

U n i v e r z i t e t u N i š u
E l e k t r o n s k i f a k u l t e t

Vladan Vučković, Petar Rajković

ZAŠTITA INFORMACIJA



Edicija: Osnovni udžbenici

U n i v e r z i t e t u N i š u
E l e k t r o n s k i f a k u l t e t

Vladan Vučković, Petar Rajković

ZAŠTITA INFORMACIJA

ZAŠTITA INFORMACIJA

Autori: Prof. dr Vladan Vučković, Petar Rajković

Izdavač: Elektronski fakultet u Nišu
P. fah 73, 18000 Niš
<http://www.elfak.ni.ac.rs>

Recenzenti: prof. dr Dejan Rančić, Elektronski fakultet u Nišu
prof. dr Daniela Milović, Elektronski fakultet u Nišu
prof. dr Miodrag Petković, Elektronski fakultet u Nišu (u penziji)

Glavni i odgovorni urednik: prof. dr _____

Odlukom Nastavno-naučnog veća Elektronskog fakulteta u Nišu, br. 07/05-

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004(075.8)

СТОЈКОВИЋ, Сузана, 1966-

Увод у рачунарство / Suzana Stojković, Natalija Stojanović, Dragan Stojanović. - Ниш: Електронски факултет, 2014 (Niš : Unigraf-X-Copy). - V, 267 str. : илустр. ; 24cm. – (#Едација #Основни уџбеници / [Електронски факултет, Ниш])

На врху насл. стр.: Универзитет у Нишу. -
Тираž 300. - Bibliografija: str. 267.

ISBN 978-86-6125-112-2

1. Стојановић, Наталија, 1974- [автор] 2. Стојановић, Драган, 1969- [автор]

а) Рачунарство

COBISS.SR-ID 210729228

Preštampavanje ili umnožavanje ove knjige nije dozvoljeno bez pismene dozvole izdavača.

Tiraž: 300 primeraka

Štampa: UNIGRAF-X-COPY, Niš

U n i v e r z i t e t u N i š u
E l e k t r o n s k i f a k u l t e t

Vladan Vučković, Petar Rajković

ZAŠTITA INFORMACIJA



Edicija: Osnovni udžbenici

2016.

Predgovor

Udžbenik „**Zaštita informacija**“ obezbeđuje neophodne informacije studentima za savladavanje gradiva iz i u skladu je sa planom i programom sličnih predmeta u ovoj oblasti. Udžbenik je namenjen studentima elektronskih i elektrotehničkih fakulteta, drugih tehničkih fakulteta, prirodno-matematičkih fakulteta, kao i ostalih srodnih fakulteta kao udžbenik u oblasti Zaštite informacija (*Information security*), a studentima računarstva i informatike obezbeđuje dobru osnovu i detaljan pregled osnovnih tema iz ove jako propulzivne oblasti nauke i tehnologije. Istoimeni predmet, baziran na knjizi **Mark Stamp-a: Information Security – Principles and Practices** autori drže na višim godinama akademskih i master studija (Kriptografija) duži niz godina, tako da je materijal i sadžaj ove knjige delimično prilagođen standardnom kursu iz ove oblasti, ali sa naglaskom na praktični deo kako u oblasti matematičke osnove tako i u delu algoritama za kodiranje. Studentima drugih studijskih programa elektronskih i elektrotehničkih fakulteta, kao i drugih tehničkih i prirodno-matematičkih fakulteta pruža uvid u ovu oblast računarstva i informatike kao i razumevanje osnovnih metoda, tehnologija i principa ove sve aktuelnije naučne i inženjerske discipline koja veoma značajno utiče na sve aspekte savremenog ljudskog društva.

Autori se zahvaljuju recenzentima, prof. dr Predragu Rančiću, prof. dr Danieli Milović i prof. dr Miodragu Petkoviću (u penziji), na angažovanju i trudu koji su uložili recenzirajući ovaj udžbenik, kao i na zapažanjima, primedbama i korisnim sugestijama, koje su doprinele kvalitetu ovog udžbenika.

Takođe, autori se zahvaljuju Nini Andrejević, apsolventu Elektronskog fakulteta u Nišu za tehničku podršku u realizaciji rukopisa.

Autori

Sadržaj

1	UVOD	1
2	OSNOVNI POJMOVI	3
3	OSNOVNI KRIPTO – ALGORITMI	7
3.1	Pomeraj za N (shift-by-N)	7
3.2	Jednostavna zamena (simple substitution cipher)	7
3.2.1	<i>Proces dekodiranja</i>	8
3.3	Jednokratni ključ (one-time-pad)	10
3.3.1	<i>Primer</i>	11
3.4	Dvostruka transpozicija (Double Transposition Cipher)	13
3.5	Kodiranje po rečniku (Codebook cipher)	13
3.5.1	<i>Elections 187</i>	15
4	SIMETRIČNI KRIPTO - KODERI TOKA	17
4.1	Kodovi toka podataka (stream ciphers)	17
4.2	A5/1 Algoritam	18
4.3	A5/2 Algoritam	20
4.4	RC4	26
4.4.1	<i>RC4 – KSA Algoritam</i>	26
4.4.2	<i>PRGA algoritam (The pseudo – random generation algoritam)</i>	27
4.4.3	<i>Celokupni RC4</i>	27
4.4.4	<i>Napad na RC4</i>	29
4.5	RC6	34
4.5.1	<i>Prednosti</i>	34
4.5.2	<i>Nedostaci</i>	34
4.5.3	<i>Specifikacije</i>	34
5	SIMETRIČNI KODERI - KODERI BLOKA	39
5.1	DES	41
5.1.1	<i>Opšta struktura</i>	42
5.1.2	<i>Fejstelova (F) funkcija kod DES algoritma</i>	43
5.1.3	<i>Algoritam za distribuciju u DES algoritmu</i>	45
5.2	Modovi kodera	47
5.2.1	<i>ECB</i>	47
5.2.2	<i>CBC</i>	47
5.2.3	<i>CRT</i>	48
5.3	TDES (3DES)	48
5.4	AES	49
5.4.1	<i>SubBytes operacija</i>	49

5.4.2 <i>ShiftRows operacija</i>	51
5.4.3 <i>MixColumns operacija</i>	51
5.4.4 <i>AddRoundKey Operacija</i>	54
5.5 Rijndael algoritam za distribuciju ključeva	54
5.5.1 <i>Rotacija</i>	54
5.5.2 <i>Rcon i SBox preslikavanja</i>	55
5.5.3 <i>Osnovna raspodela ključeva</i>	56
5.5.4 <i>Konstante</i>	57
5.5.5 <i>Opis distribucije ključeva</i>	57
5.5.6 <i>Kod</i>	57
5.6 TEA algoritam	60
5.7 Unapređenja TEA algoritma	62
5.7.1 <i>Implementacija Blok TEA algoritma</i>	64
5.8 Opis XXTEA kodera	65
6 ASIMETRIČNO KRIPTOVANJE	67
6.1 RSA	67
6.1.1 <i>Generisanje ključeva</i>	67
6.1.2 <i>Kodovanje</i>	67
6.1.3 <i>Dekodovanje</i>	68
6.2 KNAPSACK	68
6.2.1 <i>Generisanje ključeva</i>	68
6.2.2 <i>Kriptovanje</i>	69
6.2.3 <i>Dekriptovanje</i>	69
7 KRIPTOGRAFSKI HEŠVI	71
7.1 Primer za CRC	71
7.1.1 <i>Računanje CRC koda</i>	73
7.2 Tiger Hash	75
7.3 HMAC	77
7.4 MD5algoritam	79
7.4.1 <i>MD5 heš-ovi</i>	82
7.5 SHA-1	82
7.6 SHA-2 algoritam	85
7.7 Univerzalno heširanje	89
8 ODABRANA TEORETSKA POGLAVLJA	93
8.1 Moderna kripto istorija – Mašina za šifriranje Enigma	93
8.2 Režimi rada Block kodera-a	95
8.3 Integritet	96
8.3.1 <i>Diffie-Hellman</i>	96
8.4 Korišćenje eliptičnih krivih u kriptografiji	97
8.5 Hash funkcije	97
8.6 Rođendanski problem – (THE BIRTHDAY PROBLEM)	98
8.7 Tiger HASH	99
8.8 Sakrivanje informacija	100

ZAŠTITA INFORMACIJA

8.9 Kontrola pristupa	102
8.9.1 Utvrđivanje autentičnosti (<i>AUTENTIFIKACIJA</i>)	102
8.9.2 Metode autentifikacije	102
8.10 Lozinke	103
8.11 Biometrija	103
8.11.1 Otisci prstiju	105
8.11.2 Geometrija šake	106
8.11.3 Skeniranje dužice oka (<i>Iris skeniranje</i>)	107
8.11.4 Stope grešaka biometrijskih Sistema	108
8.11.5 Biometrijski metodi – zaključak	108
8.12 Identifikacija na osnovu nečega što korisnik ima	109
8.13 Ovlašćenje (<i>AUTORIZACIJA</i>)	110
8.14 Matrica kontrole pristupa	110
8.14.1 <i>ACL i C-liste</i>	110
8.15 Bell-LaPadula model sigurnosti	112
8.16 COVERT KANAL	112
8.17 CAPTCHA	112
8.18 FIREWALL	113
8.18.1 Paket filter	114
8.19 Jednostavnii sigurnosni protokoli	114
8.20 Protokoli za utvrđivanje autentičnosti	116
8.21 Greške u softveru i Malware	118
8.22 Softverske greške	118
8.23 MALWARE	120
9 ZADACI SA VEŽBI	121
9.1 One time pad	121
9.1.1 Zadatak 1	121
9.1.2 Zadatak 2	124
9.1.3 Zadatak 3	127
9.1.4 Zadatak 4	130
9.1.5 Zadatak 5	132
9.2 Double Transposition	134
9.2.1 Zadatak 1	134
9.2.2 Zadatak 2	138
9.2.3 Zadatak 3	143
9.2.4 Zadatak 4	145
9.2.5 Zadatak 5	146
9.3 A5/1	148
9.3.1 Zadatak 1	148
9.3.2 Zadatak 2	151
9.3.3 Zadatak 3	155
9.4 RC4	158
9.4.1 Zadatak 1	158
9.4.2 Zadatak 2	161

9.5 Knapsack	163
9.5.1 <i>Zadatak 1</i>	163
9.5.2 <i>Zadatak 2</i>	165
9.5.3 <i>Zadatak 3</i>	167
9.5.4 <i>Zadatak 4</i>	168
9.6 RSA	170
9.6.1 <i>Zadatak 1</i>	170
9.6.2 <i>Zadatak 2</i>	171
9.6.3 <i>Zadatak 3</i>	174
9.6.4 <i>Zadatak 4</i>	175
9.7 Modovi kodera	177
9.7.1 <i>Output feedback mod</i>	177
9.7.2 <i>COUNTER MODE (CTR)</i>	181
9.7.3 <i>Propagating Cipher Block Chaining PCBC mod</i>	184
10 ZAKLJUČAK	191

1 UVOD

Udžbenik iz predmeta "Zaštita Informacija", namenjen je pre svega studentima istoimenog kursa, koji se na Elektronskom fakultetu u Nišu sluša na završnim godinama. Sama problematika kojom se bavi priručnik je veoma aktuelna, naročito u moderno doba u kome se sve veća količina informacija prenosi digitalnim putem, uz upotrebu računara. Mogućnosti neovlaštenog pristupa, prikupljanja ili menjanja originalnih informacija su brojne, tako da potreba za njihovom zaštitom postaje veoma urgentna. Za razliku od starih sistema zaštite informacija, koji su razvijani u doba kada nisu postojale digitalne računske mašine, u priručniku je dat oslonac upravo na moderne tehnike zaštite, bazirane na algoritamskom pristupu i modernim procedurama i programima za kodiranje i dekodiranje informacija.

Ovaj udžbenik ne predstavlja teoretsko razmatranje i klasifikaciju ove naučne oblasti koja se veoma brzo razvija, već je proistekla iz praktičnog rada i iskustva koji su autori stekli držeći teoretska predavanja i računske vežbe iz istoimenog predmeta. Priručnik je namenjen pre svega savladavanju osnovnih algoritama kodiranja i dekodiranja informacija, kroz brojne primere, koji su na sto je moguće jednostavniji način objašnjeni i proradjeni. Suština zaštite informacija, pogotovo u njenom digitalnom domenu, je upravo u upotrebi tih algoritama, tako da se autori nadaju da njihov pristup predstavlja najkraći put za savladavanje obimnog, pre svega praktičnog gradiva iz ove oblasti.

Materija je izložena u nekoliko poglavlja.

Posle Uvoda, u drugom poglavlju predstavljenu su osnovni kriptoalgoritmi kao što su: pomeraj za N (*shift-by-N*) i jednostavna zamena (*simple substitution cipher*). Razmatran je zatim proces kodiranja, sistem sa jednokratnim ključem (*one-time-pad*), dvostruka transpozicija (*Double Transposition Cipher*), Kodiranje po rečniku (*Codebook cipher*), *Elections 1876 cipher* (Primenjen na čuvenim izborima u Americi 1876).

U trećem poglavlju obrađeni su simetrični koderi toka kao što su: koderi toka podataka (*stream ciphers*), A5/1 i A5/2 algoritam, RC4 - KSA algoritam, PRGA algoritam (*The pseudo-random generation algorithm*), zatim RC4 i RC6 algoritmi.

U četvrtom poglavlju su obrađeni simetrični koderi bloka. Detaljno su raymatrani: DES algoritam, Fajstelova (F) funkcija kod DES algoritma, zatim Algoritam za distribuciju ključa, modovi kodera, ECB (*Electronic Code Book*), CBC, CTR, TDES (3DES), AES algoritam sa upotrebom *SubBZtes*, *ShiftRows*, *MixColumns* i *AddRoundKeZ* operacijom, zatim *Rijndael* algoritam za distribuciju ključeva, algoritam rotacije, Rcon i SBox preslikavanja, osnovna raspodela ključeva i distribucija ključeva. Predstavljen je zatim TEA algoritam sa Blok TEA implementacijom kao i opis XXTEA kodera.

ZAŠTITA INFORMACIJA

U petom poglavlju obrađeno je asimetrično kriptovanje i osnovni RSA algoritam: Kodiranje, Dekodiranje, zatim *Knapsack* algoritam sa generisanjem ključeva, kriptovanjem i dekriptovanjem.

U šestom poglavlju razmatrane su kriptografske (*Hash*) funkcije, sa primerima za CRC, kao i računanje CRC koda. Detaljno je obrađen *Tiger Hash,HMAC,MD5* algoritam i *MD5* heš-ovi, *SHA-1* i *SHA-2* algoritam sa univerzalnim heširanjem.

U sedmom poglavlju razmatrane su izabrane teme iz široke oblasti Zaštite informacija, ključnih momenata u istorijskom razvoju oblasti, teoriji i praksi. Razmatrana je prva kriptografska mašina *Enigma* kao i prvi kriptoanalitički sistemi koji su doveli i do razvoja elektronskih računskih mašina. U nastavku su prikazani neki od osnovnih ECB/CBC modova, kao pojmove koji se odnose na integritet informacija kao i na teorijske osnove *Diffie-Hellman*-ovog algoritma sa zamenom ključeva. Obraden je i pristup sistemima kodiranja preko eliptičkih krivih. Naročita pažnja posvećena je HASH funkcijama. Sakrivanje informacija (steganografija) je obrađena u nastavku. Sistemi kontrole pristupa, autentifikacija i autorizacija, uključujući i detaljno razmatranje biometričkih metoda je takođe prikazano u nastavku poglavlja. Osnovni pojmovi u realizaciji protokola za prenos podataka kao tipovi softvera sa greškom (*malware*) su prikazani na kraju ovog poglavlja. Sadržaj ovog poglavlja prilagođen je ključnim temama i poglavljima iz osnovnog udžbenika Mark Stamp-a: *Information Security – Principles and Practices* [1] po kome se godinama drže teoretska predavanja na istoimenom predmetu.

U osmom poglavlju detaljno su obrađeni nekoliko ključnih algoritama iz kriptografije, koji se intenzivno prorađuju na računskim vežbama. Primeri su veoma ilustrativni i mogu dosta pomoći čitaocu u razumevanju funkcionisanja nekih osnovnih mehanizama za kodiranje.

Imajući u vidu da na našem jeziku ne postoji dovoljno adekvatne literature, pogotovo za ovaj predmet koji se nekoliko godina drži kao osnovni kurs prema stranoj literaturi, autori smatraju da je ovo izdanje konkretan doprinos nastavi i efikasnom usvajanju znanja iz ove oblasti, kao i pripremama studenata za ispite ili kasniji rad u ovoj veoma dinamičnoj i interesantnoj oblasti.

2 OSNOVNI POJMOVI

Bezbednost informacija je široka tema, i za razliku od nekih drugih oblasti u računarstvu nije uvek jasno koji sadržaji treba da budu uključeni u sadržaj ovog udžbenika a takođe i kako je najbolje organizovati izabrani materijal. Mi smo izabrali da organizujemo ovaj udžbenik oko sledeće četiri osnovne teme koje generalno obrađuje zaštitu informacija:

- Kriptografija,
- Kontrola pristupa,
- Protokoli,
- Softver.

Ličnosti (pojmove) koje uključujemo u dalja razmatranja su **Alisa** i **Bob** kao *dobre* ličnosti što će povremeno biti i **Charlie**.

Trudi je *loš* momak koji pokušava da napadne sistem na neki način. Neki autori se koriste tim lošim momcima gde naziv implicira određenu negativnu aktivnost. U toj upotrebi, Trudi je "uljez", Eva je "prisluškivač" i tako dalje. Trudi će biti naš opštenamenski negativac.

Alice, Bob, Trudi i ne moraju da budu ljudi. Na primer, jedan mogući scenario bi bio da Alice je laptop, Bob server, a Trudi čovek.

Pojam *poverljivost* ima za cilj da spreči neovlašćeno čitanje informacija. Na primer, Bob ne želi da Trudi zna koliko novca ima u svom štednom računu. Alice (banka) će se takođe suočiti sa pravnim problemima, ukoliko ne bi zaštitala poverljivost informacija.

Informacije imaju integritet u slučaju da je zabranjeno neovlašćeno menjanje istih. Alice - banka mora da štiti integritet podataka računa sprečavajući da Trudi, recimo, izazove povećanje stanja na svom računu ili promeni stanje na računu Bob-a.

Denial of Service, ili DoS, napadi su relativno nedavnog karaktera. Ovakvi napadi pokušavaju da ograniče pristup informacijama. Kao rezultat porasta DoS napada, raspoloživost podataka je postala osnovni problem u informacionoj bezbednosti. U primeru koji je prikazan [1] ako bezbednost nije ispunjena, onda Alice ne može zaraditi novac od transakcija klijenata i Bob-ov posao neće imati smisla.

Kriptografija ili "tajni kodovi" su temeljni sigurnosni alat za zaštitu informacija. Kriptografija obuhvata mnoge oblasti, uključujući zaštitu tajnosti i integriteta kao i mnoge druge bezbednosne funkcije od vitalnog značaja. Mi ćemo obraditi kriptografiju izuzetno detaljno imajući u vidu da je to neophodna matematička osnova za veći deo ostatka udžbenika.

Počećemo našu diskusiju o kriptografiji sa pregledom klasičnih sistema za šifrovanje. Ovi klasični sistemi ilustruju osnovne principe koji su primjenjeni u savremenim digitalnim uređajima za šifrovanje.

Kriptografski sistemi: *simetrični ključ* i *javni ključ* igraju glavne uloge u informacionoj bezbednosti, tako da ćemo najdetaljnije obraditi ovu temu. Takođe, prikazaćemo i *hash* funkcije, koje su jedan od osnovnih kriptografskih alata. Heš funkcije se koriste u različitim kontekstima u informacionoj bezbednosti. Neke od ovih upotreba su prilično iznenađujuće i ne uvek intuitivne. CRC kodovi su odličan primer.

Takođe, ukratko ćemo razmotriti nekoliko posebnih tema koje se odnose na oblast kriptografije. Na primer, prikazaćemo sakrivanje informacija gde je cilj da Alisa i Bob komuniciraju bez Trudi, koji ne zna da je informacija čak i preneta. Ovo je usko povezano sa konceptom digitalnog vodenog žiga, koji je takođe razmatran.

U početnom poglavlju, bavićemo se nekim od osnovnih elemenata kriptografije. Mi ćemo se truditi da izbegnemo matematički formalizam koliko je to moguće, ali ćemo pokušati da obezbedimo dovoljno detalja, tako da čitalac bude u stanju da odgovori ne samo na pitanje "šta" već i "zašto".

Posle ovog uvodnog poglavlja, fokus preostalih poglavlja u kriptografiji je orijentisan na četiri glavne teme:

- simetrični ključ u kriptografiji,
- javni ključ u kriptografiji,
- heš funkcije,
- napredna kriptoanaliza.



Slika 2.1 Sistem za šifriranje predstavljen kao crna kutija.

Osnovni „kripto“ termini su sledeći:

- **Kriptologija** (Cryptology) je umetnost i nauka kreiranja i dešifrovanja „tajnih kodova“.
- Postupak kreiranja tajnih kodova se zove **kriptografija** (Cryptography).
- Postupak dešifrovanja ili razbijanja tajnih kodova naziva se **kriptoanaliza** (Cryptanalysis).
- **Kripto** je sinonim za bilo koji od prethodno navedenih termina.
- Sistem, bilo softverski ili hardverski, koji služi da kreira kodove naziva se **kriptosistem** (cryptosystem) ili **koder** (cipher).

- Podatak koji treba kodovati naziva se **polazni podatak ili polazni tekst** (plaintext), a **rezultat kodovanja** (encryption) se naziva **kodirani podatak ili kodirani tekst** (ciphertext).
- Kodirani tekst se dekoduje ili dekriptuje (decrypt) kako bi se dobio polazni podatak.
- Ključ je podatak koji služi za **podešavanje** (configure) kripto sistema bilo za kodiranje ili dekodiranje.
- Ako kripto sistem koristi isti ključ i za kodiranje i za dekodiranje onda se on naziva simetrični kripto sistem. U suprotnom je **asimetričan**.
- Slova u polaznoj poruci se najčešće pišu kao mala slova, a slova u kodiranoj poruci kao velika.

3 OSNOVNI KRIPTO-ALGORITMI

3.1 Pomeraj za N (shift-by-N)

Prepostavimo da želimo da kriptujemo (šifriramo) neki tekst. Bilo koji tekst je napisan korišćenjem slova neke azbuke. Prvi korak je da ispišemo sva slova azbuke u kojoj će početni tekst biti napisan i to u redosledu koji je poznat onome kome šaljemo poruku.

Npr: abcdefghijklmnopqrstuvwxyz

Nakon toga definisemo ključ, odnosno pomeraj, odnosno N. N treba da bude broj u opsegu od 1 do broja slova u polaznoj azbuci. Zamenimo svako slovo polazne azbuke slovom koje je za N udaljeno od njega i na taj način dobijamo preslikavanje koje ćemo koristiti za kodiranje.

Npr za N=3, osnovna azbuka se pretvara u: DEFGHIJKLMNOPQRSTUVWXYZABC.

A preslikavanje koje bi koristili za kodiranje polazne poruke bi bilo: a zameni sa D, b zameni sa E, c sa F, ..., x sa A, y sa B i z sa C.

Hajde da kriptujemo reč zdravo: z se menja sa C, d sa G, r sa U, a sa D, v sa Y, o sa R.

Dakle: C(zdravo, 3) = CGUDYR.

Primer: za N=4, kriptujmo: fourscoreandsevenyearsago.

C(fourscoreandsevenyearsago, 4) = jsyvwgsvierhwizircievweks.

Dekodiranje ovakvog teksta se svodi na pronađenje vrednosti N za koju je generisana azbuka za kriptovanje. Kako su azbuke sastavljene od relativno malog broja simbola (manje od 5000), to pokazuje da je dekodiranje ovog kriptosistema izuzetno jednostavno.

3.2 Jednostavna zamena (simple substitution cipher)

Zamena jednog slova azbuke drugim je jedan od prvih načina kodiranja. Prethodno pomenuti metod Rimljani su uspešno koristili skoro 300 godina. Međutim mali broj kombinacija čini ovaj sistem prilično ranjivim. Zbog toga je nastao i sistem koji se naziva sistem jednostavne zamene. Osnovna ideja kod ovoga sistema je permutovati slova polazne azbuke [16].

Osnovna azbuka: abcdefghijklmnopqrstuvwxyz

Azbuka za kodiranje: ZPBYJRGKFLXQNWVDHMSUTOIAEC

Primer za prethodno navedeni ključ:

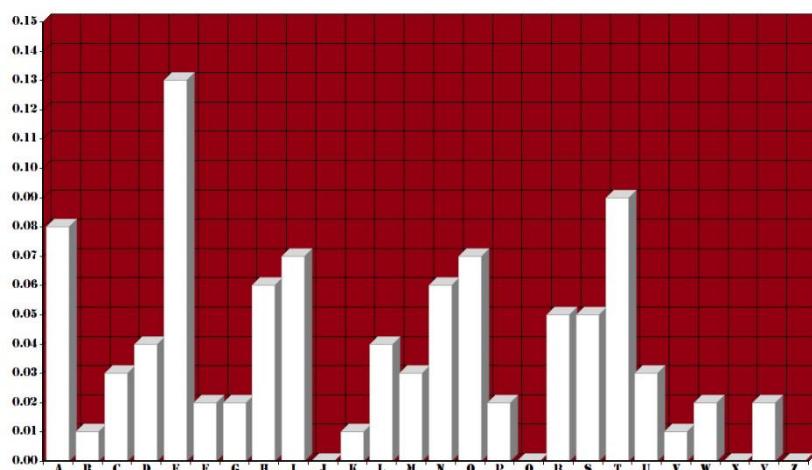
Osnovni tekst: Does this mean that a simple substitution cipher is secure

Kodirani tekst: YVJS UKFS NJZW UKZU Z SFNDQJ STPSUFUTUFVW BFDKJM FS SJBTMJ

Prednost ovog sistema je što je u opticaju mnogo veći broj ključeva. Za azbuku od 26 slova shift-by-N ima 26 mogućih ključeva dok kod ovog metoda kriptovanja, ključ je bilo koja od permutacija, s obzirom da ih ima $26!$ što je približno 288.

Današnji kompjuteri mogu da obrade 240 dekodiranja u sekundi, što znači da bi jednom takvom računaru trebalo 4450 milenijuma da probaju sve kombinacije ključa. Zbog toga se dekodiranje vrši jednim malo drugačijim pristupom.

Dekodiranje jednostavne zamene se bazira na učestalosti pojavljivanja nekog slova u rečima nekog jezika, kao i na učestalosti pojavljivanja neke reči u nekom jeziku. Sledeća slika prikazuje učestalost slova u engleskom jeziku.



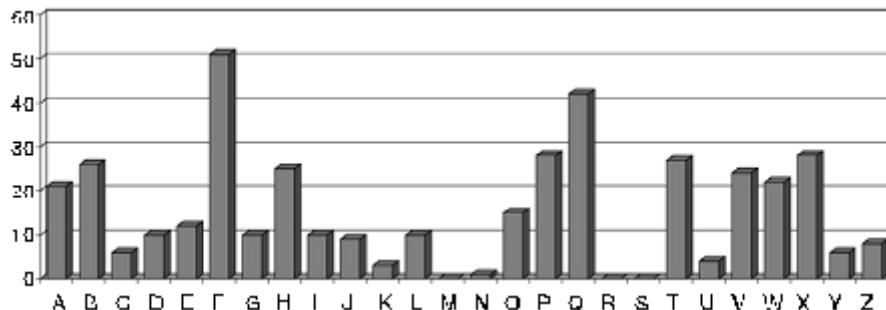
Slika 3.1 Učestalost pojavljivanja slova u rečima engleskog jezika

3.2.1 Proces dekodiranja

Uzmimo najpre kriptovan tekst:

PBFPVYFBQXZTYFPBFEQJHDXXQVAPTPQJKTOYQWIPBVWLXTOXBTFXQWA
XBVCXQWAXFQJVWLEQNTOZQGGQLFXQWAKVWLXQWAEBIPBFXFQVXGTV
JVWLBTTPQWAEBFPBFHCVLXBQUFEVWLXGDPEQVPQGVPPBFTIXPFHXZHVF
AGFOTHFEFBQUFTDHZBQPOTHXTYFTODXQHFTDPTOGHFQPBQWAQJJTODX
QHFOQPWTBDHHIXQVAPBFZQHCFWPFBFIPBQWKFABVYYDZBOTHPBQPQ
JTQOTOGHFQAPBFEQJHDXXQVAVXEBQPEFZBVFOJIWFFACFCCFHQWAUVW
FLQHGFXVAFXQHFUFHILTTAVWAFFAWTEVOITDHFFHQAITIXPFHXAFQHEFZ
QWGFLVWPTOFFA

I uradimo statistiku pojavljivanja slova u njemu:



Na osnovu toga možemo da prepostavimo inicijalnu permutaciju pomoću koje bi probali dekodovanje. I evo kako dekodiranje prethodnog teksta može da izgleda:

Tekst je na engleskom, u engleskom se najčešće pojavljuje e, a u našem tekstu F, tako da može da se prepostavi da je e kodovano sa F. Kombinacija slova PBF se najčešće javlja u tekstu, a najčešća troslovna reč je član the. Ako zamenimo ova tri slova u kriptovanom tekstu(p sa t, b sa h, f sa e) a nedekovana označimo nekim specijalnim znakom i probamo dekodovanje dobićemo:

```
thet##eh#####ethe#####t#t#####th#####h#e#####h#####e#####e#####e#####
#####e#####h#####h#the#e#####h#t#####hethe#####h#####e#####t###t###tt
he###te#####e##e##e#eh##e#####h#t#####e#####t###e#th#####e##e##t#
#####the#####e#te#the#th##e#h#####h##th#t#####e##the#####h##e#h#####
h##e#h#e####ee##e##e#####e####e##e##e#e#####e#e#####e##e#e#####
#te##e##e##e##e##t##ee###
```

Ako uvedemo dalju smenu po učestalosti dobiće zamene Q sa a, X sa s, V sa i, T sa o, H sa r. Sada deo dekodiranog teksta izgleda ovako:

```
theti#ehas#o#ethe#a#r#ssai#tota##o##a##thi##so#shoesa##shi#sa##sea##i##a#o##a##a
#esa##i##sa##h#theseais#oi#i##hota##hether#i#sha##e##i##s##t#aita#ittheo#sters#rie
##e##ore#eha#eo#r#hat#orso#eo##sareo##to##reatha##a##o##sare#at#oh##rr#sai#the#ar#e
#terthe#tha##e#hi#####h#orthata#oa#o##rea#the#a##r#ssai#is#hat#e#hie##ee##e##era##
##i##e##ar#esi#esare#er##oo##i##ee##o##rereaa##o#sters#ear#e#a##e##i#to##ee##.
```

Pogledajmo sada početak teksta `theti#ehas#o#ethe`. Prva reč je the, treća has. Druga je tiYe a četvrta ZoYe. Ako prepostavimo da su prve četiri reči „the time has come“ i uvedemo adekvatne smene (Z sa c, a Y sa m) dobićemo:

```
The time has come the#a##r#ssai#tota##o##ma##thi##so#shoesa##shi#sa##sea##i##a#o#
ca##a#esa##i##sa##h#theseais#oi#i##hota##hether#i#sha##e##i##s##t#aita#ittheo#ster
scrie##e##ore#eha#eo#rchat#orsomeo##sareo##to##reatha##a##o##sare#at#oh##rr#sai#the
car##e#terthe#tha##e#himm#c##h#orthata#oa#o##rea#the#a##r#ssai#is#hat#echie##ee##e#
##era##i##e##ar#esi#esare#er##oo##i##ee##o##rereaa##o#sters#ear#eca##e##i#to##ee##.
```

Nadalje, kriptoanaliza ovakvih tekstova se svodi na uočavanje pojedinih reči u ostatku teksta i menjanjem slova. Na kraju se dobije željena permutacija i dekodira se tekst. Ovde je bitno uočiti da se kriptovani tekstovi najčešće šalju bez interpunkcijskih znaka i razmaka kako bi se otežalo dekriptovanje.

3.3 Jednokratni ključ (one-time-pad)

U kriptografiji one-time pad, u daljem tekstu OTP, predstavlja algoritam za kriptovanje podataka koji funkcioniše tako što kombinuje izvorni tekst sa ključem, tj. one-time pad-om. OTP je trenutno jedini poznat bezuslovno siguran sistem šifriranja. Ostali sistemi šifriranja su kriptografski sigurni, što znači da su skupi za razbijanje, ali u teoriji je moguće da se razbiju ukoliko se obezbedi dovoljno vremena za obradu. Pod „pad“-om se podrazumeva tablica koja sadrži jednoznačna preslikavanja za sva slova azbuke jezika koji koristimo, u njihove kodove. OTP zahteva ključ koji je iste dužine kao i poruka koja se šalje.

OTP je dokazivo bezuslovno siguran, što znači da se ne može razbiti, bez obzira koliko vremena za analizu obezbedili. Matematički je nemoguće da se razbije OTP. Iako je OTP čudesno moćan algoritam za kriptovanje, on se ne koristi toliko često zato što se „pad“-ovi moraju predati svakoj od osoba sa kojom se komunicira. „Pad“ (ključ) treba da bude „random“ u pravom smislu te reči. Sistem se zasniva na totalnoj nasumičnosti ključa i na tome da je on poznat samo osobi koja šalje poruku i osobi kojoj je poruka namenjena.

Ovde se ispoljavaju mane ovog sistema šifriranja, jer se ključ mora na bezbedan način predati primaocu poruke, što će reći da se on predaje lično ili preko poverljive osobe.

XOR (\oplus):	
$0 \oplus 0 = 0$	$1 \oplus 0 = 1$
$0 \oplus 1 = 1$	$1 \oplus 1 = 0$
$a \oplus a = 0$	$a \oplus 0 = a$
$a \oplus b \oplus b = a$	

Slika 3.2 Osobine funkcije eksluzivno ili

U osnovi, uzima se nasumični „pad“, koji imamo mi i priamaoc poruke. Šifrirani tekst se dobija tako što se početni tekst XOR-uje sa ključem (P - početni tekst, K- ključ, C – šifrirani tekst)

$$P \oplus K = C$$

Kada primaoc primi šifrirani tekst, on ga dekriptuje sa ključem, tako što XOR-uje šifrirani tekst sa ključem

$$P = C \oplus K$$

Isti ključ („pad“) se ne sme nikad upotrebiti ponovo, jer se tada algoritam ne bi zvao „one-time“, a i algoritam će izgubiti na polju njegovog razbijanja. U slučaju ponavljanja imamo

$$\begin{aligned} C_1 &= P_1 \oplus K \text{ i } C_2 = P_2 \oplus K \\ C_1 \oplus C_2 &= P_1 \oplus K \oplus P_2 \oplus K = P_1 \oplus P_2 \end{aligned}$$

I upravo to je razlog zbog čega su neophodne čisto slučajne sekvence za ključ.

Nesalomivost OTP algoritma leži u tome što se ne može razbiti ni „brute force“ pristupom, jer šifrovani tekst sam za sebe jedino nosi informaciju o svojoj dužini i ništa više. Čak i da se primenom „brute force“ pristupa dobije tekst koji ima neki smisao ne možemo biti sigurni da je baš to pravi ključ („pad“), jer uvek postoji još neki ključ koji će kada se primeni na dati šifrirani tekst, opet dati neki drugi tekst koji ima smisla. Generisanje istinski nasumičnih „pad“-ova predstavlja veliki izazov. Na primer, korišćenje random() funkcije C programskega jezika nije ni blizu dobrog rešenja, jer ova funkcija radi sa 32-bitnim vrednostima, pa se primenom „brute force“ pristupa može za veoma kratko vreme razbiti šifrirani tekst.

3.3.1 Primer

Azbuka od 8 slova i njihovi kodovi da ti su u sledećoj tabeli.

e	000
h	001
i	010
k	011
l	100
r	101
s	110
t	111

Hoćemo da pošaljemo sledeći tekst:

heil hitler											
Originalni tekst	h	e	i	l	h	i	t	l	e	r	
Kod	001	000	010	100	001	010	111	100	000	101	
Pad	111	101	110	101	111	100	000	101	110	000	
Rezultat kodiranja	110	101	100	001	110	110	111	001	110	101	
Kodirani tekst	s	r	l	h	s	s	t	h	s	r	

Z A Š T I T A I N F O R M A C I J A

Dešifrovanje:

Šifrirani tekst	s	r	l	h	s	s	t	h	s	r
Kod	110	101	100	001	110	110	111	001	110	101
Pad	111	101	110	101	111	100	000	101	110	000
Rezultat dekodiranja	001	000	010	100	001	010	111	100	000	101

Dekodirani tekst h e i l h i t l e r

Ukoliko bi smo na isti šifrirani tekst primenili drugi ključ dobili bi smo drugi otvoreni tekst, koji bi imao smisla, ali bi bio u potpunoj suprotnosti sa originalnim otvorenim tekstrom, što pokazuje da primenom „brute force“ pristupa ne bi smo bili sigurni koji je tekst zapravo originalni. Za „ključ“: e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111 imamo sledeće,,dekodiranje“.

Šifrirani tekst	s	r	l	h	s	s	t	h	s	r
Kod	110	101	100	001	110	110	111	001	110	101
Pad	101	111	000	101	111	100	000	101	110	000
Rezultat dekodiranja	011	010	100	100	001	010	111	100	000	101

Dekodirani tekst k i l l h i t l e r

Ukratko, o ovom algoritmu se može reći da je dokazano siguran, ako se koristi ispravno. Takođe, šifra ne daje nikakvu informaciju o kodiranom tekstu i svi kodirani tekstovi su jednako verovatni. Ključ mora biti slučajan, i koristiti se samo jednokratno. Ključ je poznat samo pošiljaocu i primaocu i iste je dužine kao i otvoreni tekst.

3.4 Dvostruka transpozicija (Double Transposition Cipher)

Ideja za ovo kriptovanje je napisati slova teksta kao elemente matrice, a zatim permutovati vrste i kolone, i na kraju prevesti elemente metrice u niz i tako dobiti kriptovani tekst. U ovom slučaju ključ predstavljaju permutacije vrsta i kolona. Tekst za kriptovanje: Attack at dawn!

Prvo izbaciti sve znake interpunkcije, beline i velika slova zameniti malim. Dakle tekst za kriptovanje je attackatdawn. Ovaj tekst ima 12 slova i prebacimo ga u matricu 3×4 .

$$\begin{bmatrix} a & t & t & a \\ c & k & a & t \\ d & a & w & n \end{bmatrix}$$

Sada ćemo definisati ključ kao (3,2,1) za vrste i (4,2,1,3) za kolone. U slučaju ovog kriptoalgoritma, ako se ne kaže drugačije, prvo se vrši permutacija vrsta, a onda kolona:

$$\begin{bmatrix} a & t & t & a \\ c & k & a & t \\ d & a & w & n \end{bmatrix} \rightarrow \begin{bmatrix} d & a & w & n \\ c & k & a & t \\ a & t & t & a \end{bmatrix} \rightarrow \begin{bmatrix} n & a & d & w \\ t & k & c & a \\ a & t & a & t \end{bmatrix}$$

Dobijena kodirana reč je onda: NADWTKCAATAT

Dekodiranje se vrši u suprotnom smeru uz uslov da onaj ko dekoduje zna ključ, odnosno permutacije vrsta i kolona. Suprotno od kodiranja, ovde se najpre dekodiraju vrste, a onda kolone.

$$\begin{bmatrix} N & A & D & W \\ T & K & C & A \\ A & T & A & T \end{bmatrix} \rightarrow \begin{bmatrix} D & A & W & N \\ C & K & A & T \\ A & T & T & A \end{bmatrix} \rightarrow \begin{bmatrix} A & T & T & A \\ C & K & A & T \\ D & A & W & N \end{bmatrix}$$

Najjednostavniji postupak je označiti kolone i vrste indeksima kao u permutaciji (u našem slučaju kolone 4,2,1,3 a kolone 3,2,1) ih poređati kao 1,2,3,4 kolone, a 1,2,3 vrste.

Pročitani rezultat je attackatdawn.

3.5 Kodiranje po rečniku (Codebook cipher)

Kodiranje po rečniku je i danas prilično korišćen način kriptovanja. Ovde je ključ u rečniku gde se svaka reč nekog jezika menja nekim kodom od odgovarajućeg broja znakova.

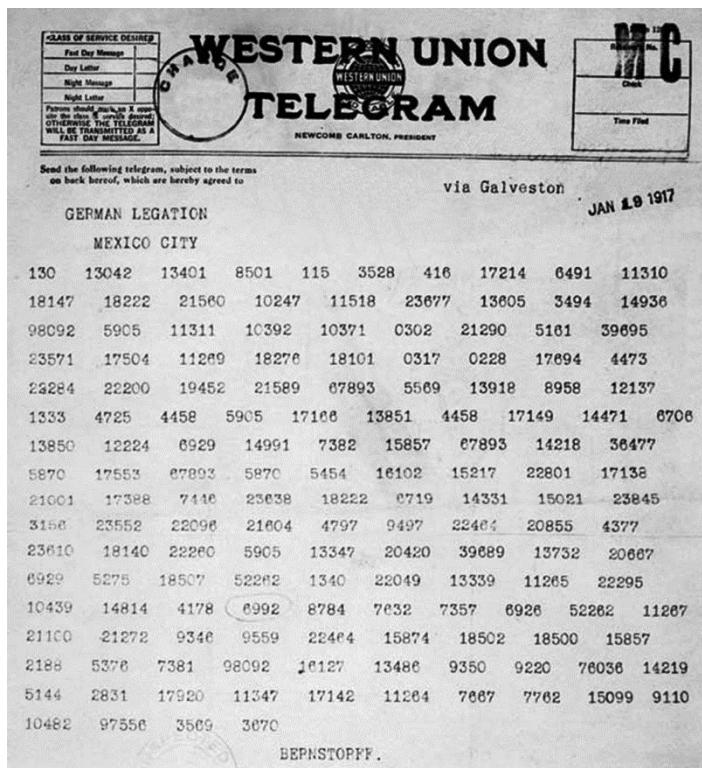
ZAŠTITA INFORMACIJA

Primer dela rečnika koji je Nemačka koristila pre i tokom Prvog svetskog rata:

Plaintext	Ciphertext
Februar	13605
fest	13732
finanzielle	13850
folgender	13918
Frieden	17142
Friedenschluss	17149
:	:

Slika 1.3 Deo rečnika koja je Nemačka koristila pre i za vreme Prvog svetskog rata

Primer poruke šifrovane tim rečnikom (čuveni Cimermanov telegram iz 1917 kojom se nemačkim podmornicama naređuje tzv. neograničeni napad na sve savezničke brodove)



Slika 3.4 Čuveni "Cimermanov telegram" kofiran rečnikom

Najefikasniji način za dekodiranje ovakvih poruka je krađa rečnika. Ovaj telegram je dešifrovan nakon što je ruski špijun mikrofilmovao delove nemačkog kodnog rečnika koji se nalazio u ambasadi u Meksiku sitiju.

3.5.1 Elections 1873

Zanimljiva modifikacija kodiranja rečnikom je tzv. kodiranje za izbore u SAD iz 1876 godine (Elections 1876 cipher). Ovaj način kodiranja se zasniva na kodiranju na bazi rečnika i delimično na dvostrukoj transpoziciji. Ideja je sledeća:

1. Napraviti rečnik koji će pokrivati ograničen broj reči, a ne ceo rečnik jednog jezika
2. Podeliti poruku na grupe od po određenog broja reči (10, 20, 30 i sl) ignorujući znake interpunkcije
3. U svakoj grupi se primeni ista permutacija reči

Dekripcija je (treba znati permutaciju) podeliti tekst na grupe reči:

1. Za svaku grupu odrediti inverznu permutaciju
2. zamjeniti specifične reči korišćenjem rečnika
3. sklopiti tekst

Plaintext	Ciphertext
Greenbacks	Copenhagen
Hayes	Greece
votes	Rochester
Tilden	Russia
telegram	Warsaw
:	:

Slika 3.5 Deo rečnika korišćenog za izbore 1876.

