicijativu i u okviru tih planova našla se nova tehnologija pod nazivom *Clipper Chip*. U pitanju je bio računarski čip koji je u sebi imao ugrađenu šifru a bio je kompatibilan sa postojećom infrastrukturom. To je podrazumevalo klasične uređaje kao što su faks mašine, telefoni i slično.

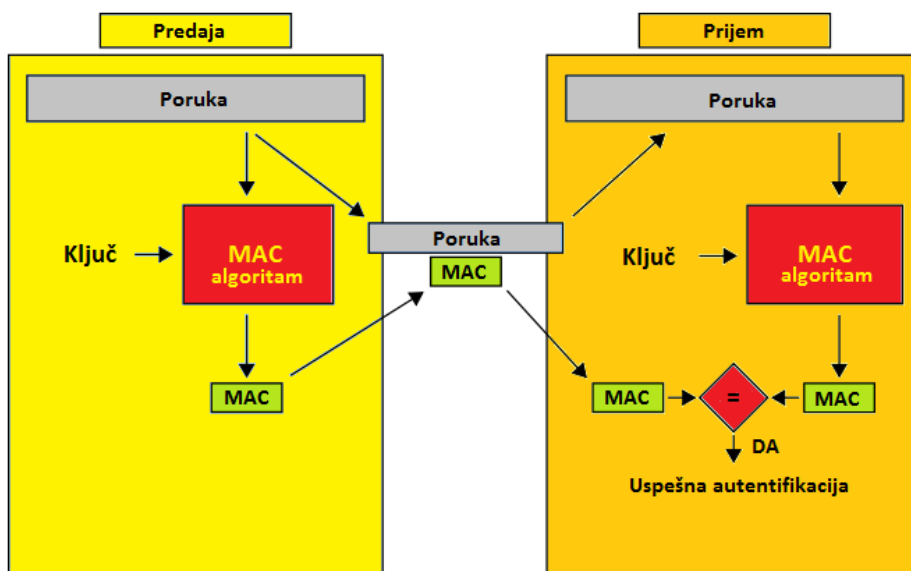
Samo funkcionisanje uređaja je jednostavno, identičan uređaj se nalazi i na predajnoj i na prijemnoj strani. Pošto je u sam uređaj ugrađena šifra, početak komunikacije podrazumeva razmenu ključa. Nadalje se komunikacija odvija kao da je u pitanju normalan telefonski razgovor ili prenos podataka. Potencijalnom uljezu koji bi presreo komunikaciju bi bile dostupne samo šifrovane poruke koji ne bi mogao da rastumači.

Algoritam za šifrovanje koji se koristi u ovim uređajima je tzv. *Skipjack* algoritam. Detalji ovog algoritma su razvijani od strane NSA i nisu u potpunosti dostupni javnosti. Zbog toga je ovaj algoritam (kao i sam uređaj) bio predmet mnogih kontroverzi i rasprava koje su se ticali nacionalne bezbednosti i ljudskih prava. U isto vreme je uređaj trebalo da obezbedi privatnost sa jedne i mogućnost prisluškivanja određenih službi sa druge strane. Ipak neke osnovne karakteristike su poznate. Svaki *Clipper chip* sadrži: ključ sesije dužine 80 bita koji se koristi za šifrovanje poruka; 80-bitni ključ grupe čipova ili *familije*; serijski broj dužine 30 bita koji je jedinstven za svaki čip; tajni ključ dužine 80 bita unikatan za svaki čip.

4.9 Integritet u simetričnim sistemima

Simetrični šifarski sistemi su prevashodno namenjeni čuvanju tajnosti odnosno poverljivosti dokumenata. Iako je čest slučaj da je poverljivost traženo svojstvo neretko se integritet podataka stavlja kao zahtev ispred poverljivosti. Ponekad nam čak poverljivost uopšte i ne treba kao svojstvo već zahtevamo samo integritet. Iz tog razloga i u simetričnim sistemima postoje mehanizmi koji pokušavaju da zadovolje zahteve za očuvanjem integriteta koji se prenose.

Jedan od metoda zaštite integriteta je tzv. **MAC** (*Message Authentication Code*) ili autentifikacioni kod poruke. U pitanju je algoritam koji za datu poruku računa njen *dodatak* (*tag - oznaku*) koristeći tajni ključ koji znaju i prijemna i predajna strana. Dodatak se dodaje na osnovnu poruku koja se zajedno sa njom prenosi (slika 24). Na prijemnoj strani se odvaja dodatak iz primljene poruke, računa se istovetnim algoritmom dodatak na primljenu poruku koristeći deljeni tajni ključ i na kraju se vrši poređenje. Ukoliko su dva dodatka (preneti i izračunati) istovetni nad prenetom porukom nisu vršene nikakve izmene u toku transporta, u suprotnom autentifikacija poruke nije uspela i zaključujemo da je sadržaj poruke kompromitovan.



sl. 24

MAC algoritam obezbeđuje integritet podataka ali nije imun na tzv. *napade ponavljanja* kod kojih napadač beleži poruku i ponavlja je nakon nekog vremena. Da bi se izbegli ovakvi napadi obično se u poruku ubacuje i neki podatak o vremenu odnosno trenutku računanja ili slično. Ovi algoritmi ne obezbeđuju svojstvo neporecivosti.

5. Asimetrični šifarski sistemi

Svi ranije spomenuti metodi za šifrovanje podrazumevali su da se dešifrovanje vrši istim ključem koji se koristio za šifrovanje ili ključem koji je moguće izvesti iz njega. Takvi sistemi se nazivaju još i **simetrični šifarski sistemi**. Sa druge strane postoje i **asimetrični šifarski sistemi** kod kojih se za proces šifrovanja i dešifrovanja koristimo različite ključeve. Ono što je posebno važno da ključ za šifrovanje može da bude i javan dok se ključ za dešifrovanje čuva u tajnosti. Svakako ovi ključevi moraju da budu u nekoj vezi. Ovakvi sistemi su poznati još i kao **kriptosistemi sa javnim ključem** i u širokoj su upotrebi.

Kriptografija sa javnim ključem ima sledeće važne karakteristike:

- upotreba matematičkih funkcija umesto operacija zamene i transpozicije
- različiti ključevi za šifrovanje i dešifrovanje
- jedan ključ je tajni, drugi može da bude i potpuno javan
- tajni i javni ključ imaju matematičku vezu
- dosta manja brzina u odnosu na simetrične sisteme
- pogodna je za upotrebu kod šifrovanja, za autentifikaciju, razmenu simetričnih ključeva, neporecivost
- nije pogodna za šifrovanje veće količine podataka

Posebna odlika asimetričnih sistema je da omogućavaju upotrebu neporecivosti. Svojstvo neporecivosti osigurava da transakcije i ostale druge akcije koje se izvršavaju preko interneta ne mogu kasnije da budu opovrgnute. Neporecivost nije moguće implementirati u simetričnim sistemima. Takođe u praksi su poznati i hibridni sistemi koji kombinuju simetrične i asimetrične metode kako bi se najbolje iskoristile prednosti oba pristupa.

Jedan od važnijih problema kriptologije je distribuiranje ključa odnosno razmena ključa između predajne i prijemne strane kao i njegovo čuvanje. Ovde će biti razmotrene osnovne metode kojima se ovaj problem prevazilazi. Takođe biće opisani sistemi koji onemogućuju da neautorizovane osobe koje poznaju algoritam i ključ za šifrovanje dešifruju poruke i utiču na njihov sadržaj.

5.1 Shamir-ov metod razmene ključa

Ideja koja je osnov za ovaj metod podrazumeva da se ključ podeli na k delova i distribuirati do k osoba. Shamir-ov metod se zasniva na interpolaciji polinoma. Kreće od pretpostavke da je potrebno k tačaka da bi se definisao polinom stepena $k-1$. Neka je funkcija:

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots + a_{k-1}x^{k-1}$$

polinom stepena $k-1$. Za $i=1, \dots, k$ određujemo tačke $(i, f(i))$, koje se dodeljuju k učesnika, osoba koje imaju dozvolu za prijem ili slanje. Pomoću poznatih tačaka moguće je interpolacijom utvrditi polinom. Koeficijent a_0 se obično koristi kao ključ za šifrovanje. Ukoliko imamo manje od k učesnika nećemo imati adekvatan broj tačaka pa utvrđivanje polinoma i dešifrovanje neće biti moguće.

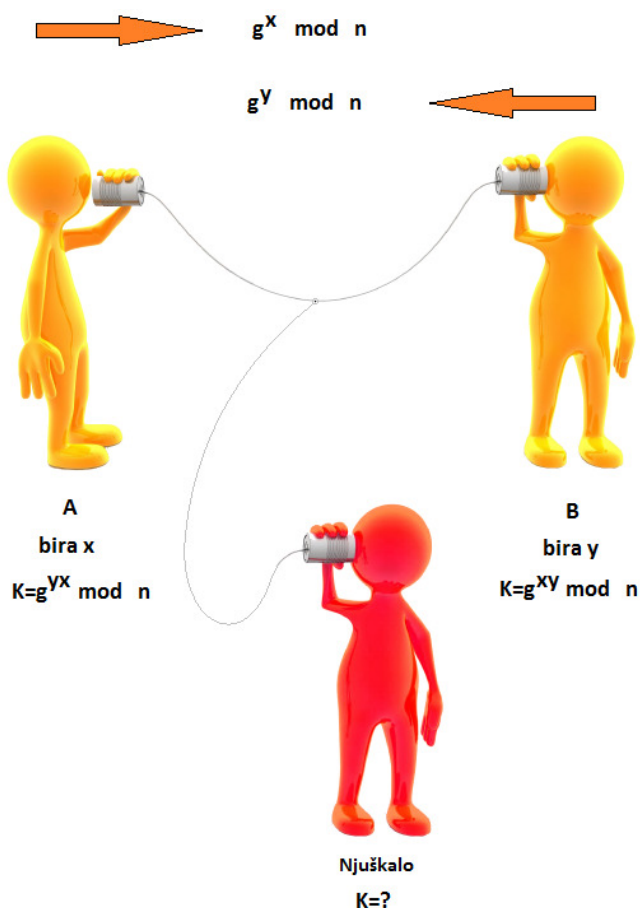
5.2 Diffie-Hellman metod razmene ključa

Kod ovog metoda predajna i prijemna strana ne razmenjuju ključeve nego izračunate vrednosti pomoću kojih je moguće doći do ključa. U tim izračunavanjima mogu da se koriste i brojevi koji nisu tajni (slika 25).