Originalni primer primene ovoga je sledeća poruka:

„Can't read last telegram. Situation unchanged. They are all idiots.“

Kodirana poruka korišćenjem permutacije od po 5 reči (4,1,5,2,3):

1. izbacimo interpunkciju i velika slova: can't read last telegram situation unchanged they are all idiots
2. podelimo reči na dve grupe od po 5:

4	1	5	2	3
can't read last telegram situation				
4	1	5	2	3
unchanged they are all idiots				

Z A Š T I T A I N F O R M A C I J A

3. zamenimo reči iz rečnika (ako postoje)
can't read last Warsaw situation
unchanged they are all idiots
4. Izvršimo permutacije
read Warsaw situation can't last
they all idiots unchanged are
5. Sklopimo dobijene grupe u jednu rečenicu:
read Warsaw situation can't last they all idiots unchanged are

Dekodiranje (kad znamo permutaciju):

1. Uzmimo kodirani tekst:
read Warsaw situation can't last they all idiots unchanged are
2. Podelimo ga na dve grupe po 5 i označimo mesta rečima:
1 2 3 4 5
read Warsaw situation can't last
1 2 3 4 5
they all idiots unchanged are
3. I poređajmo reči po permutaciji:
4 1 5 2 3
can't read last Warsaw situation
4 1 5 2 3
unchanged they are all idiots
4. Zamenimo reči koje nađemo u rečniku, i sklopimo ih u rečenicu
can't read last telegram situation unchanged they are all idiots

4 SIMETRIČNI KRIPTO – KODERI TOKA

U kriptografiji sa simetričnim ključem razlikujemo kodove toka podataka (stream ciphers) i kodove bloka podataka (block ciphers). A5/1 algoritam je pored RC4 algoritma jedan od predstavnika za dobijanje kodova toka podataka.

Kodovi toka podataka su kao one-time pad, s tim što se kod A5/1 algoritma koristi ključ koji se „izdužuje“ u dugačak niz bitova, koji se onda koristi kao kod one-time pad-a.

A5/1 i RC4 algoritmi se oba danas široko koriste, s tim što se A5/1 koristi kod GSM telefona i predstavnik je velike klase kodova toka podataka baziranih na hardveru. Služi da obezbedi privatnost i pouzdanost u bežičnoj komunikaciji. Dugo je držan u privatnosti, ali je javno postao poznat preko pukotina i reverse engineering-a (proces otkrivanja tehnoloških principa uređaja, objekta i sistema kroz analizu njegove strukture, funkcije i operacije).

4.1 Kodovi toka podataka (stream ciphers)

Kod toka podataka uzimaju ključ dužine n bita i „izdužuju“ ga u niz bitova koji služe kao ključ (keystream). Nad nizom bitova koji služe kao ključ (nad keystream-om) i osnovnim tekstrom P se onda uradi XOR operacija (ekskluzivno ili -), čime se dobije kodirani tekst C. Korišćenje keystream-a je identično korišćenju ključa u kodu one-time pad. Za dekriptovanje koda toka podataka, nad istim keystream-om i kodiranim tekstrom se odradi XOR operacija.

Funkcija koda toka podataka može biti predstavljena lako kao:

$$\text{Streamcipher}(K) = S$$

gde je K ključ i S *keystream* koji koristimo kao u one-time pad-u. Formula za enkriptovanje glasi:

$$c_0 = p_0 \oplus s_0, c_1 = p_1 \oplus s_1, c_2 = p_2 \oplus s_2, \dots$$

gde je $P = p_0 p_1 p_2 \dots$ osnovni tekst, $S = s_0 s_1 s_2 \dots$ *keystream* (niz bitova koji se koriste kao ključ) i $C = c_0 c_1 c_2 \dots$ kodirani tekst. Za dekripciju kodiranog teksta C, *keystream* se ponovo koristi:

$$p_0 = c_0 \oplus s_0, p_1 = c_1 \oplus s_1, p_2 = c_2 \oplus s_2, \dots$$

Osiguravajući i pošiljaocu i primaocu isti algoritam koda toka podataka i isti ključ, sistem je praktična generalizacija one-time pad-a.

4.2 A5/1 Algoritam

A5/1 algoritam ima algebarsku strukturu, ali takođe može biti i ilustrovan veoma jednostavnom slikom. Daćemo ovde oba opisa.

A5/1 koristi tri linearna pomeračka povratna registra (LFSRs - linear feedback *shift registers*), koje ćemo označiti sa X, Y i Z. Registar X drži 19 bita, koje ćemo označiti $(x_0, x_1, \dots, x_{18})$. Registar Y drži 22 bita $(y_0, y_1, \dots, y_{21})$ i registar Z drži 23 bita $(z_0, z_1, \dots, z_{22})$. Nije slučajno što tri LFRSs-a sadrže ukupno 64 bita.

Bez ikakve koincidencije, ključ K sadrži 64 bita. Ključ se koristi kao inicijalna popuna tri linearna pomeračka povratna registra. Pošto se registri popune ključem K, spremni smo da generišemo *keystream*. Samo pre nego što objasnimo definisanje *keystream*-a, prodiskutovaćemo registre X, Y i Z detaljnije.

Kada X pokrene pomeraje, računa se sledeće:

$$t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$$

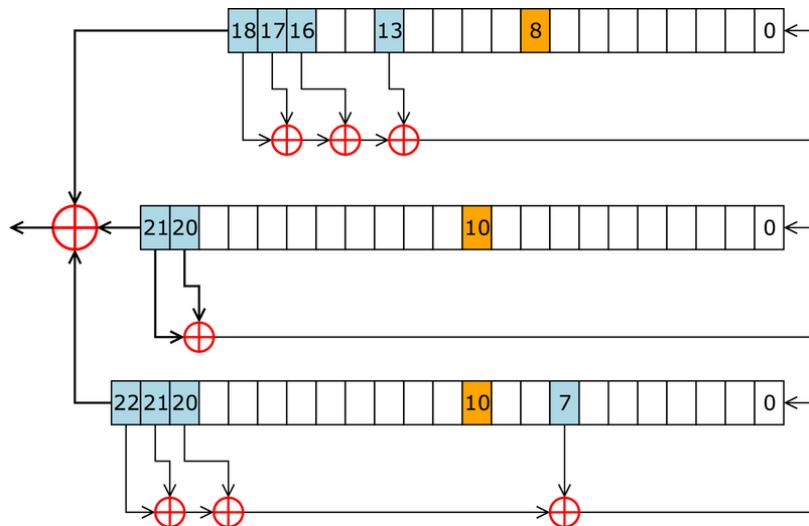
$$x_i = x_{i-1} \text{ za } i = 18, 17, 16, \dots, 1$$

$$x_0 = t$$

Slično se može napisati i za registre Y i Z. Kao što je pomenuto, inicijalni 64-bitni ključ K se koristi kao inicijalna popuna tri linearna pomeračka registra. Pošto se registri popune ključem K, spremni su da generišu *keystream*. **Error! Reference source not found.** 4.1 prikazuje strukturu koja služi za generisanje bitova ključa. Jasno je da su svakom registru definisani bitovi koji učestvuju u generisanju takozvanog t bita koji učestvuje u pomeranju registara, i on postaje multi bit odgovarajućeg registra. Kao što je pokazano vrednost bita t vezanog za registar X se računa kao $t_x = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$. Za registre Y i Z u računanju t bita, na isti način, učestvuju bitovi 20 i 21 (za Y), odnosno 7, 20, 21 i 22 (za Z).

Takođe, na svakom registru se definiše i bit koji učestvuje u većinskom glasanju. To je bit 8 na registru X, i bitovi označeni sa 10 nad registrima Y i Z. Funkcija većinskog glasanja (*majority vote*) vraća nulu ako su među vrednostima x, y i z bar dve nule, odnosno, vraća jedinicu ako su među parametrima funkcije ber dva imaju vrednost 1.

Dakle, data tri bita x, y i z, definišu $maj(x,y,z)$ da bude funkcija „majority vote (višak glasanja)“. Ako je višak od x,y i z 0, onda funkcija vraća 0, u suprotnom vraća 1.



Slika 4.1 Ilustracija rada LFRS registra kod A5/1 algoritma

A5/1 je hardverski implementiran i u svakom vremenskom pulsu vrednost:

$$m = \text{maj}(x_8, y_{10}, z_{10})$$

je izračunata. Onda se registri X, Y i Z pomeraju prema sledećim pravilima:

Ako je $x_8 = m$ onda se X pomera

Ako je $y_{10} = m$ onda se Y pomera

Ako je $z_{10} = m$ onda se Z pomera.

Napokon, keystream S je generisan kao:

$$s = x_{18} \oplus y_{21} \oplus z_{22}$$

Kome se pridružuje XOR operacija sa osnovnim tekstrom P ako se radi o enkripciji ili sa kodiranim tekstrom C pri dekripciji.

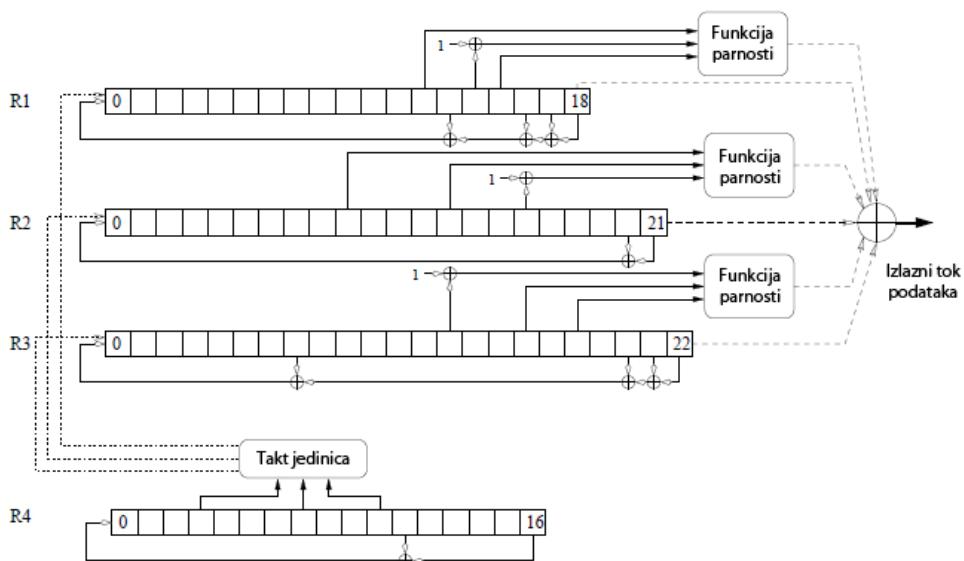
Ovo može ličiti na komplikovani način za generisanje jednog keystream bita, ali A5/1 ima laku implementaciju kroz hardver i može generisati bitove velikim brzinama koje odgovaraju bilo kom audio signalu u realnom vremenu. Takođe, broj *keystream* bitova koji se mogu generisati kroz jedan 64-ovobitni ključ je virtualno neograničen – pri čemu se eventualno keystream može ponoviti.

A5/1 algoritam je tipičan predstavnik velike klase tokova podataka koji je baziran na pomeračkim registrima i implementiran u hardveru. Ovi sistemi su bili „kraljevi“ kriptografije sa simetričnim ključem jedno vreme, ali poslednjih godina kod bloka podataka je preuzeo tu titulu.

Istorijski, pomerački registri su osnova kodova toka podataka gde je potrebno da se održi brzina sa bitovima toka podataka (kao što je u audio signalima) koji se generišu velikom brzinom. U prošlosti, softverski bazirana kriptografija nije mogla da generiše bitove tako brzo za takve aplikacije. Danas, međutim postoji više aplikacija u kojoj nam softverski bazirana kriptografija nije adekvatna.

4.3 A5/2 Algoritam

Koder toka podataka koji koristi **A5/2** algoritam prihvata 64-bitni ključ K_c i 22-bitni javni ključ, takođe poznat kao brojač koji se dobija iz javnosti poznatog broja okvira. U daljem tekstu ćemo ovaj javni ključ obeležavati sa f . Sam A5/2 se sastoji od četiri linearna pomeračka registra sa povratnom informacijom maksimalne dužine: **R1** dužine 19-bitova, **R2** dužine 22-bitova, **R3** dužine 23-bitova i **R4** dužine 17-bitova (kao što je prikazano na slici 1). Pre nego što dođe do taktovanja registra izračunava se novi nulti bit (kao XOR funkcija unapred određenih bitova registra R4).



Slika 4.2 LFRS registri za A5/2

Zatim se registar taktuje sa pomerajem za jedan bit u desno, pri čemu se odbacuje bit najveće težine, a prethodno izračunati novi bit se upisuje na mesto bita najmanje težine, tj. na lokaciju nultog bita.

A5/2 se inicijalizuje u četiri koraka pomoću K_c i f , gde se i -ti bit K_c obeležava sa $K_c[i]$, i -ti bit od f sa $f[i]$, a $i=0$ je najmanje bitan bit. Stanje koje nastaje nakon podešavanja ključa označićemo sa $(R1, R2, R3, R4) = \text{keysetup } (K_c, f)$. Ova inicijalizacija se naziva generisanje ključa. Mora se primetiti da je generisanje ključa linearno i za K_c i za f (bez bitova R1[15], R2[16], R3[18], i R4[10] koji su uvek setovani na 1).

1. Set $R1 = R2 = R3 = R4 = 0$.
2. For $i = 0$ to 63
 - Clock all four registers.
 - $R1[0] \leftarrow R1[0] \oplus K_c[i]$; $R2[0] \leftarrow R2[0] \oplus K_c[i]$; $R3[0] \leftarrow R3[0] \oplus K_c[i]$;
 - $R4[0] \leftarrow R4[0] \oplus K_c[i]$.
3. For $i = 0$ to 21
 - Clock all four registers.
 - $R1[0] \leftarrow R1[0] \oplus f[i]$; $R2[0] \leftarrow R2[0] \oplus f[i]$; $R3[0] \leftarrow R3[0] \oplus f[i]$;
 - $R4[0] \leftarrow R4[0] \oplus f[i]$.
4. Set the bits $R1[15] \leftarrow 1$, $R2[16] \leftarrow 1$, $R3[18] \leftarrow 1$, $R4[10] \leftarrow 1$.

Slika 4.3 Generisanje ključa za A5/2

A5/2 radi u ciklusima, gde se na kraju svakog ciklusa dobija po jedan izlazni bit. Tokom svakog ciklusa se taktuju dva ili tri od registra od registara R1, R2, i R3, u zavisnosti od 3 određena bita R4 registra. Nakon toga se taktuje i sam register R4. Na početku svakog ciklusa, tri bita R4[3], R4[7], i R4[10] ulaze u taktnu jedinicu. Ova jedinica sprovodi funkciju parnosti nad tri ulazna bita (majority function). Zatim na red dolazi taktovanje registara R1, R2 i R3 i to po sledećim pravilima:

1. R1 se taktuje akko se bit R4[10] slaže sa rezultatom f-je parnosti;
2. R2 se taktuje akko se bit R4[3]slaže sa rezultatom f-je parnosti;
3. R3 se taktuje akko se bit R4[7]slaže sa rezultatom f-je parnosti.

Nakon ovih taktovanja dolazi do taktovanja R4 registra, a izlazni bit se generiše it vrednosti R1,R2 i R3 registara izvršenjem XOR funkcije nad njihovim bitovima najveće težine i vrednostima funkcije parnosti (detaljni postupak je prikazan na slici 1). Vrednosti koji ulaze u funkciju parnosti nalaze se:

1. za R1 u bitovima R1[12], R1[14] i R1[15];
2. za R2 u bitovima R2[9], R2[13] i R2[16];
3. za R3 u bitovima R3[13], R3[16] i R3[18].

Važno je naglasiti da je funkcija parnosti (korišćena za generisanje izlaza) drugog stepena na svom ulaz i oblika je:

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

Na ovaj način, izlazni bit je kvadratna funkcija bitova iz registara R1, R2 i R3. Prvih 99 bitova se odbacuju (mada se po nekim kod A5/2 algoritma odbacuje prvih 100 bitova na izlazu, i izlaz je uzet sa kašnjenjem od jednog bita, što je ekvivalentno tvrđenju da ovaj algoritam odbacuje prvih 99 i da je izlaz u upotrebi bez kašnjenja), a narednih 228 bitova su uzeti kao izlazni tok ključa. Generisanje ovog toka se može ukratko objasniti na sledeći način:

1. Pokrene se generisanje ključa pomoću vrednosti K_c i f (slika 2);
2. A5/2 algoritam odradi 99 ciklusa pri čemu se izlaz odbacuje;
3. A5/2 odradi dodatnih 228 ciklusa i izlaz se upotrebni kao tok ključa.

Izlaz od 228 bitova (koji zovemo tok ključa) se deli na dva jednakata dela. Prvih 114 bitova se korite kao tok ključa koji se koristi za enkripciju veze od mreže do mobilnog uređaja dok se drugih 114 bitova koriste za enkripciju veze od mobilnog uređaja do mreže. Enkripcija se izvodi kao XOR funkcija kod koje se uporedjuju bitovi poruke i toka ključa koji imaju isti indeks.

A5/2 je baziran na A5/1 arhitekturi i predstavlja njegovu oslabljenu verziju. Funkcije generisanja novih bitova za R1,R2 i R3 registre su iste kao i kod A5/1 algoritma. Proces inicijalizacije je takođe prilično sličan procesu koji sprovodi A5/1, sa jedinom razlikom da A5/2 takođe inicijalizuje i R4 registar, i da vrednost bar jedog bita u svakom registru mora biti jednak 1 nakon inicijalizacije, dok A5/1 ne koristi R4, i nema prinudnih vrednosti za bitove. Takođe, A5/2 odbacuje prvih 99 izlaznih bitova dok A5/1 odbacuje 100. Mechanizam zadužen za taktovanje je isti, s tim što ulazni bitovi za ovaj mehanizam kod A5/2 algoritma potiču iz R4 regista, dok su u A5/1 algoritmu za njih zaduženi R1, R2 i R3.

Listing 4.1 A5/2 algoritam

```
#include <stdio.h>
/* Masks for the shift registers */
#define R1MASK 0x07FFFF /* 19 bits, numbered 0..18 */
#define R2MASK 0x3FFFFFF /* 22 bits, numbered 0..21 */
#define R3MASK 0x7FFFFFF /* 23 bits, numbered 0..22 */
#define R4MASK 0x01FFFF /* 17 bits, numbered 0..16 */
/* A bit of R4 that controls each of the shift registers */
#define R4TAP1 0x000400 /* bit 10 */
#define R4TAP2 0x000008 /* bit 3 */
#define R4TAP3 0x000080 /* bit 7 */
/* Feedback taps, for clocking the shift registers. These
correspond to the primitive polynomials
x^19 + x^5 + x^2 + x + 1, x^22 + x + 1, x^23 + x^15 + x^2 + x + 1
and x^17 + x^5 + 1. */
#define R1TAPS 0x072000 /* bits 18,17,16,13 */
#define R2TAPS 0x300000 /* bits 21,20 */
#define R3TAPS 0x700080 /* bits 22,21,20,7 */
#define R4TAPS 0x010800 /* bits 16,11 */
typedef unsigned char byte;
typedef unsigned long word;
typedef word bit;
/* Calculate the parity of a 32-bit word, i.e. the sum of its bits
modulo 2 */
bit parity(word x) {
    x ^= x>>16;
    x ^= x>>8;
    x ^= x>>4;
    x ^= x>>2;
    x ^= x>>1;
    return x&1;
}
/* Clock one shift register. For A5/2, when the last bit of the
frame is loaded in, one particular bit of
each register is forced to '1'; that bit is passed in as the last
argument. */
word clockone(word reg, word mask, word taps, word loaded_bit) {
    word t = reg & taps;
    reg = (reg << 1) & mask;
```

```

        reg |= parity(t);
        reg |= loaded_bit;
        return reg;
    }

/* The three shift registers. They're in global variables to make
the code easier to understand. A better implementation would not
use global variables. */
word R1, R2, R3, R4;
/* Return 1 iff at least two of the parameter words are non-zero.
*/
bit majority(word w1, word w2, word w3) {
    int sum = (w1 != 0) + (w2 != 0) + (w3 != 0);
    if (sum >= 2)
        return 1;
    else
        return 0;
}
/* Clock two or three of R1,R2,R3, with clock control according to
their middle bits. Specifically, we clock Ri whenever particular
bit of R4 agrees with the majority value of Ri's. Also always
clock R4. If allP == 1, clock all three of R1,R2,R3. This is
only used for key setup. If loaded == 1, then this is the last
bit of the frame number, and if we're doing A5/2, we have to set a
particular bit in each of the four registers. */
void clock(int allP, int loaded) {
    bit maj = majority(R4&R4TAP1, R4&R4TAP2, R4&R4TAP3);
    if (allP || ((R4&R4TAP1)!=0) == maj)
        R1 = clockone(R1, R1MASK, R1TAPS, loaded<<15);
    if (allP || ((R4&R4TAP2)!=0) == maj)
        R2 = clockone(R2, R2MASK, R2TAPS, loaded<<16);
    if (allP || ((R4&R4TAP3)!=0) == maj)
        R3 = clockone(R3, R3MASK, R3TAPS, loaded<<18);
    R4 = clockone(R4, R4MASK, R4TAPS, loaded<<10);
}

/* Generate an output bit from the current state. You grab a bit
from each register via the output generation taps; then you XOR the
resulting three bits. In addition to the top bit of each of R1,R2,R3,
also XOR in a majority function of three particular bits of the
register (one of them complemented) to make it non-linear. Also,
for A5/2, delay the output by one clock cycle for some reason. */
bit getbit() {
    bit topbits = (((R1 >> 18) ^ (R2 >> 21) ^ (R3 >> 22)) &
0x01);
    static bit delaybit = 0;
    bit nowbit = delaybit;
    delaybit = (
        topbits
        ^ majority(R1&0x8000, (~R1)&0x4000, R1&0x1000)
        ^ majority((~R2)&0x10000, R2&0x2000, R2&0x200)
        ^ majority(R3&0x40000, R3&0x10000, (~R3)&0x2000)
    );
    return nowbit;
}
/* Do the A5 key setup. This routine accepts a 64-bit key and a
22-bit frame number. */

```

```

void keysetup(byte key[8], word frame) {
    int i;
    bit keybit, framebit;
    /* Zero out the shift registers. */
    R1 = R2 = R3 = R4 = 0;
    /* Load the key into the shift registers, LSB of first byte
    of key array first,
     * clocking each register once for every key bit loaded.
    (The usual clock
     * control rule is temporarily disabled.) */
    for (i=0; i<64; i++) {
        clock(1,0); /* always clock */
        keybit = (key[i/8] >> (i&7)) & 1; /* The i-th bit
    of the key */
        R1 ^= keybit; R2 ^= keybit; R3 ^= keybit; R4 ^= keybit;
    }
    /* Load the frame number into the shift registers, LSB first,
    clocking each register once for every key bit loaded. (The usual
    clock control rule is still disabled.) Signal when the last bit
    is being clocked in. */
    for (i=0; i<22; i++) {
        clock(1,i==21); /* always clock */
        framebit = (frame >> i) & 1; /* The i-th bit of
    the frame # */
        R1 ^= framebit; R2 ^= framebit; R3 ^= framebit; R4 ^= framebit;
    }
    /* Run the shift registers for 100 clocks to mix the keying
    material and frame number together with output generation
    disabled, so that there is sufficient avalanche. We re-enable
    the majority-based clock control rule from now on. */
    for (i=0; i<100; i++) {
        clock(0,0);
    }
    /* For A5/2, we have to load the delayed output bit. This
    does _not_
     * change the state of the registers. For A5/1, this is a no-
    op. */
    getbit();
    /* Now the key is properly set up. */
}
/* Generate output. We generate 228 bits of keystream output. The
first 114 bits is for the A->B frame; the next 114 bits is for the
B->A frame. You allocate a 15-byte buffer for each direction, and
this function fills it in. */
void run(byte AtoBkeystream[], byte BtoAkeystream[]) {
    int i;
    /* Zero out the output buffers. */
    for (i=0; i<=113/8; i++)
        AtoBkeystream[i] = BtoAkeystream[i] = 0;
    /* Generate 114 bits of keystream for the A->B direction.
    Store it, MSB first. */
    for (i=0; i<114; i++) {
        clock(0,0);
    }
}

```

```

        AtoBkeystream[i/8] |= getbit() << (7-(i&7));
    }
/* Generate 114 bits of keystream for the B->A direction.
Store it, MSB first. */
    for (i=0; i<114; i++) {
        clock(0,0);
        BtoAkeystream[i/8] |= getbit() << (7-(i&7));
    }
}
/* Test the code by comparing it against a known-good test
vector. */
void test() {
    byte key[8] = {0x00, 0xfc, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff};
    word frame = 0x21;
    byte goodAtoB[15] = { 0xf4, 0x51, 0x2c, 0xac, 0x13, 0x59,
0x37, 0x64, 0x46, 0xb, 0x72, 0xd, 0xad, 0xd5, 0x00 };
    byte goodBtoA[15] = { 0x48, 0x00, 0xd4, 0x32, 0x8e, 0x16,
0xa1, 0xd, 0xcd, 0x7b, 0x97, 0x22, 0x26, 0x51, 0x00 };
    byte AtoB[15], BtoA[15];
    int i, failed=0;
    keysetup(key, frame);
    run(AtoB, BtoA);
/* Compare against the test vector. */
    for (i=0; i<15; i++)
        if (AtoB[i] != goodAtoB[i])
            failed = 1;
    for (i=0; i<15; i++)
        if (BtoA[i] != goodBtoA[i])
            failed = 1;
/* Print some debugging output. */
    printf("key: 0x");
    for (i=0; i<8; i++)
        printf("%02X", key[i]);
    printf("\n");
    printf("frame number: 0x%06X\n", (unsigned int)frame);
    printf("known good output:\n");
    printf(" A->B: 0x");
    for (i=0; i<15; i++)
        printf("%02X", goodAtoB[i]);
    printf(" B->A: 0x");
    for (i=0; i<15; i++)
        printf("%02X", goodBtoA[i]);
    printf("\n");
    printf("observed output:\n");
    printf(" A->B: 0x");
    for (i=0; i<15; i++)
        printf("%02X", AtoB[i]);
    printf(" B->A: 0x");
    for (i=0; i<15; i++)
        printf("%02X", BtoA[i]);
    printf("\n");

    if (!failed) {

```

```
        printf("Self-check succeeded: everything looks
ok.\n");
        exit(0);
    } else {
        /* Problems! The test vectors didn't compare*/
        printf("\nI don't know why this broke; contact the
authors.\n");
    }
}

int main(void) {

    test();

    return 0;
}
```

4.4 RC4

RC4 algoritam je razvio Ron Rivest 1987. godine, dok je radio za RSA (sigurnosni deo MC korporacije). Nije poznato da li skraćenica RC znači Ron's Code ili Rivest Cipher, ali je ime zaštićeno. Ovaj algoritam spada u grupu stream ciphera. Stream algoritam kriptira podatke bajt po bajt za razliku od block ciphera. Algoritam je čuvan u tajnosti, ali je 1994. godine kod anonimno objavljen na internetu. RC4 koristeći uneseni ključ generiše pseudoslučajni niz bitova (niz bitova ključa, eng. Keystream) koji se pomoću XOR operacije primenjuje na podatke.

Ključ može biti dug do 256 bajtova, ali najčešće dužine od 40 do 256 bitova. Upotreba ključa izvan US-a je ograničena na 40 bita. 40-bitni ključ ne predstavlja veliku sigurnost pa se sve češće upotrebljava 128 i 256-bitni ključ. RC4 je do 10 puta brži od DES-a i proglašen je veoma sigurnim algoritmom.

Ovaj algoritam se upotrebljava u mnogim komercijalnim software paketima kao što su: Lotus Notes, MS Windows, Oracle Secure SQL, Netscape Communications, Corporation SSL (Secure Socket Layer), Handshake Protocol i druge. Javlja se sve više u telekomunikacijama i mobilnim uređajima. RC4 je algoritam stanja i u svakom koraku njegovo stanje određeno je sa tri promenljive – niz S veličine 256 bajtova, index i i index j oba veličine po 1 bajt (8bita). Algoritam se izvodi u dva dela : KSA (Key-scheduling Algorithm) i PRGA (Pseudo-random Generation Algorithm).

4.4.1 RC4 – KSA Algoritam

KSA je skraćenica od (The key-Scheduling Algorithm) – algoritam za distribuciju ključa. Ovaj algoritam se koristi da inicijalizuje permutaciju u nizu S. "Keylength" se definiše kao broj bajtova u ključu i može biti u opsegu $1 \leq \text{keylength} \leq 256$, obično je izmedju 5 i 16 na odgovarajući ključ dužine od 40 do 128 bitova. Najpre, niz S je inicijalizovan kao identitet permutacije, a onda ovaj niz obrađuje 256 iteracija na sličan način kao glavni PRGA algoritam, ali takođe meša bitove ključa u isto vreme.

Listing 4.2 Kod za RC4-KSA algoritam

```

for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap(S[i],S[j])
endfor
    
```

4.4.2 PRGA algoritam (The pseudo – random generation algoritam)

PRGA algoritam menja stanje i izlaze bitova *keystream-a*, za potreban broj iteracija. U svakoj iteraciji, PRGA inkrementuje *i*, dodaje vrednost *S* ukazujući na *j* uz pomoć *i*, razmenjuje vrednosti *S* [*i*] I *S*[*j*], i onda je vrednost izlaza na lokaciji *S* jednaka *S* [*i*] + *S* [*j*], (po modulu 256).

Listing 4.3 "Lookup" faza RC4 algoritma

```

i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap(S[i],S[j])
    output S[(S[i] + S[j]) mod 256]
endwhile
    
```

Na slici 1 je prikazana lookup faza RC4. Izlazni bajt je izabran na osnovu vrednosti *S* (*i*) kao i *S* (*j*) tako što ih dodajemo zajedno po modulu 256, a zatim smestimo u sumu *S*. Izraz koji računa *S* (*S* (*i*) + *S* (*j*)) se koristi kao jedan bajt u nizu koji predstavlja ključ *K* (key stream).

4.4.3 Celokupni RC4

Celokupan algoritam radi na nivou bajtova. U prvoj fazi se inicijalizuje lookup tabela uz pomoć ključa. Označimo ključ kao *key*[*i*] for *i* = 0, 1, . . . , *N* - 1. Faza RC4 inicijalizacije se obavlja po sledećem algoritmu.

Listing 1.4 Faza inicijalizacije kod RC4 algoritma

```

for i = 0 to 255
    S[i] = i
    K[i] = key[i mod N]
next i
j = 0
for i = 0 to 255
    j = (j + S[i] + K[i]) mod 256
    swap(S[i],S[j])
next i
i = j = 0
    
```

U prikazanom algoritmu, svaki *key*[*i*] bajt, a *S*[*i*] lookup tabela gde je *S*[*i*] takođe bajt. Pseudo-kod za inicijalizaciju je prikazan gore pod naslovom „RC4 inicijalizacija”.

Interesantna stvar kod RC4 algoritma je ta da vrednost ključa može biti između 0 i 256 bajtova. Ključ se koristi samo za inicijalizaciju permutacije S.

Posle inicijalizacije, svaki keystream bajt je inicijalizovan na osnovu koda iz „RC4 inicijalizacije“. Izlaz se sastoji od jednog bajta koji može da se XOR-uje sa kodirajućim tekstom (ako se šifruje) ili sa kodiranim tekstom (ako se dešifruje).

RC4 algoritam je elegantan, jedostavan i elegantan u software-u. Međutim postoji napad koji je izvodljiv protiv upotrebe RC4[80,150,225], ali je ovaj napad nemoguć ako jednostavno odstranimo prvi 256 keystream bajtova pri generaciji. Ovo se može implementovati dodavanjem 256 koraka u inicijalnoj fazi, gde svaki korak generše – i odbacuje – keystream bajtova na osnovu sledećeg algoritma:

Listing 4.5 Generisanje bajtova ključa

```
i = (i + 1) mod 256
j = (j + S[i]) mod 256
swap(S[i], S[j])
t = (S[i] + S[j]) mod 256
keystreamByte = S[t]
```

Mnogi stream cipheri se zasnivaju na linearnim povratnim šift registrima (LFSR - *Linear Feedback Shift Registers*), koji je više efikasan u hardware-u, dok je manje efikasniji u software-u. Izrada RC4 izbegava korišćenje LFSR, ali je idealan za implementaciju software-a, zato što zahteva samo manipulaciju bajtova. Ona koristi 256 bajtova memorije za niz, od S [0] do S [255], k bajtova memorije za ključ, key [0] do key [k-1], i integer promenljive i, j, k.

Listing 4.2 primer jednostavne implementacije RC4 algoritma

```
unsigned char S[256];
unsigned int i, j;

/* KSA */
void rc4_init(unsigned char *key, unsigned int key_length) {
    for (i = 0; i < 256; i++)
        S[i] = i;
    for (i = j = 0; i < 256; i++) {
        unsigned char temp;
        j = (j + key[i % key_length] + S[i]) & 255;
        temp = S[i];
        S[i] = S[j];
        S[j] = temp;
    }
    i = j = 0;
}

/* PRGA */
unsigned char rc4_output() {
    unsigned char temp;
    i = (i + 1) & 255;
    j = (j + S[i]) & 255;
    temp = S[i];
    S[i] = S[j];
    S[j] = temp;
```

```
return S[(S[i] + S[j]) & 255];
}

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    unsigned char test_vectors[3][2][32] =
    {
        {"Key", "Plaintext"}, {"Wiki", "pedia"}, {"Secret", "Attack
at      dawn"}
    };

    int x;
    for (x = 0; x < 3; x++) {
        int y; 7
        rc4_init(test_vectors[x][0],
strlen((char*)test_vectors[x][0]));
        for (y = 0; y < strlen((char*)test_vectors[x][1]); y++)
            printf("%02X", test_vectors[x][1][y] ^ rc4_output());
        printf("\n");
    }
    return 0;
}
```

4.4.4 Napad na RC4

Kao što je već pomenuto, RC4 koristi niz bitova (keystream). Korišćenjem logičkog operatora XOR, keystream se kombinuje sa porukom koja treba da se šifruje i tako se dobija šifrirana poruka. Vraćanje originalne poruke se postiže tako što se šfrovana poruka XOR-uje keystream-om koji je korišćen za šifrovanje.

RC4 ključ ima 64 bita i sastoji se iz dva dela: 40-bitnog tajnog ključa i 24-bitnog vektora za inicijalizaciju (IV). Vektor za inicijalizaciju je broj koji se kontinualno menja kako bi se spremilo da dva parčeta teksta, koja su identična, budu ista i u šifriranom tekstu. Tako svaki paket koji se šalje ima drugi vektor za inicijalizaciju, čime se povećava zaštita.

Kao što smo mogli da vidimo, vektor za inicijalizaciju se sastoji od malog broja bitova tako da se, posle nekog vremena, određene kombinacije ponavljaju. Postoje neki vektori koji otkrivaju informacije o ključu. Ovi vektori su poznati kao slabi vektori i ima ih oko 9000 od ukupno 16000000 mogućih. Ova slabost, koristi se za razbijanje šifre.

Fluhrer, Mantin, and Shamir (FMS) je najkorišćeniji napad na WEP. On je najviše popularizovan programom AirSnort. Napad koristi mane RC4 algoritma za kreiranje ključeva i slabe vektore za inicijalizaciju. Kao što je već napomenuto, postoje slabi vektori za inicijalizaciju koji otkrivaju informacije o ključu. Kako se koristi isti ključ, a menjaju se samo vektori za inicijalizaciju, prikupljanjem velikog broja paketa sa slabim vektorima može se otkriti ključ. Za otkrivanje ključa potrebno je još znati gde je početak keystream-a.

Po standardu 802.11b, paket ima zaglavje koje je skoro uvek 0hAA. Ovo znači da prvi bajt keystream-a možemo lako da otkrijemo XORovanjem bajtova sa 0hAA.

Kada smo otkrili početak, sledeći korak je da lociramo slabi vektor. Ovaj vektor je 24-bitni (3-bajtni). Slabi vektori mogu se pronaći u formi (A+3, N-1,X), gde je A bajt keystream-a

koji napadamo, N je 256 (zato što RC4 radi sa modulom 256), i X može biti bilo koja vrednost. Ako napadamo nulti bajt keystream-a, tu će biti 256 slabih vektora u formi (3, 255, X), gde X ide od 0 do 255. Bajtovi keystream-a se napadaju redom, tako da pvi bajt ne može da se napadne dok nulti nije poznat.

Algoritam koji se ovde koristi nije komplikovan. Prvo se prolazi kroz A+3 koraka KSA (Key-Scheduling Algorithm). Ovo može da se uradi bez znanja ključa, zato što će slabi vektor zauzimati tri prva bajta u K nizu. Ako je nulti bajt poznat i A=1, može se raditi četiri koraka KSA, zato što će prva četiri bajta u nizu K biti poznata.

Ako S[0] ili S[1] budu promenjeni u zadnjem koraku, ceo napada se počinje iz početka. To može da se ispita i uslovom, da li je $j < 2$. Ako jeste početi napad iz početka, u suprotnom vrednost j i S[A+3] oduzeti od prvog bajta keystream-a, i nad rezultatom izvršiti operaciju modul 256. Ako se ovaj postupak ponovi za 60 slabih vektora za inicijalizaciju verovatnoća da se otkrije ključ je veća od 50. Kada se otkrije prvi bajt postupak se ponavlja i za ostatak ključa. Radi lakšeg objašnjenja samog algoritma umesto vrednosti 256 uzećemo 16, pri čemu će „bajtovi“ imati po 4 bita.

Prepostavimo da je ključ (1, 2, 3, 4, 5), i da se napada nulti bajt, što znači da je A=0. Ovo znači da će slabi vektor za inicijalizaciju da bude u obliku (3, 15, H). U ovom primeru H će biti 2, tako da će vektor i ključ spojeni izgledati (3, 15, 2, 1, 2, 3, 4, 5). Koristeći ovo izračunavanjem dobijamo da je prvi bajt keystream-a 9.

```

Izlaz    =   9
A        =   0
IV      = 3, 15, 2
Ključ   = 1, 2, 3, 4, 5
Seed    = IV i ključ, spojeni

K[] = 3 15 2 X X X X X 3 15 2 X X X X X
S[] = 0  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

```

Pošto je ključ trenutno nepoznat, niz K se popunjava trenutno poznatim vrednostima, a S niz se popunjava sekvencijalnim vrednostima od 0 do 15. j se postavlja na 0 i prva tri koraka KSA se izvršavaju.

KSA prvi korak:

```

i = 0
j = j + S[i] + K[i]
j = 0 + 0 + 3 = 3
Zameni S[i] i S[j]

```

```

K[] = 3 15 2 X X X X X 3 15 2 X X X X X
S[] = 3 1 2 0 4 5 6 7 8 9 10 11 12 13 14 15

```

KSA drugi korak:

```

i = 1
j = j + S[i] + K[i]
j = 3 + 1 + 15 = 3
Zameni S[i] i S[j]

```

$K[] = 3\ 15\ 2\ X\ X\ X\ X\ X\ 3\ 15\ 2\ X\ X\ X\ X\ X\ X$
 $S[] = 3\ \mathbf{0}\ 2\ \mathbf{1}\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15$

KSA treći korak:

$i = 2$
 $j = j + S[i] + K[i]$
 $j = 3 + 2 + 2 = 7$
 Zameni $S[i]$ i $S[j]$

$K[] = 3\ 15\ 2\ X\ X\ X\ X\ X\ 3\ 15\ 2\ X\ X\ X\ X\ X\ X$
 $S[] = 3\ 0\ \mathbf{7}\ 1\ 4\ 5\ 6\ \mathbf{2}\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15$

Proveravamo da li je $j < 2$, ako nije, proces može da se nastavi. $S[3]=1$, $j=7$ i prvi bajt keystream-a je bio 9, tako da bi nulti bajt ključa trebalo da bude $9-7-1=1$.

Ova informacija može da se iskoristi za nalaženje sledećeg bajta, koristeći vektor u formi $(4, 15, H)$ i prolazeći kroz KSA u četiri koraka. Ako se za vektor zzme forma $(4, 15, 9)$ dobija se da je prvi bajt keystream-a 6.

Izlaz	=	6
A	=	0
IV	=	4, 15, 9
Ključ	=	1, 2, 3, 4, 5
Seed	=	IV i ključ, spojeni

$K[] = 4\ 15\ 9\ 1\ X\ X\ X\ X\ 4\ 15\ 9\ 1\ X\ X\ X\ X\ X$
 $S[] = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15$

KSA prvi korak:

$i = 0$
 $j = j + S[i] + K[i]$
 $j = 0 + 0 + 4 = 4$
 Zameni $S[i]$ and $S[j]$

$K[] = 4\ 15\ 9\ 1\ X\ X\ X\ X\ 4\ 15\ 9\ 1\ X\ X\ X\ X\ X$
 $S[] = \mathbf{4}\ 1\ 2\ 3\ \mathbf{0}\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15$

KSA drugi korak:

$i = 1$
 $j = j + S[i] + K[i]$
 $j = 4 + 1 + 15 = 4$
 Zameni $S[i]$ and $S[j]$

$K[] = 4 \ 15 \ 9 \ 1 \ X \ X \ X \ X \ 4 \ 15 \ 9 \ 1 \ X \ X \ X \ X$
 $S[] = 4 \ 0 \ 2 \ 3 \ 1 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15$

KSA treći korak:

$i = 2$
 $j = j + S[i] + K[i]$
 $j = 4 + 2 + 9 = 15$
Zameni $S[i]$ and $S[j]$

$K[] = 4 \ 15 \ 9 \ 1 \ X \ X \ X \ X \ 4 \ 15 \ 9 \ 1 \ X \ X \ X \ X$
 $S[] = 4 \ 0 \ 15 \ 3 \ 1 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 2$

KSA četvrti korak:

$i = 3$
 $j = j + S[i] + K[i]$
 $j = 15 + 3 + 1 = 3$
Zameni $S[i]$ and $S[j]$

$K[] = 4 \ 15 \ 9 \ 1 \ X \ X \ X \ X \ 4 \ 15 \ 9 \ 1 \ X \ X \ X \ X$
 $S[] = 4 \ 0 \ 15 \ 3 \ 1 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 2$

Izlaz - $j - S[4] = \text{key}[1]$
6 - 3 - 1 = 2

U ovom primeru vrednosti promenljive H su strateški odabrane. Algoritam za razbijanje šifre prolazi kroz sve vrednosti promenljive H i kao vrednost bajta uzima onu vrednost koja se najveći broj puta javila kao rezultat. Sledi deo koda koji je u stvari implementacija prethodno opisanog algoritma.

Listing 4.7 Implementacija algoritma za napad na RC4

```
for(k=0; k < 256; k++)
    results[k] = 0;

IV[0] = A + 3;
IV[1] = N - 1;

for(x=0; x < 256; x++)
{
    IV[2] = x;
    keystream = RC4(IV, key); // ovde od kljuca i vektora
    //pravimo keystream koji kasnije koristimo za
    // razbijanje kljuca

    //seed = IV + key;
    for(k=0; k<3; k++)
```

```
seed[k] = IV[k];
for(k=0; k<13; k++)
    seed[k+3] = key[k];

// -- Key Scheduling Algorithm (KSA) --
//Initialize the arrays
for(k=0; k<256; k++)
{
    S[k] = k;
    K[k] = seed[k%16];
}

j=0;
for(i=0; i < (A + 3); i++)
{
    j = (j + S[i] + K[i])%256;
    t = S[i];
    S[i] = S[j];
    S[j] = t;
}

if(j < 2)
{
    printf("S[0] or S[1] have been disturbed, discarding..\n");
}
else
{
    known_j = j;
    known_S = S[A+3];
    keybyte = keystream - known_j - known_S;

    while(keybyte < 0)
        keybyte = keybyte + 256;

    results[keybyte] = results[keybyte] + 1;
}
}

max_result = -1;
max_count = 0;

for(k=0; k < 256; k++)
{
    if(max_count < results[k])
    {
        max_count = results[k];
        max_result = k;
    }
}
printf (max_result); //stampamo vrednost bajta kljuca
}
```

4.5 RC6

RC6 je algoritam koji je razvio Ronald Rivest iz RSA Security u saradnji sa Matt Robshaw, Ray Sidney i Yiqun Lisa Yin. Iterativna funkcija obavlja rotaciju variabli koja je regulisana kvadratnom funkcijom podataka. Svaka iteracija takođe uključuje 32-bitnu modularnu množenje, dodavanja, XOR operaciju i dodavanja ključa. Nastao je na osnovi RC5 algoritma. RC6 je vrlo jednostavan i njegova struktura je relativno poznata i testirana kroz RC. Vrlo je brz na 32-bitnim platformama. Algoritam je fleksibilan po pitanju veličine ključa, bloka i broja rundi pri enkripciji i dekripciji. Lošije strane su relativno niska margina sigurnosti, opadajuće performanse na neodgovarajućim platformama, te nedovoljna pogodnost za implementaciju na pametnim karticama.

Inače, RC6 je namerno konstruisan što jednostavnije kako bi na sebe privukao što više pažnje analitičara. Kako je RC5 bio duže vreme analiziran, od ranije je poznato na koji način pojedinačna transformacija utiče na sigurnost. U algoritmu je ostavljen širok prostor odabira veličine ključa i broja rundi. Ključna transformacija je rotacija zavisna od ključa. Po samom izlasku algoritma primećeno je da nema ni manjih ni većih sigurnosnih propusta.

4.5.1 Prednosti

- Brz na 32-bitnim platformama, vrlo brz na platformama sa sistemskom podrškom za 32-bitne rotacije i množenja.
- Velika jednostavnost omogućuje analizu, pogotovo kad je vreme ograničeno.
- Razvijen je iz RC5, koji je dosta analiziran.
- Brz postupak generisanja ključeva.
- Podržava i veličine ključa puno veće od 256 bita (teoretski do 1248 bita).
- Veličina ključa, veličina bloka i broj rundi su promenljivi u širokim granicama.

4.5.2 Nedostaci

- Relativno niska sigurnosna margina
- Performanse opadaju na platformama bez odgovarajuće sistemske podrške.
- Nemogućnost generisanja ključa u trenutku potrebe za ključem (engl. *on-the-fly*)
- Sigurnosni nedostaci u implementaciji za pametne kartice:
- Teško odbranjiv protiv timing i power napada zbog upotrebe velikog broja varijabilnih rotacija i kvadriranja.
- Manji nedostaci u nekim vrtstama napda.

4.5.3 Specifikacije

RC6 je zapravo skup algoritama gde je jedna verzija algoritma specificirana sa tri varijable w , r i b . Tako je kod algoritma RC6-w/r/b jedna reč veličine w , broj rundi r a dužina korisničkog ključa b bajta. Kako se u takmičenju za AES (*Advanced Encryption Standard*) pojavila verzija RC6-32/20/b (jer je tražena promenljiva dužina ključa) tako je to i najčešće spominjana verzija algoritma. Korišćene vrednosti za dužinu ključa su zbog zahteva takmičenja postavljene fiksno na 16, 24 i 32 reči.

Za sve varijante RC6 algoritma transformacije se obavljaju na četiri reči dužine w bita. Koristi se sledećih šest osnovnih operacija:

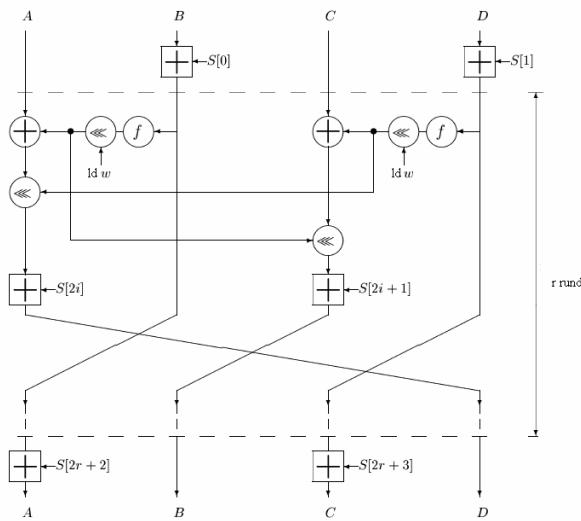
1. $a + b$ sabiranje modula 2^w
2. $a - b$ oduzimanje modula 2^w
3. $a \text{ xor } b$ ekskluzivno ili reči dužine w bita
4. $a \times b$ množenje modula 2^w
5. $a \lll b$ rotacija reči dužine w bita za b mesta u levo
6. $a \ggg b$ rotacija reči dužine w bita za b mesta u desno

U slučaju kada je kod rotacija u levo i desno b po iznosu veći od dužine reči w tada se uzima samo $\log_2 w$ najmanje značajnih bita od b (nekad se umesto $\log_2 w$ koristi oznaka ld).

Korisnički ključ dužine b bajta se na početku prepisuje u polje reči L . Poslednja reč u polju L ostaje u opštem slučaju polupotpunjena i ona se nadopuni nulama. Broj reči u L označen je oznakom c . Koriste se i dve "magične" konstante P i Q koje su konkretno određene u specifikacijama RC6 ali se mogu odabrati i proizvoljno. Generisani ključevi se nalaze u nizu S i ukupno se generiše $2^r + 4$ podključeva.

Listing 4.8 RC6 - generisanje ključeva

```
S[0] = P;
for i = 1 to 2r+3 do
    S[i] = S[i-1] + Q;
A = B = i = j = 0;
v = 3 * veci_od{c, 2*r+4};
for s = 1 to v do
    A = S[i] = (S[i] + A + B) <<< 3;
    B = L[j] = (L[j] + A + B) <<< (A + B);
    i = (i + 1) mod (2*r+4);
    j = (j + 1) mod c;
end for
```



Slika 4.4 Grafički prikaz kriptovanja algoritmom RC6

Za kriptovanje se koriste se četiri registra A, B, C i D svaki veličine jedne reči od w bita koji inicijalno sadrže čist tekst a po završetku kriptovanja kriptovani tekst. Podključevi su u postupku generisanja smešteni u polje S.

Listing 4.3 Kriptovanje kod RC6

```
B = B + S[0];
D = D + S[1];
for i = 1 to r do
    t = (B x (2B + 1)) <<< log2(w);
    u = (D x (2D + 1)) <<< log2(w);
    A = ((A xor t) <<< u) + S[2i];
    C = ((C xor u) <<< t) + S[2i+1];
    rotiraj (A, B, C, D);
end for
A = A + S[2r+2];
C = C + S[2r+3]
```

Procedura *rotiraj* je jednostavna ciklična zamena vrednosti registara (Temp = A, A=B, B=C, C=D, D=Temp). Vidi se da je postupak kriptovanja krajnje trivialan te je RC6 u velikoj prednosti jer jednostavnost olakšava samu analizu algoritma.

Za ovaj algoritam je lako napisati algoritam za postupak dekriptovanja jednostavno prateći unazad naredbu po naredbu iz postupka kriptovanja.

Listing 4.4 Algoritam za dekodiranje kod RC6

```
C = C - S[2r+3];
A = A - S[2r+2];
for i = r to 1 do
    inv_rotiraj (A, B, C, D);
    t = (B x (2B + 1)) <<< log2(w); //ove vrednosti ostaju
    u = (D x (2D + 1)) <<< log2(w); //iste
```

4. SIMETRIČNI KRIPTO-KODERI TOKA

```
C = ((C - S[2i+1]) >>> t) xor u;
A = ((A + S[2i]) >>> u) xor t;
end for
D = D - S[1];
B = B - S[0];
// Funkcija inv_rotiraj je inverzna funkcija funkcije
rotiraj.
```

Prethodnik algoritmu RC6 bio je RC5 i on je duže vreme, pored silnih analiza, smatran sigurnim. Kako je vreme pravi pokazatelj jačine sigurnosti algoritma, RC6 je samo naprednija verzija RC5 i neće biti nekih prelomnih otkrića u snazi ovog algoritma. Jedini poznati napad primenjiv na RC6 je pretraživanje celog prostora i zbog toga se smatra sigurnim.

5 SIMETRIČNI KODERI – KODERI BLOKA

Osnova simetričnih kodera je Fajstelov koder. Pojam Fajstelov koder (**Feistel cipher, Feistel network**), u kriptografiji i kriptoanalizi, označava bilo koji simetrični koder koji radi nad blokovima podataka. Ime je dobio po Horstu Fajstelu jednom od najznačajnijih IBM-ovih kriptografa koji je po obrazovanju bio fizičar [26],[27].
(<http://domino.research.ibm.com/comm/pr.nsf/pages/bio.feistel.html>).

Veliki broj današnjih simetričnih kodnih sistema koristi i dalje Fajstelovu mrežu kao osnovu. Prednost Fajstelove mreže je u tome što su operacije kriptovanja i dekriptovanja veoma slične, a u nekim slučajevima čak i identične. Sa tačke gledišta implementacije kodera u hardveru, ovakvi koderi donose uštedu do 50%. Fajstelova mreža je jednostavne strukture, ali implementira višestruko ponavljanje svih koraka u algoritmu. Jedno ponavljanje svih algoritamskih koraka se zove runda (kao runda u borilačkim sportovima). Najčešći koraci u okviru jedne runde su:

- Mešanje (permutovanje) bitova (Bit-shuffling). Strukture podataka koje se koriste pri ovom koraku se najčešće nazivaju P – matrice ili P – kutije (slovo P dolazi od reči permutacija)
- Izvršenje jednostavne nelinearne funkcije nad bitovima. Strukture podataka koje se ovde koriste zovu se matrice zamene (substitution boxes) ili S – matrice.
- Linearno mešanje (u smislu modularne algebre) koristeći XOR funkciju nad bitovima.

Izvršenje navedena tri koraka u svakoj od rundi imaće za rezultat stvaranje „konfuzije i difuzije“, na način kako je to opisao Klod Šenon (Claude Shannon) u svom članku "Communication Theory of Secrecy Systems" objavljenom još 1949. godine. Konkretno, mešanje bitova doprinosi efektu difuzije, dok zamena podataka i mešanje doprinose konfuziji.

Na sledećoj slici dat je prikaz Fajstelove mreže. Sa F je označen blok koji izvršava takozvanu funkciju runde (engleski termin je round function, koji ne treba mešati sa funkcijama zaokruživanja brojeva). Ovu funkciju definiše autor kodera i ona je specifična za svaki algoritam ponaosob.

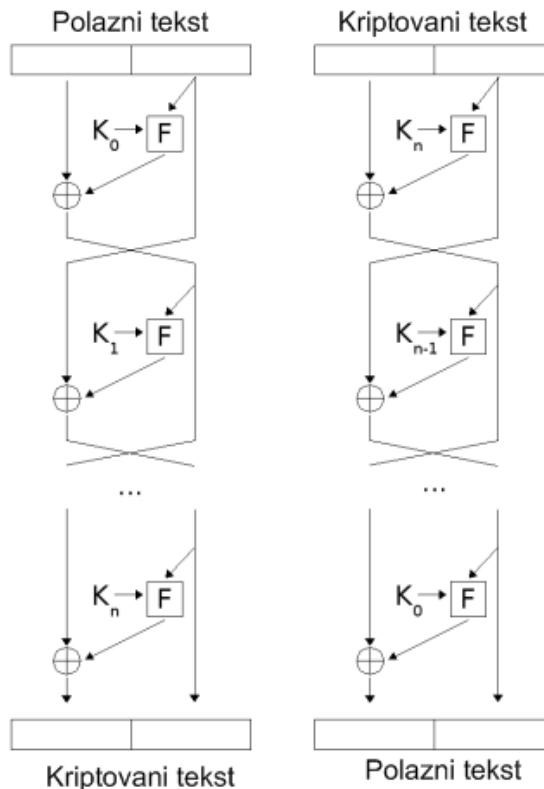
Sa K0, K1, do Kn, označeni su podključevi koji koriste u svakoj od rundi (brojevi rundi su 0,1 do n). Ovde je potrebno da autor algoritma generiše opšti ključ, najčešće se obeležava

sa K, kao i algoritam za distribuciju ključa (key schedule algorithm) na osnovu koga će se generisati K₀, K₁, do K_n.

Još jedan preduslov pre početka operacije kriptovanja je podela podatka koji treba kriptovati na blokove, pošto je ovo koder koji radi na nivou blokova podataka. Sada se svaki blok podataka kriptuje u toku definisanog broja rundi, tako što se u svakoj od njih ponovi sledeća sekvenca koraka:

- Uzeti jedan blok podataka koje treba kodovati i podeliti ga na dva dela koji imaju jednak broj bitova. Ti inicijalno dobijeni blokovi se najčešće označavaju kao L₀ i R₀.
- Ako naš sistem ima m rundi, u svakoj od njih koder računa sledeće: L(i + 1) = R(i); R(i + 1) = L(i) \oplus F(R(i), K(i)). Na kraju, L(m) i R(m) predstavljaju kodovane podatke. Ovde treba uočiti da u procesu kodovanja indeks i na početku ima vrednost 0, a na kraju procesa ima svoju maksimalnu vrednost, dok je kod dekodovanja situacija sa indeksima obrnuta.

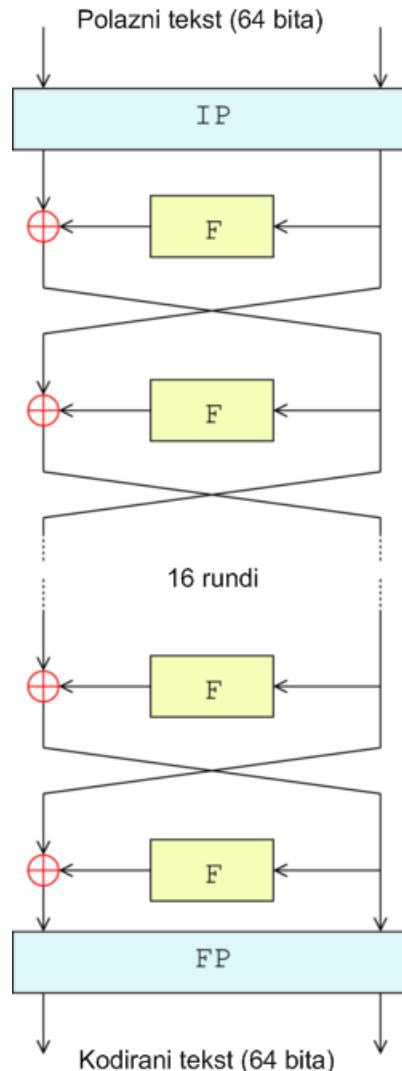
Proces dekriptovanja, u tom slučaju, podrazumeva izvršenje sledećih operacija u svakoj rundi: R(i) = L(i + 1); R(i + 1) = R(i + 1) \oplus F(L(i + 1), K(i)). Prednost ovog modela je u tome što funkcija runde F ne mora da bude inverzna, a pri tome može da bude i izuzetno kompleksna. Sve ovo može se dodatno usložiti, tako što inicijalni L₀ i R₀ neće biti iste veličine – u tom slučaju govorimo o nebalansiranim Fajstelovim koderima.



Slika 5.1 Osnovna šema fajstelovog kodera

5.1 DES

DES je blok koder direktno implementiran po Fajstelovoj definiciji kodera i kaže se da je arhetipski blok koder. On uzima string fiksne dužine kao polazni tekst, koduje ga kroz niz operacija i na kraju vraća kriptovani tekst iste dužine. U slučaju DES-a početni tekst se deli na blokove dužine 64 bita [23].



Slika 5.2 Fajstelova mreža kod DES algoritma

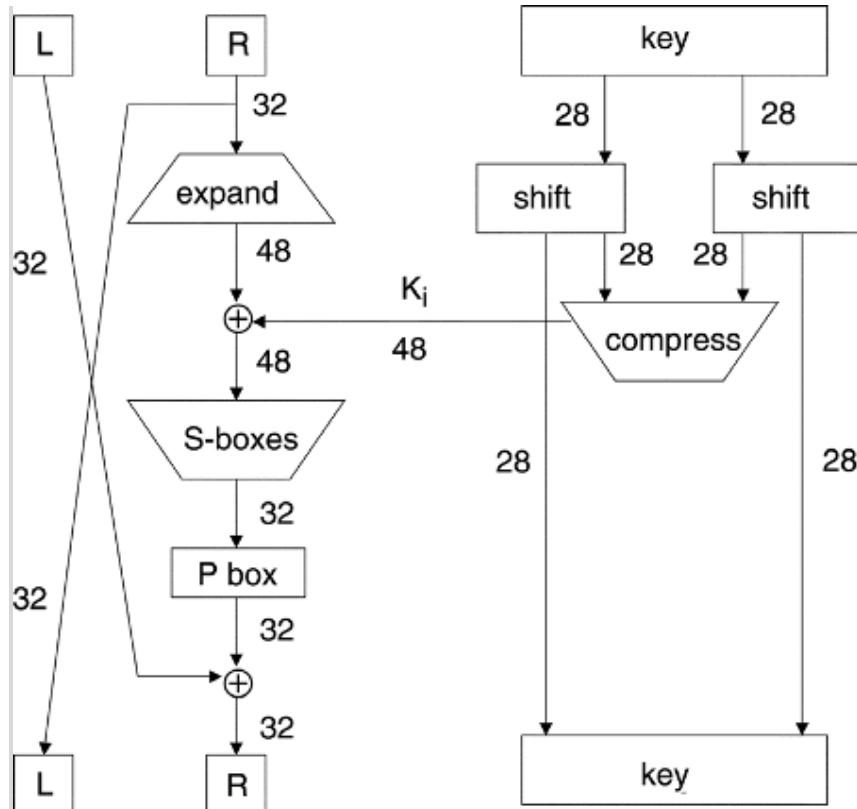
DES naravno koristi i ključ kako bi izvršavao transformacije, tako da dekripciju može da izvrši samo onaj ko zna ključ. Ključ se sastoji, realno, od 64 bita, ali se za kodiranje u

algoritmu koristi samo njih 56. Preostalih 8 se koristi za složenu kontrolu parnosti (parity-check). Dakle, efektivni ključ ima 56 bitova, i najčešće se u definiciji kodera navodi da DES koristi ključ od 56 bitova.

5.1.1 Opšta struktura

Opšta struktura kodera je data na slici 1. Proces se sastoji od 16 identičnih delova procesa koji se nazivaju runde. Takođe, tu su i inicijalna (IP) i završna (FP) permutacija koje se obavljaju pre prve (IP) i nakon poslednje (FP) runde. IP i FP su inverzne permutacije [24].

Pre nego što glavni proces počne blok koji treba kodovati, podeli se na dve polovine od po 32 bita, i procesira se po Fajstelovoj šemi sa slike 1. Fajstelova šema obezbeđuje da kodovanje i dekodovanje budu vrlo slični procesi, jedina je razlika ta što se podključevi tokom dekriptovanja primenjuju u obrnutom redu u odnosu na proces kriptovanja. Ovo je veoma zgodno kada ovaj algoritam treba implementirati u hardveru, jer se onda isti hardver može koristiti za oba procesa.



Slika 5.3 Detaljni prikaz jedne runde u DES-u

F – funkcija kombinuje pola ulaznog bloka sa nekim od ključeva. Izlaz iz F funkcije se zatim kombinuje sa drugom polovinom bloka, i polovine zamene mesta pre početka sledeće runde (slika 2). Nakon poslednje runde polovine ne menjaju mesta; ovo dodatno obezbeđuje da se ista mreža može koristiti i za kodovanje i za dekodovanje [25].

5.1.2 Fejstelova (F) funkcija kod DES algoritma

F funkcija prikazana na slikama 2 i 3, kao ulaze ima pola početnog bloka podataka (32 bita) i podključ od 48 bitova i ima sledeće korake:

1. Proširenje početnog polu – bloka (Expansion) — 32-bitni ulazni podataka treba proširiti na 48 bita koristeći takozvane ekspanzionate permutacije (blok E na slici 3) koje dupliciraju neke od bitova. Permutaciona tablica (vektor) izgleda ovako:

```
public static int[] pc_e = { 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8,
                            9, 10, 11, 12, 13, 12, 13, 14, 15, 16,
                            17, 16, 17, 18, 19, 20, 21, 20, 21, 22,
                            23, 24, 25, 24, 25, 26, 27, 28, 29, 28,
                            29, 30, 31, 32, 1 };
```

Dakle, vektor pc_e kaže u kome će rasporedu bitovi početnog (32-bitnog) podatka biti raspoređeni u proširenom 48-bitnom podatku. Ovde treba oučiti da su u vektoru indeksi početnog podatka „1-based“, a ne „0-based“ kao što je uobičajeno.

2. *Mešanje sa ključem (Key mixing)* — rezultat koraka 1 se sada kombinuje sa ključem koji odgovara tekućoj rundi, koristeći XOR operaciju. I tekući ključ i rezultat koraka 1 su 48-bitni tako da se nad njima direktno može izvršiti XOR nad bitovima.
3. *S-box transformacija (Substitution)* — nakon mešanja sa ključem dobijeni 48-bitni blok se podeli na osam grupa po 6 bitova i nad svakom od njih se vrši S-box transformacija sa odgovarajućom S-box matricom (slika 2). Svaka od 8 S-box matrica zamenjuje šestobini ulaz četvorobitnim izlazom u skladu sa nelinearnom transformacijom koja je u njoj zapisana. S-box matrice se koriste kao lookup tabele. Ulazni podatak, kao što je rečeno je šestobitan, a izlazni četvorobitan. Četvorobini izlazi su predstavljeni sa svojim hex vrednostima.

Tabela 1 S-box 1 u DES-u

$b_0 b_5$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7
1	0	F	7	4	E	2	D	1	A	6	C	B	9	5	3	8
2	4	1	E	8	D	6	2	B	F	C	9	7	3	A	5	0
3	F	C	8	2	4	9	1	7	5	B	3	E	A	0	6	D

Iz ulaznog 6-bitnog podatka izdvoje se prvi i zadnji bit i oni zajedno čine indeks reda (označimo ga sa **row**), dok ostala 4 bita čine indeks kolone (**column**). Nakon toga vrednost kojom menjamo ulazni šestobitni podatak izabraćemo iz matrice kao S1[row, column].

Nakon što svih 8 ulaznih 6-bitnih konvertujemo u 4-bitne podatke, i spojimo ih, dobijamo 32-bitni podatak koji dalje možemo da transformišemo.

S-box matrice predstavljaju osnovu sigurnosti koju DES obezbeđuje, bez njih koder bi bio linearan i veoma bi bilo lako razbiti ga. Sledi prikaz svih 8 S-box matrica.

```
public static readonly int[,] s1 ={
{ 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },
{ 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },
{ 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },
{ 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } };

public static readonly int[,] s2 ={
{ 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },
{ 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },
{ 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },
{ 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } };

public static readonly int[,] s3 ={
{ 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },
{ 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },
{ 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },
{ 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } };

public static readonly int[,] s4 ={
{ 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },
{ 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },
{ 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
{ 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } };

public static readonly int[,] s5 ={
{ 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },
{ 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
{ 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
{ 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } };

public static readonly int[,] s6 ={
{ 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },
{ 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
{ 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
{ 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } };

public static readonly int[,] s7 ={
{ 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },
```

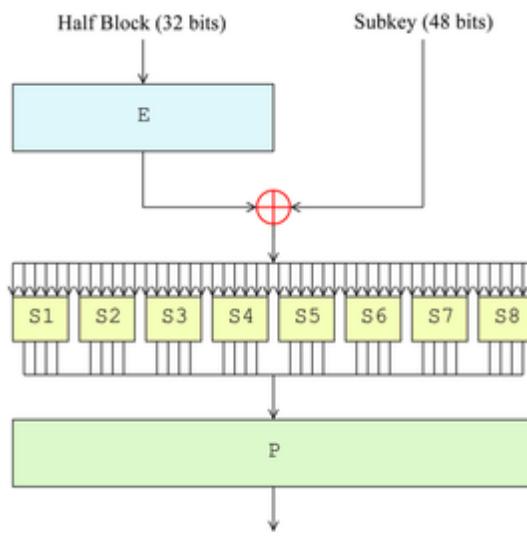
```
{ 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
{ 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
{ 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } };
```

```
public static readonly int[,] s8 ={
{ 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },
{ 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
{ 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
{ 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } };
```

4. *Permutacija (Permutation)* — na kraju, izlaz iz S-box transformacije (32 bita) se preuređuje uz pomoć P-niza, koji uvodi fiksnu permutaciju nad bitovima. Niz koji definiše tu permutaciju zove se *P-box* i izgleda ovako:

```
public static readonly int[] pc_p ={
16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23,
26, 5, 18, 31, 10, 2, 8, 24, 14, 32,      27, 3, 9, 19, 13, 30,
6, 22, 11, 4, 25 };
```

Zamena vrednosti i permutacija koju donose S-box matrice i P-box niz, obezbeđuju „konfuziju“ dok proširenje podataka sa početka algoritma, obezbeđuje difuziju. Ovo su dva osnovna zahteva koje koderi treba da ispunile kako bi ispunili Šenonov uslov za siguran i praktičan koder.



Slika 5.4 Fajstelova funkcija u DES algoritmu

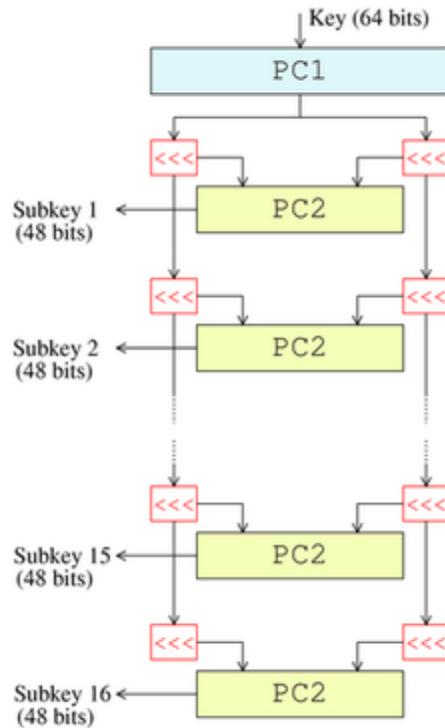
5.1.3 Algoritam za distribuciju u DES algoritmu

Slika ilustruje distribuciju ključa za proces kriptovanja, tj algoritam koji generiše 48-bitne podključeve koji se koriste u svakoj od rundi. Na početku se od 64-bitnog prosleđenog ključa

izdvaja njih 56 koji će se koristiti dalje. Oni se izdvajaju pomoću matrice (niza) koja se zove *Permuted Choice 1* ili *PC-1*. Preostalih osam bitova se ili odbace ili se koriste za kontrolu parnosti.

```
public static readonly int[] pc_1 =
{ 57, 49, 41, 33, 25, 17, 9, 1, 58, 50,
  42, 34, 26, 18, 10, 2, 59, 51, 43, 35,
  27, 19, 11, 3, 60, 52, 44, 36, 63, 55,
  47, 39, 31, 23, 15, 7, 62, 54, 46, 38,
  30, 22, 14, 6, 61, 53, 45, 37, 29, 21,
  13, 5, 28, 20, 12, 4 };
```

Ulas u blok PC1 na slici 4 je pomenuti 64-bitni ključ, dok izlaz ima 56 bitova i generiše se na osnovu ulaza i matrice pc-1, dakle prvi bit u izlazu je 57.-mi bit ulaza, drugi u izlazu je 49.-ti sa ulaza, itd.



Slika 5.5 Distribucija ključa kod DES algoritma

Nakon dobijanja 5-to bitnog ključa, on se podeli na dva dela od po 28 bitova, i svaka polovina se tretira zasebno (slika 4). U svakoj rundi se svaka polovina rotira uлево ili udesno za jedan ili dva bita po rasporedu datom u vektoru:

nrOfShifts = { 0, 1, 1, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };

Nakon rotacije može se iz svake polovine se izdvojiti 24 bita i od njih se sklopiti 48-bitni podključ, ili se mogu obe polovine najpre sklopiti u jedan niz bitova, a onda korišćenjem vektora *Permuted Choice 2* (*PC-2*) izdvojiti potrebnih 48 bitova.

```
public static readonly int[] pc_2 =
{ 14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10,
 23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2,
 41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,
 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32 };
```

5.2 Modovi kodera

5.2.1 ECB

Kada su u pitanju koderi poput A5/1 može se zaključiti da oni sve podatke koje kodiraju tretiraju kao jedan jedini blok podataka, tako da se ne mora razmišljati o tome kako deliti podatke na blokove i šta dalje sa njima. Međutim, kod kodera koji rade sa blokovima podataka situacija je malo drugačija.

Naime, ako podelimo ulazni tekst na blokove P0, P1, P2, ... i iskodiramo ih po nekom blokkoderu (DES npr) kao rezultat ćemo dobiti niz kodiranih blokova koje možemo da nazovemo C0, C1, C2, Kako uvek za kodiranje koristimo isti ključ i isti postupak možemo da napišemo sledeće:

$$C_i = E(P_i, K)$$

gde je K, korišćeni ključ, a E je zapravo algoritam koji smo koristili da kodujemo ulazni podatak P_i . Ovo je osnovni mod funkcionalnosti kodera, dakle direktna primena algoritma na svaku ulaznu reč. Na ovaj način se dobija „codebook“ kodiranje gde uvek za isti ulaz dobijemo isti izlaz. Zbog toga se ovaj osnovni mod rada kodera naziva *Electronic codebook mode* ili ECB mod.

Dekripcija za ECB mod je, dakle, kao po algoritmu:

$$P_i = D(C_i, K)$$

gde je D funkcija dekodiranja.

ECB mod nažalost, čini napade veoma lakim jer je za slučaj $P_i = P_j$ i $C_i = C_j$, što omogućava lakšu rekonstrukciju ključa.

5.2.2 CBC

Na sreću, postoji nekoliko načina da se ova slabost izbegne. Prvi od njih je CBC mod kodera (CBC je skraćenica od cipher block chaining (ulančavanje kodiranih blokova)). Enkripciona formula u ovom slučaju je:

$$C_i = E(P_i \oplus C_{i-1}, K), \text{ za } i = 0, 1, 2, \dots$$

Dekriptovanje je u tom slučaju

$$P_i = D(C_i, K) \oplus C_{i-1}, \text{ za } i = 0, 1, 2, \dots$$

Ovde možemo da uočimo da će nam nedostajati C_{-1} . Zbog toga se ta vrednost postavlja na početku procesa kodiranja. Ona se ili automatski generiše, ili se unosi kao sekundarni ključ. Pomenuti C_{i-1} se zove vektor inicijalizacije i u algoritmima se najčešće označava kao IV.

Dakle IV se koristi za kodovanje i dekodovanje prvog bloka, dakle kodovanje prvog bloka je $C_0 = E(P_0 \oplus IV, K)$ a dekodovanje $P_0 = D(C_0, K) \oplus IV$.

Nedostatak ove metode je propagacija greške, ukoliko greška nastane.

5.2.3 CTR

CTR je skraćenica od counter mode. Kao i CBC, i CTR koristi vektor inicijalizacije IV, ali su formule po kojima se vrše kriptovanje i dekriptovanje su malo drugačije:

$$C_i = P_i \oplus E(IV + i, K), \text{ za } i = 0, 1, 2, \dots$$

$$P_i = C_i \oplus E(IV + i, K), \text{ za } i = 0, 1, 2, \dots$$

CTR mod omogućava zapravo, transformaciju iz blok-kodera u koder koji radi nad nizom podataka.

5.3 TDES (3DES)

Trostruki DES, 3DES ili TDES predstavlja algoritam dobijen na osnovu običnog DES- a tako što se DES kriptovanje sukcesivno ponovi 3 puta nad istim blokom podataka sa 3 različita ključa.

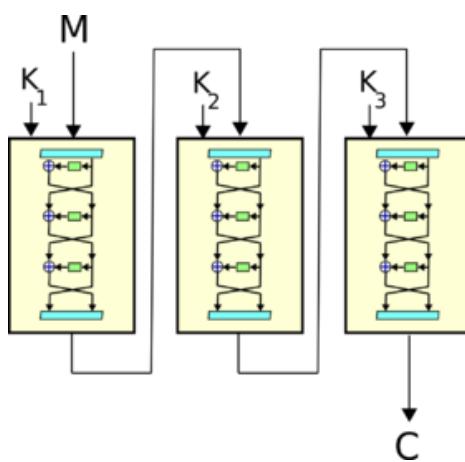
Ako osnovno DES kriptovanje obeležimo sa $C = E(P, K)$, a odgovarajuće dekriptovanje sa $P = D(C, K)$, dolazimo do inicijalnih formula za TDES:

$$C = E(E(E(P, K_1), K_2), K_3) \text{ i } P = D(D(D(C, K_3), K_2), K_1)$$

Međutim, danas je najkorišćenija konfiguracija TDES-a u kojoj se naizmenično zovu algoritmi za kriptovanje i dekriptovanje po svakom od ključeva. Pa je u tom slučaju:

$$C = E(D(E(P, K_1), K_2), K_3)$$

Ovde je interesantno da se za $K_1 = K_2$, TDES svodi na običan DES.



Slika 5.6 Tri sukcesivna poziva DES-a u 3DES-u

5.4 AES

AES je nastao tokom 1990-tih kao rezultat konkursa za koder koji bi zamenio DES, a koji je bio raspisani od strane američkog NIST instituta. Pobednik je bio Rijndael algoritam, koji je kasnije prihvaćen kao AES algoritam.

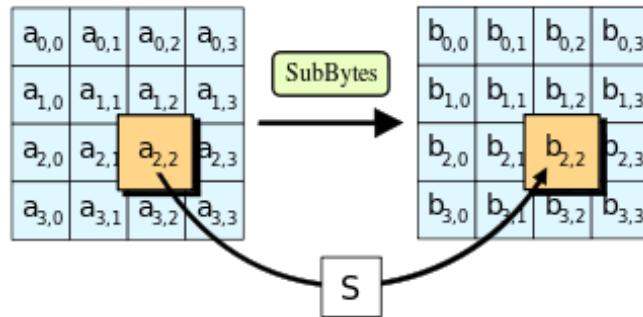
Poput DES-a, i AES je koder koji radi nad blokovima podataka i koji proces kodiranja izvršava kroz određeni broj rundi, međutim AES nije Fajstelov koder. Posledica toga je da procesi kriptovanja i dekriptovanja nisu ekvivalentni. Zapravo, ovde su dekriptovanje i kriptovanje inverzni procesi. Takođe, u odnosu na DES, AES je, sa matematičke tačke gledišta, izuzetno složen.

Ukratko, osnovne osobine standardnog AES algoritma su sledeće:

- Veličina bloka može biti 128, 192 i 256 bitova
- Dužina ključa (nije zavisna od dužine bloka) može biti takođe 128, 192 i 256 bitova
- Broj rundi je 10 do 14, u zavisnosti od dužine ključa
- U inicijalnoj rundi se postavljaju inicijalne vrednosti ključa (samo AddRoundKey operacija)
- U svakoj od narednih (regularnih) rundi obavljaju se sledeće akcije:
 - SubBytes
 - ShiftRows
 - MixColumns
 - AddRoundKey
- U finalnoj rundi izostavlja MixColumns operacija
- Distribucija ključa se obavlja po Rijndaelovom algoritmu za distribuciju ključa.

5.4.1 SubBytes operacija

Ulaz u SubBytes operaciju su bajtovi podatka koji treba da bude kriptovan, i oni su na slici 1 označeni kao elementi matrice A. U ovom koraku svaki ulazni bajt menja svoju vrednost nakon interakcije sa odgovarajućim elementom iz osmobilne supstitucione matrice (S-box). Ova operacija obezbeđuje nelinearnost dobijenog koda i obezbeđuje da takozvani linearni algebarski napadi nemaju efekta protiv ovog algoritma. S-matrica koja se ovde koristi je zapravo originalna Rijndaelova S-matrica.



Slika 5.7 SubBytes korak u AES-u

U kodu se S – matica implementira kao najjednostavnija lookup tabela i elementi resultantne matrice B se računaju kao $b_{ij} = S(a_{ij})$.

Direktna S matica izgleda ovako:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Za svaku moguću vrednost bajta iz A matrice definiše se vrednost kojom se taj bajt menja. Tako na primer ako je sekvenca ulaznih bajtova bila 12, a3, cc, f0, de, 74 rezultat SubBytes operacije bi bio c9, 0a, 4b, 8c, 1d , 92.

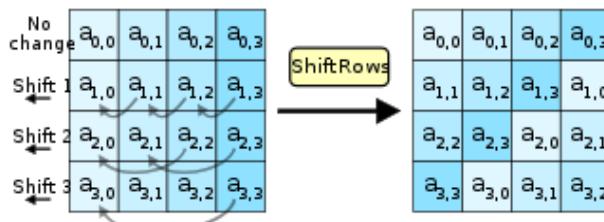
S matica se može koristiti (u programu) i kao jednodimenzionalni niz od 256 elemenata i onda je čak jednostavnija za korišćenje. Tada se jednostavno ulazni bajt iskoristi kao indeks niza i jednostavno se pročita vrednost sa odabrane pozicije. Pročitana vrednost se onda uvrsti u matricu B (vidi sliku 1) na odgovarajuće mesto.

Inverzna S-matrica za ovaj algoritam je sledeća:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
10	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
20	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
30	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
40	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
50	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
60	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
70	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
80	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
90	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a0	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b0	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c0	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d0	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e0	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f0	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

5.4.2 ShiftRows operacija

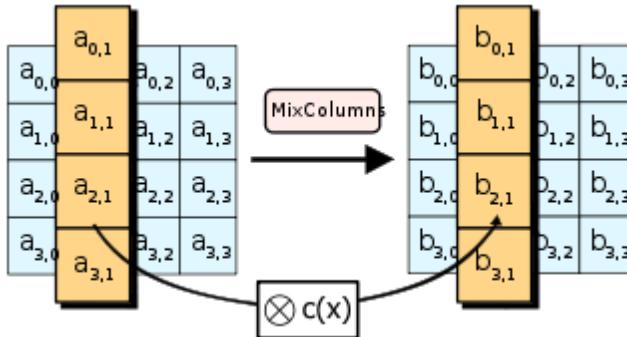
Ulaz u ovu operaciju je matrica koja predstavlja rezultat prethodnog (SubBytes) koraka. Tu matricu ćemo označiti sa A. Operacija ShiftRows radi nad elementima jednog reda matrice tako što ih ciklično pomera za odgovarajuću vrednost koja se naziva offset. U našem slučaju prvi red se ne menja. Drugi red se pomera za jedno mesto uлево, treći za dva, četvrti za tri (vidi sliku 2). Izlaz iz ove operacije je promenjena matrica A.



Slika 5.8 ShiftRows operacija u AES-u

5.4.3 MixColumns operacija

Ulaz u ovaj korak je matrica koju smo dobili kao rezultat ShiftRows koraka. Označićemo je sa A. U koraku MixColumns sva četiri bajta svake kolone promene mesta koristeći neku inverznu linearnu transformaciju. Dakle, MixColumns operacija uzima četiri bajta kao ulaz i vraća četiri bajta na izlazu. Zajedno sa ShiftRows ovaj korak obezbeđuje difuziju podataka u ovom kodnom sistemu.



Slika 5.9 MixColumns operacija u AES-u

Svaka kolona se ovde tretira kao polinom nad $\text{GF}(2^8)$ poljem i predstavlja rezultat množenja modula po $x^4 + 1$ i fiksnog polinoma $c(x) = 3x^3 + x^2 + x + 2$ (slika 3).

Međutim, ovde ćemo iskoristiti jednu pogodnost koju nam daju Galoaova polja a kako bi se izbeglo kompleksno računanje polinoma i modula. Ovaj korak ćemo izvesti tako što ćemo za množenje elemenata iskoristiti Rijndael-ovu MDS matricu (kad je on već izračunao):

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Ovo takođe možemo i da pišemo kao skup od četiri jednačine:

$$r_0 = 2a_0 + a_3 + a_2 + 3a_1$$

$$r_1 = 2a_1 + a_0 + a_3 + 3a_2$$

$$r_2 = 2a_2 + a_1 + a_0 + 3a_3$$

$$r_3 = 2a_3 + a_2 + a_1 + 3a_0$$

Međutim, problem je što sva ova množenja i sabiranja nisu „obična“ nego množenja i sabiranja u tzv. Rijndaelovom Galoaovom polju (tj. GF(64)). Tako da je sabiranje zapravo XOR operacija, a množenje je prilično komplikovana operacija, a za množenje sa 1, dva i tri možete koristiti sledeće metode:

```
private static byte gfmultby01(byte b)
{
    return b;
}

private static byte gfmultby02(byte b)
```

```
{  
    if (b < 0x80)  
        return (byte) (int) (b << 1);  
    else  
        return (byte) ( (int) (b << 1) ^ (int) (0x1b) );  
}  
  
private static byte gfmultby03(byte b)  
{  
    return (byte) ( (int)gfmultby02(b) ^ (int)b );  
}  
Za inverznu MixColumns koriste se izrazi:  

$$r_0 = 14a_0 + 9a_3 + 13a_2 + 11a_1$$

$$r_1 = 14a_1 + 9a_0 + 13a_3 + 11a_2$$

$$r_2 = 14a_2 + 9a_1 + 13a_0 + 11a_3$$

$$r_3 = 14a_3 + 9a_2 + 13a_1 + 11a_0$$

```

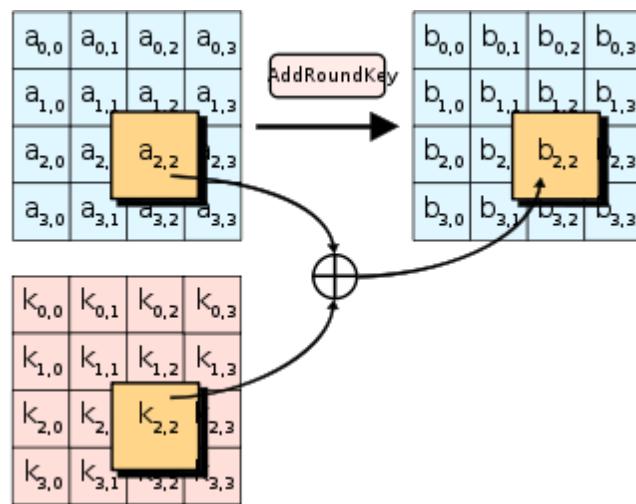
Odgovarajuće metode za množenje koje se mogu koristiti su:

```
private static byte gfmultby09(byte b)  
{  
    return (byte) ( (int)gfmultby02(gfmultby02(gfmultby02(b))) ^  
                  (int)b );  
}  
  
private static byte gfmultby0b(byte b)  
{  
    return (byte) ( (int)gfmultby02(gfmultby02(gfmultby02(gfmultby02(b)))) ^  
                  (int)gfmultby02(b) ^  
                  (int)b );  
}  
  
private static byte gfmultby0d(byte b)  
{  
    return (byte) ( (int)gfmultby02(gfmultby02(gfmultby02(gfmultby02(b)))) ^  
                  (int)gfmultby02(gfmultby02(b)) ^  
                  (int)(b) );  
}  
  
private static byte gfmultby0e(byte b)  
{  
    return (byte) ( (int)gfmultby02(gfmultby02(gfmultby02(gfmultby02(b)))) ^  
                  (int)gfmultby02(gfmultby02(b)) ^
```

```
(int) gfmultby02(b) );
}
```

5.4.4 AddRoundKey Operacija

Ulaz u ovaj korak je matrica dobijena nakon MixColumnOperacije. Nazovimo tu matricu A. Da bi smo dobili kriptovanu matricu (označićemo je sa B) iskombinovaćemo matricu A sa matricom koja sadrži ključ koristeći XOR operaciju nad odgovarajućim bajtovima (slika 4). Za svaku rundu se generiše poseban ključ koji se generiše na osnovu polaznog ključa po Rijndaelovom algoritmu za distribuciju ključeva. Svaki podključ je matrica koja je po dimenzijama jednaka matrici A.



Slika 5.10 AddRoundKey operacija u kod AES algoritma

5.5 Rijndael algoritam za distribuciju ključeva

Algoritam koji se koristi za distribuciju ključeva u okviru AES algoritma u originalu se zove Rijndael key schedule algoritam. Ovaj algoritam ne predstavlja deo gradiva za ispit. Studenti koji budu imali AES kao laboratorijsku vežbu, preuzeće generisanje i distribuciju ključa kao unapred pripremljenu metodu.

Pomoćne operacije koje se koriste u ovom algoritmu su rotacija (označava se sa Rotate ili RotWord), Rcon preslikavanje, S-box preslikavanje i osnovna raspodela ključeva.

5.5.1 Rotacija

Operacija RotWord ima funkciju 32-bitnog desnog kružnog pomeračkog registra. Tako na primer ako je ulazna reč 1d2c3a4f, nakon RotWord operacije reč će biti transformisana u 2c3a4f1d, ako su članovi stringa ASCII karakteri.

```
private byte[] RotWord(byte[] word)
```

```
{
    byte[] result = new byte[4];
    result[0] = word[1];
    result[1] = word[2];
    result[2] = word[3];
    result[3] = word[0];
    return result;
}
```

Rijndael's key schedule utilizes a number of operations, which will be described before describing the key schedule.