Strana A bira broj x , izračunava K i šalje. Sa druge strane B nezavisno od A bira y i izračunava K . Polazeći od osobina modularne aritmetike $g^{xy} \bmod n$ je isto što i $g^{yx} \bmod n$ pa u suštini i A i B koriste identične ključeve za šifrovanje.

Što se tiče potencijalnog uljeza koji bi prisluškivao komunikacionu liniju, njegov posao razbijanja je znatno otežan. Pošto strane A i B kriju x i y uljez ne može da kompletira izračunavanja da bi došao do ključa. U praksi se g i n biraju da budu veliki brojevi reda hiljadu bita pa je time dodatno otežano razbijanje.

Ova metoda je međutim osetljiva na vrstu napada poznatom kao *man in the middle*. Osnovna ideja ovog napada je da njuškalo presreće poruke menja im vrednost i šalje ih dalje. Na ovaj način moguće je da presreatač sluša sve poruke koje se šalju a da ni predajna ni predajna strana uopšte nisu svesne toga.



sl. 25

5.3 RSA algoritam

Algoritam je dobio naziv po svojim autorima *Rivest, Shamir i Adleman*. Koristi modularnu i eksponencijalnu aritmetiku, proste brojeve i faktORIZACIJU velikih brojeva. Metod možemo ilustrovati pomoću uprošćenog primera.

Recimo da želimo da pošaljemo neku tekstualnu poruku. Šifrovanjem primenom RSA algoritma bi podrazumevao sledeće korake:

- svakom karakteru pridružujemo prirodan broj
- biramo broj n koji je zapravo proizvod dva prosta broja p i q
- pronalazimo broj k koji je relativno prost u odnosu na $(p-1)(q-1)$
- poruku delimo na komponente (karaktere)
- svakoj komponenti pridružujemo njen binarni kod
- poruka se zatim šifrjuje tako što se svaka komponenta stepenuje sa k a zatim računa po modulu n .

Na prijemnoj strani je potrebno izvršiti dešifrovanje koje podrazumeva sledeće korake:

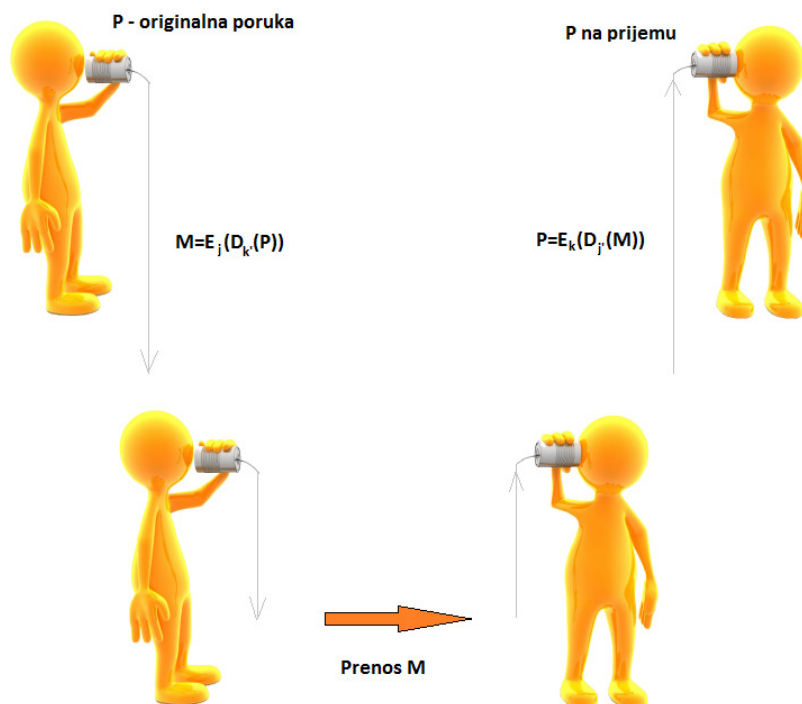
- potrebno je da pronađemo broj k' (što je naš ključ za šifrovanje) iz sledeće relacije $k \cdot k' - 1 = 0 \bmod (p-1) \cdot (q-1)$
- svaka prethodno šifrovanana komponenta se sada stepenuje sa k' a zatim se računa po modulu n .

Ovi algoritmi šifrovanja i dešifrovanja deluju jednostavno i zaista nisu preterano komplikovani. Na prvu pomisao ovo bi moglo da znači da su jednostavniji i za razbijanje ali nije tako. Bezbednost ovog algoritma se krije u tome što se koriste veliki brojevi koji i ako mogu da se izračunaju to izračunavanje bi odnelo jako puno vremena.

5.4 Digitalni potpis i digitalni sertifikati

Kriptosistemi sa javnim ključevima imaju veliku primenu u verifikaciji odnosno autentifikaciji dokumenata. U današnje vreme elektronskih bankarskih usluga i državne uprave radi efikasnosti često se koristi digitalno potpisivanje dokumenata (*eng. Digital Signature*). Principijelno je to šifrovanje kod koga način šifrovanja zna samo strana koja šalje. Pošto prijemna strana nema nikakav zapis ključa ograda je od svake zloupotrebe. Vlasnik ključa odnosno potpisa snosi svu odgovornost za čuvanje.