5.5.2 Rcon i SBox preslikavanja

Rcon preslikavanje se vrši na osnovu Rcon matrice. Opis kako je Rcon matrica dobijena ovde će biti zaobiđen. U nastavku je data puna Rcon matrica koja se može koristiti za implementaciju kodera. Za svaki bajt se može izračunati odgovarajuća Rcon vrednost direktnim čitanjem iz Rcon tabele. Ako je vrednost ulaznog bajta označena sa B, njegov Rcon ekvivalentn se računa kao RB = Rcon[B].

```
Rcon[255] = {0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91,
0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01,
0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a,
0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3,
0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83,
0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10,
0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f,
0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa,
0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8,
0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6,
0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,
0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02,
0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab,
0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4,
0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d,
0x3a, 0x74, 0xe8, 0xcb}
```

Za računanje može da se iskoristi i redukovana Rcon matrica koja izgleda ovako:

```
private void BuildRcon()
{
    this.Rcon = new byte[11,4] { {0x00, 0x00, 0x00, 0x00},
                                {0x01, 0x00, 0x00, 0x00},
                                {0x02, 0x00, 0x00, 0x00},
```

```
    {0x04, 0x00, 0x00, 0x00},  
    {0x08, 0x00, 0x00, 0x00},  
    {0x10, 0x00, 0x00, 0x00},  
    {0x20, 0x00, 0x00, 0x00},  
    {0x40, 0x00, 0x00, 0x00},  
    {0x80, 0x00, 0x00, 0x00},  
    {0x1b, 0x00, 0x00, 0x00},  
    {0x36, 0x00, 0x00, 0x00} };  
} // BuildRcon()
```

U tom slučaju Rcon transformacija za grupu od 4 bajta računa na sledeći način:

```
if (row % Nk == 0)  
{  
    temp = SubWord(RotWord(temp));  
  
    temp[0] = (byte)( (int)temp[0] ^ (int)this.Rcon[row/Nk,0] );  
    temp[1] = (byte)( (int)temp[1] ^ (int)this.Rcon[row/Nk,1] );  
    temp[2] = (byte)( (int)temp[2] ^ (int)this.Rcon[row/Nk,2] );  
    temp[3] = (byte)( (int)temp[3] ^ (int)this.Rcon[row/Nk,3] );  
}  
else if ( Nk > 6 && (row % Nk == 4) )  
{  
    temp = SubWord(temp);  
}
```

gde je Nk veličina ključa u bitovima, a metoda SubWord je definisana na sledeći način.

```
private byte[] SubWord(byte[] word)  
{  
    byte[] result = new byte[4];  
    result[0] = this.Sbox[ word[0] >> 4, word[0] & 0x0f ];  
    result[1] = this.Sbox[ word[1] >> 4, word[1] & 0x0f ];  
    result[2] = this.Sbox[ word[2] >> 4, word[2] & 0x0f ];  
    result[3] = this.Sbox[ word[3] >> 4, word[3] & 0x0f ];  
    return result;  
}
```

5.5.3 Osnovna raspodela ključeva

Ovo je operacija koja se odvija u unutrašnjoj petlji algoritma za raspodelu ključeva i za nju je karakteristično sledeće:

- Ulaz je 32-bitna reč, i redni broj iteracije **i**. Izlaz je 32-bitna reč
- Izlazna reč zamenjuje ulaznu u daljim računanjima.
- Prethodno navedena RotWord operacija se primenjuje na svaku grupu od 32 bita izlazne reči

- Primeni se S-box preslikavanje na svali bajt izlazne reči
- Primeniti Rcon operaciju nad vrednosti i , pa onda uraditi XOR nad izlazom iz pomenute Rcon operacije i prvog bajta izlazne reči.

5.5.4 Konstante

Pošto je distribucija ključa vrlo slična za svaku od mogućih dužina ključeva (i za 128 bita, i za 192 i za 256-bit), kao i sam proces kriptovanja definisaćemo sledeće konstante koje zavise od dužine ključa:

- **n** (broj bajtova u osnovnom ključu) ima vrednost 16 za ključ od 128 bita, 24 za 192 bita u ključu i 32 za 256-bitne ključeve
- **b** (broj bitova u proširenem ključu) ima vrednost 176 za 128-bitni ključ, 208 za 192-bitne i 240 za 256-bitne.

5.5.5 Opis distribucije ključeva

Rijndaelova distribucija ključeva se odvija po sledećim koracima:

- Prvih **n** bajtova proširenog ključa je zapravo uneti kodni ključ
- Vrednost **i** koja se koristi za Rcon iteracije je postavljena na 1.
- Sve dok ne dobijemo svih **b** bajtova proširenog ključa, radimo sledeće kako bi generisali nedostajuće bajtove:
 - Sledeće korake izvodimo kako bi generisali po 4 nedostajuća bajta u ključu:
 - Kreiramo privremenu četvorobajtnu promenljivu **t**
 - Prethodna 4 bajta u proširenom ključu dodelimo bajtovima u promenljivoj **t**
 - Izvedemo osnovnu raspodelu ključeva nad **t**, gde **i** koristimo kao Rcon ulaznu vrednost
 - Uvećamo **i** za 1
 - Izvršimo XOR nad **t** i četvorobajtnim blokom **n**, pre nego što krenemo da računamo novi prošireni ključ. Ovo će postati sledeća 4 bajta u proširenom ključu.
 - Sada radimo sledeća dva potkoraka ukupno tri puta kako bi kreirali sledećih dvanaest bajtova u proširenom ključu:
 - Prethodna 4 bajta proširenog ključa dodelimo u **t**
 - Izvršimo XOR nad **t** i četvorobajtnim blokom **n**, pre nego što krenemo da računamo novi prošireni ključ. Ovo će postati sledeća 4 bajta u proširenom ključu
- Korak 3 ponavljamo sve dok ne generišemo **b** bajtova u proširenom ključu.

5.5.6 Kod

AES kriptografski algoritam je jedan od kriptografskih blok algoritama koji je u osnovi baziran na Fajstelovo mreži, i izvodi se u više rundi. U svakoj rundi se koristi ključ koji je

izведен iz osnovnog, korišćenjem Rijandel algoritma za distribuciju ključeva. Za rad ovog algoritma, potrebno je da definišemo nekoliko osnovnih operacija.

Listing 5.1 8-bit cirkularna rotacija 32-bitne reči

```
void rotate(unsigned char *in) {
    unsigned char a,c;
    a = in[0];
    for(c=0;c<3;c++)
        in[c] = in[c + 1];
    in[3] = a;
    return;
}
```

Listing 5.2 rcon operacija – eksponent od 2 u Galoaovom polju

```
unsigned char rcon(unsigned char in) {
    unsigned char c=1;
    if(in == 0)
        return 0;
    while(in != 1) {
        unsigned char b;
        b = c & 0x80;
        c <= 1;
        if(b == 0x80) {
            c ^= 0x1b;
        }
        in--;
    }
    return c;
}
```

Listing 5.3 Schedule core operacija

```
void schedule_core(unsigned char *in, unsigned char i) {
char a;
/* Rotate the input 8 bits to the left */
rotate(in);
```

```
/* Apply Rijndael's s-box on all 4 bytes */
for(a = 0; a < 4; a++)
    in[a] = sbox(in[a]);
/* On just the first byte, add 2^i to the byte */
in[0] ^= rcon(i);
}
```

Operacija „schedule core“ predstavlja računanje promenljive **temp** pri ekspanziji ključa, i radi se tako što:

1. poziva se operacija **rotate** nad ulaznim podatkom.
2. za svaki od 4 bajta u reči se radi S-Box operacija.
3. na kraju se prvi bajt XOR-uje sa **rcon(i)**.

Listing 5 Algoritam za ekspanziju ključa

```
key_expansion(byte Key[key_size/8],
              word W[block_size/32*(number_rounds+1)])
{
    for(i = 0; i < keysize/32; i++)
        W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
    for(i = keysize/32; i < block_size/32*(number_rounds+1);
i++)
    {
        temp = W[i - 1];
        if (i % (keysize/32) == 0)
            temp = schedule_core(temp, i / keysize/32);
        W[i] = W[i - keysize/32] ^ temp;
    }
}
```

Kao što je već bilo pomenuto, prvi korak u Rijandel algoritmu za distribuciju ključeva se sastoji u ekspanziji osnovnog (Cipher) ključa, da bi se dobilo (number_rounds+1)*block_size bitova u ključevima runde.

1. Algoritam deli ključ i lokaciju na 32bitne reči.
2. Prvih **key_size/32** reči sadrže sam Cipher key.
3. Ostali deo ključa se dobija generisanjem po **key_size/32** grupa 32bitnih reči.

- a. Kod prve od ovih reči se operacijom schedule_core sa parametrom i koji se pri svakom prolazu inkrementira za 1 (počinje se od 1), nad poslednje generisanom reči, dobija se promenjiva temp.
 - b. Kod svih ostalih, promenljiva temp je vrednost poslednje generisane reči
 - c. key_size/32 reči se generišu tako što se uzima vrednost reči koja je po indeksu za key_size/32 pre reči koju generišemo, i nad njom se radi XOR sa promenjivom temp.
4. Dok god nemamo (number_rounds+1)*block_size bitova, ponavlja se postupak pod 3.

Ključ runde je veličine **block_size** i dobija se tako što se iz ključa dobijenog u prethodnom postupku uzimaju bitovi indeksa **i*block_size** do **(i+1)*block_size-1**.

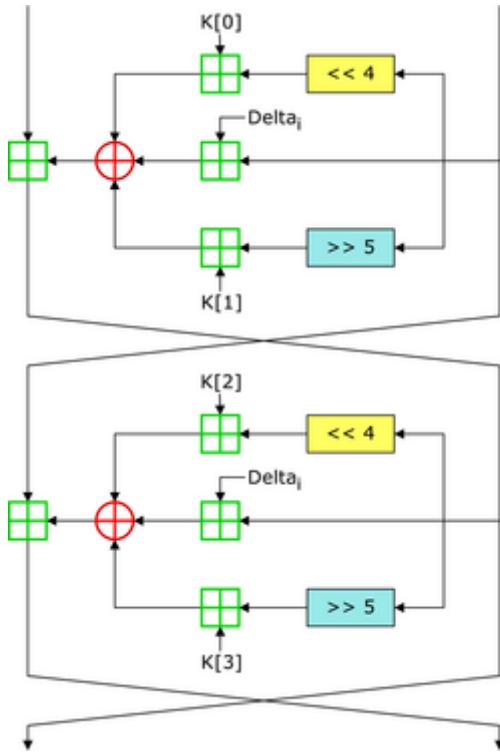
Napomene:

- Ovde je prikazan način ekspanzije ključa za ključeve veličine 128 i 192 bita. Za ključeve veličine 256 bita, operacija se nezatno razlikuje.
- block_size je veličina bloka podataka u bitovima
- key_size je veličina ključa u bitovima
- number_rounds je broj rundi u AES algoritmu.

5.6 TEA algoritam

TEA je skraćenica od **Tiny Encryption Algorithm**. Spada u kodere koji rade sa blokovima podataka. Pridev "tiny" je dobio zbog izuzetno jednostavne definicije i implementacije. Ovaj algoritam su razvili Dejvid Viler (David Wheeler) i Rodžer Nidam (Roger Needham) iz Cambridge Computer Laboratory, i prvi put je predstavljen javnosti 1994. godine, i zbog jednostavnosti su smatrali da ga ne treba patentirati.

TEA radi nad 64-bitnim blokovima i pri tome koristi ključ od 128 bitova. Implementiran je na osnovu Fajstelove mreže. Inicijalno je bilo predviđeno da koristi 64 runde u toku obrade, ali je danas uobičajena implementacija sa 32 runde. Distribucija ključa (key schedule) je ovde izuzetno jednostavna pošto se u svakoj rundi ključ računa istim postupkom, i pri tome ne koristi nikakvu pomoćnu tabelu kao drugi algoritmi koji obrađuju podatke na nivou blokova. Kako bi se izbegli napadi bazirani na simetriji algoritma, TEA uvodi u algoritam i takozvanu magičnu delta vrednost koja ima heksadecimalnu vrednost 9E3779B9 i izabrana je tako da deli interval od 0 do 2^{32} po zlatnom preseku.



Slika 5.11 Dve Fajstelove runde u TEA algoritmu

Veliki nedostatak TEA algoritma je suviše jednostavna metoda za distribuciju ključa. Ovo je imalo za posledicu da je Majkrosoftova X-Box igračka konzola izuzetno brzo dekodovana, pošto je ona koristila TEA kao heš algoritam. Nakon toga razvijeno je nekoliko novijih i sigurnijih verzija algoritma.

Međutim, iako ne preterano bezbedan, TEA se, sa svojim modifikacijama, i dalje učestalo koristi u svim sistemima koji imaju malo RAM memorije i koji treba da obezbede kodiranje u realnom vremenu.

Listing 5.5 Kod za TEA algoritam

```
// metoda za kriptovanje jednog bloka podataka
// v predstavlja ulazni blok od 2 reči
// k predstavlja ključ od 4 reči
void encrypt (unsigned long* v, unsigned long* k)
{
    // ulazna reč se podeli na svoja dva dela (levi v0 i desni
    v1)
    // vrednost pomoćne promenljive sum se postavi na 0
    unsigned long v0=v[0], v1=v[1], sum=0, i; /* set up
*/
    // magična delta konstanta
```

```

unsigned long delta=0x9e3779b9;      /* a key schedule
constant */

// ključ se podeli na svoje četiri nezavisne reči
unsigned long k0=k[0], k1=k[1], k2=k[2], k3=k[3]; // cache
key
for (i=0; i < 32; i++) { // algoritam ima 32 runde
    // algoritam kodiranja se sastoji samo od sledeća tri
    reda
    sum += delta;
    v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
    v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
}
v[0]=v0; v[1]=v1;
}// kriptovani podatak se opet nalazi u v parametru funkcije

// metoda za dekriptovanje jednog bloka podataka
void decrypt (unsigned long* v, unsigned long* k) {
    // inicijalizacija, ovde se može napisati i sum = delta << 5
    unsigned long v0=v[0], v1=v[1], sum=0xC6EF3720, i; /* set up
*/
    unsigned long delta=0x9e3779b9; /* a key schedule constant */
    unsigned long k0=k[0], k1=k[1], k2=k[2], k3=k[3];
    for (i=0; i<32; i++) {
        // proces dekodiranja u okviru jedne runde se sastoji
        // samo od sledeća tri reda
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum -= delta;
    }
    v[0]=v0; v[1]=v1;
}

```

TEA (Tiny Encryption Algorithm) algoritam razvijen je 1994. godine. Međutim, relativno brzo nakon toga otkriveni su neki nedostaci, odnosno slabosti. Otkrio ih je Dejvid Vagner (*David Wagner*), a one su: postojanje ekvivalentnih ključeva: svaki ključ ekvivalentan je sa još tri ključa, što čini efektivnu dužinu ključa 126 bita. Na osnovu ovog nedostatka izvršen je napad prilikom koga je ishakovana Xbox konzola kompanije Microsoft, koja je koristila TEA algoritam za zaštitu.

5.7 Unapređenja TEA algoritma

XTEA (eXtended Tiny Encryption Algorithm) algoritam projektovan je sa namerom da nadomesti određene primenjene slabosti TEA algoritma. Ovaj algoritam projektovali su Dejvid Viler (*David Wheeler*) i Rodžer Nidhem (*Roger Needham*) iz Računarske laboratorije u Kembbridžu (*Cambridge Computer Laboratory*), a prvi put je predstavljen u neobjavljenom tehničkom izveštaju iz 1997. godine. XTEA algoritam nije zaštićen patentom, usled čega je moguće koristiti ga za najrazličitije namene, bez potrebe plaćanja nadoknade ili traženja posebnih dozvola.

Poput TEA algoritma, XTEA predstavlja Feistelovu mrežu 64-bitnih blokova sa 128-bitnim ključem, a predložen broj rundi je 64. Lako je uočiti nekoliko razlika u odnosu na

osnovni TEA algoritam, između ostalog i drugačiji raspored šift, XOR i elemenata za sabiranje. Kako bi se nadomestile pomenute slabosti TEA algoritma, uvedene su i promene u načinu dodeljivanja podključa (*key schedule*), a i samo uvođenje ključa je sporije.

U originalnom izveštaju je, uz XTEA algoritam, bio predstavljen i algoritam sa blokovima promenljive dužine, nazvan *Block TEA*, koji koristi funkciju XTEA runde, ali je primenjuje ciklično na celoj poruci u nekoliko iteracija. Napad na *Block TEA* opisan je već 1998. godine (*Saarinen*), a ujedno su uočeni i nedostaci naslednika tog algoritma – XXTEA.

Od 2004. godine, najbolji zabeleženi napad na XTEA algoritam je diferencijalni *relatedkey* napad na 26 od 64 rundi XTEA algoritma. Veoma mali memorijski prostor koji XTEA algoritam zahteva čini da ovaj algoritam bude veoma dobra opcija za uređaje sa većim hardverskim ograničenjima.

Kao što je već navedeno **TEA** radi nad 64-bitnim blokovima i pri tome koristi ključ od 128 bitova. Implementiran je na osnovu Fajstelove mreže. Inicijalno je bilo predviđeno da koristi 64 runde u toku obrade, koje se danas implementiraju sa 32 ciklusa (jedan ciklus pokriva dve uparene uzastopne runde). Distribucija ključa (*key schedule*) je ovde izuzetno jednostavna pošto se u svakoj rundi ključ računa istim postupkom, i pri tome ne koristi nikakvu pomoćnu tabelu kao drugi algoritmi koji obrađuju podatke na nivou blokova. Kako bi se izbegli napadi bazirani na simetriji algoritma, TEA uvodi u algoritam i takozvanu magičnu delta vrednost koja ima heksadecimalnu vrednost 9E3779B9 i izabrana je tako da deli interval od 0 do 2^{32} po zlatnom preseku.

XTEA je blok koder dizajniran da ispravi slabosti TEA kodera. Poput njegovog prethodnika, i XTEA predstavlja Feistelovu mrežu 64-bitnih blokova sa 128-bitnim ključem, a predložen broj rundi je 64. Lako je uočiti nekoliko razlika u odnosu na osnovni TEA algoritam, između ostalog i drugačiji raspored šift, XOR i elemenata za sabiranje. Kako bi se nadomestile pomenute slabosti TEA algoritma, uvedene su i promene u načinu dodeljivanja podključa (*key schedule*), a i samo uvođenje ključa je sporije.

Block TEA algoritam je predstavljen u originalnom izveštaju uz XTEA algoritam. On ima blokove promenljive dužine i koristi funkciju XTEA runde, ali je primenjuje ciklično na celoj poruci u nekoliko iteracija. To znači da se različite faze kriptovnja obavljaju istovremeno što rezultira potencijlnom uštemom vremena i prirodnijem povezivanju kompletног bloka od n reči. Algoritam za dodelu kluča je takođe malo promenjen, ali su karakteristike ostale iste. Operacija koju izvodimo nad rečima bloka se može opisati preko formule:

$$v[n] += \text{mix}(v[n-1], \text{key}, \text{delta})$$

Napad na *Block TEA* opisan je već 1998. godine (*Saarinen*), a ujedno su uočeni i nedostaci naslednika tog algoritma – XXTEA. Naime, usled nekompletnosti round funkcije, dva velika sifirana teksta od 53 ili više 32-bitnih reči, koji se razlikuju u najmanje 12 reči dok su kod ostalih identični, se mogu dešifrovati. Dovoljno je upotrebiti *brute-force collision* pretraživanje koje zahteva 2^{96-N} memorije, 2^n vremena. Ukupna kompleksnost vreme*memorija je 2^{96} , što je ustvari $2^{\text{wordsize} * \text{fulcycles}/2}$ koa i za bilo koji koder tog tipa. Trenutno nije poznato da li ova činjenica predstavlja opasnost po sigurnost XXTEA kodera. Ono što je poznato je da bi 8 punih ciklusa povećalo složenost razbijanja ovog kodera iznad one koja je potrebna ako se koristi paralelni *brute-force* napad.

5.7.1 Implementacija Blok TEA algoritma

Listing 5.6 Block TEA algoritam - implementacija

```
#define MX (z>>5^y<<2) + (y>>3^z<<4)^sum^y + (k[p&3^e]^z);

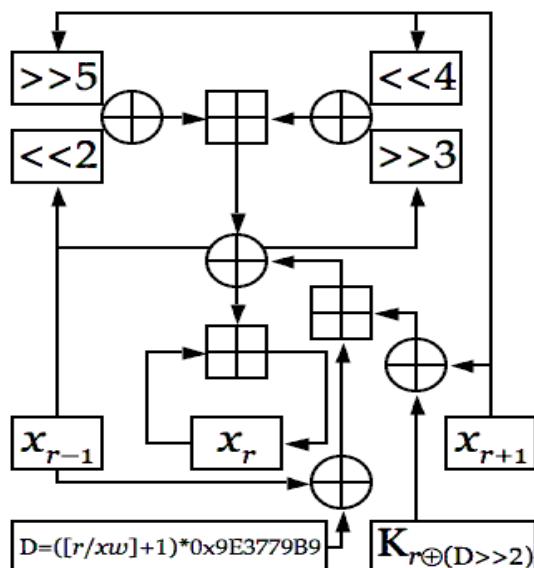
long btea(long* v, long n, long* k) {
    unsigned long z=v[n-1], y=v[0], sum=0, e, DELTA=0x9e3779b9;
    long p, q ;
    if (n > 1) { /* Deo za kodiranje */
        q = 6 + 52/n;
        while (q-- > 0) {
            sum += DELTA;
            e = (sum >> 2) & 3;
            for (p=0; p<n-1; p++) y = v[p+1], z = v[p] += MX;
            y = v[0];
            z = v[n-1] += MX;
        }
        return 0 ;
    } else if (n < -1) { /* Deo za dekodiranje */
        n = -n;
        q = 6 + 52/n;
        sum = q*DELTA ;
        while (sum != 0) {
            e = (sum >> 2) & 3;
            for (p=n-1; p>0; p--) z = v[p-1], y = v[p] -= MX;
            z = v[n-1];
            y = v[0] -= MX;
            sum -= DELTA;
        }
        return 0;
    }
    return 1;
}
```

Objašnjenje navedenog koda:

- Koder enkodira ili dekodira n reči kao jedan blok za svako $n > 1$
- Niz reči koje treba kodirati je predstavljen simbolom v
- Ukoliko želimo da dekodiramo n će imati negativnu vrednost
- Ako je n nula, funkcija će vratiti vrednost 1 i neće se izvršiti nikakvo kodiranje ili dekodiranje, u suprotnom funkcija vraća vrednost 0
- Prepostavljamo da kompjuteri koji dekodiraju i kodiraju podatke rade sa 32-bitnim podacima i na isti način predstavljaju podatke (little i big endian notacija)

5.8 Opis XXTEA kodera

XXTEA je blok koder koji je u suštini nedovršena, heterogena, nebalansirana Fajstelova mreža*. Može da radi sa blokovima različitih dužina, ali ti blokovi moraju biti celobrojni umnošci 32 bita (minimum je 64). Originalni Blok TEA algoritma primenjuje XTEA funkciju runde na svaku reč u bloku i kombinuje je (sabira je) sa susedom koji se nalazi najviše levo.



Slika 5.12 Mreža XXTEA algoritma

Ovaj korak se ponavlja u nekoliko rundi, u zavisnosti od veličine bloka, ali mora se izvršiti najmanje 6 puta (kod implementacije se zbog toga javlja uslov $q=6+52/n$). Prednost ovakvog pristupa je što eliminiše potrebu za različitim modovima operacije (CBC, OFB, CFB itd.) i koder se može primeniti direktno na poruku. Takođe ovaj koder će biti efikasniji kod dužih poruka od XTEA algoritma.

Spori difuzioni stepen procesa dekriptovanja je odmah eksplorisan u cilju probijanja šifre. Ispravljeni Blok TEA algoritam koristi efikasniju round funkciju jer ona koristi oba najbliža suseda prilikom procesuiranja svake od reči u bloku i može se zapisati u obliku:

$$v[m] += f(v[m-1], v[m+1], \text{key}, \text{delta})$$

Ako je veličina bloka jednaka veličini cele poruke, XXTEA je tako osmišljen da se proces sifriranja može direktno primeniti za enkriptovanje cele poruke. XXTEA ima tendenciju da bude efikasnija za duže poruke od XTEA algoritma.

Za ispravno korišćenje i opštu sigurnost bolje je koristiti velike blokove sa više reči pri kodiranju zbog sledećih razloga:

- Ako se na ulazu promeni jedan bit poruke, na izlazu će se promeniti skoro polovina bitova celog tog bloka, pritom je nemoguće odrediti gde se promena prvo desila, odnosno koji je bit prvi promenjen.
- Napad deljenjem i spajanjem su se pokazali kao nemogući

Z A Š T I T A I N F O R M A C I J A

- Ako nije prihvatljivo imati velike poruke, one se mogu podeliti u parčiće od po 60 reči i ulančati na isti način kao što se to radi kod DES algoritma

Neobično mala veličina ovog algoritma čini ga praktičnim i jako dobrom rešenjem kod situacija u kojima postoje ekstremna ograničenja, npr. starijim (integrisanim ili običnim) hardverskim sistemima (*legacy perhaps embedded hardware systems*) gde je količina raspoložive RAM memorije minimalna.

6 ASIMETRIČNO KRIPTOVANJE

6.1 RSA

RSA pripada grupi algoritama koji kriptuju pomoću javnog i privatnog ključa. On je prvi algoritam koji je bio koristan ne samo za kriptovanje, već i za digitalni potpis. RSA je široko korišćen u protokolima za elektronsku trgovinu, jer je zahvaljujući korišćenju dugih ključeva i unapređenjima u implementaciji [17],[18].

6.1.1 Generisanje ključeva

Kao što je poznato, javni ključ je poznat svima i koristi se za kriptovanje poruka. Poruke kriptovane javnim ključem mogu se dekriptovati samo pomoću privatnog ključa. Ključevi za ovaj algoritam se generišu na sledeći način:

1. Izaberu se dva različita velika prosta broja p i q .
2. Iračuna se $n = p * q$. Izračunati n ćemo koristiti kao moduo i kod kriptovanja i kod dekriptovanja.
3. Izračuna se Ojlerova ϕ funkcija od n , $\phi(n) = (p - 1)(q - 1)$.
4. Izabere se vrednost e tako da je $1 < e < \phi(n)$, gde su e i $\phi(n)$ uzajamno prosti.
Izabrano e , zajedno sa n , predstavlja javni ključ.
5. Izabere se tako da zadovoljava uslov $d * e = 1 \pmod{\phi(n)}$. Dobijeno d , zajedno sa n , predstavlja privatni ključ.

Veoma čest izbor za vrednost e je broj $2^{16} + 1 = 65537$, jel je potvrđen kao dovoljno siguran. Međutim, i danas se u pojedinim aplikacijama koriste i male vrednosti poput 3, 5 ili 35 kod uređaja koji nemaju visoke performanse. Takvi uređaji su npr. čitači smart ili kreditnih kartica. Ovde treba imati u vidu da je korišćenje manjih eksponenata veći bezbednosni rizik, jel se samo kriptovanje i dekriptovanje izvršava u manje koraka.

6.1.2 Kodovanje

- Ulazni podaci (podaci koje treba kodovati) se prevedu u niz brojeva
- Kodovanje se radi za svaki elemenat ulaznog niza
- Obeležimo podatak koji kodujemo sa M , a rezultat koji će se dobiti kodovanjem sa C
- izračunamo C na sledeći način $C = M^e \pmod{n}$
- sačuvamo C u izlazni niz brojeva

6.1.3 Dekodovanje

- Ulazni podaci (podaci koje treba dekodovati) se prevedu u niz brojeva (ako već nisu u takvom formatu)
- Dekodovanje se radi za svaki elemenat ulaznog niza
- Obeležimo podatak koji dekodujemo sa C , a rezultat koji će se dobiti dekodovanjem sa M
- izračunamo M na sledeći način $M = C^d \text{ mod } n$
- sačuvamo M u izlazni niz brojeva

6.2 Knapsack

Kanpsack algoritam je nastao „slučajno“ tj. nastao je kao posledica rešavanja takozvanog „problema pakovanja“ (http://en.wikipedia.org/wiki/Knapsack_problem). Knapsack pripada grupi algoritama koji kriptuju pomoću javnog i privatnog ključa [19].

6.2.1 Generisanje ključeva

- Odlučimo kakve podatke želimo da kriptujemo (bajtove, polureči, reči i sl)
- Izaberimo jedan superrastući niz (superincreasing) niz koji ima onoliko elemenata koliko bitova imaju podaci koje želimo da kriptujemo, i obeležimo ga sa P . Superrastući je onaj niz kod koga važi pravilo da je svaki njegov elemenat veći od zbiru svih svojih prethodnika.
- Izaberemo jedan broj koji je logički nastavak izabranog niza (tj. veći od zbiru svih njegovih članova) i obeležimo ga sa n . Izaberemo još jedan broj (multiplikator, obeležimo ga sa m) koji je uzajamno prost sa n . Izračunamo članove još jednog niza koji ćemo obeležiti sa J , na sledeći način $J[i] = (P[i] * m) \text{ mod } n$. Dobijeni niz biće opšti.
- Javni ključ je opšti niz J
- Izračunamo inverznu vrednost broju m u odnosu na moduo n i označimo je sa im ili m^{-1} , tj treba da važi: $m^{-1} * m = 1 \pmod{n}$. Privatni ključ čine niz P , kao i vrednosti m^{-1} i n .

Primer:

- Izaberemo bajtove, bajtovi imaju 8 bitova
- $P = (2, 3, 7, 14, 30, 57, 120, 251)$
- $m = 41, n = 491$
 $J[0] = 2 * 41 = 82 \pmod{491}$
 $J[1] = 3 * 41 = 123 \pmod{491}$
 $J[2] = 7 * 41 = 287 \pmod{491}$
 $J[3] = 14 * 41 = 83 \pmod{491}$
 $J[4] = 30 * 41 = 248 \pmod{491}$
 $J[5] = 57 * 41 = 373 \pmod{491}$
 $J[6] = 120 * 41 = 10 \pmod{491}$

$$J[7] = 251 * 41 = 471 \bmod 491$$

$$J = (82, 123, 287, 83, 248, 373, 10, 471)$$

- Javni ključ je niz J
- $m^{-1} = 12$, jer je $12 * 41 = 492 = 1 \pmod{491}$
- Privatni ključ čine niz P , $m^{-1} = 12$ i $n = 491$

6.2.2 Kriptovanje

Uzmimo bajt koji želimo da kriptujemo i označimo ga sa M . Kriptovanje se obavlja pomoću javnog ključa. Neka bajt ima vrednost 150, dakle $M = 150$.

Prevedimo vrednost bajta iz dekadnog u binarni sistem, dakle $M = 150_{10} = 10010110_2$. Ako M u binarnoj reprezentaciji ima manje od 8 cifara, dopunimo ga nulama sa desne strane, i predstavimo binarnu reprezentaciju broja M nizom bitova, počev od bita najveće težine, pa imamo $B = (1, 0, 0, 1, 0, 1, 1, 0)$. Uzmimo za javni ključ vektor J koji smo odredili u prethodnom primeru, dakle $J = (82, 123, 287, 83, 248, 373, 10, 471)$. Kako B i J imaju isti broj elemenata, kriptovanje izvršimo tako što izračunamo sumu svih proizvoda $B(i) \cdot J(i)$ i označimo je sa C . Dakle, u našem slučaju:

Listing 6.1 Algoritam za kriptovanje

```
int C = 0;
for (int i = 0; i < 8; i++)
{
    C += J[i] * B[i];
}
```

Odnosno $C = 1 * 82 + 0 * 123 + 0 * 287 + 1 * 83 + 0 * 248 + 1 * 373 + 1 * 10 + 0 * 471 = 548$. Znači, rezultat kriptovanja broja $M = 150$, po ključu J je $C = 548$. Ovde treba uočiti da rezultat kriptovanja vrednosti koja je bajt, može da bude vrednost veća od one koja može da se smesti u fizičku osmobiltnu lokaciju.

6.2.3 Dekriptovanje

Kod dekriptovanja dobijemo kriptovani broj C , i iz njega treba da ekstrakujemo osnovnu informaciju M . Sekvenca koraka u dekriptovanju je sledeća:

- Uzmemo C , u našem primeru $C = 548$.
- Izračunamo transformisano C (TC) po formuli $TC = C * m^{-1} \bmod n = 548 * 12 \bmod 491 = 193$
- TC predstavimo kao zbir elemenata iz skupa P . Tako je kod nas $193 = 2 * 1 + 3 * 0 + 7 * 0 + 14 * 1 + 30 * 0 + 57 * 1 + 120 * 1 + 251 * 0$. Dobijene faktore, kojima smo množili elemente niza P , predstavimo kao binarni broj: 10010110_2 . Prevedemo dobijeni binarni broj u dekadni sistem i dobijemo M . Tj. $M = 10010110_2 = 150_{10}$

7 KRIPTOGRAFSKI HEŠEVI

7.1 Primer za CRC

Prepostavimo da nam treba da izračunamo osmobitni CRC, osmobitne poruke (realniji slučaj je da nam treba da izračunamo osmobitni CRC niza osmobitnih poruka). Uzmimo kao primer poruke čiji CRC hoćemo da računamo ASCII karakter „W“. Njegova decimalna reprezentacija je 87_{10} a heksadecimalna 57_{16} . Za ilustraciju računanja uzećemo jedan od poznatih CRC polinoma koji se zove CRC-8-ATM i koji služi da detektuje greške u zaglavljima poruka prilikom asinhronog prenosa. Taj polinom se nad Galoaovim poljem izgleda ovako: $x^8 + x^2 + x + 1$. Ovaj polinom možemo da transformišemo u njegovu devetobitnu binarnu reprezentaciju kao: „100000111“. Kao što se može uočiti prvi monom odgovara prvoj jedinici (monom sa stepenom 8). Druga jedinica odgovara monomu sa stepenom 2, treća onom koji ima stepen 1, a četvrta linearnom članu. Svi nedostajući stepeni promenljive x (7, 6, 5, 4 i 3) su kodovani nulama [20],[21].

Najpoznatiji CRC polinomi, kao i njihove heksa reprezentacije date su u sledećoj tabeli:

Ime	Polinomi	Brojčanana reprezentacija (normalna ili inverzna)
CRC-1	$x + 1$ (poznat kao bit parnosti)	0x1 ili 0x1 (0x1)
CRC-4-ITU	$x^4 + x + 1$	0x3 ili 0xC (0x9)
CRC-5-ITU	$x^5 + x^4 + x^2 + 1$	0x15 ili 0x15 (0x0B)
CRC-5-USB	$x^5 + x^2 + 1$ (koristi se za verifikaciju paketa kod USB prenosa)	0x05 ili 0x14 (0x9)
CRC-6-ITU	$x^6 + x + 1$	0x03 ili 0x30 (0x21)
CRC-7	$x^7 + x^3 + 1$ (koriste ga digitalne telefonske centrale)	0x09 ili 0x48 (0x11)
CRC-8-ATM	$x^8 + x^2 + x + 1$	0x07 ili 0xE0 (0xC1)
CRC-8-CCITT	$x^8 + x^7 + x^3 + x^2 + 1$ (koriste ga jednobitne magistrale)	0x8D ili 0xB1 (0x63)

Z A Š T I T A I N F O R M A C I J A

CRC-8-Dallas/Maxim	$x^8 + x^5 + x^4 + 1$ (koriste ga jednabitne magistrale)	0x31 ili 0x8C (0x19)
CRC-8	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$	0xD5 ili 0xAB (0x57)
CRC-8-SAE J1850	$x^8 + x^4 + x^3 + x^2 + 1$	0x1D ili 0xB8
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x + 1$	0x233 ili 0x331 (0x263)
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$ (koriste ga digitalne telefonske centrale)	0x80F ili 0xF01 (0xE03)
CRC-15-CAN	$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$	0x4599 ili 0x4CD1 (0x19A3)
CRC-16-Fletcher	Nije CRC; vidite Fletcher's checksum	Koristi se u Adler-32 A & B CRCs
CRC-16-CCITT	$x^{16} + x^{12} + x^5 + 1$ (Koristi se u sledećim protokolima: X.25, V.41, Bluetooth, PPP, IrDA, BACnet; known as "CRC-CCITT")	0x1021 ili 0x8408 (0x0811)
CRC-16-IBM	$x^{16} + x^{15} + x^2 + 1$	0x8005 ili 0xA001 (0x4003)
CRC-24-Radix-64	$x^{24} + x^{23} + x^{18} + x^{17} + x^{14} + x^{11} + x^{10} + x^7 + x^6 + x^5 + x^4 + x^3 + x + 1$	0x864CFB ili 0xDF3261 (0xBE64C3)
CRC-32-Adler	Nije CRC; vidite Adler-32	Vidite Adler-32
CRC-32-MPEG2	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	0x04C11DB7 ili 0xEDB88320 (0xDB710641)
CRC-32-IEEE 802.3	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	0x04C11DB7 ili 0xEDB88320 (0xDB710641)
CRC-32C (Castagnoli)	$x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^6 + 1$	0x1EDC6F41 ili 0x82F63B78 (0x05EC76F1)
CRC-32K (Koopman)	$x^{32} + x^{30} + x^{29} + x^{28} + x^{26} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{11} + x^{10} + x^7 + x^6 + x^4 + x^2 + x + 1$	0x741B8CD7 ili 0xEB31D82E
CRC-64-ISO	$x^{64} + x^4 + x^3 + x + 1$ (use: ISO 3309)	0x0000000000000001B ili 0xD800000000000000 (0xB000000000000001)

CRC-64-	$x^{64} + x^{62} + x^{57} + x^{55} + x^{54} + x^{53} + x^{52} +$
	$x^{47} + x^{46} + x^{45} + x^{40} + x^{39} + x^{38} + x^{37} + \quad 0x42F0E1EBA9EA3693$ ili
ECMA-182	$x^{35} + x^{33} + x^{32} + x^{31} + x^{29} + x^{27} + x^{24} + \quad 0xC96C5795D7870F42$
	$x^{23} + x^{22} + x^{21} + x^{19} + x^{17} + x^{13} + x^{12} + \quad (0x92D8AF2BAF0E1E85)$
	$x^{10} + x^9 + x^7 + x^4 + x + 1$

7.1.1 Računanje CRC koda

Vrednost bajta 57_{16} se može preneti na dva različita načina, počev od bita najveće težine ili počev od bita najmanje težine. Kao posledicu ovoga, u oba slučaja ćemo imati različite polinomne predstave poruke koju želimo da kodujemo. U slučaju „najpre bit najveće težine“ imamo $M(x) = x^6 + x^4 + x^2 + x + 1 = 01010111$, dok je u slučaju „najpre bit najmanje težine“ $M(x) = x^7 + x^6 + x^5 + x^3 + x = 11101010$.

Nakon što i poruku i CRC polinom prevedemo u binarni zapis, poruku moramo da prepravimo kako bi računali CRC. To radimo tako što na nju sa desne strane nalepimo onoliko nula koliko je najveći stepen u CRC polinomu. U polinomnoj reprezentaciji, ovo odgovara množenju monoma sa najvećim stepenom iz CRC polinoma sa polinomom poruke $M(x)$. U našem konkretnom primeru, pomnožićemo polinom $M(x)$ sa x^8 da bi dobili $x^8M(x)$, odnosno početni podatak proširen sa osam pratećih nula.

U sledećoj tabeli prikazan je postupak računanja CRC koda. CRC kod je zapravo ostatak pri deljenju polinoma $M(X)$ sa polinomom koji predstavlja CRC. Znači računanje CRC koda je kao računanje ostatka pri deljenju, s tim što je ovo deljenje polinoma, pa je čak i jednostavnije od deljenja običnih brojeva.

Pravilo je sledeće:

Pozicionirajmo se najpre skroz levo i izdvojimo onoliko bitova iz modifikovane poruke, koliko nam je potrebno za računanje (nama treba 9, ali je ovde „zaplavljen“ 8 kako bi se pratila promena rezultata).

- Ako izdvojena grupa počinje nulom, pomerimo se jedno mesto u desno i ne radimo ništa.
- Ako izdvojena grupa počinje jedinicicom, napišemo ispod nje binarnu reprezentaciju CRC polinoma i uradimo XOR operaciju nadbitovima izdvojene grupe i CRC polinoma. Pomerimo se za jedno mesto u desno.

U desno se pomeramo sve dok nam se krajnji desni član izdvojene grupe ne poklopi sa krajnjim desnim članom reči za koju tražimo CRC.

Most significant bit first	Least-significant bit first
	
-000000000	-100000111
$=0101011100000000$	$=0110100110000000$
-100000111	-100000111
$=0001011011000000$	$=0010100001000000$
-000000000	-100000011
$=0001011011000000$	$=0000100010100000$
-100000111	-000000000
$=0000011010110000$	$=0000100010100000$
-000000000	-100000111
$=0000011010110000$	$=0000000010011000$
-100000111	-000000000
$=0000001010101100$	$=0000000001001100$
-100000111	-000000000
$=0000000101000100$	$=0000000001001100$
-000000000	-000000000
$=0000000010100010$	$=0000000001001100$

U slučaju „najpre bit najveće težine“ rezultat je polinom $x^7 + x^5 + x$. Ako ga konvertujemo u heksa broj to je $A2_{16}$. U drugom slučaju, „najpre bit najmanje težine“ dobijeni polinom je $x^7 + x^4 + x^3$. Ako ga konvrtujemo u heksa gledajući bitove s leva u desno dobćemo heksa vrednost 98, a ako ga pročitamo s desna u levo rezulat je 19_{16} . Oba načina čitanja su korektna, dok se drugi način smatra ispravnijim, zato što je i početni podatak zapisan kao „najpre bit najmanje težine“.

U slučaju kada treba odrediti CRC kod za niz ulaznih podataka, koristi se sledeća procedura:

zbirniCRCKod = 0

Za svaki ulazni podatak:

zbirniCRCKod = zbirniCRCKod XOR CRCKod(ulaznog podatka)

7.2 Tiger Hash

Opis za Tiger hash možete naći u knjizi. Ovde će biti predstavljen pseudokod za Tiger hash algoritam sa komentarima, a u drugom fajlu moći ćete da nadjete S – box matrice:

Listing 7.1 Tiger hash - pseudokod

```
// Napomena: konkatenacija je obeležena rečju append
// Promenljiva koja čuva poruku obeležena je sa message
// h0, h1 i h2 su inicijalni a, b i c respektivno.

// Inicijalizacija promenljivih:
var int h0 := 0x0123456789ABCDEF
var int h1 := 0xFEDCBA9876543210
var int h2 := 0xF096A5B4C3D2E187

// Preprocesiranje. Svodjenje poruke na dužinu koja je umnožak od 512 // (u bitovima). Može se uraditi bilo kako, ali dva najčešća načina su // dodavanje potrebnog broja nula na kraju ili sledeće:
-append "1" bit to message
-append "0" bits until message length in bits ≡ 448 (mod 512)
-append bit (bit, not byte) length of unpadded message as 64-bit little-endian integer to message

// Procesirati poruku (message) kao niz 512-bitnih segmenata:
for each 512-bit chunk of message

    podeliti segment na 8 64-bitnih big-endian reči w[i], 0 ≤ i ≤ 7

// Inicijalizovati heš vrednosti za jedan segment:
var int a := h0
```

```
var int b := h1
var int c := h2

//Glavna petlja:
for i from 0 to 3

    // Za svaku rundu u okviru grupe
    for j from 0 to 7

        c = c XOR w[j]
        // c (64-bitna reč) se podeli na 8 bajtova: c0, c1, ..., c7
        // S0, S1, S2 i S3 su S-boks nizovi
        a = a - (S0[c0] XOR S1[c2] XOR S2[c4] XOR S3[c6])
        b = b + (S3[c1] XOR S2[c3] XOR S1[c5] XOR S3[c7])
        b = b * (i + 1)
    next

    // key schedule
    if i = 0 then
        w[0] = w[0] - (w[7] XOR 0xA5A5A5A5A5A5A5A5)
        w[1] = w[1] XOR w[0]
        w[2] = w[2] + w[1]
        w[3] = w[3] - (w[2] XOR ((w[1] XOR 0xFFFFFFFFFFFFFF)<< 19))
        w[4] = w[4] XOR w[3]
        w[5] = w[5] + w[4]
        w[6] = w[6] - (w[5] XOR ((w[4] XOR 0xFFFFFFFFFFFFFF)>> 23))
        w[7] = w[7] XOR w[6]
    else if i = 1 then
        w[0] = w[0] + w[7]
        w[1] = w[1] - (w[0] XOR ((w[0] XOR 0xFFFFFFFFFFFFFF)<< 19))
        w[2] = w[2] XOR w[1]
        w[3] = w[3] + w[2]
```

```

w[4] = w[4] - (w[3] XOR ((w[2] XOR 0xFFFFFFFFFFFFFF) >> 23))
w[5] = w[5] XOR w[4]
w[6] = w[6] + w[5]
w[7] = w[7] - (w[6] XOR =0x0123456789ABCDEF)
end if

next

// Doda se rezultat kodovanja jednog bloka na ukupni rezultat:

h0 := h0 + a
h1 := h1 + b
h2 := h2 + c
next

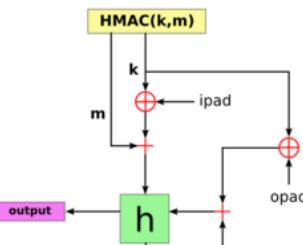
// definiše se krajnji rezultat kodovanja
var int digest := h0 append h1 append h2

```

7.3 HMAC

U kriptografiji, heš-ključ kod za autentifikaciju poruke (engl. keyed-Hash Message Authentication Code ili HMAC), predstavlja jedan od načina kodovanja koji se koriste za proces u kome se ustanavljava da li je dobijena poruka autentična (engl. message authentication code – MAC). On se dobija korišćenjem kriptografskih heš funkcija u kombinaciji sa tajnim ključem. Kao bilo koji drugi MAC, i ovaj se može koristiti da istovremeno potvrdi i integritet podataka i autentičnost poruke. Bilo koja iterativna kriptografska heš funkcija, kao što je Tiger Hash, MD5 ili SHA-1, može se koristiti u proračunu HMAC-a. U tom slučaju rezultujući MAC algoritam nazivamo HMAC-MD5 ili HMAC-SHA-1, respektivno. Kriptografska snaga HMAC-a zavisi od kriptografske snage heš funkcija koje čine njegovu osnovu, od veličine i kvaliteta ključa kao i od dužine heš izlaza u bitovima.

Iterativna heš funkcija razbija poruku na blokove fiksne veličine i vrši nad njima odgovarajuću transformaciju pomoću funkcije za kompresiju. Npr, MD5 i SHA-1 rade sa 512-bitnim blokovima. Veličina izlaza HMAC-a je ista ista kao i kod heš funkcija koje ga čine (128 ili 160 bitova u slučaju MD5 i SHA-1), iako može biti i skraćen po potrebi. Skraćivanje heš lika smanjuje sigurnost MAC-a koji je onda pogodan za ono što se zove „birthday attack“.



Slika 7.1 HMAC - blok šema

Konstrukciju i analizu HMAC-a prvi put su objavili 1996 Mihir Bellare, Ran Canetti i Hugo Krawczyk, koji je napisao RFC 2104. FIPS PUB 198 generalizuje i standardizuje korišćenje HMAC-a. HMAC-SHA-1 i HMAC-MD5 koričeni su u okviru IPsec-a i TLS protokola.

Na slici je data šema koja opisuje HMAC. Simbolom h je označena korišćena kriptografska funkcija, k je tajni ključ dopunjeno nulama do veličine bloka heš funkcije, m je poruka koja se autentificuje, \oplus označava konkatenaciju, \oplus označava eksluzivno ili (XOR), a spoljašnje dopunjavanje $opad=0x5c5c5c...5c5c$ i unutrašnje dopunjavanje $ipad=0x363636...3636$ predstavljaju dve heksadecimalne konstante dužine jednog bloka.

Listing 7.2 HMAC - pseudokod

```

function hmac (key, message)
  // Vrednost blocksize je veličina bloka poruke kod heša
  opad = [0x5c ponavljanje do blocksize]
  ipad = [0x36 ponavljanje do blocksize]

  if (length(key) > blocksize) then
    // ovo skrati ključ ako je veći od blocksize
    // 'hash' je korišćena heš funkcija u ovom postupku
    key = hash(key)
  end if

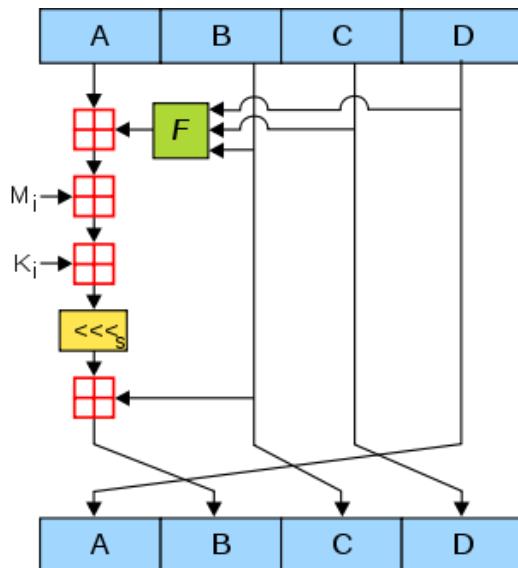
  if (length(key) < blocksize) then
    // dopuni ključ nulama ako je manji od blocksize
    // Simbol || označava konkatenaciju
    key = key || [0x00 * (blocksize - length(key))]
  end if

  ipad = ipad XOR key
  opad = opad XOR key

  return hash(opad || hash(ipad || message))

end function
  
```

7.4 MD5 algoritam



Slika 7.2 MD5 algoritma - struktura jedne runde

Na slici je prikazana jedna MD5 runda, MD5 se sastoji od 64 ovakve runde, grupisane u četiri kruga od po 16 rundi. F je nelinearna funkcija. M_i označavba 32-bitni blok ulaza poruke, a K_i označava 32-bitnu konstantu, različitu za svaku operaciju. Simbol $<<<_s$ označava rotaciju bita u levo za s pozicija, s se menja za svaku operaciju, dok $+$ u kvadratu označava sabiranje po modulu 2^{32} .

MD5 obraduje poruke promenljive dužine u izlaz fiksne dužine od 128 bitova. Ulazna poruka se razbija na 512-bitne blokove (šesnaest 32-bitnih little endian celih brojeva), poruka se proširuje kako bi bila deljiva sa 512. Proširivanje se vrši na sledeći način, prvo jednim bitom, 1, koji se dodaje na kraju poruke. Nakon ovoga dodaje se onoliko nula koliko je potrebno da bi se dužina poruke povećala na 64 bita manje od 512. Preostalih 64 bita se nadopunjuju sa 64-bitnim celim brojem koji predstavlja dužinu originalne poruke, u bitovima.

MD5 algoritam radi sa 128-bitnim vektorom stanja, podeljenim na četiri 32-bitne reči, označene sa A , B , C i D . Ove reči su inicijalizovane sa određenim fiksним konstantama. Glavni algoritam radi pojedinačno sa svakom 512-bitnom blok porukom, svaki blok menja stanje. Obrada bloka poruke sastoji se od četiri slične faze, označene kao runde, svaka runda se sastoji od 16 sličnih operacija zasnovanih na nelinearnoj funkciji F , modularnom sabiranju, i levoj rotaciji. Postoje četiri moguće F funkcije, u svakom krugu se koristi različita funkcija:

$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$

$$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$

$$H(X, Y, Z) = X \oplus Y \oplus Z$$

$$I(X, Y, Z) = Y \oplus (X \vee \neg Z)$$

$\oplus, \wedge, \vee, \neg$ označavaju XOR, AND, OR i NOT operacije, respektivno.

Listing 7.3 MD5 - pseudokod

```
Pseudokod za MD5 algoritam:
var int[64] r, k

// r niz, čuva podatke o tome koliko u kojoj rundi treba šiftovati
// podatke. Ove vrednosti su unapred definisane, dakle poput S-
// box
// nizova.

r[ 0..15] := {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7,
12, 17, 22}
r[16..31] := {5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5,
9, 14, 20}
r[32..47] := {4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4,
11, 16, 23}
r[48..63] := {6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6,
10, 15, 21}

// Inicijalizacija ključeva. U svakoj rundi se koristi p0o jedan:
for i from 0 to 63
    k[i] := floor(abs(sin(i + 1)) × (2 pow 32))
next

// Inicijalne vrednosti za A, B, C i D registre:
var int h0 := 0x67452301
var int h1 := 0xEFCDAB89
var int h2 := 0x98BADCFC
var int h3 := 0x10325476

// Preprocesiranje:
-append "1" bit to message
-append "0" bits until message length in bits ≡ 448 (mod 512)
-append bit (bit, not byte) length of unpadded message as 64-bit
little-endian integer to message

// Poruka se podeli na 512-bitne segmente „chunks“ i svaki se
procesira // posebno:
for each 512-bit chunk of message

// svaki segment se podeli na niz od po 16 little endian reči
```

```

break chunk into sixteen 32-bit little-endian words w[i], 0 ≤ i ≤
15

var int a := h0
var int b := h1
var int c := h2
var int d := h3

//Glavna petlja:
for i from 0 to 63
    if 0 ≤ i ≤ 15 then
        f := (b and c) or ((not b) and d)
        g := i
    else if 16 ≤ i ≤ 31
        f := (d and b) or ((not d) and c)
        g := (5×i + 1) mod 16
    else if 32 ≤ i ≤ 47
        f := b xor c xor d
        g := (3×i + 5) mod 16
    else if 48 ≤ i ≤ 63
        f := c xor (b or (not d))
        g := (7×i) mod 16

    temp := d
    d := c
    c := b
    b := b + leftrotate((a + f + k[i] + w[g]) , r[i])
    a := temp
next

//Dodaj rezultat za jedan segment ukupnom rezultatu:
h0 := h0 + a
h1 := h1 + b
h2 := h2 + c
h3 := h3 + d
end for each

// definisanje izlazne vrednosti
var int digest := h0 append h1 append h2 append h3

//definicija funkcije leftrotate
leftrotate (x, c)
    return (x << c) or (x >> (32-c));

```

Postoji jedno moguće uprošćenje funkcija:
 $(0 \leq i \leq 15) : f := d \text{ xor } (b \text{ and } (c \text{ xor } d))$
 $(16 \leq i \leq 31) : f := c \text{ xor } (d \text{ and } (b \text{ xor } c))$

7.4.1 MD5 heš-ovi

128-bitni (16-bajtova) MD5 heš-ovi (poznati i kao izlazi poruke) obično se predstavljaju kao niz od 32 heksadecimalne cifre. Sledi demonstracija 43-bajtnog ASCII ulaza i odgovarajućeg MD5 heš-a:

MD5("The quick brown fox jumps over the lazy dog")

= 9e107d9d372bb6826bd81d3542a419d6

Čak i najmanja promena u ovoj poruci (sa velikom verovatnoćom) dovešće do sasvim drugačijeg heš-a, zahvaljujući efektu lavine. Npr, promenom d u e:

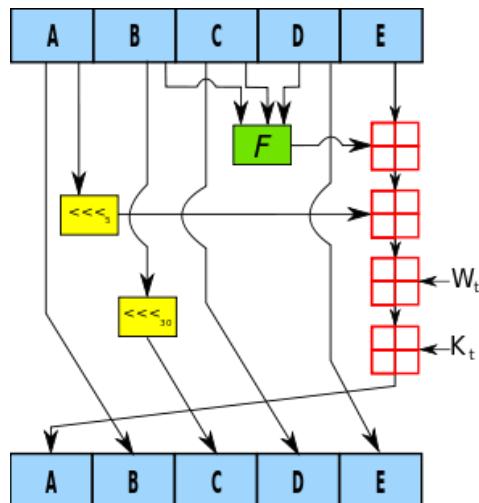
MD5("The quick brown fox jumps over the lazy eog")

= ffd93f16876049265fbaef4da268dd0e

Heš praznog stringa je:

MD5("") = d41d8cd98f00b204e9800998ecf8427e

7.5 SHA-1



Slika 7.3 Šema jedne runde u okviru SHA-1

Na slici je prikazana jedna iteracija u okviru SHA -1 kompresione funkcije. A, B, C, D i E su 32-bitne reči koje definišu tzv. stanje runde, F je nelinearna funkcija koja se menja, <<<_n označava rotaciju levog bita za n pozicija, crveni pravougaonici sa plusom u sredini označavaju sabiranje po modulu 2³². K_t je konstanta ključa koja se koristi u odgovarajućoj

rundi, a W_t su blokovi podataka dobijeni na osnovu ulazne reči (ima ih onoliko koliko ima rundi), a rundi ima 80.

Originalna specifikacija algoritma izdata je 1993 godine u Secure HAsh Standard-u, FIPS PUB 180, od strane agencije za standarde američke vlade – NIST (National Institute of Standards and Technology). Danas se ova verzija često naziva SHA-0. NSA je povukla ovaj naziv odmah nakon njenog nastanka i zamenjen je prepravljenom verzijom, izdatom 1995 u FIPS PUB 180-1 i njen naziv je SHA-1. SHA-1 se razlikuje od SHA-0 samo po jednom smeru rotacije bita u rasporedu poruke kompresovane funkcije, ovo je urađeno, prema NSA, kako bi se ispravio tok u originalnom algoritmu što smanjuje kriptografsku sigurnost.

Međutim, NSA nije pružila dalja objašnjenja niti je pokazala zašto je tok ispravljen. Kasnije su uočavane slabosti i u SHA-0 i u SHA-1. Izgleda da SHA-1 obezbeđuje veću otpornost na napade, opravdavajući NSA-ovu tvrdnju da je izmena povećala sigurnost, mada to nikada zvanično nije potvrđeno.

SHA-1 (kao i SHA-0) pravi 160-bitni izlaz iz poruke sa maksimalnom dužinom od $2^{64} - 1$ bitova, i bazira se na principima sličnim onima koji se koriste kod Ronald L. Rivest u MIT-u u projektovanju MD4 i MD5 algoritama za izlazne poruke [22].

Listing 7.4 SHA-1 - pseudokod

Napomena: Sve promenljive su neoznačeni 32-bitni celi brojevi

Inicijalne promenljive:

```
h0 := 0x67452301
h1 := 0xEFCDAB89
h2 := 0x98BADC9E
h3 := 0x10325476
h4 := 0xC3D2E1F0
```

Preprocesiranje:

- doda se jedan bit '1' na poruku
- doda se k bitova '0', gde je k minimalni broj ≥ 0 takav da dužina rezultujuće poruke daje ostatak 448 pri deljenju sa 512.
- vrednost dužine poruke, pre preprocesiranja, se pretvori u 64-bitni „big-endian“ ceo broj i konkatenira se na rezultat prethodnog koraka

Nakon svega ovoga dužina poruke (u bitovima) je deljiva sa 512. Možemo da kažemo da se poruka sada sastoji iz celog broja, označimo ga sa t , blokova dužine 512 bitova:

```
for s from 0 to t // za svaki 512-bitni blok
```

```
    podelimo blok na 16 32-bitnih reči i obeležimo ih  $w[i]$ ,  $0 \leq i \leq 15$ 
```

Preračunamo ostale elemente niza w po sledećoj formuli:

ZAŠTITA INFORMACIJA

```
for i from 16 to 79
    w[i] := (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16])
leftrotate 1
next

Inicijalizacija tzv. promenljivih stanja za aktuelni blok
a := h0
b := h1
c := h2
d := h3
e := h4

Runde:
for i from 0 to 79
// U svakoj od rundi se izračunava vrednost funkcije f, kao i
// vrednost konstante ključa k, po sledećim formulama
// (zavisno od rednog broja runde)
if 0 ≤ i ≤ 19 then
    f := (b and c) or ((not b) and d)
    k := 0x5A827999
else if 20 ≤ i ≤ 39
    f := b xor c xor d
    k := 0x6ED9EBA1
else if 40 ≤ i ≤ 59
    f := (b and c) or (b and d) or (c and d)
    k := 0x8F1BBCDC
else if 60 ≤ i ≤ 79
    f := b xor c xor d
    k := 0xCA62C1D6

// prearačunavanje promenljivih stanja (vidi sliku)
temp := (a leftrotate 5) + f + e + k + w[i]
e := d
d := c
c := b leftrotate 30
b := a
a := temp
next

// Rezultat dobijen nakon svih rundi za aktuelni blok podataka
se
// dodaje u inicijalizacione promenljive
h0 := h0 + a
```

```

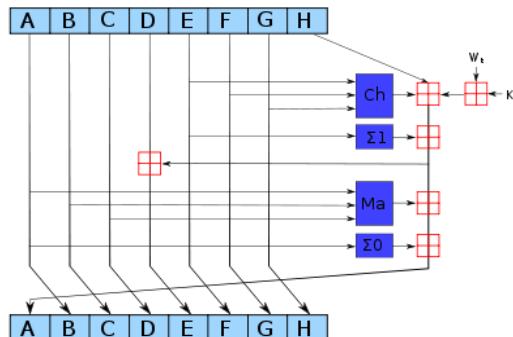
h1 := h1 + b
h2 := h2 + c
h3 := h3 + d
h4 := h4 + e
next

// Na kraju se sve inicijalizacione promenljive stapaju u jednu
// vrednost koja predstavlja rezultat računanja heš-a
hash = h0 append h1 append h2 append h3 append h4

```

7.6 SHA-2 algoritam

NIST je objavio četiri dodatne heš funkcije u okviru SHA familije, svaka sa dužim izlazom, zajedno poznate kao SHA-2. Pojedinačne varijante su dobine nazine po dužini svojih izlaza (u bitovima): SHA-224, SHA-256, SHA-384 i SHA-512. Poslednje tri su objavljene 2001 u planu FIPS PUB 180-2, i tafđ su prihvacieni i ponovni pregled i komentari. FIPS PUB 180-2, koji je uključivao i SHA-1, proglašen je zvaničnim standardom 2002 godine. Februara 2004, objavljena je izmena za FIPS PUB 180-2, specificirajući dodatnu varijantu, SHA-224, definisanu da odgovara dužini ključa u dva-ključa Triple DES. Ove varijante su patentirane u US patent 6829355. SHA-256 i SHA- 512 su nove heš funkcije izračunate sa 32-bitnim i 64-bitnim rečima, respektivno. Koriste druge veličine za pomeranje i konstante za sabiranje, ali ako se to ostavi po strani njihova struktura je identična, razlikuje se samo u broju krugova. SHA-224 i SHA-384. Predstavljaju skraćene verzije prve dve, izračunate sa različitim inicijalnim vrednostima. Ove nove heš funkcije nisu toliko ispitivane od strane javnosti kriptografskog udruženja kao SHA-1, tako da njihova kriptografska sigurnost jkoš uvek nije dobro utvrđena. Gilbert i Handschuh (2003) proučavali su novije varijante i nisu našli slabosti.



Slika 7.4 Šema jedne runde algoritma SHA-2

Na slici je prikazana jedna runda u algoritmu SHA-2 (crveni kvadrati sa plusevima predstavljaju i ovde sabiranje po modulu 2^{32}). Za razliku od SHA-1, ovde imamo više promenljivih stanja (a do h), i kompleksnije računanje. Dok smo kod SHA-1 imali samo jednu funkciju (f), ovde ih imamo 4 – Ch, Ma, S1 i S0. Računanje ključeva je takođe malo kompleksnije, ali, za utehu, imamo „samo“ 64 runde.

Listing 7.5 SHA2 - pseudokod

Napomena: Sve promenljive su neoznačeni 32-bitni celi brojevi

Inicijalne promenljive, ima ih 8, su dobijene od razlomljenih delova (prva 32 bita) kvadratnih korena prvih 8 prostih brojeva (2, 3, 5, 7, 11, 13, 17, 19):

```
h0 := 0x6a09e667  
h1 := 0xbbb67ae85  
h2 := 0x3c6ef372  
h3 := 0xa54ff53a  
h4 := 0x510e527f  
h5 := 0x9b05688c  
h6 := 0x1f83d9ab  
h7 := 0x5be0cd19
```

Definisanje niza ključeva (za svaku od runde po jedan). Ovi ključevi su dobijeni tako što su uzimana po prva 32 bita kubnog korena svakog od prva 64 prosta broja:

```
k[0..63] :=  
    0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b,  
    0x59f111f1, 0x923f82a4, 0xab1c5ed5,  
    0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74,  
    0x80deb1fe, 0x9bdc06a7, 0xc19bf174,  
    0xe49b69c1, 0xefbe4786, 0xfc19dc6, 0x240calcc, 0x2de92c6f,  
    0xa7484aa, 0x5cb0a9dc, 0x76f988da,  
    0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3,  
    0xd5a79147, 0x06ca6351, 0x14292967,  
    0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354,  
    0x766a0abb, 0x81c2c92e, 0x92722c85,  
    0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819,  
    0xd6990624, 0xf40e3585, 0x106aa070,  
    0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3,  
    0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,  
    0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90beffffa,  
    0xa4506ceb, 0xbef9a3f7, 0xc67178f2
```

Preprocesiranje:

- doda se jedan bit '1' na poruku
- doda se k bitova '0', gde je k minimalni broj ≥ 0 takav da dužina rezultujuće poruke daje ostatak 448 pri deljenju sa 512.
- vrednost dužine poruke, pre preprocesiranja, se pretvori u 64-bitni „big-endian“ ceo broj i konkatenira se na rezultat prethodnog koraka

Nakon svega ovoga dužina poruke (u bitovima) je deljiva sa 512. Možemo da kažemo da se poruka sada sastoji iz celog broja, označimo ga sa t, blokova dužine 512 bitova:

```
for s from 0 to t // za svaki 512-bitni blok
```

podelimo blok na 16 32-bitnih reči i obeležimo ih $w[i]$, $0 \leq i \leq 15$

Preračunamo ostale članove niza w (ima ih koliko i runde, dakle 64), po sledećoj formuli:

```
for i from 16 to 63
    s0 := (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18)
    xor (w[i-15] rightshift 3)
    s1 := (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19)
    xor (w[i-2] rightshift 10)
    w[i] := w[i-16] + s0 + w[i-7] + s1
next
```

Inicijalizacija promenljivih stanja za aktuelni blok:

```
a := h0
b := h1
c := h2
d := h3
e := h4
f := h5
g := h6
h := h7
```

Runde:

```
for i from 0 to 63

// Računanje s0, ch, ma i s1 u svakoj u rundi
// t1 i t2 su pomoćne promenljive

    s0 := (a rightrotate 2) xor (a rightrotate 13) xor (a
rightrotate 22)
    ma := (a and b) xor (a and c) xor (b and c)
    t2 := s0 + ma
    s1 := (e rightrotate 6) xor (e rightrotate 11) xor (e
rightrotate 25)
    ch := (e and f) xor ((not e) and g)
    t1 := h + s1 + ch + k[i] + w[i]

    // prearačunavanje promenljivih stanja (vidi sliku)
    h := g
    g := f
    f := e
    e := d + t1
    d := c
    c := b
    b := a
    a := t1 + t2
next

// Rezultat dobijen nakon svih rundi za aktuelni blok podataka
se
// dodaje u inicijalizacione promenljive
h0 := h0 + a
h1 := h1 + b
h2 := h2 + c
h3 := h3 + d
```

```

h4 := h4 + e
h5 := h5 + f
h6 := h6 + g
h7 := h7 + h

next

// Na kraju se sve inicijalizacione promenljive stapaju u jednu
// vrednost koja predstavlja rezultat računanja heš-a

hash = h0 append h1 append h2 append h3 append h4 append h5 append
h6 append h7

```

7.7 Univerzalno heširanje

Ovde je opisan jedan, ali ne i jedini, postupak za računanje 24-bitnog UMAC-a. Opšti algoritam za UMAC – ove se u najkraćem može opisati na sledeći način:

- Treba nam neki „slučajno“ izabrani polinom reda 24, u primeru koji sledi to je $x^{24} + x^4 + x^3 + x + 1$.
- Treba nam inicijalna „tajna“ vrednost (zvaćemo je nadalje secret), čija dužina u bitovima treba da bude deljiva sa 24. Ekvivalent ovoga je tzv. IV vektor kod kodera blokova podataka.
- Podelimo poruku za koju tražimo heš vrednost na jednakе blokove koji nisu duži od dužine podatka zvanog secret
- Rezultat kodovanja se na početku postavi na 0
- Za svaki blok se izvrši heš funkcija (kao Uhash24 u primeru), i rezultat kodovanja se preračunava kao stari rezultat XOR rezultat kodovanja jednog bloka
- Napomena: ovo nije jedini opšti način za računanje UMAC-ova.

Sledeća C funkcija generiše 24-bitni UMAC za jedan blok podataka.

Listing 7.6 Univerzalno heširanje - pseudokod

```

#define uchar unsigned char;
void UHash24 (uchar *msg, uchar *secret, int len, uchar *result)
{
    // - dužina vrednosti secret (u bitovima) treba da bude deljiva sa
    24
    // - blok poruke (msg) treba da je po dužini manji ili jednak
    vrednosti
    // dužine za parametar secret
    // - niz result se sastoji od 3 bajta (24 bita)
    // - len je dužina bloka poruke (msg) u bajtovima
}

```

ZAŠTITA INFORMACIJA

```
uchar r1 = 0, r2 = 0, r3 = 0, s1, s2, s3, byteCnt = 0, bitCnt,
byte;

while (len-- > 0) // za svaki bajt u poruci
{
    if (byteCnt-- == 0)
    {
        // ako je byteCnt jednako 0, postave se vrednosti za s1, s2 i s3
        // nakon svaka 3 bajta s1, s2 i s3 se ponovo postave
        s1 = *secret++;
        s2 = *secret++;
        s3 = *secret++;
        byteCnt = 2;
    }

    // uzme se sledeći bajt iz bloka poruke
    byte = *msg++;

    // za svaki bit u promenljivoj byte
    for (bitCnt = 0; bitCnt < 8; bitCnt++)
    {
        if (byte & 1)
        {
            // ako se bajt iz poruke završava jedinicom, postave se
            // privremene vrednosti rezultata r1, r2 i r3
            r1 ^= s1;
            r2 ^= s2;
            r3 ^= s3;
        }
    }

    // byte se pomeri za jedno mesto u desno
    byte >>= 1;

    // isečak vrednosti secret, koji čine s3,s2 i s1 se pomeri za 1
    // u levo
    s3 <= 1;
    if (s2 & 0x80) s3 |= 1;
    s2 <= 1;
    if (s1 & 0x80) s2 |= 1;
    s1 <= 1;

    // sada, ako isečak vrednosti secret počinje sa 1 uraditi
    // XOR sa unapred definisanim polinomom stepena 24
```

```
if (s3 & 0x80)
{
    s1 ^= 0x1B; /* x^24 + x^4 + x^3 + x + 1 */
}

} /* za svaki bit u poruci */
} /* za svaki bajt u poruci */

// postavljanje rezultata
*result++ ^= r1;
*result++ ^= r2;
*result++ ^= r3;
}
```


8 ODABRANA TEORETSKA POGLAVLJA

U ovom udžbeniku prikaćemo odabrana teoretska poglavља sa predavanja [1], koja će čitaocu pružiti osnovnu sliku kompleksnosti tema kojima se bavi Zaštita informacija.

8.1 Moderna kripto istorija – Mašina za šifriranje *Enigma*

U dvadesetom veku kriptografija je igrala važnu ulogu u velikim svetskim događajima. Krajem 20. veka, kriptografija je postala kritična tehnologija za komercijalne i poslovne komunikacije. Čuveni *Zimmermann*-ov telegram je jedan od prvih primera uloge koju kriptoanaliza imala u političkim i vojnim pitanjima. U ovom poglavlju, spomenućemo i nekoliko drugih istorijskih činjenica iz prošlog veka.

Godine 1929., državni sekretar Henri L. Stimson okončao je zvaničnu kriptoanalitičku aktivnost američke vlade, pravdujući svoje postupke čuvenom rečenicom "gospoda ne čitaju jedni drugima poruke". Ovo će se pokazati kao skupa greška kada je kasnije došlo do japanskog napada na Perl Harbor. Ubrzo posle napada 7. decembra 1941. godine, SAD je ponovo započela svoj kriptoanalitički program. Uspeh savezničkih kriptoanalitičara tokom II svetskog rata su bili izvanredni, a ovaj period se često posmatra kao "zlatno doba" kriptoanalyse. Praktično svi značajni tadašnji kriptosistemi su otkriveni u to doba.

U to vreme, *Black* šifra se koristi za visok nivo komunikacije unutar vlade Japana. Ova šifra je otkrivena od strane američkih kriptoanalitičara pre napada na Perl Harbor, tako da je obaveštajna agencija dobila (pod šifrom *Magic*) jasnu indikaciju predstojećih napada. Japanska carska mornarica koristila je šifru poznatu kao JN-25, koja je takođe otkrivena od strane Amerikanaca. Informacije dobijene otkrivanjem JN-25 su gotovo sigurno bile odlučujuće u borbama u Koralnom moru, gdje je inferiorna američka vojna grupa uspela da zaustavi napredovanje Japanaca na Pacifiku, po prvi put od početka ratovanja. Japanska mornarica kasnije nije bila u stanju da se oporavi od gubitaka nanesenih tokom ove borbe.

U Evropi, otkrivanje Enigma šifre (šifra je bila poznata pod nazivom *Ultra*) takođe je bilo ključan korak u pomoći saveznicima u ratu. Često se tvrdi da je Ultra bila toliko vredna da u novembru 1940. godine, Čerčil odlučuje da ne obavesti britanski grad Coventri o predstojećem napadu od strane nemačkog Luftwaffe, da se ne bi otkrilo da je Enigmina šifra bila već razbijena. Enigma je prvobitno analizirana od strane poljskih kriptoanalitičara. Nakon pada Poljske, poljski kriptoanalitičari su mahom pobegli u Francusku. Ubrzo nakon toga, Francuska je pala nacistima u ruke i poljske kriptoanalitičari

beže zatim u Englesku, gde su kasnije svoje znanje udružili sa britanskim kriptoanalitičarima.

Neobično je primetiti da poljskim kriptoanalitičarima nije bilo dozvoljeno da nastave svoj rad na Enigmi. Međutim, britanski tim uključujući i pionira računarstva, Alana Turinga je kasnije razvio poboljšan napad.

Posle Drugog svetskog rata, kriptografija konačno prelazi u oblasti nauke. Objavljanje rada 1949. Pod naslovom *Information Theory*, Claude Shannon je u teoriji tajnosti sistema uspeo da odredi prekretnicu. Shannon je dokazao da je *One-Time Pad* system šifriranja siguran i ponudio dva osnovna šifra principa dizajna sistema: *konfuziju* i *difuziju*.

Konfuzija je potrebna da sakrije vezu između osnovnog i šifriranog teksta, dok difuzija treba da omogući pristup preko statistike u kriptoanalitičkom sistemu.



Slika 8.1. Mašina Enigma (ljubaznošću TB Perera i Enigma muzeja [1]).

Ova dva koncepta - konfuzija i difuzija su i dalje vodeći principi u dizajniranju šifara danas. U narednim poglavljima, postaće jasno koliko su bitni ovi koncepti u modernom dizajnu blok šifara.

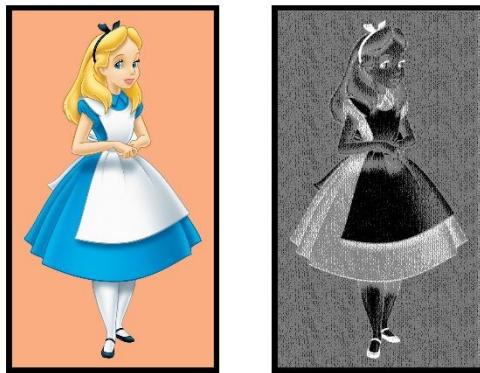
Do nedavno, kriptografija je bila isključivo u domenu vlada. To se dramatično promenio 1970., pre svega zbog računarske revolucije, što je dovelo do potrebe da se zaštiti velika količina elektronskih podataka. Do sredine 1970-ih, čak je i američka vlada shvatila da je postojala komercijalna potreba za uslugama kriptografije, i bilo je jasno da su komercijalni proizvodi upravo ti koji nedostaju. Nacionalni biro za standarde, (NBS4) izdao zahtev za razvoj kriptografskih algoritama. Krajnji rezultat ovog procesa bio je standard poznat kao *Data Encryption Standard*, ili DES, koji je postao zvanični standard američke vlade.

Nakon pojave DES-a, i akademski interes u kriptografiji je brzo rastao. Sistem javnog ključa u kriptografiji je otkriven ubrzo nakon DES-a. Do 1980-ih, počele su i godišnje Cripto konferencije, koje konstantno prikazuju visok kvalitet rada kako u teoretskom tako i na praktičnom polju.

Vlade nastavljaju da finansiraju velike organizacije koje rade u kripto i srodnim oblastima. Međutim, jasno je da je kriptografija kao oblast već otišla iz striktno državnog okvira.

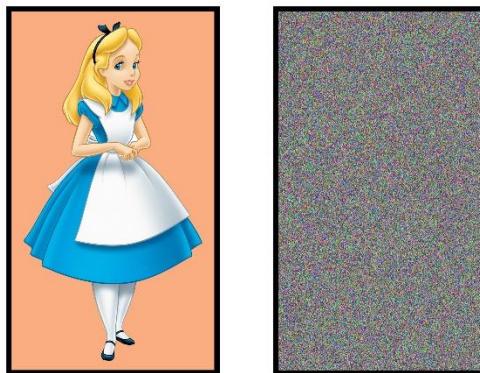
8.2 Režimi rada Block kodera-a

Korišćenje *Stream* šifratora je jednostavno – generiše se niz koja je iste dužine kao i osnovni tekst i primeni se operacija XOR. Korišćenje blok šifre je takođe lako, sve dok postoji tačno jedan blok za šifriranje. Ali kako bi bilo ako postoje više blokova ili ako bi se išlo sa delimičnim kodiranjem? Odgovori na ova pitanja rešavaju se primenom različitih režima rada blok kodera, što je detaljno objašnjeno u delu sa vežbama. Kao ilustraciju, prikazaćemo modifikaciju slike korišćenjem CBC i ECB blok režima.



Slika 8.2. Alice i ECB režim.

Primećuje se da ECB režim zadržava strukturu slike posle kodiranja, tako da se može vizuelno prepoznati. Da bi se ovo onemogućilo, umesto da se slika Alice kodira korišćenjem ECB moda, može se primeniti CBC mod tako da finalna kodirana slika izgleda kao da je transponovana belim šumom tako da se unutrašnja struktura ne može više prepoznati.



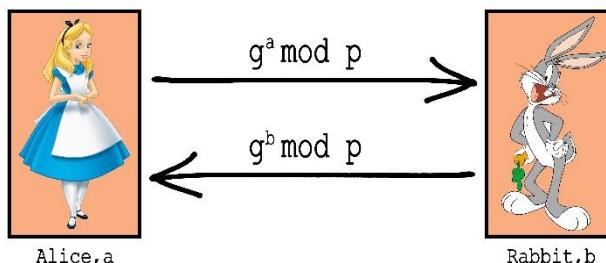
Slika 8.3. Alisa u CBC režimu kodiranja.

8.3 Integritet

Dok se kriptografija bavi sprečavanjem neovlašćenog čitanja, zaštita integriteta podataka se bavi sprečavanjem neovlašćenog pisanje i menjanja sadržaja informacija. Važno je shvatiti da su poverljivost i integritet dva veoma različita pojma. Kodiranje sa bilo kojom šifrom u one-time pad sistemu, u jednom od svojih režima ne štiti podatke od zlonamernih ili nemamernih promena. Ako se manipuliše šiframa (*cut-and-paste* napad) ili ako se javljaju greške u prenosu, integritet podataka je uništen. Naš je cilj da budemo u stanju da automatski detektujemo da li je do promene u prenosu došlo i da li originalni tekst identičan primljenom, posle dešifriranja.

8.3.1. Diffie-Hellman

Algoritam sa izmenom ključa (*Key Exchange Algoritam*) tipa Diffie-Hellman, ili DH skraćeno, pronašao je Malcolm Williamson iz GCHK i ubrzo nakon toga su nezavisno otkriveni od strane Whitfield Diffie i Martin Hellman-a. DH je u osnovi algoritam sa razmenom zajedničkog simetričnog ključa.

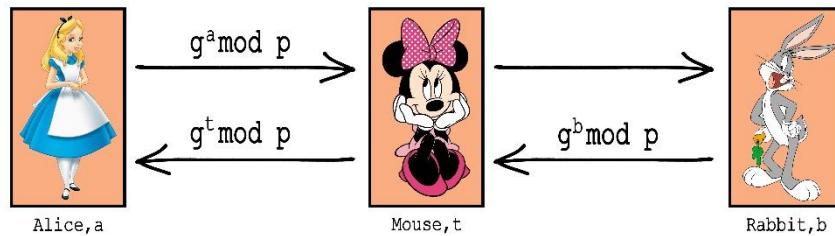


Slika 8.4. Ključ Diffie-Hellman - razmena.

MiM (*Man in the Middle*) napad prikazan na sledećoj slici predstavlja veliki problem kada se koristi DH. Kako možemo sprečiti ovaj napad? Postoji nekoliko mogućnosti, uključujući:

1. Šifriranje razmene DH sa zajedničkim simetričnim ključem,

2. Šifriranje razmene DH korišćenjem javnih ključeva,
3. Potpisivanje vrednosti DH sa privatnim ključevima.



Slika 8.5. Diffie-Hellman *man-in-the-middle* napad.

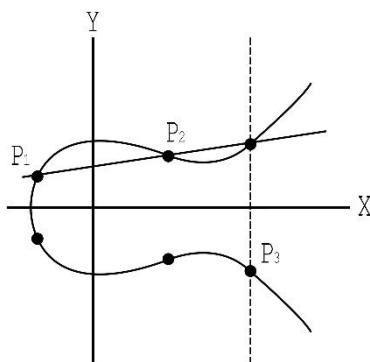
Ovaj algoritam će se detaljnije pokazati u delu sa računskim vežbama i zadacima.

8.4 Korišćenje eliptičnih krivih u kriptografiji

Eliptična kqua se ne može svrstati u poseban kriptosistem. Umesto toga, ove krive jednostavno obezbeđuju još jedan način za obavljanje složenih matematičkih operacija koje su potrebne u kriptografiji javnog ključa. Na primer, postoji verzija Diffie-Hellman algoritma korišćenjem eliptičnih krivih. Prednost eliptičke kqua u kriptografiji (ECC) je da je potrebno manje bitova za isti nivo sigurnosti u poređenju sa ne-eliptičnim krivama. Sa druge strane, rad sa eliptičnim krivama je složeniji i kao rezultat toga, matematika primenjena na eliptičnim krivama je komplikovanija. Sve u svemu, eliptične krive u izračunavanju imaju prednost. Iz tog razloga, ECC je posebno popularna u okruženjima sa ograničenim resursima kao što su mobilni uređaji.

Eliptička kqua E je grafik funkcije oblika:

$$E: y^2 = x^3 + ax + b$$



Slika 8.6 Eliptička kqua.

Kodiranje i dekodiranje u ovom sistemu je jednostavno i svodi se primenu zadate formule najčešće u *integer* domenu. Podaci koji se kodiraju pripadaju x domenu a šifrovani podaci z domenu.

8.5 Hash funkcije

Hash sistem i hash funkcije su veoma popularne jer omogućavaju da se veoma jednostavnim sredstvima, najšešće upotreboom jednostavne dvodimenzionalne matrice, vrši obrada informacija kroz process prekodiranja. Korišćenje indeksa kao ulazne informacije koja se kodira u hash tabelu je osnovni mehanizam rada sa ovom strukturom podataka.

Kriptografska *hash funkcija* $h(k)$ mora da obezbedi sledeće funkcionalnosti:

- *Kompresija:* Za bilo koju veličinu ulaznog k , izlaz dužine $i = h(k)$ je mali. U praksi, kriptografske heš funkcije proizvede određenu veličinu na izlazu, bez obzira na dužinu ulaza.
- *Efikasnost:* Mora se lako izračunati $h(k)$ za bilo koji ulaz k . Vreme potrebno da se izračuna $h(k)$ će sigurno rasti sa dužinom od k , ali ne bi trebalo da raste prebrzo.
- *Jednosmernost (One-way):* Sa obzirom na bilo koju vrednost i , računski je neizvodljivo da se nađe vrednost k ako je $H(k) = i$. Drugim rečima, teško je obrnuti hash.
- *Slaba otpornost na hash kolizije:* S obzirom $h(i)$, to je neizvodljivo da se pronađu i , sa $I = k$, tako da $H(i) = h(k)$.
- *Jaka otpornost na hash kolizije:* To je neizvodljivo da pronađu bilo kakav K_s i I , sa $K_s \neq I$, tako da je $H(k) = h(I)$.

Kolizije moraju postojati jer je ulazni prostor stanja mnogo veći od izlaznog prostora. Na primer, prepostavimo da hash funkcija generiše 128-bitni izlaz. Ako uzmemu u obzir, recimo, sve moguće 150-bitne ulazne vrednosti onda, u proseku više od 4.000.000 od ovih ulaza ide na svaku moguću izlaznu vrednost. Kolizije se teško nalaze.

8.6 Rođendanski problem – (The Birthday Problem)

Takozvani *rođendanski problem* je fundamentalno pitanje u mnogim oblastima kriptografije. Prepostavimo da ste u sobi sa N drugih ljudi. Koliko veliki broj N mora da bude pre nego što se očekuje da će se pronaći bar jedna osoba sa istim danom rođendana? Problem može da se postavi i da drugačiji način: koliki mora biti N da bi verovatnoća da neko ima isti rođendan bila veća od $1/2$? Kao i kod mnogih proračuna verovatnoće, lakše je izračunati suprotnu verovatnoću odnosno verovatnoću da nijedan od N ljudi imaju isti rođendan i da se dobijeni rezultat oduzme od jedan.

Rođendan se pada jednog dana u godini. Ako druga osoba nema isti rođendan, njegov rođendan mora biti jednom od ostalih 364 dana. Pod prepostavkom da svi datumi rođenja podjednako verovatni, verovatnoća da slučajno izabrana osoba nema isti rođendan je $364/365$. Verovatnoća da niko od N ljudi imaju isti rođendan je $(364/365)^N$, a verovatnoća da bar jedna osoba ima isti rođendan je $1 - (364/365)^N$. Rešavanjem tog izraza za $1/2$, nalazimo $N = 253$.

Sada želimo da odgovorimo na pitanje, koliko mora da glasi N da bi sa verovatnoćom većaom od $1/2$ bilo koja dva ili više osoba imaju isti rođendan?

I u ovom slučaju je lakše rešiti za verovatnoću komplementa i oduzeti rezultat od 1. U ovom slučaju, dopuna uslova je da svih N ljudi imaju različite rođendane.

Neka su N ljudi u prostoriji numerisani $0, 1, 2, \dots, N - 1$. Osoba 0 ima poseban rođendan. Ako svi ljudi imaju različite rođendane, onda osoba 1 mora imati svoj rođendan koji se razlikuje od osobe 0; to jest, osoba 1 može svoj rođendan imati na bilo koji od preostalih 364 dana. Slično tome, osoba 2 može svoj rođendan na bilo preostalih 363 dana, i tako dalje. Opet, pod pretpostavkom da su svi datumi rođenja podjednako verovatni i izračunavanjem komplementa, dobijena verovatnoća je:

$$1 - 365/365 \cdot 364/365 \cdot 363/365 \cdots (365 - N + 1)/365.$$

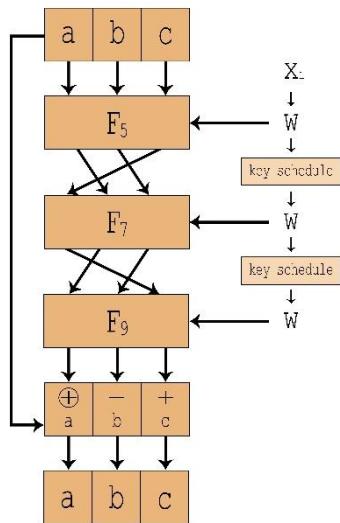
Izjednačavanjem ovog izraza sa $1/2$ nalazimo da je $N = 23$. Ovo se često naziva *paradoks rođendana*, I na prvi pogled, izgleda paradoksalno da sa samo 23 ljudi u jednoj prostoriji, očekujemo da se nađu dva ili više sa istim rođendanom. U ovom zadatku, mi smo računali u odnosu na rođendane svih parova ljudi. Sa N ljudi u jednoj prostoriji, broj poređenja je $n(n - 1)/2 \approx N^2$. Pošto postoje samo 365 različitih datuma rođenja, treba očekivati da se pronađe podudaranje na mestu gde je $N^2 = 365$, ili $N = \sqrt{365} \approx 19$. Gledano u tom svetlu, rođendanski paradoks nije tako paradoksalan kao što se to čini na prvi pogled.

8.7 Tiger HASH

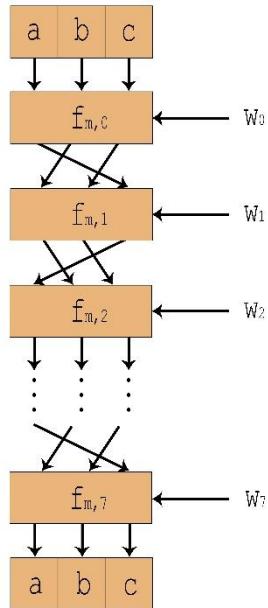
U ovom delu ćemo obratiti pažnju na kriptografske heš funkcije. Dve posebne heš funkcije su najpopularnije danas. Jedna od njih je MD5, gde je "MD" skraćenica za "Message Digest" i pronalazak je Ron Rivesta. MD5 je naslednik MD4, koja je i sama bila naslednik MD2. Kao i kod svih hash sistema, problem su kolizije. MD5 kolizije su nedavno otkrivene. MD5 proizvodi 128-bitni izlaz.

MD5 i SHA-1 algoritmi nisu posebno zanimljivi jer se oba sastoje od slučajnih transformacija. Umesto da raspravljamo o bilo kome od njih, prikazaćemo ukratko osnovnu postavku Tiger hash koji će kasnije biti detaljnije obrađen. Tiger hash, koji je razvijen od strane Ross Anderson i Eli Biham, ima više strukturiran dizajn od SHA-1 ili MD5.