Na slici 26 prikazano je kako slanje digitalnog potpisa funkcioniše. Koriste se dva para metoda šifrovanja javnim ključem (E_k, D_k) i (E_j, D_j) gde su j i k javni ključevi, k' je privatni ključ koji zna samo predajna strana a j' je privatni ključ koji zna samo prijemna strana.



sl. 26

Uobičajeno je da se digitalni potpis šalje zajedno sa porukom ali ne mora uvek da bude tako, može se slati i odvojeno ali mora uvek postojati veza između dokumenta i potpisa. Kao i kod klasičnog potpisa on je vezan za original i nema smisla odvajati ga potpuno od dokumenta.

Digitalni potpis je zapravo šifrovani broj (npr. *RSA* algoritmom) određene dužine koji se dodaje na *hash* vrednost osnovne poruke koja se šalje. Digitalni potpis i dokument koji se potpisuje se mogu izraziti i sledećom formulom:

$$S = E[H(M), d]$$

gde je: S - digitalni potpis, M - dokument koji se potpisuje, $H(M)$ - hash funkcija, d - privatni ključ potpisnika, E - funkcija asimetričnog šifrovanja.

Kreiranje digitalnog potpisa vrši se u dva koraka. U prvom koraku se određuje digitalni sažetak poruke primenom odgovarajuće *hash* funkcije. U drugom koraku potpisnik šifrue dobijeni sažetak svojim privatnim ključem odgovarajućim asimetričnim algoritmom. Prilikom kreiranja potpisa podrazumeva se da je korisnik jasno definisao granice svog dokumenta.

Verifikacija potpisa sa druge strane vrši se u tri koraka. Prvi je izdvajanje potpisa iz poslate poruke i dešifrovanje javnim ključem pošaljioca. Drugi je kreiranje sažetka informacionog dela poruke istim postupkom kao na predajnoj strani. Na kraju se vrši poredenje sažetaka i ukoliko su identični verifikacija je uspešna.

Digitalni sertifikat je potpisani dokument koji omogućava mehanizam provere identiteta učesnika u komunikaciji. Veza između korisnika i javnog ključa se utvrđuje tako što se sertifikat digitalno potpisuje. Sastavni delovi sertifikata su: javni ključ, identitet korisnika, podaci o vlasniku sertifikata i digitalni potpis. Upotreba digitalnih sertifikata obezbeđuje se: identitet korisnika, integritet poruka, autorizacija, neporicanje.

Kada neki korisnik šalje neku poruku identitet utvrđujemo preko potpisa treće strane kojoj možemo da verujemo. Tada podrazumevamo da poruka nije kompromitovana. Potpis treće strane od poverenja je ustvari stigao od nekog sertifikacionog tela (*Certification Authority - CA*) koje je zaduženo da generiše, izdaje i poništava digitalne sertifikate. Potpis na sertifikatu se može proveriti pomoću javnog ključa. CA takođe potpisuje sertifikate svojim privatnim ključem. Svako može proveriti integritet sertifikata korišćenjem javnog ključa. Zajednički format svih sertifikata određen je standardom poznatim pod oznakom X.509. On određuje format zapisa i definiše polja sertifikata. Mnoge kompanije u svetu upravo koriste ovaj standard.

Infrastruktura sistema javnih ključeva (eng. *Public Key Infrastructure - PKI*) se sastoji iz sledećih komponenti:

- Korisnici
- Ovlašćene CA organizacije
- Sertifikati
- Katalozi
- Lista povučenih sertifikata

Za *PKI* ne postoji opšte prihvaćeni standard već razlikujemo više tzv. modela poverenja. Neki od modela su: *monopolski model* - predviđa jedinstvenu CA za sve korisnike; *oligarhijski model* - podrazumeva veći broj sertifikacionih tela od kojih korisnik odabira kome će ukazati poverenje, koristi se kod veb brauzera; *anarhijski model* - kod kojeg svako ima mogućnost da bude CA a korisnik sam bira kome će ukazati poverenje - koristi se kod programa *PGP* (*Pretty Good Privacy*).

5.5 Program PGP (*Pretty Good Privacy*)

PGP predstavlja integrisani kriptografski algoritam koji je dostupan na različitim platformama (*Unix, Windows, Macintosh*). Tvorac ovog algoritma je *Phill Zimmerman*. Postoji u besplatnoj verziji što ga čini jako pogodnim za razne primene. Ovaj sistem sa uspehom ispunjava zahteve **poverljivosti, autentifikacije, integriteta i neporecivosti**. Takođe *PGP* je bio jedan od prvih hibridnih kriptosistema. On koristi metode CAST, 3DES i IDEA za šifrovanje, za računanje digitalnog sažetka koristi SHA1 i MD5, za distribuciju ključa i potpisivanje koristi RSA ili DSS/Diffie-Hellman.

Objedinjeni su servisi za poruke i distribuciju ključeva. Servis poruke u sebi sadrži servise za: autentifikaciju, poverljivost, kompresiju i kompatibilnost sa programima za elektronsku poštu. Servis za distribuciju ključa integriše servis za generisanje, distribuciju i povlačenje javnih i privatnih ključeva i servis za generisanje i prenos simetričnih ključeva ili inicijalnih vektora.