Tiger je dat u obliku koji veoma podseća na blok šifre. Tiger je dizajniran da bude "brz i jak" pa otuda i ime. Takođe je dizajniran za optimalan rad na 64-bitnim procesorima zamjenjujući MD5 i SHA-1, ili bilo koji drugi sličan algoritam. Na sledećim slikama su prikazane osnovne blok šeme algoritma u spoljnem i unutrašnjem prolazu.



Slika 8.7. Tiger spoljašnji ciklus.

Slika 8.8. Tiger unutrašnji ciklus za F_m .

8.8 Sakrivanje informacija

Dva osnovna metoda sakrivanja informacija koje ćemo ovde obraditi su steganografija i digitalni vodeni žigovi.

Steganografija ili "skriveno pisanje" je pokušaj da se sakrije informacija da se prenose poruke koje su tajne prirode. Veoma poznat primer je sa vodenim žigovima koji kriju informacije koje otkrivaju identitet digitalne muzike u cilju identifikacije onih koji su odgovorni za ilegalne redistribucije. U poznatoj priči o Herodotu (oko 440 pre nove ere), grčki general je obrijao glavu roba i napisao poruku na glavi roba radi upozorenja na persijsku invaziju koja je predstojila. Nakon toga, kada je kosa roba izrasla dovoljno da pokrije poruku, rob je poslat kroz neprijateljske linije da isporuči svoju skrivenu poruku. Tokom vojne istorije, steganografija je korišćena mnogo češće nego kriptografija. Moderna verzija steganografije uključuje sakrivanje informacija u medijima, kao što su slikovne datoteke, audio podatake ili čak softver. Ova vrsta informacija koja se krije u nekoj drugoj moguće je takođe posmatrati kao oblik tajnog prenosnog kanala informacija.

Digitalni vodeni žig je podatak namenjen nešto različitoj svrsi. Postoji nekoliko varijanti vodenih žigova; u jednoj vrsti tzv. "nevidljivi" identifikator se dodaje podacima. Na primer, identifikator može da se doda uz digitalnu muziku, tako da, u principu, ako se pojavi piratska verzija muzike, vodeni žig se može pročitati iz nje i kupac ili prepostavljen pirat-distributer mogu biti identifikovani. Takve tehnike su razvijene za praktično sve tipove digitalnih medija, kao i za softver. Digitalni vodeni žigovi se realizuju u nekoliko formi, uključujući:

- *Nevidljivi vodeni žigovi*, koji ne bi trebalo da bude vidljivi u medijima.
- *Vidljivi vodenih žigovi*, zamišljeni da se respektuju. TOP SECRET pečat na dokumentu je primer takvog žiga.

Vodeni žigovi se mogu dalje kategorizovani kao:

- *Robusni vodeni žigovi*, koje bi trebalo da ostanu čitljivi čak i ako su napadnuti.
- *Fragile vodeni žig*, koji su dizajnirani da budu uništeni ili oštećeni ako dođe do zloupotreba.

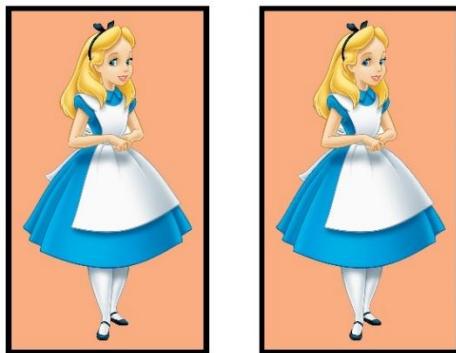
Na primer, možemo ubaciti snažan nevidljivi trag u digitalnu muziku u nadi da će se otkriti piraterija. Onda kada se piratska muzika pojavi na internetu, možemo naći trag do njegovog izvora. Možemo ubaciti *fragile* nevidljivi trag u audio datoteke. Ako je vodeni žig je nečitljiv, primalac zna da je došlo do neovlašćenog pristupa. Ovaj drugi pristup je od suštinskog značaja za oblik provere integriteta. Mogu biti korišćene različite druge kombinacije vodenih žigova.

Na mnogim savremenim valutama već postoje vodeni žigovi. Nekoliko sadašnjih američkih državnih zapisa, uključujući i 20 dolara kao na slici ima vidljive vodene žigove. U novčanici od 20 dolara, slika predsjednika Jacksona je ugrađena u samom radu, u desnom delu, a vidljiv je kada se drži prema svetlosti. Ovaj vidljivi vodeni žig je dizajniran da je falsifikovanje teže, jer je i sa posebnim papirom puno teže umnožiti ovu novčanicu – jednostavno se lako može proveriti vodeni žig.



Slika 8.9 Novčanica sa vodenim žigom.

Holografski vodeni žig je šema koja je predložena da se ubace informacije u fotografije na takav način da, ako dođe do oštećenja iste može biti moguće da se rekonstruiše celu sliku iz tog malog komada originala. Jedan kvadratni centimetar slici može da sadrži dovoljno informacija da se rekonstruiše čitava fotografija, bez uticaja na kvalitet slike. Ovaj veoma jednostavan pristup u steganografiji se primenjuje na digitalnim slikama, kao što je dato u nastavku:



Slika 8.10. Priča o dve Alice.

8.9 Kontrola pristupa

8.9.1 Utvrđivanje autentičnosti (AUTENTIFIKACIJA)

Mi ćemo koristiti termin *kontrola pristupa* za pitanja koja se tiču pristupa sistemskim resursima. U okviru ove široke definicije, postoje dve osnovne oblasti za kontrolu pristupa: *autentifikacija* i *autorizacija*.

Autentifikacija se bavi problemom utvrđivanja da li korisniku treba dozvoliti pristup određenom sistemu ili resursu. U ovom poglavlju, prikazaće se metode koje se koriste od strane ljudi da se potvrdila autentičnost pristupu mašinama (kompjuterima). U ovom delu se otvaraju mnoga pitanja u vezi sa protokolima, naročito kada se provera autentičnosti javlja preko mreže. Ovo je ujedno i okruženje u kome se najčešće realizuje mašina-mašina autentifikacija. Po definiciji, korisnici koji su prošli proveru autentičnosti mogu imati pristup sistemskim resursima.

Međutim, korisniku koji je prošao potvrdu autentičnosti se obično ne daje *neograničen* pristup svim resursima sistema. Na primer, mora se omogućiti da samo privilegovani korisnik kao što je administrator instalira softver na sistemu. Kao rezultat toga, moraju se ograničiti delovanja proverenih korisnika. To već potпадa pod polje ovlašćenja. Dok je autentifikacija binarna odluka da bi se dobio pristup, procedura autorizacije se bavi ograničenjima na pristup različitim sistemskim resursima.

Autentifikacija se često koristi kao pogrešan sinonim za autorizaciju. Međutim, u zaštiti informacija, kontrola pristupa je šire definisana uključujući i potvrdu identiteta i autorizaciju pod okriljem termina kontrola pristupa. Ova dva aspekta kontrole pristupa mogu se svesti na:

- Autentifikacija: Pitanje: Ko je tamo?
- Autorizacija: Pitanje: Da li dozvoljeno da to uradite?

8.9.2 Metode autentifikacije

Osnovni problem koji ćemo razmotriti u ovom poglavlju je rešavanje problema utvrđivanja autentičnosti čoveka prema mašini. Drugim rečima, želimo da ubedimo sistem da je neko ili nešto to što tvrdi da jeste.

Sa strane korisnika, može se potvrditi autentičnost prema mašini na osnovu bilo koje kombinacije sledećih informacija:

- Nešto što korisnik zna,
- Nešto što korisnik ima,
- Nešto korisnik jeste.

Lozinka (*Password*) je primer za "nešto što korisnik zna". Postoji opšta saglasnost da lozinke predstavljaju ozbiljnu slabu kariku u mnogim savremenim informacionim bezbednosnim sistemima. Primer za "nešto što korisnik ima" je bankomat kartica ili pametna kartica. Primer za "nešto što korisnik jeste" je kategorija sinonim za oblast biometrije koja se ubrzano širi. Na primer, danas možete kupiti digitalni skener otiska palca i koristiti rezultat za autorizaciju. Ukratko ćemo diskutovati nekoliko biometrijskih metoda kasnije u ovom poglavlju.

8.10 Lozinke

Idealna lozinka je nešto što znate, nešto što računar može da proveri da li znate ili nešto što niko drugi ne može da sazna čak i sa pristupom neograničenim računarskim resursima. U praksi je teško prići ni blizu ovom idealu.

Nesumnjivo je da su korisnici upoznati sa sistemom lozinki. Danas se masovno koriste kompjuteri pri čemu je značaj korišćenja lozinke izuzetno važan. Poznata činjenica u vezi lozinke je da mnoge stvari deluju kao lozinke. Na primer, PIN broj za ATM kartice je ekvivalent lozinke. A ako ste zaboravili "pravu" lozinku, sajt može da potvrdi autentičnost na osnovu vašeg broja socijalnog osiguranja, vašeg datuma rođenja ili drugih informacija koje se u ovom slučaju kao "stvari koje poznajete" ponašaju kao lozinke. Očigledan problem je što ove stvari nisu tajna. Kada korisnici izabiraju lozinke, oni imaju tendenciju da izaberu loše lozinke, što čini otkrivanje lozinke iznenađujuće lako. Postoje značajni matematički argumenti koji pokazuju da je veoma teško postići sigurnost preko lozinke. Jedno od rešenje problema lozinke bi bilo da se koristi nasumično generisani kriptografski ključevi umesto lozinke. Problem sa takvim pristupom je da ljudi moraju da pamte svoje lozinke.

Najpre da objasnimo zašto su lozinke tako popularne. Zašto je "nešto znate" popularnije od "nečega imate" i "nešto što jeste" kada druga dva su, po svoj prilici, sigurnija?

Odgovor su pre svega troškovi sistema kao i pogodnost upotrebe. Lozinke su besplatne, dok pametne kartice i biometrijski uređaji za očitavanje koštaju. Za administratora koji je preopterećen poslovima održavanja sistema za izdavanje nove lozinke je potrebno daleko manje napora nego da se obezbedi konfigurisanje nove pametne kartice.

8.11 Biometrija

Biometrija kao identifikacija osobina koje poseduje korisnik i kao metod provere identiteta se može jednostavno definisati kroz besmrtnе reči Schneider-a: "Ti si sam svoj ključ!"

Danas postoji mnogo različitih tipova biometrije, uključujući i takve tradicionalne metode kao što je sistem identifikacije preko otiska prstiju ili rukom pisanih potpisa. U skorije vreme, biometrija je zasnovana na automatizovanom prepoznavanju lica, prepoznavanja govora, hoda, strukture šake pa čak i prepoznavanja mirisa. Oblast biometrije je trenutno veoma propulzivna oblast istraživanja.

U oblasti zaštite informacija, glavni podsticaj za razvoj biometrije je zamena za lozinke. Da bi to bilo praktično, jeftin i pouzdan biometrijski sistem je potreban. Danas, upotrebljivi biometrijski sistemi postoje, uključujući i otisk prsta na ručnim skenerom zatim sistem za otisk dlana za kontrolu pristupa objektima, upotrebom otiska prstiju za otključavanje vrata automobila i tako dalje. Oblast biometrije je zaista široka i za istraživanja i za praktičnu primenu.

Idealno, biometrijski treba da ima sledeće osobine:

- **Univerzalnost:** Idealna biometrijski sistem mora da ima mogućnost da se univerzalno primeni. U stvarnosti, biometrijski sistem ne važi generalno. Na primer, mali procenat ljudi nema otiske.
- **Razlikovanje detalja:** Idealan biometrijski sistem treba razlikovati ulazne informacije sa sigurnošću. U stvarnosti, ne postoji 100% sigurnosti, iako u teoriji, neke metode mogu imati veoma niske stope grešaka.
- **Stalnost karakteristike:** Idealno, fizička karakteristika koja se meri nikada ne sme da se promeni. U praksi, to je dovoljno ako karakteristika ostaje stabilna tokom razumno dugog vremenskog perioda.
- **Sposobnost lakog prikupljanja podataka:** Izabrana fizička karakteristika bi trebalo da bude takva da se može lako prikupiti neinvanzivno (bez fizičke interakcije sa korisnikom).
- **Pouzdanost, robustnost, razumljivost:** Da bi bio koristan u praksi, biometrijski sistem mora biti pouzdan, robustan i razumljiv pod realnim uslovima. Neki biometrijski sistemi koji su se pokazali odlični u laboratorijskim uslovima kasnije nisu uspeli da pruže slične performanse u praksi.

Biometrija može se koristiti za *identifikaciju* ili *autentifikaciju*. U identifikaciji, cilj je da se identificuje korisnik sa liste mnogih mogućih korisnika. Ovo se dešava, na primer, kada je sumnjiv otisk sa mesta zločina je upućen u bazu podataka otiska policije za poređenje sa svim podacima o datoteci. U ovom slučaju, poređenje je tipa jedan sa mnogima.

U autentifikaciji poređenje je jedan prema jedan. Na primer, ako neko tvrdi da je određeni korisnik, otisk prsta se biometrijskih poredi samo sa sačuvanim otisk prsta te osobe. Problem identifikacije je teži i ima višu stopu grešaka. U ovom delu, mi smo prvenstveno bavimo problemom provere identiteta.

U principu, postoje dve faze u biometrijskom sistemu. Prvo, postoji *faza upisa*, gde subjekti imaju svoje biometrijske podatke koje se unose u bazu podataka. Tipično, tokom ove faze potrebno je pažljivo merenje odgovarajuće fizičke informacije. Pošto je ovo jednokratna procedura (po svakom predmetu), to je prihvatljivo da je proces spor i više merenja su obavezna. U nekim trening sistemima, upis u bazu se pokazao kao slaba tačka, jer može biti teško da se u fazi korišćenja postignu rezultati koji su robustni kao i oni dobijeni

u laboratorijskim uslovima. Druga faza u biometrijskom sistemu je *faza poređenja-provere*. Ovo se dešava kada se koristi biometrijski sistem za detekciju da se utvrdi da li je (problem autentikacije) identitet korisnika ispravan. Ova faza mora biti brza, jednostavna i tačna.

Mi ćemo prepostaviti da subjekti kooperativni, to jest, oni da su spremni da mere odgovarajuću fizičku karakteristiku. Ovo je razumna pretpostavka u slučaju autentifikacije, jer je potvrda identiteta obično potrebna za pristup određenim informacijama ili ulazak u sigurnu sonu u nekom objektu. U vezi problema identifikacije, čest slučaj je da subjekti ne sarađuju.

Bila je predložena upotreba takvog sistema u kazinima, gde se sistem može koristiti za detekciju poznatih prevaranata koji pokušavaju da uđu u kazino. Takvi sistemi su predloženi i kao način za detekciju terorista na aerodromima. U takvim slučajevima, uslovi upisa su verovatno daleko od idealnih, a u fazi priznavanja, subjekti svakako ne sarađuju i verovatno će učiniti sve što je moguće da izbegnu otkrivanje. U nastavku, mi ćemo se fokusirati na problem provere identiteta pretpostavljajući da su subjekti kooperativni.

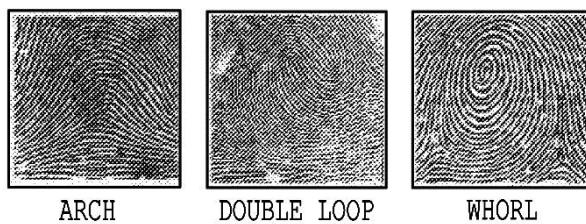
U nastavku opisacemo tri zajednička biometrička sistema. Prvo, razmotrićemo sistem identifikacije preko otiska prstiju. Otisci prstiju se počinju široko koristiti u kompjuterskim aplikacijama. Takođe, prikazaćemo identifikaciju putem otiska dlanova i iris skeniranja.

8.11.1 Otisci prstiju

Otisci prstiju su još korišćeni u drevnoj Kini kao oblik potpisa. Kasnije je to bio oblik potpisivanja ljudi koji nisu znali pismo. Međutim, upotreba otiska kao naučni oblik identifikacije je relativno nova pojava. Prva značajna analiza otiska dogodila se 1798. kada je J.C. Majer dokazao da su otisci biti jedinstven. Johannes Evangelist Purkinje je 1823. godine pronašao devet "obrasca otiska prsta", ali to bila rasprava iz oblasti biologije i autor nije sugerisao korišćenje otiska prstiju kao oblik identifikacije.

Prva moderna upotreba otiska prstiju za identifikaciju dogodio se 1858. godine u Indiji, kada Ser Vilhelm Heršel koristi šake i prste kao oblik potpisa na ugovorima. Godine 1880., dr Henri Folds objavio je članak u časopisu *Nature* koji je razmatrao upotrebu otiska prstiju radi identifikacije. U knjizi Marka Tvena *Life on the Mississippi*, koja je objavljena 1883., ubica je identifikovan preko otiska prsta. Široka upotreba otiska je postala moguća 1892. godine, kada je Ser Francis Galton razvio sistem klasifikacije zasnovan na "Minutia" linijama koji je i danas u upotrebi.

Primeri različitih tipova Minutia u sistemima klasifikacije Galton-a se pojavljuju na slikama u nastavku. Klasifikacija sistem Galton-a je pogodan za efikasno pretraživanje, čak i bez kompjutera. Galton je takođe utvrdio da se otisci prstiju ne menjaju tokom vremena. Danas, otisci prstiju se rutinski koriste za identifikaciju, posebno u krivičnim predmetima. Zanimljivo je da je standard za određivanje utakmicu varira.



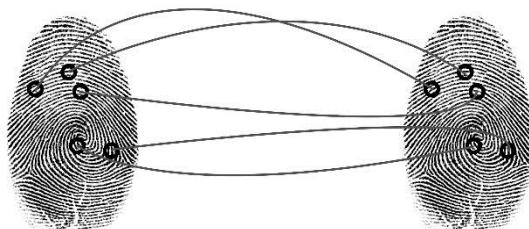
Slika 8.11 Primeri Galton-ovih Minutia.



Slika 8.12 Automatsko određivanje Minutia.

Na primer u Velikoj Britaniji, otisci moraju da odgovaraju u tačkama 16 Minutia, dok u Sjedinjenim Američkim Državama taj limit ne postoji.

Procedura za otiske prsta je sledeća. Najpre se vrši skeniranje otiska. Slika se zatim unapređuje primenom različitih tehnika za obradu slike i Minutie se identifikuju i izdvajaju iz poboljšane slike. Ovaj proces je ilustrovan na sledećoj slici.



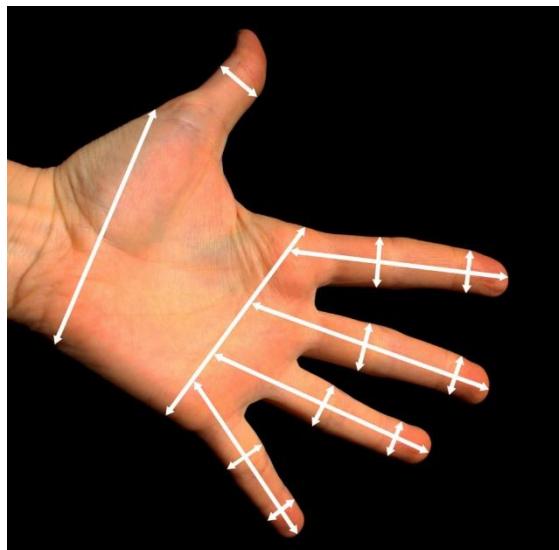
Slika 8.13 Poređenje Minutia.

Na taj način se ekstrahovan biometrijski sistem poređi na način koji je analogan sa manuelnom analizom otiska. Za potvrdu identiteta se ekstraktovane Minutie porede sa Minutia-ma koje su prethodno (u fazi upisa) pohranjene u bazi podataka. Sistem određuje statistički da li se javlja poklapanje, sa nekim unapred utvrđenom prakom odluke.

8.11.2 Geometrija šake

Drugi popularan oblik biometrijskog skeniranja namenjen posebno za ulazak u sigurne objekte je geometrija šake. U ovom sistemu, oblik ruku se pažljivo meri, uključujući i širinu i dužinu šake i prstiju. Obično se rade 16 takvih merenja na šaci, od kojih 14 su ilustrovane na sledećoj slici (druga dva merenja traže debljinu ruke). Ljudske šake nisu ni približno

jedinstvene kao otisci prstiju, ali geometrija šake se lako i brzo meri a sistem je dovoljno robustan za mnoge primene. Međutim, geometrija šake ne bi bila dovoljna za identifikaciju, budući da postoji mnogo lažnih rezultata.



Slika 8.14 Geometrija šake - merenja.

Jedna od prednosti geometrijskih sistema šake je da su brzi, uzimanje podataka traje manje od jednog minuta u fazi upisa u bazu i manje od pet sekundi u fazi prepoznavanja. Još jedna prednost je da ljudske ruke simetrične, pa ako je ruka, recimo u gipsu, sa druge ruke se može koristiti merenje parametara šake za prepoznavanje. U ovo smislu, sistem je veoma robustan. Neki nedostaci geometrije šake uključuju da se ne može koristiti kod veoma mlađih ili starih osoba a takođe i da sistem ima relativno visoku stopu grešaka.

8.11.3 Skeniranje dužice oka (Iris skeniranje)

Od svih biometrijskih metoda, u teoriji je najbolje za potvrdu identiteta upotrebiti iris skeniranje. Razvoj irisa (obojeni deo oka) je haotičan tako da manje varijacije dovode do velikih razlika. Postoji vrlo malo ili nimalo genetskog uticaja na uzorak irisa, tako da su podaci van korelacije za identične blizanace, pa čak i za dva oka jednog pojedinca. Još jedna poželjna osobina je činjenica da je obrazac stabilan tokom celog ljudskog života.

Proces razvoj irisa tehnologije skeniranja je relativno nov. Ideju o korišćenju ljudskih irisa za identifikaciju je predložio Frank Burch, 1936. godine. Tokom 1980-ih, ideja se pojavila u James Bond filmovima, ali su se tek 1986. godine pojavili prvi patenti u ovoj oblasti. Džon Daugman, istraživač na Univerzitetu Kembridž, je 1994. patentirao metod koji se danas smatra kao opšte prihvaćen najbolji pristup skeniranja irisa.

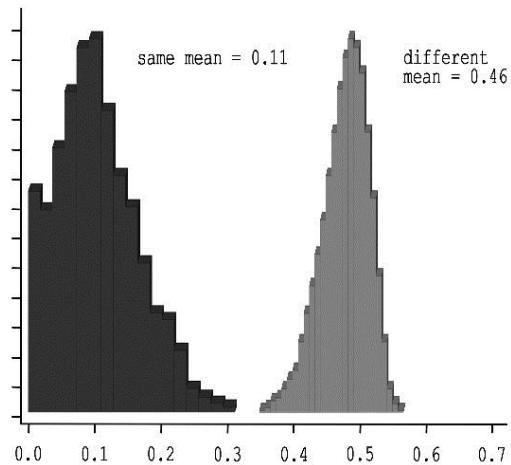
Automatski iris skener mora prvo da locira iris. Onda se generise crno-bela sliku oka. Dobijena slika se obrađuje dvodimenzionalnom *wavelet* transformacijom, čiji su rezultat 256 bajtova (2048-bit) "iris kod." Iris kodovi se računaju u odnosu na Hammingove udaljenosti.

Iris skeniranja se često navodi kao najbolji biometrijski sistem za proveru autentičnosti. Histogram na slici, koja je zasnovana na 2,3 miliona poređenja, teži da podrži ovaj stav, jer je preklapanje regiona između "istog" (*match*) i "drugačijeg" (*no match*) skoro nemoguće pa tako pogrešna identifikacija ne može ni nastati.

Iris scan rastojanja za podacima dato je na histogramu na slici u nastavku, gde vidimo da je jednaka stopa greške, odnosno *crossover* tačka između dva grafikona – javlja se negde u blizini na udaljenosti 0.34.



Slika 8.15. Iris skeniranje.



Slika 7.16. Histogram irisa - rezultati skeniranja [1].

8.11.4 Stope grešaka biometrijskih sistema

Poređenje biometrijskih stopa grešaka otkriva detalje u vezi prikazanih sistema za identifikaciju. Jednaka stopa greške, tačka u kojoj je stopa greške jednaka pogotka, je najbolja mera za upoređivanje biometrijskih sistema. Za sistem otiska prstiju u biometriji jednaka stopa greške je obično oko 5%, dok geometrija šake ima jednak stepen greške od oko 3/10. Iako ovo može izgledati iznenađujuće, većina biometrija otiska prstiju se radi na relativno jeftinim uređajima. S druge strane, za geometriju ruke biometrijski uređaji su

skuplji i sofisticirani. Prema tome, biometrija otiska prstiju nije ni blizu pravom potencijalu za tehnologiju. U teoriji, skeniranje irisa ima jednak stepen greške od oko 5/10. Ali u praksi se teško postižu takvi rezultati. To je očigledno zato faza upisa mora biti izuzetno precizna u cilju postizanja optimalne stope grešaka. U praksi, ljudi odgovorni za bazu uzorka (kao i sama oprema) ne može biti po laboratorijskim standardima na kojima je teoretski rezultat proveren. U praksi, većina drugih biometrija su gore od otiska prstiju.

8.11.5 Biometrijski metodi – zaključak

Biometrija ima ogromne potencijalne prednosti. Biometrijski sistem je teško, mada ne i nemoguće, da se falsificuje. Poznati su slučajevi falsifikovanja otiska prstiju voskom sa negativom otiska. Takođe, potencijalni su napadi na softver koji radi poređenja ili manipulišu sa bazom podataka koja sadrži podatke - uzorke. Dok se otkriven ključ za šifrovanje ili lozinka može ukinuti i zameniti, nije jasno kako može da se opozove otkriven biometrijski sistem.

Biometrija ima veliki potencijal kao zamena za lozinke. Imajući u vidu ogromne probleme sa lozinkama i veliki potencijal biometrije, iznenadujuće je da biometrija nije više rasprostranjena danas. Ovo bi trebalo promeniti u budućnosti, kada biometrija postaje još robusnija i jeftinija.

8.12. Identifikacija na osnovu nečega što korisnik ima

Pametne (*Smart*) kartice mogu da se koriste za proveru identiteta na osnovu "nečega što imate". Pametna kartica izgleda kao kreditna kartica, ali sadrži malu količinu memorije i ugrađeni CPU, tako da u stanju da skladišti kriptografske ključeve ili druge podatke:

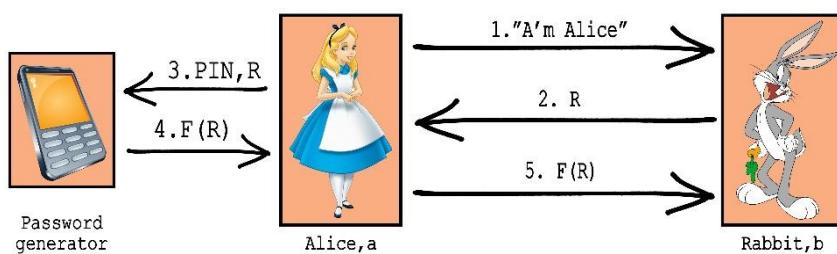


Slika 8.17 Čitač pametnih kartica. (Ljubaznošću Athena, Inc. [1])

Takođe, moguće je raditi proračune na samoj kartici. Namenske pametne kartice se najčešće koriste za proveru informacija koje se nalaze na kartici. Postoje još nekoliko

poznatih namena uključujući i identifikaciju laptop računara (ili njegove MAC adresa), ATM kartice ili generatora lozinke.

Generator lozinke je uređaj, veličine kalkulatora, koji korisnik mora da poseduje da bi se prijavio na sistem. Pretpostavimo da Alisa ima generator lozinke i da želi da potvrdi identitet prema Bob-u. Bob šalje nasumično "izazov" R koji Alisa onda ukucava u generator lozinke, zajedno sa svojim PIN-om. Generator lozinke zatim daje odgovor, koji Alisa šalje nazad Bobu. Bob onda može da proveri da li je odgovor tačan ili ne. Ako je odgovor tačan, Bob je uveren da je on zaista razgovarao sa Alice, jer samo Alice poseduje generator lozinke. Ovaj proces je ilustrovan na sledećoj slici. Videćemo kasnije da ima još mnogo primera upotrebe mehanizama izazov-odgovor u delu sa protokolima.



Slika 8.18 Password generator.

8.13. Ovlašćenje (AUTORIZACIJA)

Ovlašćenje je deo kontrole pristupa koji se bavi sistemom ograničenja u akcijama već proverenih korisnika. U našoj terminologiji, ovlašćenje je jedan dodatni aspekt kontrolе pristupa. Negde se u literaturi koristi izraz kontrola pristupa kao sinonim za autorizaciju. U prethodnom poglavlju, izložen je princip autentifikacije, gde je pitanje vezano za utvrđivanja identiteta. U svom tradicionalnom obliku, pitanje ovlašćenja se bavi situacijom u kojoj već identifikovanom korisniku želimo da sprovedemo ograničenja o tome šta je dozvoljeno da radi. Iako je provera identiteta binarna rad sa sistemo ovlašćenje je višezačan.

U ovom poglavlju, mi ćemo produžiti tradicionalnom pojmu ovlašćenja koji obuhvata nekoliko dodatnih oblika kontrole pristupa. Biće objašnjen sistem CAPTCHA, koji je dizajniran da ograniči pristup kompjuterima. Takođe, razmotrićemo i *firewall*, koji se može prikazati kao oblik kontrole pristupa na kompjuterskim mrežama.

8.14. Matrica kontrole pristupa

Prvo je potrebno definisati *subjekat* kao korisnika sistema (ne nužno ljudsko korisnika) i *objekat* kao sistem resursa. Dva osnovna koncepta koja se pojavljuju su *pristupne kontrolne liste*, ili ACL i *liste mogućnosti* ili C-liste. ACL i C-liste su izvedene iz *matrice kontrole pristupa*, koja ima red za svaki subjekt i kolone za svaki objekat. Pristup dozvoljen

predmetu S prema objektu O čuva se na preseku reda indeksiranog sa S i kolone indeksirane sa O .

U ovim tabelama program se tretira i kao objekat i subjekat. Na ovaj način, možemo primeniti ograničenje da se podaci mogu menjati samo iz programa. Na taj način je menjanje podataka teže, jer bilo kakve promene u programskom sistemu se moraju obaviti iz softvera koji, po svoj prilici, uključuje standardne kontrole. Međutim, to ne sprečava sve moguće napade, jer sistem administrator može da zameni program sa korumpiranim (ili lažnom) verzijom i prevaziđe ovu zaštitu.

8.14.1 ACL i C-liste

Pošto se svi subjekti i svi predmeti pojavljuju u matrici za kontrolu pristupa, ona sadrži sve relevantne informacije na kojima se mogu zasnivati odluke o ovlašćenjima. Međutim, postoji praktično pitanje u upravljanju velikom matricom kontrole pristupa. Realno, sistem može imati stotine predmeta i desetine hiljada objekata i u tom slučaju kontrola pristupa matrići sa milionima unosa bi trebalo da se obavi pre bilo kakve operacije. Rad sa tako velikom matricom se nameće kao neprihvatljiv teret prilikom realizacije operativnog sistema koji treba da podržava funkcije sigurnosti.

TABELA 8.1 Matrica kontrole pristupa.

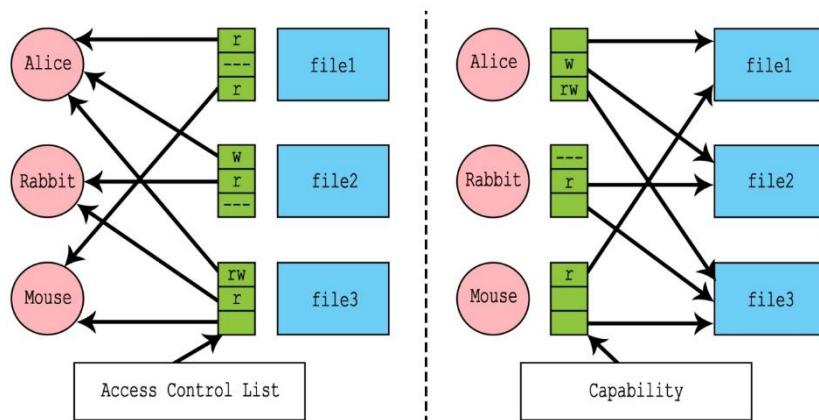
Matrica kontrole pristupa					
	OS	Accounting Program	Accounting Data	Insurance Data	Payroll Data
Rabbit	rx	rx	r	—	—
Alice	rx	rx	r	rw	rw
Mouse	rwx	rwx	r	rw	rw
acct.program	rx	rx	rw	rw	r

Na prvi pogled, ACL i C-liste su ekvivalentne jer jednostavno obezbeđuju različite načine čuvanja iste informacije. Međutim, postoje neke suptilne razlike između ova dva pristupa.

Uzmite u obzir poređenje ACL-a i C-listi prikazanih na sledećoj slici (8.19). Strelice na slici kod oba sistema se kreću u suprotnim pravcima; to jest, za ACL, strelice ukazuju od objekata

ka korisnicima, a kod C-listi, strelice ukazuju od korisnika do objektima (resursima). Ova naizgled trivijalna razlika ima veliki značaj.

Sa C-listama, povezanost između korisnika i datoteka je ugrađena u sistem, dok je za sistem sa ACL listama, potreban poseban metod za integrisanje podataka sa fajlovima. Ova osobina ilustruje prednosti i mogućnosti obe metode. C-liste imaju nekoliko prednosti u pitanju bezbednosti i iz tog razloga C-liste su mnogo popularnije.



Slika 8.19. ACL naspram mogućnosti (C lista).

Ma sledećoj slici je prikazan problem koji može da nastane primenom ACL liste. Jednostavno, Alice nema pristuma resursima Bill-a prema matrici, ali korišćenjem indirektnog pristupa preko Compiler-a može pročitati resurse koje poseduje Bill.

TABELA 8.2. Kontrola pristupa matrica preko zamenika .

Matrica kontrole pristupa		
	Compiler	Bill
Alice	x	—
Compiler	rx	rw

8.15 Bell-LaPadula model sigurnosti

Prvi model sigurnosti koji ćemo napomenuti je *Bell-LaPadula*, ili BLP, koji je dobio ime po svojim pronalazačima, Elliot Bell i Len LaPadula-i. Svrha BLP je određivanje minimalnih

uslova u pogledu poverljivosti koji svaki model sigurnosti mora da zadovolji. BLP se sastoji od sledećih dve prepostavke:

Jednostavan uslov bezbednosti : Predmet M može da čita objekat O ako i samo ako $P(o) \leq L(s)$.

Osobine: Subjekat N može menjati objekat O ako i samo ako je $L(s) \leq P(o)$.

Ova jednostavna pravila omogućavaju građenje hijerarhije besbednosti.

8.16 COVERT KANAL

Definisaćemo *tajni kanal* kao komunikacionu liniju namenjenu ulogama koje dizajneri sistema nisu predvideli. Tajni kanali nastaju u mnogim situacijama, naročito u mrežnim komunikacijama. Tajne kanale je praktično nemoguće eliminisati, a akcenat je umesto toga na ograničavanje kapaciteta tih kanala.

8.17 CAPTCHA

Turingov test je predložio pionir u računskim naukama (i glavni kriptoanalitičar Enigme) Alan Turing 1950. Test je koncipiran tako da čovek postavlja pitanja drugom čoveku i jednom računaru. Ispitivač, koji ne može da vidi ispitankice, mora da pokuša da odredi koji ispitnik je čovek a koji je računar.

Ako ljudski ispitivač ne može da reši ovo pitanje, računar prolazi Tjuringov test. Ovaj test je "zlatni standard" veštacke inteligencije, ali nijedan kompjuter još nije došao ni blizu da prođe Tjuringov test. Slično Tjuringovom testu, ali sa drugačijom idejom, koncipiran je *CAPTCHA* test koji čovek može da prođe, ali ne i računar sa verovatnoćom većom od pogodanja. Ovo se može smatrati kao suprotnost Turingovom testu. Prepostavka je da je test nastao računarskim programom i ocenjen od strane drugog računarskog programa koji ne može da prođe taj test. Drugim rečima, "CAPTCHA" je program koji može da generiše i gradi testove koji sama ne može da prođe. Čini se paradoksalno da računar može da stvori test koji sam ne može proći.



Slika 8.20 CAPTCHA (Ljubaznošću Luis von Ahn [1]).

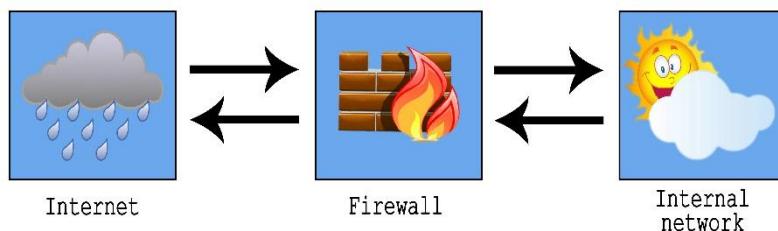
Objašnjenje ovog paradoksa je da je mnogo lakše izvršiti generisanje RANDOM slova sa smetnjama, nego ih prepoznati sistemom za optičko prepoznavanje [28],[29].

8.18 FIREWALL

Pretpostavimo da želite da se sastanete sa presedavajućim sa odeljenja informatike na nekom fakultetu. Prvo, verovatno ćete se morati obratiti sekretaru fakulteta. Ako sekretar smatra da je sastanak opravдан, on će ga zakazati - inače se to neće desiti. Na ovaj način, sekretar filtrira mnoge zahteve koji ne bi trebalo da budu realizovani.

Firewall deluje kao da sekretar ima svoju mrežu (filter). Firewall ispituje zahteve za pristup mreži i odlučuje da li će proći test prolaznosti. Drugim rečima, odlučuje koji će paket proći prema kojoj aplikaciji.

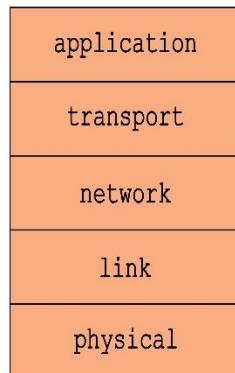
- *Packet filter* je zaštitni zid koji deluje u mrežnom sloju.
- *Stateful Packet Filter* je zaštitni zid koji radi u transportnom sloju.
- *Proxy aplikacija* je, kao što ime sugerira, zaštitni zid koji funkcioniše na aplikacionom sloju gde funkcioniše kao *Proxy*.



Slika 8.21 Firewall.

8.18.1 Paket filter

Paket filter za *firewall* ispituje pakete sve do mrežnog sloja, kao što je prikazano na sledećoj slici. Kao rezultat toga, ova vrsta zaštitnog zida može samo da filtrira pakete na osnovu informacija koje si na mrežnom sloju.



Slika 8.22 Paket filter – hijerarhijski slojevi.

8.19 Jednostavnii sigurnosni protokoli

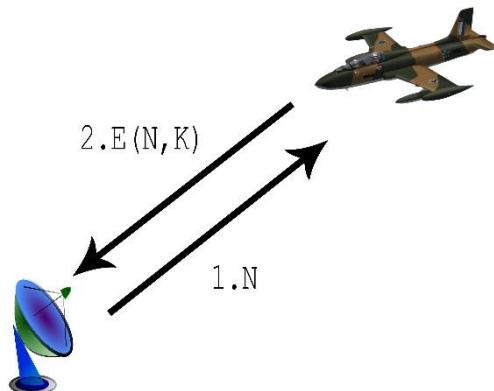
Bezbednosni protokol koji ćemo razmotriti je protokol koji se može koristiti za ulazak u siguran objekat. Zaposleni su dobili značku koju moraju nositi u svakom trenutku kada su u sigurnom objektu. Da bi se ušlo u zgradu, značka mora da se ubaci u čitač kartica i zaposleni moraju da obezbede PIN broj. Protokol se može opisati na sledeći način:

1. Stavi se značka u čitač
2. Unese se PIN
3. Da li je PIN ispravan?
 - Da: Unese se broj zgrade
 - Nema: Obezbeđenje reaguje

Takođe, odličan primer su bankomati koji su danas u masovnoj upotrebi. Kada se podiže novac iz bankomata, protokol je praktično identičan već prikazanom protokolu:

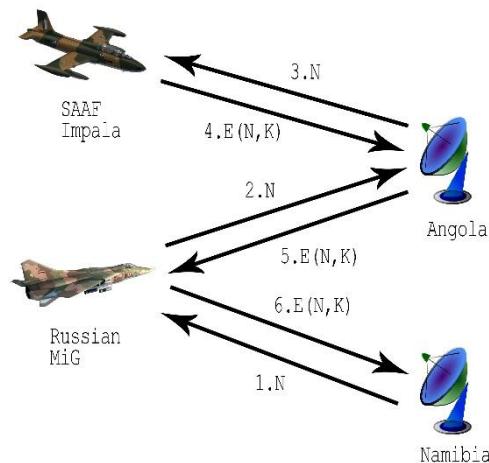
1. Ubacite bankomat karticu u čitač,
 2. Unesite PIN,
 3. Da li je PIN ispravan?
 - Da: Sprovesti svoje transakcije
 - Ne: Mašina zadržava ATM karticu, posle tri uzastopna neuspešna pokušaja identifikacije.
- Standardni protocol za radare je takođe ilustrativni primer realizacije jednostavnog protokola. Radar zrači digitalni numerik N koji stiže do aviona u kontrolnoj oblasti i avion

reaguje tako što šalje proračunati rezultat. Taj rezultat se poredi u radaru, i ako se ključevi slože u pitanju je prijateljski avion, u suprotnom protivnički.



Slika 8.22 Identifikuje se prijatelj ili neprijatelj.

U praksi je zabeležen uspešan napad na IFF sistem koji je prikazan na predhodnoj slici. Anderson je nazvao ovaj napad je "*Mig-in-the-middle*", što je igra reči od *man-in-the-middle*. Scenario za napad, ilustrovan je sledećoj slici.



Slika 8.23 MIG-in-the-middle protokoli za autentifikaciju

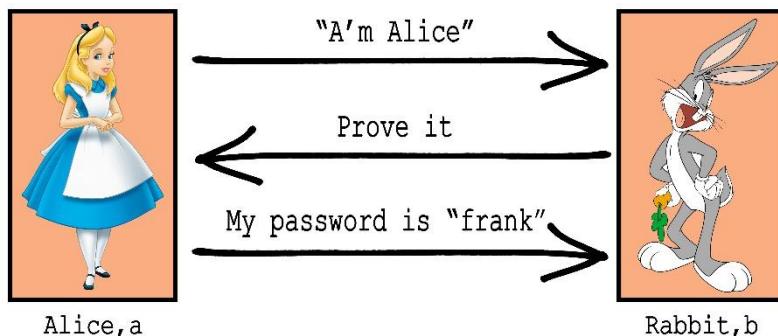
Avion MiG (protivnik u SAAF) je izvan dometa radara SAAF u Namibiji, dok je *SAAF Impala* borac u misiji nad Angolom. Kada je avion Impala u dometu kubanske radarske stanice u Angoli, MIG ulazi u domet SAAF radara. Kao što je navedeno u protokolu sa komunikaciju, SAAF radar šalje upit N . Kako bi se izbeglo obaranje, MIG treba da odgovori sa $E(n, k)$ i brzo. Zbog toga što MIG ne zna ključ za upit K , njegova situacija se čini beznadežnom.

Međutim, sistem SAAF se može zavarati na sledeći način. U momentu kada MIG prihvati upit K, taj upit odmah prosleđuje prijateljskom radaru u Angoli (korak 2). Radar u Angoli zatim zrači upit K u prostor (korak 3). Protivnički avion na taj upit odgovara sifrom E(N,K) (korak 4). Prihvaćeni odgovor radar u Angoli prosleđuje MIG-u (korak 5) koji ga odmah vraća protivničkom radaru (korak 6). Na taj način je MIG iskoristio dešifrator protivničkog aviona Impala, da bi se protivničkom radaru predstavio kao prijateljski avion. Tako je MIG neopaženo isvršio zadatak u protivničkom vazdušnom prostoru.

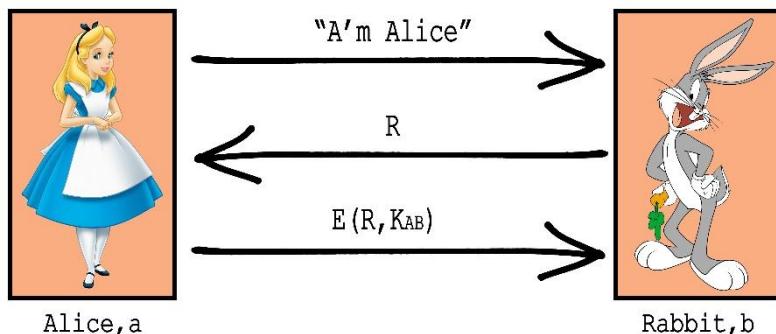
8.20 Protokoli za utvrđivanje autentičnosti

Pretpostavimo da Alisa mora da dokaže Bobu da je ona Alisa. Po pravilu, Alisa i Bob komuniciraju preko mreže. Treba imati na umu da Alisa može da bude čovek ili mašina. U mnogim slučajevima, dovoljno je da Alisa dokaže svoj identitet Bobu, bez potrebe da Bob dokazuje svoj identitet. Ali, ponekad je *uzajamna autentifikacija* potrebna; to jest, Bob takođe mora da dokaže svoj identitet Alice. Čini se očiglednim da ako Alisa može da dokaže svoj identitet Bobu, upravo isti protokol se može koristiti u drugom smjeru za Boba. Međutim, u okviru sigurnosnih protokola, očigledan pristup često nije siguran.

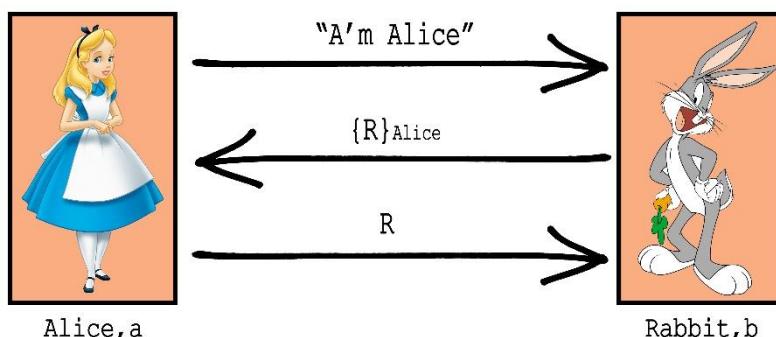
Za proveru identiteta potreban je simetrični ključ. Simetrični ključ se obično koristi kao ključ sesije, koji figuriše kao ključ za zaštitu poverljivosti ili integriteta (ili oba) za trenutnu sesiju. U određenim situacijama, moguće je da postoje zahtevi koji se posebno postavljaju pred protokol bezbednosti. Na primer, može se zatražiti da protokol koristi javne ili simetrične ključeve. Nekoliko jednostavnih protokola je prikazano na slikama u nastavku:



Slika 8.24. Prosta autorizacija.



Slika 8.25. Autentifikacija sa simetričnim ključem.



Slika 8.26. Autentifikacija sa javnim ključem - šifrovanje.

8.21 Greške u softveru i *Malware*

U zaštiti informacija, softver se tretira na istom nivou kao kriptografija, kontrola pristupa i protokoli. Praktično sve procedure informacione bezbednosti su implementirane u softveru. U stvari, softver je temelj na koji se oslanjaju svi drugi mehanizmi bezbednosti. Pokazalo se ipak da softver daje veoma slabu osnovu na kojoj se gradi pouzdana sigurnost podataka.

U nastavku prikazaćemo osnovne detalje o nekoliko bezbednosnih pitanja u vezi sa softverom. Prvo, razmotrićemo nenamerne softverske greške koje mogu dovesti do bezbednosnih problema. Klasičan primer takvog tipa grešaka je u vezi grešaka sa *bafer*-om, koji ćemo diskutovati u nekim detaljima. Dalje prikazaćemo osnovne tipove zlonamernog softvera (*malware*), koji je namerno dizajniran da deluje protiv sistema.

8.22 Softverske greške

Greške u softveru su masovna pojava. Na primer, NASA Mars Lander, koji je koštao 165 miliona dolara, udario je Mars zbog greške u softveru u pretvaranju između engleskog i metričkog sistema mere. Još jedan neslavni primer lošeg softvera se desio na aerodromu u Denveru u sistemu za transport prtljaga. Greske u ovom softveru odložile su otvaranje aerodroma za 11 meseci po ceni od oko 1 miliona dolara dnevno. Opasnije greške u software-u su se desile i u SN-22 Osprey, naprednim vojnim avionima sa vertikalnim poletanjem. Životi pilota su izgubljeni zbog tog pogrešnog softvera. Postoji još mnogo primera takvih propusta.

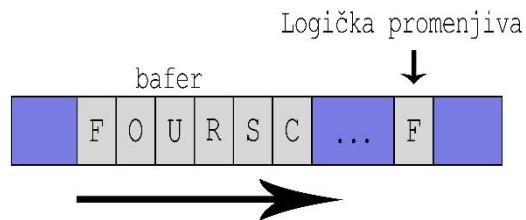
U ovom delu, prikazaćemo neke bezbednosne implikacije softverskih grešaka. Sa greškama softvera koje su već prisutne, nije čudno što su napadači na računske sisteme brzo našli mogućnosti da iskoriste ovu situaciju. Obični korisnici softvera mogu pronaći greške i propuste više ili manje slučajno. Napadači, sa druge strane, vide loš softver kao mogućnost, a ne kao problem. Oni aktivno traže greške i propusta u softveru, i vole da rade sa lošim softverom. Napadači pokušavaju da pronađu propuste u softver-u i ti propusti se obično pokazuju veoma korisnim u tom pogledu. Poznato je da je sistemski softver u središtu mnogih napada. Opšte prihvaćena teza među bezbednosnim profesionalacima je da je "kompleksnost neprijatelj bezbednosti", a moderan softver je izuzetno složen. U stvari, složenost softvera je već prevazišla sposobnosti ljudi da upravljaju tom kompleksnošću.

Konzervativna procena određuje broj grešaka u softveru sa 5 na 1.000 linija koda. Ako se proširi ova kalkulacija u srednje korporativne mreže sa 30.000 čvorova, očekujemo oko 4,5 milijardi grešaka u mreži. Naravno, mnogi od tih grešaka će biti duplikati, ali to je ipak zapanjujući broj. Ako prepostavimo da je samo 10% od tih grešaka kritične i da samo 10% od njih bezbednosno-kritično to nas vodi do nivoa od 4,5 miliona ozbiljnih nedostataka bezbednosti zbog lošeg softvera u mreži!

Isdvaja se nekoliko grešaka u softveru koje mogu imati bezbednosne implikacije:

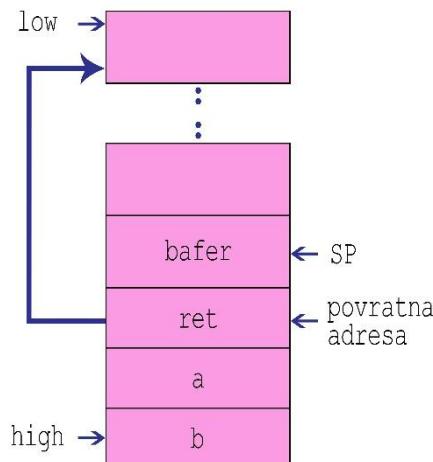
- Prekoračenje bafera (*Buffer overflow*),
- *Race uslovi*,
- Nepotpuno posredovanje.

U praksi se u velikoj većini slučajeva u toku programiranja previda situacija oko dimenzija radnih nizova. Tipičan scenario u kome može doći do takvog napada možemo prikazati na sledeći način. Pretpostavimo da program traži od korisnika da unese podatke, kao što su ime, starost, datum rođenja i tako dalje. Ovi podaci se zatim šalju na server koji upisuje podatke u bafer koji može da primiti N znakova. Ako server ne potvrđi da je dužina podataka N znakova, onda u baferu može doći do greške tako što će produžena veličina radnog niza jednostavno "prebrisati" već postojeću logičku promenjivu T . To je prikazano na sledećoj slici:



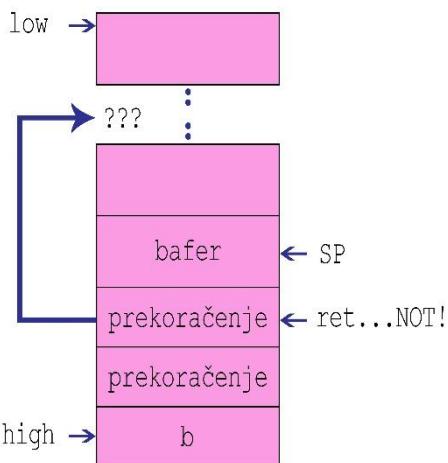
Slika 7.27. Logičke promenjive

Dalje, čest je slučaj pogrešnog rada sa stekom usled grešaka u dimenzijama steka čime dolazi do upisivanja novih vrednosti preko već postojećih aktivnih vrednosti. To je prikazano na sledećim slikama:



Slika 7.28. Stek – primer

U slučaju da je došlo do greške na steku ,povratne adrese i druge informacije koje su ranije tamo smeštene biće korumpirane, a samim tim i program će početi da ispoljava greške ili da se jednostavno blokira.



Slika 8.29. Prekoračenje steka izaziva problem.

8.23 MALWARE

U ovom odeljku ćemo nabrojati primere softvera koji su dizajnirani da probiju linije bezbednosti sa namerom činjenja štete, brisanja podataka ili drugih malicioznih zahvata u originalnom sistemu. Pošto je ovaj tip softvera zlonameran u svojoj osnovi, možemo ga klasifikovati pod imenom *malware*.

Malware se može podeliti u više različitih kategorija. Mi ćemo koristiti sledeći klasifikacioni sistem, iako postoji značajno preklapanje između različitih tipova.

- Kompjuterski virus je malware koji se oslanja na neki drugi softver sa ciljem da se širi iz jednog sistema u drugi. Na primer, virus se prikači na e-mail koji se šalje od jednog korisnika ka drugom, čime se inficiraju svi računari koji su bili u kontaktu sa tom porukom. Donedavno, virusi su bili najpopularniji oblik malware-a.
- *Crv (worm)* je sličan virusu osim što se širi sam bez potrebe za spoljnom pomoći. Ovaj tip kompjuterskih virusa je postao najpopularniji među hakerima.
- *Trojanski konj (trojan horse)*, ili trojan, je softver koji izgleda normalno, ali ima neku neočekivanu funkcionalnost. Na primer, obična video igra preuzeta kao freeware-a sa interneta može učini nešto zlonamerno na sistemu dok se žrtva (vlasnik sistema) igra.
- *Zadnja vrata (trapdoor or backdoor)* omogućava neovlašćen pristup sistemu kroz posebno dizajnirani prolaz kroz firewall.
- *Zec (rabbit)* je zlonamerni program koji iscrpljuje sistemske resurse. Zec može biti biti implementiran kroz viruse, crve, ili drugim sredstvima.

9 ZADACI SA VEŽBI

9.1 One time pad

9.1.1 Zadatak 1

Kriptovanjem četiri reči po one-time-pad algoritmu dobijene su sledeće vrednosti: acestse, aeikee, ciiee, eeesk. Ako se zna da je jedna od 4 polazne reči krasta i da je korišćena sledeća azbuka [e, c, i, k, a, r, s, t] (kodirana sa tri bita), odrediti ostale tri reči.

Rešenje:

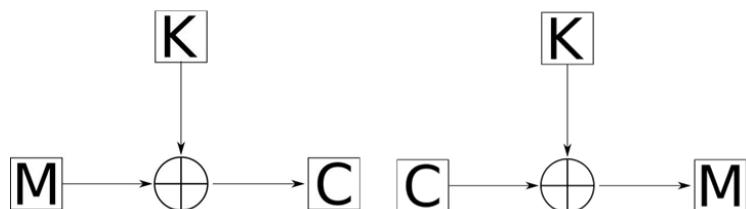
- Algoritam - One-time-pad
- Kriptovane reči - acestse, aeikee, ciiee, eeesk
- Azbuka - [e, c, i, k, a, r, s, t]

Slova	Binarne vrednosti	
e	0 0 0	
c	0 0 1	
i	0 1 0	
k	0 1 1	
a	1 0 0	
r	1 0 1	
s	1 1 0	
t	1 1 1	

Zna se da je jedna od 4 kriptovane reči **krasta**. Ova reč je dužine 6 karaktera, tako da potencijalne kriptovane reči koje odgovaraju su **acestse** (6 karaktera) ili **aeikee** (6 karaktera).

One-time-pad algoritam je algoritam koji pripada kategoriji simetričnih kripto algoritama, kako se kodira (encode) podatak isti tako se i dekodira (decode).

- PODATAK (**M**)
- KLJUČ (**K**)
- KODIRANI PODATAK (**C**)



Slika 9.1 Proces kodiranja i dekodiranja kod One-Time-Pad algoritma

Ključ ćemo dobiti xor-ovanjem kodirane reč i date reči **krasta**.

	a	c	e	t	s	e
podatak	1 0 0	0 0 1	0 0 0	1 1 1	1 1 0	0 0 0
Data reč	k	r	a	s	t	a
Ključ	0 1 1	1 0 1	1 0 0	1 1 0	1 1 1	1 0 0

Ključ koji smo dobili je **K = [000, 011, 011, 110, 110, 011]**. Moramo da proverimo da li je ovo pravi ključ. Kada xor-ujemo kodiranu reč **aeikee** sa **K**, ako je ovo pravi ključ trebali bi da dobijemo smisleni podatak, smislenu reč.

Provera ključa

	a	e	i	k	e	e
podatak	1 0 0	0 0 0	0 1 0	0 1 1	0 0 0	0 0 0
Ključ	0 0 0	0 1 0	0 1 1	1 1 0	1 1 0	0 1 1
Dekodir ana reč	0 1 1	1 0 1	1 1 0	0 1 0	0 0 1	1 0 0
	k	r	s	i	c	a

9. ZADACI SA VEŽBI

Reč koju smo dobili dekodiranjem date reči **aeikee** je **KRSICA**. Ova reč u srpskom, engleskom jeziku nije smislena, tako da zaključujemo da ovaj ključ nije pravi. Jedina preostala kombinacija koju možemo da proverimo jeste da xor-ujemo reč **aeikee** (6 karaktera) i poznatu reč **krasta** i dobijemo novi ključ **K**.

	a	c	i	k	e	e
podatak	1 0 0	0 0 1	0 1 0	0 1 1	0 0 0	0 0 0
Data reč	k	r	a	s	t	a
	0 1 1	1 0 1	1 0 0	1 1 0	1 1 1	1 0 0
Ključ	0 0 0	0 1 0	0 0 1	0 1 0	0 0 0	0 1 1

Dobijen ključ je **K = [000, 010, 001, 010, 000, 011]**. Ukoliko je ovo pravi ključ, xor-ovanjem preostale tri kodirane reči i ključa dobićemo smislene reči, to će ujedno biti i rešenje zadatka.

Provera ključa

	a	c	e	t	s	e
podatak	1 0 0	0 0 1	0 0 0	1 1 1	1 1 0	0 0 0
Ključ	0 0 0	0 1 0	0 0 1	0 1 0	0 0 0	0 1 1
Dekodirana reč	0 1 1	1 0 0	1 1 0	0 1 0	0 0 1	1 0 0
	k	a	s	i	c	a

KASICA je smislena reč. Čini se da ovaj ključ jeste pravi, nastavićemo dekodiranje preostale dve reči.

Kodirani	c	i	i	e	e	-
podatak	0 0 1	0 1 0	0 1 0	0 0 0	0 0 0	
Ključ	0 0 0	0 1 0	0 0 1	0 1 0	0 0 0	-
Dekodirana	1 0 0	1 1 1	1 0 0	1 0 1	1 1 1	-
	s	t	a	r	t	

ZAŠTITA INFORMACIJA

START je takođe smislena reč na engleskom. Nastavljamo dalje dekodiranje.

Kodirani podatak	e	e	e	s	k	-
0 0 0	0 0 0	0 0 0	1 1 0	0 1 1		
Ključ	0 0 0	0 1 0	0 0 1	0 1 0	0 0 0	-
Dekodirana	1 1 1	1 0 1	1 1 0	0 1 1	1 0 0	-
reč	t	r	s	k	a	

TRSKA je takođe smislena reč. Došli smo do kraja, sve reči su dekodirane.

Zaključak

Ključ:

$K = [000, 010, 001, 010, 000, 011]$ Kodirane reči:

aceutse, aeikee, ciiee, eesk

aceutse \oplus K = kasica
aeikee \oplus K = krasta
ciiee \oplus K = start
eesk \oplus K = trska

Dekodirane reči:

kasica, krasta, start, trska

9.1.2 Zadatak 2

Kriptovanjem četiri reči po one-time pad algoritmu dobijene su sledeće vrednosti: nmmmtm, lktmam, mssalt, msmmmm. Ako se zna da je jedna od 4 polazne reči ananas i da je korišćena sledeća abzuka [n, a, k, s, t, l, m, i] (kodirana sa 3 bita), odrediti ostale tri reči.

Rešenje:

Originalne reči se ne mogu direktno odrediti jer u zadatku nije dat ključ koji je korišćen za kodiranje one-time pad algoritmom. Za dekodiranje se može iskoristiti činjenica da je za kodiranje sve četiri reči korišćen isti ključ.

Označimo sa M_1, M_2, M_3, M_4 originalne reči, a sa C_1, C_2, C_3, C_4 kodirane reči.

Za bilo koje dve reči tada važi

$$C_i = M_i \oplus K$$

$$C_j = M_j \oplus K$$

Na ove dve jednačine se može primeniti xor operacija

$$C_i \oplus C_j = M_i \oplus K \oplus M_j \oplus K$$

$$C_i \oplus C_j = M_i \oplus M_j \oplus K \oplus K$$

$$C_i \oplus C_j = M_i \oplus M_j$$

čime smo se oslobođili ključa.

Uzmimo da je, na primer, $M_j = \text{ananas}$ (isto bi važilo i za $M_i = \text{ananas}$).

Ako se sada na obe strane poslednje jednačine primeni xor operacija sa rečju ananas, dobija se

$$C_i \oplus C_j \oplus \text{ananas} = M_i \oplus M_j \oplus \text{ananas}$$

$$C_i \oplus C_j \oplus \text{ananas} = M_i$$

Na ovaj način smo predstavili nepoznatu reč M_i preko poznatih elemenata C_i , C_j , i ananas .

Dekodiranje se sada vrši tako što se za svaku kombinaciju indeksa i i j i izračuna vrednost $C_i \oplus C_j \oplus \text{ananas}$. Kada se dobije smislena reč kao rezultat znači da smo na levoj strani izraza imali šifrovanu reč ananas ali i šifru rezultujuće reči M_i .

Data azbuka se kodira binarno:

n 000

a 001

k 010

s 011

t 100

l 101

m 110

i 111

Na osnovu toga se nađu kodovi za svaku od šifriranih reči i za reč ananas:

$C_1 = nnmmtm$	0	0	0	0	0	0	1	1	0	1	1	0	1	0	0	1	1	0
$C_2 = lktnam$	1	0	1	0	1	0	1	0	0	1	1	0	0	0	1	1	1	0
$C_3 = mssalt$	1	1	0	0	1	1	0	1	1	0	0	0	1	1	0	1	0	0
$C_4 = msmmmm$	1	1	0	0	1	1	1	1	0	1	1	0	1	1	0	1	1	0
<i>ananas</i>	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1

Sada mogu da se isprobaju kombinacije.

Prva kombinacija: $C_1 \oplus C_2 \oplus ananas$

C_1	0	0	0	0	0	0	1	1	0	1	1	0	1	0	0	1	1	0
C_2	1	0	1	0	1	0	1	0	0	1	1	0	0	0	1	1	1	0
<i>ananas</i>	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1
$C_1 \oplus C_2 \oplus ananas$	1	0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	1	1
	<i>t</i>		<i>k</i>		<i>i</i>		<i>n</i>		<i>t</i>		<i>s</i>							

Dobijena je besmislena reč, pa zaključujemo da ni C_1 ni C_2 ne predstavljaju šifrovanu reč ananas.

Druga kombinacija: $C_1 \oplus C_3 \oplus ananas$

C_1	0	0	0	0	0	0	1	1	0	1	1	0	1	0	0	1	1	0
C_3	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	1	1	0
<i>ananas</i>	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1
$C_1 \oplus C_3 \oplus ananas$	1	1	1	0	1	1	1	0	0	1	1	1	0	0	0	0	0	1
	<i>i</i>		<i>s</i>		<i>t</i>		<i>i</i>		<i>n</i>		<i>a</i>							

Dobijena je smislena reč, pa zaključujemo da C_3 predstavlja kodiranu reč **ananas**, a C_1 kodiranu reč **istina** (prethodno smo već zaključili da C_1 ne može da predstavlja šifriranu reč ananas).

Treća kombinacija: $C_2 \oplus C_3 \oplus \text{ananas}$

C_2	1	0	1	0	1	0	1	1	0	0	0	1	1	0
C_3	1	1	0	0	1	1	0	1	1	0	0	1	1	0
ananas	0	0	1	0	0	0	0	0	1	0	0	0	0	1
$C_2 \oplus C_3 \oplus \text{ananas}$	0	1	0	0	0	1	1	0	1	1	1	1	0	1

Na osnovu ovoga zaključujemo da C_2 predstavlja kodiranu reč **kamila**.

Četvrta kombinacija: $C_3 \oplus C_4 \oplus \text{ananas}$

C_3	1	1	0	0	1	1	0	1	1	0	1	1	0	0
C_4	1	1	0	0	1	1	1	1	0	1	1	0	1	1
ananas	0	0	1	0	0	0	0	0	1	0	0	0	0	1
$C_3 \oplus C_4 \oplus \text{ananas}$	0	0	1	0	0	0	1	0	0	1	1	1	0	0

Na osnovu ovoga zaključujemo da C_4 predstavlja kodiranu reč **antika**.

Dakle, originalne reči su: **istina, kamila, ananas, antika**.

9.1.3 Zadatak 3

Kriptovanjem četiri reči po one-time-pad algoritmu dobijene su sledeće vrednosti: stnatac, nictsa, rnctgg, atiali. Ako se zna da je jedna od 4 polazne reči string i da je korišćena abzuka [a, n, i, c, g, r, s, t] (kodirana sa tri bita), odrediti ostale tri reči.

Najpre napišimo kodnu tablicu:

N	0	1	2	3	4	5	6	7
Slovo	a	n	i	c	g	r	s	t
Kod	000	001	010	011	100	101	110	111

Z A Š T I T A I N F O R M A C I J A

I reč string zapišimo datim kodom:

Slovo	s	t	r	i	n	g
Kod	110	111	101	010	001	100

Pretpostavimo da je “stnac” rezultat dekodiranja reči “string”. Da bi to bilo tačno, ključ mora biti:

Slovo	s	t	n	a	t	c
Kod	110	111	001	000	111	011
String	110	111	101	010	001	100
Ključ	000	000	100	010	110	111

Dobijamo da je ključ 000 000 100 010 110 111. Proverićemo da li dekodiranjem ostalih reči dobijemo smislene reči (reči koje imaju neko značenje). Dekodiranjem reči “nictsa” dobijamo:

Slovo	n	i	c	t	s	a
Kod	001	010	011	111	110	000
Ključ	000	000	100	010	110	111
Rezultat	001	010	111	101	000	111
Reč	n	i	t	r	a	t

S' obzirom da reč “nitrat” ima značenje, nastavljamo dekodiranje istim ključem. Sledеća reč je “rnctgg”.

Slovo	r	n	c	t	g	g
Kod	101	001	011	111	100	100
Ključ	000	000	100	010	110	111
Rezultat	101	001	111	101	010	011
Reč	r	n	t	r	i	c

9. ZADACI SA VEŽBI

Kako reč „rntric” nema smisleno značenje, zaključujemo da 000 000 100 010 110 111 nije traženi ključ. Pretpostavimo sada da je „nictsa” vrednost koja se dobija kodiranjem reči „string”. Da bi se to ostvarilo, ključ mora biti:

Slovo	n	i	c	t	s	a
Kod	001	010	011	111	110	000
String	110	111	101	010	001	100
Ključ	111	101	110	101	111	100

Dobili smo ključ 111 101 110 101 111 100. Proveravamo da li dekodiranjem ostalih reči dobijemo smislene reči. Dekodiranjem reči „stnate” dobijamo:

Slovo	s	t	n	a	t	c
Kod	110	111	001	000	111	011
Ključ	111	101	110	101	111	100
Rezultat	001	010	111	101	000	111
Reč	n	i	t	r	a	t

Kako reč “nitrat” ima značenje nastavljamo sa daljim dekodiranjem. Sledеća reč je „rnctgg”:

Slovo	r	n	c	t	g	g
Kod	101	001	011	111	100	100
Ključ	111	101	110	101	111	100
Rezultat	010	100	101	010	011	000
Reč	i	g	r	i	c	a

Reč “igrica” ima smisao, pa nastavljamo sa daljim dekodiranjem. Sledеća reč je „atiari”.

Slovo	a	t	i	a	r	t
Kod	000	111	010	000	101	111
Ključ	111	101	110	101	111	100
Rezultat	111	010	100	101	010	011
Rec	t	i	g	r	i	s

Uspešnim dekodiranjem svih reči, zaključujemo da je **111 101 110 101 111 100** zaista ključ koji je korišćen za kodiranje. Dobijeni rezultat nakon dekodiranja je sledeći: stnatac => **nitrat**, nictsa => **string**, rnctgg => **igrica**, atiali => **tigris**.

9.1.4 Zadatak 4

Kriptovanjem četiri reči po one-time-pad algoritmu dobijene su sledeće vrednosti: dalce, ecsci, pcpci, elpde. Ako se zna da je jedna od 4 polazne reči sesil i da je korišćena sledeća abzuka (p, c, I, s, a, l, e, d) kodirana sa tri bita, odrediti ostale tri reči.

Azbuka:

p	000
c	001
i	010
s	011
a	100
l	101
e	110
d	111

1)	Šifrirani tekst:	d	a	l	c	e
		111	100	101	001	110
	Originalni tekst:	s	e	s	i	l
		011	110	011	010	101
		100	010	110	011	011 – pad(ključ)

Provera:

Šifrirani tekst:	e	c	s	c	i
	110	001	011	001	010
Pad:	100	010	110	011	011
	010	011	101	010	001
	i	s	l	i	c

Dobijeni tekst nema smisla, pa zaključujemo da dobijeni ključ nije ispravan.

9. ZADACI SA VEŽBI

2)	Šifrirani tekst	e	c	s	c	i
		110	001	011	001	010
	Originalni tekst:	s	e	s	i	1
		011	110	011	010	101
		101	111	000	011	111 – pad(ključ)

Provera:

Šifrirani tekst:	p	c	p	c	i
	000	001	000	001	010
Pad:	101	111	000	011	111
	101	110	000	010	101
	1	e	p	i	1

Dobijeni tekst nema smisla, pa zaključujemo da dobijeni ključ nije ispravan.

3)	Šifrirani tekst:	p	c	p	c	i
		000	001	000	001	010
	Originalni tekst:	s	e	s	i	1
		011	110	011	010	101
		011	111	011	011	111 – pad(ključ)

Provera:

Šifrirani tekst:	d	a	l	c	e
	111	100	101	001	110
Pad:	001	111	011	011	111
	100	011	110	010	001
	a	s	e	i	c

Dobijeni tekst nema smisla, pa zaključujemo da dobijeni ključ nije ispravan.

4)	Sifrirani tekst:	e	l	p	d	e
		110	101	000	111	110
	Originalni tekst:	s	e	s	i	1
		011	110	011	010	101
		101	011	011	101	011 – pad(ključ)

Provera:

Šifrirani tekst:	d	a	l	c	e
	111	100	101	001	110
Pad:	101	011	011	101	011
	010	111	110	100	101
	i	d	e	a	l - Ispravna reč

Provera2:

Šifrirani tekst:	e	c	s	c	i
	110	001	011	001	010
Pad:	101	011	011	101	011
	011	010	000	100	001
	s	i	p	a	c - Ispravna reč

Provera3:

Šifrirani tekst:

p	c	p	c	i
000	001	000	001	010

Pad:

101	011	011	101	011
101	010	011	100	001

1 i s a c - Ispravna reč

Iz ovoga zaključujemo da za izabrani ključ: 101 011 011 101 011 dešifrovanjem dobijamo sledeće reči: ideal, sipac, lisac.

9.1.5 Zadatak 5

Data je azbuka za „one-time-pad“ kriptovanje: a = 000, t = 001, n = 010, o = 011, s = 100, v = 101, d = 110 i e = 111. Istim ključem (pad-om) kriptovane su četiri naše reči (2 sa 5 i 2 sa 6 slova) i dobijeni rezultati su ananv, stovv, vodeea i ttovvs. Ako znamo da je jedna od te četiri reči „tavan“, odrediti preostale tri, kao i korišćeni ključ (pad).

Rešenje:

azbuka:

a	000
t	001
n	010
o	011
s	100
v	101
d	110
e	111

kodirane reči:

ananv stovv vodeea ttovvs

Xor(\oplus) tablica:

		x1,x2 xor(\oplus)
		0 0 0
		0 1 1
		1 0 1
		1 1 0

poznata reč:

tavan

osobine:

(P-početni tekst, K-ključ, C-kriptovani tekst)

$$P \oplus K = C$$

$$P = C \oplus K$$

Iz ove dve osobine možemo izvesti još jednu osobinu koja je nama potrebna , a to je da ključ možemo dobiti kada \oplus (xor-ujemo) polazni tekst i kriptovani tekst.

$$K = P \oplus C$$

U samom tekstu su nam date po dve reči dužine 5 slova i dve reči dužine 6 slova, a kako je poznata reč „tavan“ ona je dužine 5 slova, to znači da će jedna od reči „ananv“ ili „stovv“ biti baš ta reč, jer su samo te dve reči dužine 5 slova. Da bismo dobili ključ, moramo primeniti \oplus (xor) operaciju nad rečima „tavan“ i „ananv“ ili „tavan“ i „stovv“ (u ovom slučaju dobićemo dva ključa od kojih će jedan moći da dešifruje sve ostale reči, ali da bismo otkrili koji je od ta dva ključa ispravan moramo isprobavati kombinacije).

Prvo dobijamo jedan ključ na osnovu reči „tavan“ i „ananv“:

9. ZADACI SA VEŽBI

$$\begin{array}{l} \text{anav - } 000\ 010\ 000\ 010\ 101 \\ \text{tavan - } 001\ 000\ 101\ 000\ 010 \\ \oplus \quad \cdots \cdots \cdots \\ \quad \quad \quad 001\ 010\ 101\ 010\ 111\ \text{ključ1} \end{array}$$

Ovaj ključ koji smo dobili ključ1 primenjujemo na ostale reči:

$$\begin{array}{l} \text{stovv - } 100\ 001\ 011\ 101\ 101 \\ \text{ključ1 - } 001\ 010\ 101\ 010\ 111 \\ \oplus \quad \cdots \cdots \cdots \\ \quad \quad \quad 101\ 011\ 110\ 111\ 010 \\ \quad \quad \quad \text{v} \quad \text{o} \quad \text{d} \quad \text{e} \quad \text{n} \end{array}$$

Pošto nam je ključ dužine 5 slova, a reč „vodeea“ je dužine 6 slova, sam ključ moramo dopuni s onoliko bita koliko su nam potrebna kako bi mogli da dekodiramo reč. Ključ se dopunjuje tako što se sam ključ ponavlja sve dok ne stigne do kraj poruke. U našem primeru su nam potrebna još tri bita i ključ se dopuni, tako što mu se na kraju dodaju prva tri bita.

$$\begin{array}{l} \text{vodeea - } 101\ 011\ 110\ 111\ 111\ 000 \\ \text{ključ1 - } 001\ 010\ 101\ 010\ 111\ \mathbf{001} \\ \oplus \quad \cdots \cdots \cdots \\ \quad \quad \quad 100\ 001\ 011\ 101\ 000\ 001 \\ \quad \quad \quad \text{s} \quad \text{t} \quad \text{o} \quad \text{v} \quad \text{a} \quad \text{t} \end{array}$$

Kako ova reč nema smisla, vratimo se na početak da probamo sada sa rečima „tavan“ i „stovv“ i ključem koji se dobija iz njih.

$$\begin{array}{l} \text{stovv - } 100\ 001\ 011\ 101\ 101 \\ \text{tavan - } 001\ 000\ 101\ 000\ 010 \\ \oplus \quad \cdots \cdots \cdots \\ \quad \quad \quad 101\ 001\ 110\ 101\ 111\ \text{ključ2} \end{array}$$

$$\begin{array}{l} \text{anav - } 000\ 010\ 000\ 010\ 101 \\ \text{ključ2 - } 101\ 001\ 110\ 101\ 111 \\ \oplus \quad \cdots \cdots \cdots \\ \quad \quad \quad 101\ 011\ 110\ 111\ 010 \\ \quad \quad \quad \text{v} \quad \text{o} \quad \text{d} \quad \text{e} \quad \text{n} \end{array}$$

$$\begin{array}{l} \text{vodeea - } 101\ 011\ 110\ 111\ 111\ 000 \\ \text{ključ2 - } 101\ 001\ 110\ 101\ 111\ 101 \\ \oplus \quad \cdots \cdots \cdots \\ \quad \quad \quad 000\ 010\ 000\ 010\ 000\ 101 \\ \quad \quad \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{n} \quad \text{a} \quad \text{v} \end{array}$$

ZAŠTITA INFORMACIJA

ttovvs - 001 001 011 101 101 100
ključ2 - 101 001 110 101 111 101
 \oplus -----
100 000 101 000 010 001
s a v a n t

Kako ovde može da se primeti da reči „ananav“ i „savant“ podsećaju na naše reči „ananas“ i „savana“ i to u slučaju da produženi deo ključa nije 101 već 100, onda umesto toga može se praktično namesti produženi deo ključa kako bi se doatile smislene reči, jer možda je došlo do greške na mreži tokom slanja samog ključa, a samim tim to je i najveći neprijatelj „one-time-pad“ algoritma greška na mreži.

vodeea - 101 011 110 111 111 000
ključ2 - 101 001 110 101 111 100
 \oplus -----
000 010 000 010 000 100
a n a n a s
ttovvs - 001 001 011 101 101 100
ključ2 - 101 001 110 101 111 100
 \oplus -----
100 000 101 000 010 000
s a v a n a

Dekodirane reči su:

anav - voden
stovv - tavan
vodeea - ananas
ttovvs - savana

Ključ koji je korišćen je:

ključ2 - 101 001 110 101 111 100

9.2 Double Transposition

9.2.1 Zadatak 1

„Double transposition“ metodom kriptovana je rečenica i dobijen je sledeći rezultat:

ertduvjefrigajoikpkrttoaecjtniaidnknacpenot.

Ako se zna da je korišćena matrica dimenzija 7x6 i da je sadržaj zadnje vrste pre permutovanja bio **nikadni** naći polaznu rečenicu.

9. ZADACI SA VEŽBI

Krajnje stanje:

	1	2	3	4	5	6	7
1	e	r	t	d	u	v	j
2	e	f	r	i	g	a	j
3	o	i	k	p	k	r	t
4	o	a	e	c	j	t	n
5	i	a	i	d	n	k	n
6	a	c	p	e	n	o	t

Početno stanje:

	1	2	3	4	5	6	7
1	n	i	k	a	d	n	i
2	?	?	?	?	?	?	?
3	?	?	?	?	?	?	?
4	?	?	?	?	?	?	?
5	?	?	?	?	?	?	?
6	?	?	?	?	?	?	?

Upoređujući početnu sa krajnjom matricom možemo videti da se sadržaj petog reda krajnje matrice poklapa sa sadržajem prvog reda početne matrice.

	1	2	3	4	5	6	7
5	i	a	i	d	n	k	n
1	n	i	k	a	d	n	i

Gledajući dva reda možemo uvideti sledeće mogućnosti za permutaciju kolona kako bi od početnog stigli do krajnjeg stanja:

1)

	1	2	3	4	5	6	7
5	i	a	i	d	n	k	n
1	n	i	k	a	d	n	i

→

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

2)

	1	2	3	4	5	6	7
5	i	a	i	d	n	k	n
1	n	i	k	a	d	n	i

→

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

3)

	1	2	3	4	5	6	7
5	i	a	i	d	n	k	n
1	n	i	k	a	d	n	i

→

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

4)

1	2	3	4	5	6	7	7	3	6	2	4	5	1		
5	i	a	i	d	n	k	n	1	n	i	k	a	d	n	i

→

Uzimajući u obzir ove mogućnosti permutacija dobijamo sledeće matrice:

1)

1	2	3	4	5	6	7	5	1	6	2	4	7	3	
1	e	r	t	d	u	v	1	u	e	v	r	d	j	t
2	e	f	r	i	g	a	2	g	e	a	f	i	j	r
3	o	i	k	p	k	r	3	k	o	r	i	p	t	k
4	o	a	e	c	j	t	4	j	o	t	a	c	n	e
5	i	a	i	d	n	k	5	n	i	k	a	d	n	i
6	a	c	p	e	n	o	6	n	a	o	c	e	t	p

→

2)

1	2	3	4	5	6	7	5	3	6	2	4	7	1	
1	u	t	v	r	d	j	1	u	t	v	r	d	j	e
2	g	r	a	f	i	j	2	g	r	a	f	i	j	e
3	k	k	r	i	p	t	3	k	k	r	i	p	t	o
4	j	e	t	a	c	n	4	j	e	t	a	c	n	o
5	n	i	k	a	d	n	5	n	i	k	a	d	n	i
6	n	p	o	c	e	t	6	n	p	o	c	e	t	a

→

3)

1	2	3	4	5	6	7	7	1	6	2	4	5	3	
1	j	e	v	r	d	u	1	j	e	v	r	d	t	
2	j	e	a	f	i	g	2	j	e	a	f	i	g	r
3	t	o	r	i	p	k	3	t	o	r	i	p	k	k
4	n	o	t	a	c	j	4	n	o	t	a	c	j	e
5	n	i	k	a	d	n	5	n	i	k	a	d	n	i
6	t	a	o	c	e	n	6	t	a	o	c	e	n	p

→

4)

1	2	3	4	5	6	7	→
1	e	r	t	d	u	v	j
2	e	f	r	i	g	a	j
3	o	i	k	p	k	r	t
4	o	a	e	c	j	t	n
5	i	a	i	d	n	k	n
6	a	c	p	e	n	o	t

7	3	6	2	4	5	1	→
1	j	t	v	r	d	u	e
2	j	r	a	f	i	g	e
3	t	k	r	i	p	k	o
4	n	e	t	a	c	j	o
5	n	i	k	a	d	n	i
6	t	p	o	c	e	n	a

Iz priloženog se vidi da je druga mogućnost jedina odgovarajuća za permutaciju kolona, jer jedino primenom nje može se dobiti čitljiv tekst. Ostalo je samo naći permutaciju redova. Videli smo da prvi red početne matrice odgovara petom redu krajnje matrice. Shodno tome dobijamo (primenjujući i permutaciju kolona):

1	2	3	4	5	6	7	→
1	e	r	t	d	u	v	j
2	e	f	r	i	g	a	j
3	o	i	k	p	k	r	t
4	o	a	e	c	j	t	n
5	i	a	i	d	n	k	n
6	a	c	p	e	n	o	t

5	3	6	2	4	7	1	→
5	n	i	k	a	d	n	i
4	j	e	t	a	c	n	o
1	u	t	v	r	d	j	e
6	n	p	o	c	e	t	a
3	k	k	r	i	p	t	o
2	g	r	a	f	i	j	e

Analizom dobijene matrice može se zaključiti da je izgled početne matrice sledeći:

5	3	6	2	4	7	1	
5	n	i	k	a	d	n	i
4	j	e	t	a	c	n	o
1	u	t	v	r	d	j	e
6	n	p	o	c	e	t	a
3	k	k	r	i	p	t	o
2	g	r	a	f	i	j	e

Polazna rečenica glasi:

Nikad nije tačno utvrđen početak kriptografije.

9.2.2 Zadatak 2

„Double transposition“ metodom kriptovana je rečenica i dobijen je sledeći rezultat:

isnattiamezredatvkeseebzvopukurdrcaeaesbljeoairrtkeiialmojfaric.

Ako se zna da je korišćena matrica dimenzija 8x8 i da je sadržaj zadnje vrste pre permutovanja bio **skevezbe** naći polaznu rečenicu.

Krajnje stanje:

		1	2	3	4	5	6	7	8
1		l	s	n	a	t	t	i	a
2	m	e	z	r	e	d	a	a	t
3	v	k	e	s	e	e	b	z	
4	v	o	p	u	i	k	u	r	
5	d	r	c	a	e	a	e	s	
6	b	l	j	e	o	a	i	r	
7	r	t	k	e	i	i	a	l	
8	m	o	j	f	a	r	i	c	

Početno stanje:

		1	2	3	4	5	6	7	8
1	?	?	?	?	?	?	?	?	?
2	?	?	?	?	?	?	?	?	?
3	?	?	?	?	?	?	?	?	?
4	?	?	?	?	?	?	?	?	?
5	?	?	?	?	?	?	?	?	?
6	?	?	?	?	?	?	?	?	?
7	?	?	?	?	?	?	?	?	?
8	s	k	e	v	e	z	b	e	

Upoređujući početnu sa krajnjom matricom možemo videti da se sadržaj trećeg reda krajnje matrice poklapa sa sadržajem zadnjeg reda početne matrice.

9. ZADACI SA VEŽBI

1	2	3	4	5	6	7	8	
3	v	k	e	s	e	e	b	z

1	2	3	4	5	6	7	8	
8	s	k	e	v	e	z	b	e

Gledajući data dva reda možemo uvideti sledeće mogućnosti za permutaciju kolona kako bi od početnog stigli do krajnjeg stanja:

1)

1	2	3	4	5	6	7	8		
3	v		k	e	s	e	e	b	z

4	2	3	1	5	8	7	6	
8	s	k	e	v	e	z	b	e

→

1	2	3	4	5	6	7	8	
3	v	k	e	s	e	e	b	z

4	2	3	1	6	8	7	5	
8	s	k	e	v	e	z	b	e

2)

→

1	2	3	4	5	6	7	8	
3	v	k	e	s	e	e	b	z

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

3)

→

1	2	3	4	5	6	7	8	
3	v	k	e	s	e	e	b	z

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

4)

→

1	2	3	4	5	6	7	8	
3	v	k	e	s	e	e	b	z

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

5)

→

1	2	3	4	5	6	7	8	
3	v	k	e	s	e	e	b	z

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

6)

→

Uzimajući u obzir ove mogućnosti permutacija dobijamo sledeće matrice:

1)

	1	2	3	4	5	6	7	8	
1	i	s	n	a	t	t	i	a	
2	m	e	z	r	e	d	a	t	
3	v	k	e	s	e	e	b	z	
4	v	o	p	u	i	k	u	r	
5	d	r	c	a	e	a	e	s	
6	b	l	j	e	o	a	i	r	
7	r	t	k	e	i	i	a	l	
8	m	o	j	f	a	r	i	c	



	4	2	3	1	5	8	7	6	
1	a	s	n	i	t	a	i	t	
2	r	e	z	m	e	t	a	d	
3	s	k	e	v	e	z	b	e	
4	u	o	p	v	i	r	u	k	
5	a	r	c	d	e	s	e	a	
6	e	l	j	b	o	r	i	a	
7	e	t	k	r	i	l	a	i	
8	f	o	j	m	a	c	i	r	

2)

	1	2	3	4	5	6	7	8	
1	l	s	n	a	t	t	i	a	
2	m	e	z	r	e	d	a	t	
3	v	k	e	s	e	e	b	z	
4	v	o	p	u	i	k	u	r	
5	d	r	c	a	e	a	e	s	
6	b	l	j	e	o	a	i	r	
7	r	t	k	e	i	i	a	l	
8	m	o	j	f	a	r	i	c	



	4	2	3	1	6	8	7	5	
1	a	s	n	i	t	a	i	t	
2	r	e	z	m	d	t	a	e	
3	s	k	e	v	e	z	b	e	
4	u	o	p	v	k	r	u	i	
5	a	r	c	d	a	s	e	e	
6	e	l	j	b	a	r	i	o	
7	e	t	k	r	i	l	a	i	
8	f	o	j	m	r	c	i	a	

9. ZADACI SA VEŽBI

3)

	1	2	3	4	5	6	7	8
1	I	S	N	A	T	T	I	A
2	M	E	Z	R	E	D	A	T
3	V	K	E	S	E	E	B	Z
4	V	O	P	U	I	K	U	R
5	D	R	C	A	E	A	E	S
6	B	L	J	E	O	A	I	R
7	R	T	K	E	I	I	A	L
8	M	O	J	F	A	R	I	C

	4	2	3	1	6	8	7	5
1	A	S	N	I	T	A	I	T
2	R	E	Z	M	D	T	A	E
3	S	K	E	V	E	Z	B	E
4	U	O	P	V	K	R	U	I
5	A	R	C	D	A	S	E	E
6	E	L	J	B	A	R	I	O
7	E	T	K	R	I	L	A	I
8	F	O	J	M	R	C	I	A



4)

	1	2	3	4	5	6	7	8
1	I	S	N	A	T	T	I	A
2	M	E	Z	R	E	D	A	T
3	V	K	E	S	E	E	B	Z
4	V	O	P	U	I	K	U	R
5	D	R	C	A	E	A	E	S
6	B	L	J	E	O	A	I	R
7	R	T	K	E	I	I	A	L
8	M	O	J	F	A	R	I	C

	4	2	5	1	3	8	7	6
1	A	S	T	I	N	A	I	T
2	R	E	E	M	Z	T	A	D
3	S	K	E	V	E	Z	B	E
4	U	O	I	V	P	R	U	K
5	A	R	E	D	C	S	E	A
6	E	L	O	B	J	R	I	A
7	E	T	I	R	K	L	A	I
8	F	O	A	M	J	C	I	R



5)

	1	2	3	4	5	6	7	8
1	I	S	N	A	T	T	I	A
2	M	E	Z	R	E	D	A	T
3	V	K	E	S	E	E	B	Z
4	V	O	P	U	I	K	U	R
5	D	R	C	A	E	A	E	S
6	B	L	J	E	O	A	I	R
7	R	T	K	E	I	I	A	L
8	M	O	J	F	A	R	I	C

	4	2	6	1	3	8	7	5
1	a	s	t	i	n	a	i	t
2	r	e	d	m	z	t	a	e
3	s	k	e	v	e	z	b	e
4	u	o	k	v	p	r	u	i
5	a	r	a	d	c	s	e	e
6	e	l	a	b	j	r	i	o
7	e	t	i	r	k	l	a	i
8	f	o	r	m	j	c	i	a



6)

	1	2	3	4	5	6	7	8
1	I	S	N	A	T	T	I	A
2	M	E	Z	R	E	D	A	T
3	V	K	E	S	E	E	B	Z
4	V	O	P	U	I	K	U	R
5	D	R	C	A	E	A	E	S
6	B	L	J	E	O	A	I	R
7	R	T	K	E	I	I	A	L
8	M	O	J	F	A	R	I	C

	4	2	6	1	5	8	7	3
1	a	s	t	i	t	a	i	n
2	r	e	d	m	e	t	a	z
3	s	k	e	v	e	z	b	e
4	u	o	k	v	i	r	u	p
5	a	r	a	d	e	s	e	c
6	e	l	a	b	o	r	i	j
7	e	t	i	r	i	l	a	k
8	f	o	r	m	a	c	i	j



Iz priloženog se vidi da je šesta mogućnost jedina odgovarajuća za permutaciju kolona, jer jedino primenom nje može se dobiti čitljiv tekst. Ostalo je samo naći permutaciju redova. Videli smo da treći red početne matrice odgovara zadnjem redu krajnje matrice. Shodno tome dobijamo (primenjujući i permutaciju kolona):

1	2	3	4	5	6	7	8	
1	I	s	n	a	t	t	i	a
2	m	e	z	r	e	d	a	t
3	v	k	e	s	e	e	b	z
4	v	o	p	u	i	k	u	r
5	d	r	c	a	e	a	e	s
6	b	l	j	e	o	a	i	r
7	r	t	k	e	i	i	a	l
8	m	o	j	f	a	r	i	c

4	2	6	1	5	8	7	3	→
1	a	s	t	i	t	a	i	n
2	r	e	d	m	e	t	a	z
8	f	o	r	m	a	c	i	j
4	u	o	k	v	i	r	u	p
5	a	r	a	d	e	s	e	c
6	e	l	a	b	o	r	i	j
7	e	t	i	r	i	l	a	k
3	s	k	e	v	e	z	b	e

Analizom dobijene matrice može se zaključiti da je izgled početne matrice sledeći:

4	2	6	1	5	8	7	3	
4	u	o	k	v	i	r	u	p
2	r	e	d	m	e	t	a	z
1	a	s	t	i	t	a	i	n
8	f	o	r	m	a	c	i	j
5	a	r	a	d	e	s	e	c
7	e	t	i	r	i	l	a	k
6	e	l	a	b	o	r	i	j
3	s	k	e	v	e	z	b	e

Polazna rečenica glasi:

U okviru predmeta zaštita informacija rade se četiri lake *laborijske* vežbe.