PGP je opšteprihvaćeni model zaštite kod komunikacije elektronskom poštom. Kada instalira *PGP* korisnik je dužan da kreira par ključeva javni i tajni. Tajni ključ je zaštićen šifrom koja se mora uneti kad god se pristupa tajnom ključu. Poruka koja se šalje nije tajna ali se garantuje da nije izmenjena u toku prenosa.

Prilikom slanja pošaljilac potpisuje svoju poruku svojim privatnim ključem. Primalac koristi javni ključ kako bi proverio potpis. Javni ključ se uzima iz takozvanog prstena ključeva (*keyring*) primalaca kome pristupa preko e-mail adrese pošaljioaca. Putem javnih ključeva strane razmenjuju ključ sesije kojim se šifruje sama poruka (slika 27).

```

-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
MessageID: DAdVB3wzpBr3YRunZwYvhK5gBKXOb/m

qANQR1DBwU4D/TlT68XXuiUQCAdfj2o4b4aFYBcWumA7hR1Wvz9rbv2BR6WbEUsy
ZBIEFtjyqCd96qF38sp9IQiJIK1NaZfx2GLRWikPZwchUXxB+AA5+1qsG/ELBvRa
c9XefayPbbAZ6z6LkOQ+eEOXASe7aEEPfdxvZZT37dVyiYxuBBRYNLN8Bphdr2zv
z/9Ak4/OLnLiJRk05/2UNE5Z0a+3lcvITMmfGajvRhkXqocavPOKiin3hv7+Vx88

uLLem2/fQHZhGcQvkqZVqXx8SmNw5gzuvwjVlWHj9muDGBY0MkjiZIRI7azWnoU9
3KcNmpR60VO4rDRAS5uGl9fioSvze+q8XqxubaNsgdKkoD+tB/4u4c4tznLfwlL2
YBS+dzFDw5desMFS07JkecAS4NB9jAu9K+f7PTAsesCBNETDd49BTOFFTWwvAfE
gLYcPrcn4s3EriUgvL3OzPR4P1chNu6sa3ZJkTBbriDoA3VpnqG3hxqfNyOlqAka

mJJuQ53Ob9ThaFH8YcE/VqUFdw+bQtrAJ6NpjIxi/x0Ff0InhC/bBw7pDLXBFNaX
HdlLQRQqdrmnWskKznOSarxq4GjprTQo4hpCRJJ5aU7tZ09HPTZXF6iRIT0wa47

AR5nvkEkoIAjW5HaDKiJriuWldtN4OXecWvxFsJR32ebz76U8aLpAK87GEyTzBx
dV+1H0hwyT/y1cZQ/E5USePP4oKWF4uquPeelOPeFMB04CvuGyhZXD/18Ft/53Y
WIEbvdiCqsOoabK3jEfdGExce63zDI0=
=MpRf
-----END PGP MESSAGE-----

```

sl. 27

Kada poruka stigne primalac koristi svoj privatni ključ i dešifruje poruku (slika 28). PGP obezbeđuje i kompresiju podataka (najčešće se koristi ZIP algoritam) međutim ukoliko se poruka šifruje stepen kompresije ne može biti naročito velik jer poruka naizgled ima oblik niza slučajnih brojeva.

"Mislim da televizija veoma obrazuje. Kad god neko uključi TV, ja odem u drugu sobu i čitam dobru knjigu."

sl. 28

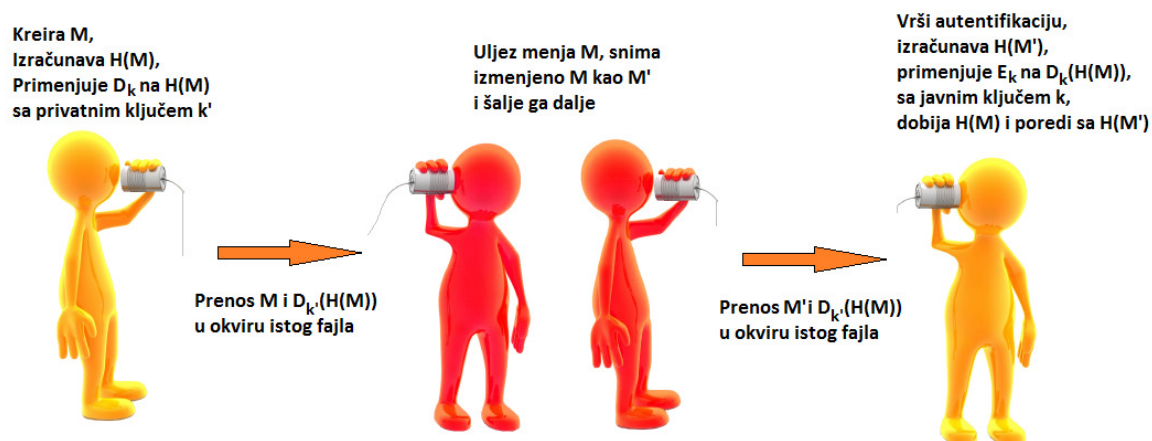
5.6 Digitalni sažetak poruke - *hash* funkcije

U prethodno opisanom metodu autentifikacije primenjeno je šifrovanje cele poruke. Međutim, u praksi se često dešava da je mnogo važnija autentičnost nekog dokumenta nego njegova tajnost. Tada nam je mnogo korisniji neki metod koji će umesto šifrovanja celog dokumenta omogućiti da detektujemo eventualne izmene koje su nastale prilikom transfera.

Pogodan način da se ovaj problem reši je upotreba tzv. *hash funkcije* ili *message digest* - digitalni sažetak poruke. Generalno gledano *hash* funkcije mogu biti različite. Jednostavna *hash* funkcija može biti i obična čeksuma. Za nas su interesantne sigurne odnosno kriptološke *hash* funkcije koje ispunjavaju neke posebne uslove.

Kao što je ranije pomenuto kod digitalnog potpisa *hash* funkcije se ne koriste za šifrovanje i zavise od sadržaja poruke koja se šalje. One kompresuju poruku sa proizvoljnim brojem bitova u niz bitova fiksne dužine (ne zavisi od dužine poruke), prema tome *hash* funkcija vrši kompresiju sa gubicima. Drugim rečima *hash* funkcija je jednosmerna funkcija, iz *hash* funkcije ne možemo nikako dobiti originalnu poruku. Dobra *hash* funkcija bi trebala da vrednosti raspoređuje uniformno tako da dobijena vrednost ima sadržan udeo slučajnosti.

Za dati dokument M postoji *hash* funkcija H koja daje *hash* vrednost $H(M)$ odnosno digitalni sažetak date poruke. Na tu vrednost se primenjuje algoritam dešifrovanja D_k sa privatnim ključem k' i rezultat ove operacije se smešta u sam dokument (slika 29).



sl. 29

Na prijemnoj strani se vrši autentifikacija dokumenta tako što se izračunava nanovo $H(M)$, zatim se primenjuje algoritam za šifrovanje sa javnim ključem k koji je odgovarajući prethodnom algoritmu za dešifrovanje privatnim ključem k' , rezultat koji se dobija je $H(M)$ i on mora biti jednak sa izračunatom vrednošću $H(M)$. Ukoliko se ove dve vrednosti razlikuju autentifikacija nije uspešna i time zaključujemo da je u toku prenosa neko menja sadržaj dokumenta.

Pitanje napada je ustvari pitanje da li je moguće izmeniti sadržaj poruke tako da se i kod izmenjenog dokumenta dobije ista vrednost $H(M)$. Pošto je to i teorijski i praktično moguće dobro odabrana *hash* funkcija treba bude jednosmerna i da zadovolji sledeće uslove:

- Za poruku M , funkcija H dodeljuje jedinstvenu vrednost fiksne dužine $H(M)=V$
- Kompresija - za proizvoljan broj bitova poruke, *hash* funkcija daje vrednost fiksne dužine koja je najčešće manja od dužine poruke
- $H(M)$ bi trebalo da se što lakše izračunava zbog efikasnosti
- Utvrđivanje M za koje je $H(M)=V$ bi trebalo da bude što teže odnosno pronalaženje poruke M' za koje je $H(M')=V$
- Što teže pronalaženje kolizije ili sudara odnosno poruka M_1 i M_2 za koje je $H(M_1)=H(M_2)$
- Lavinski efekat - promena jednog bita na ulazu menja bar polovinu bitova na izlazu

Algoritmi za izračunavanje *hash* vrednosti (*hashing* algoritmi) koji se koriste u praksi su **MD5** (*Message Digest algorithm 5*) i **SHA-1** (*Secure Hash Algorithm*).

MD5 je složen algoritam računanja *hash* vrednosti. Generiše 128-bitne vrednosti pri čemu poruku deli na 512-bitne blokove. Bitski blokovi prolaze kroz četiri ciklusa bit-operacija i faktorizacije upotrebom sinusnih funkcija. Ova *hash* funkcija je zbog svojih slabosti na kriptografske napade zamenjena modernijim rešenjima ali se zbog brzine i dalje koristi za preveru integriteta velikih fajlova.

SHA je klasa nešto bezbednijih funkcija. Najčešće se koristi SHA-1 (naslednik prvobitne SHA-0 verzije) koja koristi 160-bitnu vrednost digitalnog sažetka. Slično kao kod MD5 tako i ovaj algoritam radi sa blokovima od 512-bita pri čemu se blokovi dele na šesnaest 32-bitnih reči. Iz više ciklusa u kojima se primenjuju logičke operacije i operacije pomeranja na kraju se dobija izlazna 160-bitna vrednost. Poboljšane verzije ovog algoritma

koje su kasnije nastale nose naziv SHA-256, SHA-384, SHA-512 i SHA-224, međutim SHA-1 i dalje danas predstavlja industrijski standard u kriptografiji.

Napadi na ovakve sisteme se popularno nazivaju rođendanski napadi - *birthday attack* (naziv je potekao iz teorije verovatnoće i tzv. *rođendanskog problema*¹). Primer napada koji koristi rezultate rođendanskog problema bi mogao da bude sledeći. Recimo da hoćemo da pošaljemo nalog da se na neki račun prebaci 1.000,00 RSD. Napadač je smislio da izmeni poruku u toku prenosa i da značenje iznosa promeni jednostavnim dodavanjem dve 00 na kraju tako da umesto 1.000,00 RSD prebacujemo 100.000,00. Sama tajnost ovakve poruke nam nije važna ali nam je jako važno da ona ne bude modifikovana u toku prenosa.

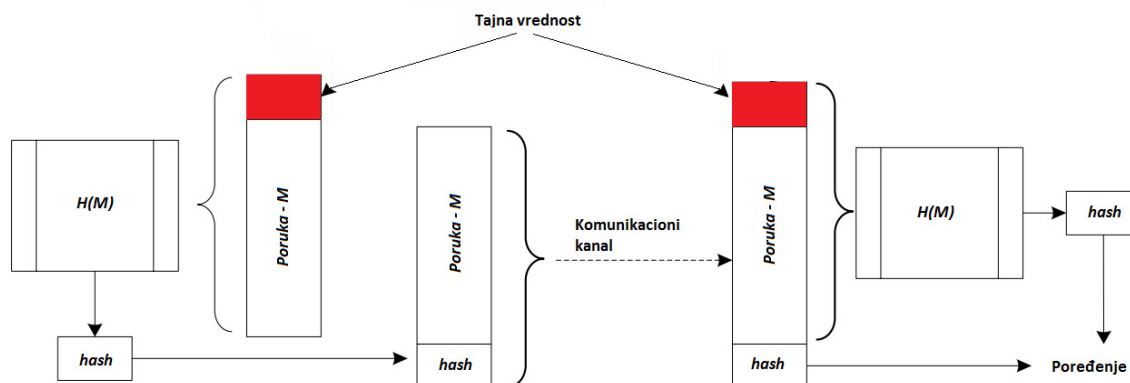
Napadač će se poslužiti sledećim trikom. Generisaće poruku koja mu je potrebna za malverzaciju međutim nedostajaće mu naš potpis. Da bi nas nadmudrio generisaće neku bezvrednu poruku koju rutinski potpisujemo jer nam ne može naneti nikakvu štetu. Pretpostavimo da *hash* funkcija koju koristimo generiše N bitova. Napadač će sada generisati $2^{\frac{n}{2}}$ bezvrednih poruka sa sitnim beznačajnim izmenama, ono što se razlikuje kod

svake iteracije je zapravo sama *hash* vrednost. Uporedo napadač generiše $2^{\frac{n}{2}}$ poruka od interesa takođe sa sitnim izmenama koje rezultuju različitim *hash* vrednostima. Cilj ovog upornog traženja je da se pronađu iste *hash* vrednosti bezvredne poruke i poruke od interesa. Kada se podudare vrednosti napadač nam šalje poruku od interesa koja ustvari ima istu *hash* vrednost kao bezvredna, naivna poruka koju potpisujemo bez razmišljanja. Rezultat je da je sada poruka koja će se zloupotребiti potpisana i ostajemo bez 99.000,00 RSD.

Ovde je u pitanju napad tzv. *brutalne sile* kod kojeg se izračunava ogroman broj *hash* vrednosti. Da bi se ovakvi napadi onemogućili u šifarskim sistemima bi trebalo birati *hash* funkcije koje daju duže vrednosti. Izračunavanje velikog broja dužih *hash* vrednosti je vremenski zahtevno pa je stoga šansa da ovakvi napadi uspeju manja.

5.7 Integritet u asimetričnim sistemima

Ranije smo videli da se kod simetričnih sistema za očuvanje integriteta koristi koncept koda za autentifikaciju poruke (*Message Authentication Code* - *MAC*). Slično tome u asimetričnim sistemima koji koriste *hash* vrednost poruke razvijen je *HMAC* (*Hashed MAC*) za verifikaciju integriteta. Osnovna ideja je da se *hash* vrednost šifrjuje



sl. 30

¹ rođendanski problem - verovatnoća da se u skupu od N ljudi nađu osobe sa istim rođendanom

međutim, moguće je izbeći šifrovanje, koje je uvek kompleksno, korišćenjem *HMAC*-a bez šifrovanja. Ovakav pristup podrazumeva da ključ - tajnu vrednost, dodamo poruci na početak ili na kraj pre izračunavanja *hash* vrednosti. Nakon toga se izračunava *hash* vrednost ukupne poruke i dodaje poruci koja se šalje (slika 30).

Slabosti ovakvog sistema je u tome što korišćene *hash* funkcije MD5 i SHA-1 rade blokovski i izračunavaju vrednost kroz više ciklusa. Napadač može primenom sile da izračuna *hash* vrednost bez znanja ključa a time se otvara mogućnost i da utiče na sadržaj poruke. Ovo je rezultat prostog lepljenja tajne vrednosti na početak ili na kraj. Problem se prevazilazi tako što se tajna poruka izmeša sa osnovnom porukom pre nego što se izračuna *hash* vrednost.

Definicija *HMAC* je određena sledećom formulom:

$$HMAC(K, m) = H((K \oplus opad) \parallel H((K \oplus ipad) \parallel M))$$

gde je: *H* - kriptografska *hash* funkcija; *K* - tajni ključ; *M* - poruka; $\parallel$ - simbol spajanja stringa; $\oplus$ - XOR operacija; *opad* - spoljna dopuna (heksadecimalni blok fiksne dužine); *ipad* - unutrašnja dopuna (heksadecimalni blok fiksne dužine).

Sigurnost ovog postupka zavisi naravno od dužine ključa. *HMAC* nije osetljiv na napad *kolizije*, napad na ovakve sisteme je brutalnom silom i sastoji se u potpunoj pretrazi tajnog ključa. Vrednosti *opad* i *ipad* je poželjno da budu što različitije.

6. Zaključak

Sistemi digitalnih telekomunikacija su doživele u poslednjoj dekadi značajna poboljšanja u smislu povećanja protoka odnosno brzine prenosa podataka. Pogotovo je to uočljiv napredak koji se tiče običnih korisnika telekomunikacionih usluga u vidu korišćenja interneta, mobilnih telefona, digitalne televizije i sl.

Brzine prenosa su se povećale mnogostruko i ide se ka tome da se ostvare nekada nezamislive brzine i do 100Mb/s ka korisniku. Ovakav napredak je najviše omogućilo usavršavanje bežičnog prenosa (*Wireless*) i *ADSL* sistema prenosa. Međutim iako su današnje brzine dostigle zavidan nivo, apetiti današnjeg prosečnog korisnika su takođe porasli. Svedoci smo fuzije interneta i televizije i radija, video konferencija u realnom vremenu i sl. Da bi svi ovi zahtevi mogli da se ostvare i dalje je neophodno razvijati algoritme kompresije, detekcije i korekcije grešaka. Poslednjih godina su se pojavili noviji algoritmi za kompresiju i dekompresiju video i audio sadržaja koji su bolje prilagođeni izazovima današnjeg kvaliteta multimedije. Ipak svi algoritmi predstavljaju manje ili više proširenje osnovnih postupaka koji su prikazani u ovom radu. Možemo da zaključimo da uprkos porastu brzine prenosa, potreba za kvalitetnim algoritmima kompresije, detekcije i korekcije grešaka nije opala već naprotiv porasla pa je i dalje ključni deo svakog današnjeg sistema digitalnih telekomunikacija.

Sa druge strane posebno je značajno korišćenje računarskih mreža u poslovnom domenu. Internet trgovina, bankarstvo, reklamiranje i slične primene su doživele pravu eksploziju u poslednjih desetak godina. Upravo ovakva primena telekomunikacija je najosetljivija i najpogodnija za zloupotrebe. Iz tih razloga razvijeni su različiti algoritmi za zaštitu podataka odnosno za garantovanje bezbednosti podataka u toku prenosa. U ovom radu su prikazane osnovne ideje i metode za šifrovanje i dešifrovanje podataka, osnovni standardi po kojima funkcionišu današnji sistemi i osnovne metode za autentifikaciju dokumenata, dakle obezbeđivanje integriteta podataka na različitim nivoima.

Sa stanovišta integriteta podataka možemo da zaključimo da su uprkos složenosti kriptosistema danas, korisnici i dalje izloženi brojnim napadima raznih vrsta. Moćni računari su danas dostupni za relativno mala sredstva a oni su osnovna alatka svakom potencijalnom uljezu koji pokušava da zloupotrebi podatke koji se šalju. Stalno poboljšavanje i unapređivanje metoda zaštite i osiguravanje integriteta podataka pri prenosu je neophodno na današnjem nivou razvijenosti globalnih komunikacija.

Literatura

- [1] A. S. Tanenbaum, **Computer Networks**, Prentice Hall, 2002.
- [2] S. William, **Savremene komunikacione tehnologije i mreže**, Cengage Learning, 2003.
- [3] C. Douglas, **Povezivanje mreža, principi protokoli i arhitekture**, Pearson, 2003.
- [4] M. Veinović, S. Adamović, **Kriptologija 1**, Univerzitet Singidunum, 2013.
- [5] M. Veinović, A. Jevremović, **Računarske mreže**, Univerzitet Singidunum, 2016.
- [6] J. Kurose, K. Ross, **Umrežavanje računara od vrha ka dnu**, Računarski fakultet, 2014.
- [7] D. Pleskonjić, Nj. Maček, B. Đorđević, M. Carić, **Sigurnost računarskih sistema i mreža**, Mikro knjiga, 2007.
- [8] W. Marcus, **How Video Works: From Analog to High Definition**, Focal Press, 2007.
- [9] G. Jerry, **Multimedia Communications - Directions & Innovations**, San Diego, 2001.
- [10] D. Salomon, **Data Compression. The Complete Reference**, Springer, 2004.
- [11] C. Wootton, **A Practical Guide to Video and Audio Compression**, Focal Press, 2005.
- [12] I. E. G. Richardson, **H.264 and MPEG-4 Video Compression**, Wiley, 2003.
- [13] S. Turk, **Računarske mreže**, Školska knjiga, 1991.
- [14] D. V. Davis, **Računarske mreže i protokoli**, Vojnoizdavački zavod, 1986.
- [15] W. Feibel, **Novellova Enciklopedija računarskih mreža**, Mikro knjiga, 1997.
- [16] Daemen, J., and V. Rijmen. **The Design of Rijndael: AES, the Advanced Encryption Standard**, Heidelberg, Germany: Springer-Verlag, 2002.
- [17] Denning, P., ed. **Computers Under Attack: Intruders, Worms, and Viruses**, Reading, MA: Addison-Wesley, 1990.
- [18] Mollin, R. **An Introduction to Cryptography**, Boca Raton, FL, Chapman & Hall/CRC Press, 2001.