9.2.3 Zadatak 3

„Double Transposition“ metodom kriptovanja kriptovana je rečenica i dobijen sledeći rezultat: **helsnevsoitneeoaoikjlajadiijerasdejnzikmdkleee**. Ako se zna da je sadržaj prve vrste pre permutovanja bio **Ilijada** naći polaznu rečenicu.

Rešenje:

Pošto je sadržaj prve vrste pre permutovanja bio **iliJada** (koja je dužine 7 slova) dobijeni kriptovani tekst (koji ima 49 slova) pišemo u obliku matrice dimenzija 7x7:

$$\begin{bmatrix} h & e & l & s & n & e & o \\ v & s & o & i & t & n & e \\ e & o & a & o & i & k & j \\ l & a & j & a & d & i & i \\ i & j & e & r & a & s & d \\ e & j & n & z & i & k & k \\ m & d & k & l & e & e & e \end{bmatrix}$$

Nakon toga vršimo permutaciju kolona sve dok u nekoj vrsti ne dobijemo niz slova **Ilijada**.

$$\begin{bmatrix} o & h & e & l & e & n & s \\ e & v & n & o & s & t & i \\ j & e & k & a & o & i & o \\ \textcolor{red}{i} & l & i & j & a & d & a \\ d & i & s & e & j & a & r \\ k & e & k & n & j & i & z \\ e & m & e & k & d & e & l \end{bmatrix}$$

Nakon toga vršimo permutaciju vrsta. Prva vrsta postaje vrsta koja ima niz slova **Ilijada**, ostale vrste „redjamo“ smisleno, tako da kada čitamo slova redom (vrstu po vrstу) to ima nekog smisla.

$$\begin{bmatrix} i & l & i & j & a & d & a \\ j & e & k & a & o & i & o \\ d & i & s & e & j & a & r \\ e & m & e & k & d & e & l \\ o & h & e & l & e & n & s \\ k & e & k & n & j & i & z \\ e & v & n & o & s & t & i \end{bmatrix}$$

Kada čitamo redom vrstu po vrstу dobijamo sledeće:
ilijadajekaoiodisejaremekdelohelenskeknjizevnosti.

Dakle, polazna rečenica je bila „ilijada je kao i odiseja remekdelo helenske knjizevnosti“.

9.2.4 Zadatak 4

„Double transposition“ metodom kriptovana je rečenica i dobijen je sledeći rezultat: **ugnrdaamaeginaristikseiredokdjekadanjeretsvrigud**. Ako se zna da je sadržaj zadnje vrste pre permutovanja bio **aenigma** naći polaznu rečenicu.

Na osnovu sadržaja zadnje vrste vidimo da polazna matrica ima 7 kolona, a na osnovu toga i broja slova u rezultatu može se konstatovati da je matrica dimenzija 7x7. Popunjavamo matricu redom slovima iz rezultata i dobijamo sledeće:

$$\begin{array}{ccccccc} \text{I} & 6 & 1 & 2 & 5 & 4 & 3 & 7 \\ \text{II} & 6 & 7 & 2 & 5 & 4 & 3 & 1 \end{array}$$

$$\begin{vmatrix} u & g & n & r & d & a & a \\ m & a & e & g & i & n & a \\ r & a & t & i & k & s & e \\ i & r & e & d & o & k & d \\ j & e & k & a & d & a & n \\ s & e & j & e & r & e & t \\ s & v & r & i & g & u & d \end{vmatrix}$$

Vidimo da u drugoj vrsti transformacijom kolona možemo dobiti reč **aenigma** i to na dva načina zbog toga što imamo dva slova **a** u zadatoj reči.

I način (Transformacija I – (6, 1, 2, 5, 4, 3, 7)):

$$\begin{vmatrix} g & n & a & d & r & u & a \\ a & e & n & i & g & m & a \\ a & t & s & k & i & r & e \\ r & e & k & o & d & i & d \\ e & k & a & d & a & j & n \\ e & j & e & r & e & s & t \\ v & r & u & g & i & s & d \end{vmatrix}$$

Na osnovu ovoga može se ustanoviti da bilo kojom transformacijom vrsta matrice ne možemo dobiti smislenu rečenicu, pa koristimo drugi način (Transformacija II – (6, 7, 2, 5, 4, 3, 1)):

$$\begin{vmatrix} a & n & a & d & r & u & g \\ a & e & n & i & g & m & a \\ e & t & s & k & i & r & a \\ d & e & k & o & d & i & r \\ n & k & a & d & a & j & e \\ t & j & e & r & e & s & e \\ d & r & u & g & i & s & v \end{vmatrix}$$

Transformacijom vrsti ove matrice može se dobiti smislena rečenica.

(Transformacija vrsti – (6, 7, 2, 5, 4, 3, 1)):

$$\begin{vmatrix} d & r & u & g & i & s & v \\ e & t & s & k & i & r & a \\ t & j & e & r & e & s & e \\ n & k & a & d & a & j & e \\ d & e & k & o & d & i & r \\ a & n & a & d & r & u & g \\ a & e & n & i & g & m & a \end{vmatrix}$$

Čitanjem vrsti ove matrice dobija se:

drugisvetskiratjeresenkadajedekoriranadrugaenigma. Prevedeno na naš jezik: Drugi svetski rat je rešen kada je dekodirana druga enigma.

9.2.5 Zadatak 5

„Double transposition“ metodom kriptovana je rečenica i dobijen je sledeći rezultat: **dгivianlgiasndeјeibjrsiuliatiatniounvpanretps**. Ako se zna da je sadržaj zadnje vrste pre permutovanja bio **signal** naći polaznu rečenicu.

Rešenje:

Pošto se radi o „Double transposition“ metodu prvo kreiramo matricu čiji su elementi dobijeni kao rezultat kriptovanja tj. sledeći niz **dгivianlgiasndeјeibjrsiuliatiatniounvpanretps** a iz uslova da je poslednja vrsta pre permutacije **signal** zaključujemo da je matrica dimenzija 7x6 tj. ima 7 vrsta i 6 kolona.

$$\begin{bmatrix} d & g & i & v & i & a \\ n & l & g & i & a & s \\ n & d & e & j & e & i \\ b & j & r & s & i & u \\ l & i & a & t & n & i \\ o & u & n & v & p & a \\ n & r & e & t & p & s \end{bmatrix}$$

početni izgled matrice

Numeracija kolona je u prvom slučaju (1,2,3,4,5,6,7), a numeracija kolona je (1,2,3,4,5,6). Iz ove matrice lako možemo da uočimo da permutacijom kolona na sledeći način (6,4,3,1,5,2) dobijamo **signal** u drugoj vrsti što je dobar početak pri dekodiranju, dok numeracija vrsta ostaje ista (1,2,3,4,5,6,7). Nakon ove permutacije matrica izgleda ovako

$$\begin{bmatrix} a & v & i & d & i & g \\ s & i & g & n & a & l \\ i & j & e & n & e & d \\ u & s & r & b & i & j \\ l & t & a & l & n & i \\ a & v & n & o & p & u \\ s & t & e & n & p & r \end{bmatrix}$$

Na osnovu ovog izgleda matrice polako se naziru već neke reči koje imaju nekog smisla i zbog toga radimo permutaciju po vrsti na sledeći način (4,3,6,7,1,5,2), dok numeracija kolona ostaje kao u prethodnom slučaju (6,4,3,1,5,2). Sada matrica izgleda ovako:

$$\begin{bmatrix} u & s & r & b & i & j \\ i & j & e & n & e & d \\ a & v & n & o & p & u \\ s & t & e & n & p & r \\ a & v & i & d & i & g \\ i & t & a & l & n & i \\ s & i & g & n & a & l \end{bmatrix}$$

Pošto smo kao rezultat dobili da je **signal** zadnja vrsta u ovoj matrici, što je i bio uslov zadatka, možemo da zaključimo da je ova matrica u stvari skup elemenata polazne rečenice. Polazna rečenica bi u ovom slučaju izgledala ovako (ovo je bez razmaka baš kao što je u matrici, element po element sve spojeno samo pročitana matrica)

usrbijijenedavnopustenpravidigitalnisignal

Kada se to sve sredi i razdvoji blanko znacima dobijamo da je početna rečenica izgledala ovako.

u srbiji je nedavno posten pravi digitalni signal

9.3 A5/1

9.3.1 Zadatak 1

Dat je algoritam sličan standardnom A5/1. Bitovi koji učestvuju u XOR operacijama su:

$x[8], x[11], x[17]$

$y[4], y[15], y[16], y[17]$

$z[15], z[16], z[17], z[18]$

Bitovi koji učestvuju u Majority Vote operaciji su: $x[11], y[11], z[11]$.

Korišćenjem principa rada standardnog A5/1 algoritma kodirati poruku $M = 00101$

Rešenje:

Pocetno stanje registra:

X:

0	0	1	1	1	0	0	0	1	1	0	0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Y:

0	0	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Z:

0	1	1	1	0	0	1	1	0	0	1	1	0	0	0	1	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Prva iteracija:

U majority vote-u pobedjuje 0 pa se menjaju (rotiraju za 1 bit udesno umetanjem bita na nullu poziciju) registri X i Y.

Za X : $1 \oplus 0 \oplus 1 = 0$

Za Y : $1 \oplus 0 \oplus 0 \oplus 1 = 0$

X:

0	0	0	1	1	1	0	0	0	1	1	0	0	0	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

9. ZADACI SA VEŽBI

Y:

0	0	0	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Z:

0	1	1	1	0	0	1	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$$1 \oplus 0 \oplus 1 \oplus 0 = 0 \rightarrow C = 0$$

Druga iteracija:

U majority vote-u pobedjuje 0 pa se menjaju (rotiraju za 1 bit udesno umetanjem bita na nultu poziciju) registri X i Y.

$$\text{Za } X : 0 \oplus 0 \oplus 0 = 0$$

$$\text{Za } Y : 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

X:

0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Y:

0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Z:

0	1	1	1	0	0	1	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$$0 \oplus 1 \oplus 1 \oplus 0 = 0 \rightarrow C = 00$$

Treća iteracija:

U majority vote-u pobedjuje 1 pa se menjaju (rotiraju za 1 bit udesno umetanjem bita na nultu poziciju) registri X, Y i Z.

ZAŠTITA INFORMACIJA

Za X : $0 \oplus 1 \oplus 0 = 1$

Za Y : $0 \oplus 1 \oplus 1 \oplus 0 = 0$

Za Z : $1 \oplus 1 \oplus 1 \oplus 0 = 1$

X:

1	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Y:

0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Z:

1	0	1	1	1	0	0	1	1	0	0	1	1	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$0 \oplus 1 \oplus 0 \oplus 1 = 0 \rightarrow C = 000$

Četvrta iteracija:

U majority vote-u pobedjuje 1 pa se menjaju (rotiraju za 1 bit udesno umetanjem bita na nultu poziciju) registri X, Y i Z.

Za X : $0 \oplus 1 \oplus 1 = 0$

Za Y : $0 \oplus 1 \oplus 1 \oplus 1 = 1$

Za Z : $0 \oplus 1 \oplus 1 \oplus 1 = 1$

X:

0	1	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Y:

1	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Z:

1	1	0	1	1	1	0	0	1	1	0	0	1	1	0	0	0	1	1	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$$1 \oplus 1 \oplus 0 \oplus 0 = 0 \quad \rightarrow C = 0000$$

Peta iteracija:

U majority vote-u pobedjuje 0 pa se menjaju (rotiraju za 1 bit udesno umetanjem bita na nultu poziciju) registri X i Z.

$$\text{Za } X : 1 \oplus 0 \oplus 1 = 0$$

$$\text{Za } Z : 0 \oplus 0 \oplus 1 \oplus 1 = 0$$

X:

0	0	1	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Y:

1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Z:

0	1	1	0	1	1	1	0	0	1	1	0	0	1	1	0	0	0	1	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$$1 \oplus 1 \oplus 1 \oplus 1 = 0 \quad \rightarrow C = 00000$$

Rezultat kodovanja za ulaznu sekvencu M = 00101 je C = 00000.

9.3.2 Zadatak 2

Dato je trenutno stanje registra za modifikovani A5/1 algoritam:

X="10100011100111001", Y="01111110011111000001" I

Z="01111110101111001100001". Bitovi koji učestvuju u funkciji većinskog glasanja su X[4], Y[12] i Z[18]; dok su bitovi koji učestvuju u step operacijama sledeći: 2, 4, 9 i 13 za registar X, 4, 16 i 20 za registar Y i 9, 13, 15 i 20 za registar Z. Odrediti rezultat kodovanja za ulaznu sekvencu bitova: 10000.

Rešenje: Početno stanje registara je kao na slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	1	0	1	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0	1				
Y	0	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	0	0	1	1		
Z	0	1	1	1	1	1	1	0	1	0	1	1	1	1	0	0	1	1	0	0	0	1	

Za svaki bit P koji pristigne na ulaz generiše se ključ, bit S, koji se XOR-uju se ulaznim bitom i dobija se kriptovani bit C. Generisanje ključa se vrši na sledeći način:

- 1) izvrši se funkcija većinskog glasanja i dobija se bit m = maj(X[4],Y[12],Z[18]);
- 2) ako je X[4]=m, onda se računa t_x kao $t_x = x_1 \oplus x_3 \oplus x_8 \oplus x_{12}$ i vrši se rotacija u desno ($x_i = x_{i-1}$)
- 3) ako je Y[12]=m, onda se računa t_y kao $t_y = y_3 \oplus y_{15} \oplus y_{19}$ i vrši se rotacija u desno ($y_i = y_{i-1}$)
- 4) ako je Z[18]=m, onda se računa t_z kao $t_z = z_8 \oplus z_{12} \oplus z_{14} \oplus z_{19}$ i vrši se rotacija u desno ($z = z_{i-1}$)

dok 0-i bit kod registra koji se rotira postaje t_x , t_y ili t_z .

Bit S dobija se nakon rotiranja kao $S = x_{18} \oplus y_{21} \oplus z_{22}$.

Kriptovani bit na kraju se dobija kao $C = S \oplus P$.

Prvi ulazni bit je bit 1. P=1.

m=maj(0,1,0) => m=0

Rotiramo registre X i Z.

$$t_x = 0 \oplus 0 \oplus 1 \oplus 1 = 0$$

$$t_z = 1 \oplus 1 \oplus 0 \oplus 0 = 0$$

Novo stanje registara dato je na sledećoj slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	0	1	0	1	0	0	0	1	1	1	1	0	0	1	1	1	0	0	0				
Y	0	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	0	0	1	1		
Z	0	0	1	1	1	1	1	1	0	1	0	1	1	1	1	0	0	1	1	0	0	0	

$$\text{Keystream } S = 0 \oplus 1 \oplus 0 = 1$$

$$\text{Kodirani bit } C = 1 \oplus 1 = 0$$

Sledeći ulazni bit je 0. P=0.

$$m = \text{maj}(0,1,1) \Rightarrow m=1$$

Rotiramo registre Y i Z.

$$t_y = 1 \oplus 0 \oplus 0 = 1$$

$$t_z = 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

Novo stanje registara dato je na sledećoj slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	0	1	0	1	0	0	0	1	1	1	1	0	0	1	1	1	0	0	0	0	0	0	
Y	1	0	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	0	0	0	0	0	
Z	0	0	0	1	1	1	1	1	1	0	1	0	1	1	1	0	0	0	1	1	0	0	

$$\text{Keystream } S = 0 \oplus 1 \oplus 0 = 1$$

$$\text{Kodirani bit } C = 1 \oplus 0 = 1$$

Sledeći ulazni bit je 0. P=0.

$$m = \text{maj}(0,1,1) \Rightarrow m=1$$

Rotiramo registre Y i Z.

$$t_y = 1 \oplus 1 \oplus 0 = 0$$

$$t_z = 1 \oplus 1 \oplus 1 \oplus 1 = 0$$

Novo stanje registara dato je na sledećoj slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	0	1	0	1	0	0	0	1	1	1	1	0	0	1	1	1	0	0	0	0	0	0	
Y	0	1	0	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	0	0	0	
Z	0	0	0	0	1	1	1	1	1	0	1	0	1	1	1	1	0	0	0	1	1	0	

$$\text{Keystream } S = 0 \oplus 0 \oplus 0 = 0$$

$$\text{Kodirani bit } C = 0 \oplus 0 = 0$$

Sledeći ulazni bit je 0. P=0.

$$m = \text{maj}(0,1,0) \Rightarrow m=0$$

ZAŠTITA INFORMACIJA

Rotiramo registre X i Z.

$$t_x = 1 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$t_z = 1 \oplus 0 \oplus 1 \oplus 1 = 1$$

Novo stanje registara dato je na sledećoj slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	1	0	1	0	1	0	0	0	1	1	1	1	0	0	1	1	1	0	0				
Y	0	1	0	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	0	0	0	
Z	1	0	0	0	0	1	1	1	1	1	0	1	0	1	1	1	1	0	0	1	1	0	

$$\text{Keystream } S = 0 \oplus 0 \oplus 0 = 0$$

$$\text{Kodirani bit } C = 0 \oplus 0 = 0$$

Sledeći ulazni bit je 0. P=0.

$$m = \text{maj}(1, 1, 0) \Rightarrow m = 1$$

Rotiramo registre X i Y.

$$t_x = 0 \oplus 0 \oplus 1 \oplus 0 = 1$$

$$t_y = 1 \oplus 1 \oplus 0 = 0$$

Novo stanje registara dato je na sledećoj slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	1	1	0	1	0	1	0	0	0	1	1	1	1	0	0	1	1	1	0				
Y	0	0	1	0	1	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	0	0	
Z	1	0	0	0	0	1	1	1	1	1	0	1	0	1	1	1	1	0	0	1	1	0	

$$\text{Keystream } S = 0 \oplus 0 \oplus 0 = 0$$

$$\text{Kodirani bit } C = 0 \oplus 0 = 0$$

Ulazni podatak nam je bio 10000 a izlazni, kodirani je 01000.

9.3.3 Zadatak 3

Dato je trenutno stanje registra za modifikovani A5/1 algoritam:

$X = "1101001110110001100"$, $Y = "111000110011011111001"$ I

$Z = "11001100011101001001100"$. Bitovi koji učestvuju u funkciji većinskog glasanja su $X[9]$, $Y[9]$ i $Z[9]$; dok su bitovi koji učestvuju u step operacijama sledeći: 2 i 19 za registar X , 3, 19 i 20 za registar Y i 9, 13, 15 i 20 za registar Z . Odrediti rezultat kodovanja za ulaznu sekvencu bitova: 11001.

Rešenje: Početno stanje registara je kao na slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	1	1	0	1	0	0	1	1	1	0	1	1	0	0	0	1	1	1	0	0			
Y	1	1	1	0	0	0	1	1	0	0	1	1	0	1	1	1	1	1	1	0	0	1	
Z	1	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	1	0	0	1	1	0	0

Za svaki bit P koji pristigne na ulaz generiše se ključ, bit S , koji se XOR-uju se ulaznim bitom i dobija se kriptovani bit C . Generisanje ključa se vrši na sledeći način:

- 1) izvrši se funkcija većinskog glasanja i dobija se bit $m = \text{maj}(X[9], Y[9], Z[9])$;
- 2) ako je $X[9]=m$, onda se računa t_x kao $t_x = x_1 \oplus x_{18}$ i vrši se rotacija u desno ($x_i = x_{i-1}$)
- 3) ako je $Y[9]=m$, onda se računa t_y kao $t_y = y_2 \oplus y_{21} \oplus y_{22}$ i vrši se rotacija u desno ($y_i = y_{i-1}$)
- 4) ako je $Z[9]=m$, onda se računa t_z kao $t_z = z_8 \oplus z_{12} \oplus z_{14} \oplus z_{19}$ i vrši se rotacija u desno ($z = z_{i-1}$)

dok 0-i bit kod registra koji se rotira postaje t_x , t_y ili t_z .

Bit S dobija se nakon rotiranja kao $S = x_{18} \oplus y_{21} \oplus z_{22}$.

Kriptovani bit na kraju se dobija kao $C = S \oplus P$.

Prvi ulazni bit je bit 1. $P=1$.

$m=\text{maj}(0,0,1) \Rightarrow m=0$

Rotiramo registre X i Y .

$$t_x = 1 \oplus 0 = 1$$

$$t_y = 1 \oplus 1 \oplus 0 = 0$$

Novo stanje registara dato je na sledećoj slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	1	1	1	0	1	0	0	1	1	1	0	1	1	0	0	0	1	1	0				
Y	0	1	1	1	0	0	0	1	1	0	0	1	1	0	1	1	1	1	1	0	0		
Z	1	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	1	0	0	1	1	0	0

Keystream $S = 0 \oplus 0 \oplus 0 = 0$

Kodirani bit $C = 0 \oplus 1 = 1$

Ulaz 1 => Izlaz 1

Sledeći ulazni bit je 1. P=1.

$m = \text{maj}(1,0,1) \Rightarrow m=1$

Rotiramo registre X i Z.

$$t_x = 1 \oplus 0 = 1$$

$$t_z = 0 \oplus 0 \oplus 0 \oplus 1 = 1$$

Novo stanje registara dato je na sledećoj slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	1	1	1	1	0	1	0	0	1	1	1	0	1	1	0	0	0	1	1				
Y	0	1	1	1	0	0	0	1	1	0	0	1	1	0	1	1	1	1	1	0	0		
Z	1	1	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	1	0	0	1	1	0

Keystream $S = 1 \oplus 0 \oplus 0 = 1$

Kodirani bit $C = 1 \oplus 1 = 0$

Ulaz 1 => Izlaz 0

Sledeći ulazni bit je 0. P=0.

$m = \text{maj}(1,0,0) \Rightarrow m=0$

Rotiramo registre Y i Z.

$$t_y = 1 \oplus 1 \oplus 1 = 1$$

$$t_z = 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

Novo stanje registara dato je na sledećoj slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	1	1	1	1	0	1	0	0	1	1	1	0	1	1	0	0	0	0	1	1			
Y	1	0	1	1	1	0	0	0	1	1	0	0	1	1	0	1	1	1	1	1	1	0	
Z	0	1	1	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	1	0	0	1	

Keystream $S = 1 \oplus 0 \oplus 1 = 0$

Kodirani bit $C = 0 \oplus 0 = 0$

Ulez 0 => Izlez 0

Sledeći ulazni bit je 0. P=0.

$m = \text{maj}(1, 1, 0) \Rightarrow m = 1$

Rotiramo registre X i Y.

$$t_x = 1 \oplus 1 = 0$$

$$t_y = 1 \oplus 1 \oplus 1 = 1$$

Novo stanje registara dato je na sledećoj slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	0	1	1	1	1	0	1	0	0	1	1	1	0	1	1	0	0	0	1				
Y	1	1	0	1	1	1	0	0	0	1	1	0	0	1	1	0	1	1	1	1	1	1	
Z	0	1	1	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	1	0	0	1	

Keystream $S = 1 \oplus 1 \oplus 1 = 1$

Kodirani bit $C = 1 \oplus 0 = 1$

Ulez 0 => Izlez 1

Sledeći ulazni bit je 1. P=1.

$m = \text{maj}(1, 1, 0) \Rightarrow m = 1$

Rotiramo registre X i Y.

$$t_x = 1 \oplus 1 = 0$$

$$t_y = 0 \oplus 1 \oplus 1 = 0$$

Novo stanje registara dato je na sledećoj slici:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
X	0	0	1	1	1	1	0	1	0	0	1	1	1	0	1	1	0	0	0	0			
Y	0	1	1	0	1	1	1	0	0	0	1	1	0	0	1	1	0	1	1	1	1	1	
Z	0	1	1	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	1	0	0	1	1

Keystream $S = 0 \oplus 1 \oplus 1 = 0$

Kodirani bit $C = 0 \oplus 1 = 1$

Ulaz 1 => Izlaz 1

Ulagani podatak nam je bio 11001 a izlazni, kodirani je 10011.

9.4 RC4

9.4.1 Zadatak 1

Za kodiranje po RC4 algoritmu dato je trenutno stanje ključa $S = [5 7 0 4 1 3 2 6]$ i trenutne vrednosti indeksa $i = 4$ i $j = 3$. Kodirati broj $14340_{(8)}$ po RC4 algoritmu i rezultat predstaviti kao oktalni broj.

Rešenje:

Prevešćemo cifre datog broja u binarni oblik:

$$1 \rightarrow 001$$

$$4 \rightarrow 100$$

$$3 \rightarrow 011$$

$$0 \rightarrow 000$$

Dalje, dati su trenutno stanje ključa $S = [5 7 0 4 1 3 2 6]$ i trenutne vrednosti brojača $i = 3$ i $j = 4$.

U odnosu na standardni RC4 algoritam, ovaj problem se specijalizuje na sledeći način:

```
i := 3  
j := 4  
while GeneratingOutput:  
    i := (i + 1) mod 8  
    j := (j + S[i]) mod 8  
    swap(S[i], S[j])  
    output S[(S[i] + S[j]) mod 8]  
endwhile
```

Operacija mod se ne vrši sa 256 već sa 8 jer u datom primeru niz S ima 8 elemenata. Kako je broj petocifren, do ključa dolazimo primenom datog algoritma kroz pet iteracija:

Prva iteracija:

$$i = 4 \text{ mod } 8 = 4$$

$$j = (4 + S[4]) \text{ mod } 8 = (4 + 1) \text{ mod } 8 = 5$$

$$S = [5 \ 7 \ 0 \ 4 \ 3 \ 1 \ 2 \ 6]$$

$$\text{output} = S[(S[i] + S[j]) \text{ mod } 8] = S[(S[4] + S[5]) \text{ mod } 8] = S[(3 + 1) \text{ mod } 8] = S[4] = 3$$

Druga iteracija:

$$i = 5 \text{ mod } 8 = 5$$

$$j = (5 + S[5]) \text{ mod } 8 = (5 + 1) \text{ mod } 8 = 6$$

$$S = [5 \ 7 \ 0 \ 4 \ 3 \ 2 \ 1 \ 6]$$

$$\text{output} = S[(S[i] + S[j]) \text{ mod } 8] = S[(S[5] + S[6]) \text{ mod } 8] = S[(2 + 1) \text{ mod } 8] = S[3] = 4$$

Treća iteracija:

$$i = 6 \text{ mod } 8 = 6$$

$$j = (6 + S[6]) \text{ mod } 8 = (6 + 1) \text{ mod } 8 = 7$$

$S = [5 \ 7 \ 0 \ 4 \ 3 \ 2 \ 6 \ 1]$

$output = S[(S[i] + S[j]) \ mod \ 8] = S[(S[6] + S[7]) \ mod \ 8] = S[(6 + 1) \ mod \ 8] = S[7] = 1$

Četvrta iteracija:

$i = 7 \ mod \ 8 = 7$

$j = (7 + S[7]) \ mod \ 8 = (7 + 1) \ mod \ 8 = 0$

$S = [1 \ 7 \ 0 \ 4 \ 3 \ 2 \ 6 \ 5]$

$output = S[(S[i] + S[j]) \ mod \ 8] = S[(S[7] + S[0]) \ mod \ 8] = S[(5 + 1) \ mod \ 8] = S[6] = 6$

Peta iteracija:

$i = 8 \ mod \ 8 = 0$

$j = (0 + S[0]) \ mod \ 8 = 1 \ mod \ 8 = 1$

$S = [7 \ 1 \ 0 \ 4 \ 3 \ 2 \ 6 \ 5]$

$output = S[(S[i] + S[j]) \ mod \ 8] = S[(S[0] + S[1]) \ mod \ 8] = S[(7 + 1) \ mod \ 8] = S[0] = 7$

Kako se svaki generisani triplet ključa (a iz svake iteracije dobili smo po jedan) uparuje sa jednim tripletom originalnog podatka, kodirani podatak dobićemo na sledeći način:

Nazovimo $output$ iz i -te iteracije $output[i]$. Dakle imamo da je $output[1] = 3, output[2] = 4, output[3] = 1, output[4] = 6, output[5] = 7$. $output$ takođe treba biti kodiran binarno tako da je

$output[1] = 011,$
 $output[2] = 100,$
 $output[3] = 001,$
 $output[4] = 110,$
 $output[5] = 111.$

Svaki bajt triplet podatka obeležimo sa $data[i]$, odakle sledi da je

$data[1] = 001,$

$data[2] = 100,$

$data[3] = 011,$
 $data[4] = 100,$
 $data[5] = 000.$

Primenom XOR (\oplus) operacije među parovima ključ – podatak dobijamo:

$$\begin{aligned}data[1] \oplus output[1] &= 001 \oplus 011 = 010 \\data[2] \oplus output[2] &= 100 \oplus 100 = 000 \\data[3] \oplus output[3] &= 011 \oplus 001 = 010 \\data[4] \oplus output[4] &= 110 \oplus 100 = 010 \\data[5] \oplus output[5] &= 000 \oplus 111 = 111\end{aligned}$$

Dakle dobijeni kodirani podatak je:

010 000 010 010 111

Odnosno, predstavljeno oktalno:

20227

9.4.2 Zadatak 2

Odrediti sadržaj niza S od 16 elemenata nakon postupka generisanja ključa standardnim RC4 algoritmom. Za spoljni ključ uzeti niz hex cifara 0x2FFFA113.

Spoljni ključ je 0x2FFFA113. Kako se sastoji od 8 hex cifara, vrednost parametra keylength je 8. Broj elemenata niza S je 16, po uslovu zadatka. Dakle, N = 16 i za dobijanje ključa potrebno je 16 iteracija.

ZAŠTITA INFORMACIJA

U priloženoj tabeli su date promene vektora S uz prateće određivanje indeksa koji menjaju vrednosti.

S							i = 0; i = 8;	j = (0 + 0 + 2) % 16 = 2; j = (6 + 8 + 2) % 16 = 0
0	0	2	8					
1	1	0	3	A	C		i = 1; i = 9;	
2	2	0	1				j = (2 + 1 + F) % 16 = 2; j = (0 + 9 + F) % 16 = 8;	
3	3	4					i = 2; i = A;	
4	4	3	0				j = (2 + 1 + F) % 16 = 2 j = (8 + A + F) % 16 = 1	
5	5	7					i = 3; i = B	
6	6	E	5	6	D		j = (2 + 3 + F) % 16 = 4 j = (1 + B + F) % 16 = 11	
7	7	5	E				i = 4; i = C;	
8	8	2	9				j = (4 + 3 + A) % 16 = 1 j = (B + C + A) % 16 = 1	
9	9	2					i = 5; i = D	
A	A	3					j = (1 + 5 + 1) % 16 = 7; j = (1 + D + 1) % 16 = F	
B	B						i = 6; i = E	
C	C	A					j = (7 + 6 + 1) % 10 = E j = (F + 6 + 1) % 10 = 6	
D	D	F					i = 7; i = F	
E	E	6	5					j = (E + 5 + 3) % 10 = 10
F	F	D	6				= 6 j = (6 + D + 3) % 10 = 6	

$$S = [8 \ C \ 1 \ 4 \ 0 \ 7 \ D \ E \ 9 \ 2 \ 3 \ B \ A \ F \ 5 \ 6]$$

9.5 Knapsack

9.5.1 Zadatak 1

Neka je dat privatni ključ po Knapsack algoritmu niz **(2,6,9,21,46,99)**, vrednost **m=39** i moduo **n=194**. Odrediti vrednost “inverzno m” (**im**) i javni ključ, i izračunati sledeće:

- Kriptovanu vrednost, ako je polazni podatak $M=56$
- Polazni podatak, ako je kriptovana vrednost $C=165$
- Polazni podatak, ako je kriptovana vrednost $C=293$

Koraci Knapsack algoritma:

1. Odlučimo kakve ćemo podatke da kriptujemo (koje su veličine podaci koje kriptujemo)
2. Odredimo super-rastući niz **P**, čiji je broj elemenata jednak veličini podataka koje želimo da kriptujemo (veličina u bitovima)
3. Odredimo broj **n** kao nasatavak super-rastućeg niza. Pogodno je da **n** bude neparan broj, a najbolje ako je moguće, da bude prost broj
4. Odredimo broj **m** takav da bude uzajamno prost sa brojem **n**: $(m,n)=1$
5. Generisemo novi niz **J** takav da je: $J[i] = (P[i]*m) \text{mod}(n)$, $i=0,..P.\text{Length}$
6. Odredimo “inverzno m” **im** kao: $im*m = 1 \text{mod}(n)$.

Rešenje zadatka:

Najpre treba odrediti vrednost „inverznog m“ **im**:

$$im*m = 1 \text{ mod } (n)$$

Kada zamenimo vrednosti za **n** i **m** dobijamo:

$$39*im = 1 \text{ mod}(194)$$

Najmanji pozitivan broj **m** za koji važi ova jednačina je $im=5$. Kada smo odredili broj **m**, možemo da odredimo članove niza **J** po formuli: $J[i] = (P[i]*m) \text{mod}(n)$, $i=0,..P.\text{Length}$:

$$J[0] = (P[0]*m) \text{mod}(n) = (2*49) \text{mod}(194) = 78$$

$$J[1] = (6*39) \text{mod}(194) = 40$$

$$J[2] = (9*39) \text{mod}(194) = 157$$

$$J[3] = (21*39) \text{mod}(194) = 43$$

$$J[4] = (46*39) \text{mod}(194) = 48$$

$$J[5] = (99*39) \text{mod}(194) = 175$$

Dakle, niz **J = (78,40,157,43,48,15)**

Z A Š T I T A I N F O R M A C I J A

Odredujemo kriptovanu vrednost zapolazni podatak $M = 56$. Najpre prebacimo broj M u binarni oblik:

$$M = 111000_2$$

i	0	1	2	3	4	5
J[i]	78	40	157	43	48	175
M[i]	1	1	1	0	0	0
J[i]*M[i]	78	40	157	0	0	0

Kriptovan podatak C se dobija kao: $\sum_{i=0}^{J.Length} J[i] * M[i]$, odnosno $C = 78+40+157+0+0+0 = 275$.

Odredujemo polazni podatak, ako je kriptovan podatak: $C=165$.

Prilikom dekodiranja, ne dekodiramo podatak C , već podatak: $TC = (C*im)mod(n)$.

Za $C=165$, $TC = (165*5)mod(194) = 49$.

Dekodiranje se izvršava na sledeći način:

i	0	1	2	3	4	5
TCtrenutno	3	3	3	3	49	49
P[i]	2	6	9	21	46	99
M[i]	1	0	0	0	1	0

Tabela se popunjava sa desne strane. Na početku je $TCtrenutno=TC$. Ukoliko je:

$TCtrenutno-P[i]>0$, onda je $M[i] = 1$, a $TCtrenutno = TCtrenutno - P[i]$. U suprotnom, $P[i] = 0$, a $TCtrenutno$ ostaje nepromenjeno. Ukoliko nakon zadnje iteracije $TCtrenutno$ ne postane 0, onda nije moguce dekriptovati polazni podatak. U ovom primeru za $C = 165$, u zadnjoj iteraciji za $TCtrenutno$ dobijamo: $TCtrenutno = 3-2 = 1 \neq 0$ pa nije moguce dekriptovati kodirani podatak.

Isti postupak ponavljamo za kriptovan podatak $C = 293$. Najpre odredujemo:

$TC = (293*5)mod(194) = 107$ i dekriptujemo podatak:

i	0	1	2	3	4	5
TCtrenutno	2	8	8	8	8	107
P[i]	2	6	9	21	46	99
M[i]	1	1	0	0	0	1

Sada je nakon poslednje iteracije $TC_{trenutno}=0$ pa je dekodiranje moguce I dobijeni podatak je:

$$M = 110001_2 = 49.$$

9.5.2 Zadatak 2

Neka je dat privatni ključ po Knapsack algoritmu niz **(3,7,11,29,61,155)**, vrednost **m = 43** i moduo **n = 386**. Odrediti vrednost “inverzno m” (**im**) I javni ključ, i izračunati sledeće:

- Kriptovau vrednost, ako je polazni podatak $M = 49$
- Polazni podatak, ako je kriptovana vrednost $C = 622$
- Polazni podatak, ako je kriptovana vrednost $C = 826$

Resenje:

Obeležimo sa **P** niz **(3,7,11,29,61,155)**. Pošto niz **P** ima 6 članova zaključujemo da su ulazni podaci dužine 6 bitova.

Javni kljuc:

Da bi dobili *javni kljuc* potrebno je da zadati superrastući niz **(3,7,11,29,61,155)** konvertujemo u generalni knapsack niz **J** po sledećoj formuli:

$$\begin{aligned} J[i] &= (P[i] * m) \bmod n, \quad 0 \leq i < 6. \\ J[0] &= (3 * 43) \bmod 386 = 129 \\ J[1] &= (7 * 43) \bmod 386 = 301 \\ J[2] &= (11 * 43) \bmod 386 = 87 \\ J[3] &= (29 * 43) \bmod 386 = 89 \\ J[4] &= (61 * 43) \bmod 386 = 307 \\ J[5] &= (155 * 43) \bmod 386 = 103. \end{aligned}$$

Dobijeni niz **(129,301,87,89,307,103)** pretstavlja javni ključ.

Privatni ključ:

Privatni kljuc čine početni niz **(3,7,11,29,61,155)** i **im** koje se računa kao:

$$m^{-1} \bmod n = 43^{-1} \bmod 386 = 9.$$

Vrednost **im** može se lakše uočiti iz formule:

$$im * m = 1 \bmod n$$

$$im * 43 = 1 \bmod 386$$

Odavde se vidi da ako 43 pomnožimo sa 9 dobijemo 387, što daje ostatak 1 pri deljenju sa 386.

a) **Naći kriptovau vrednost, ako je polazni podatak $M = 49$.**

Kriptovana vrednost se dobija tako sto se prvo svaki element niza javnog ključa pomnoži odgovarajućom binarnom cifrom broja M a zatim nadje suma dobijenih vrednosti.

$$\begin{array}{r} J \quad 129 \quad 301 \quad 87 \quad 89 \quad 307 \quad 103 \\ * \quad * \quad * \quad * \quad * \quad * \\ M \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \end{array}$$

$$C = 1 * 129 + 1 * 301 + 0 * 87 + 0 * 89 + 0 * 307 + 1 * 103 = 533$$

b) **Naći polazni podatak, ako je kriptovana vrednost $C = 622$.**

Za dekriptovanje koristi se privatni ključ, niz P i broj im . Potrebno je prvo izračunati vrednost TC kao:

$$TC = C * im \bmod n = 622 * 9 \bmod 386 = 194.$$

Ako polazni podatak označimo sa M i njegove cifre u binarnom obliku sa a_0, a_1, \dots, a_5 $a_i \in \{0,1\}$, treba izabrati vrednosti a_0-a_5 tako da važi jednakost:

$$a_0 * 3 + a_1 * 7 + a_2 * 11 + a_3 * 29 + a_4 * 61 + a_5 * 155 = 194$$

Vidimo da jednakost važi za vrednosti:

$$a_0 = 1, a_1 = 1, a_2 = 0, a_3 = 1, a_4 = 0, a_5 = 1$$

$$1 * 3 + 1 * 7 + 0 * 11 + 1 * 29 + 0 * 61 + 1 * 155 = 194$$

i zaključujemo da je polazni podatak $M=110101$, odnosno $M=53$.

c) **Naći polazni podatak, ako je kriptovana vrednost $C = 826$.**

Istim postupkom kao pod b) nalazimo da je $TC = 100$ i $M=110110$ odnosno $M=54$.

9.5.3 Zadatak 3

Neka je dat privatni ključ po Knapsack algoritmu niz (3,7,11,29,61,155), vrednost m = 43 i moduo n = 386. Odrediti vrednost „inverzno m“ (im) i javni ključ, i izračunati sledeće:

1. Kriptovanu vrednost, ako je polazni podatak M = 53
2. Polazni podatak, ako je kriptovana vrednost C = 533
3. Polazni podatak, ako je kriptovana vrednost C = 519

Poznato je :

P = (3,7,11,29,61,155) - privatni ključ (superrastući niz)

m = 43

n = 386

Elementi javnog kljuca se dobijaju po formuli : **J[i] = (P[i]*m) mod n**

Izračunati elementi su :

$$J[0]=(3*43) \text{ mod } 386 = 129$$

$$J[1]=(7*43) \text{ mod } 386 = 301$$

$$J[2]=(11*43) \text{ mod } 386 = 87$$

$$J[3]=(29*43) \text{ mod } 386 = 89$$

$$J[4]=(61*43) \text{ mod } 386 = 307$$

$$J[5]=(155*43) \text{ mod } 386 = 103$$

J = (129, 301, 87, 89, 307, 103) - javni ključ

Inverzno m se računa po formuli : **im*m = 1 mod n**

im*43 = 1 mod 386 iz čega sledi da je **im=9**

$$1) \quad M = (53)_{10} = (110101)_2$$

$$J = (129 \quad 301 \quad 87 \quad 89 \quad 307 \quad 103)$$

$$M = 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1$$

$$C = 129 + 301 + 0 + 89 + 0 + 103 = \mathbf{622}$$

Kriptovani podatak je **C = 622**

2) **M=?**, **C=533**

$$TC = C * im \text{ mod } n = 533 * 9 \text{ mod } 386 = 165$$

$$3 \quad <- \quad 10 \quad <- \quad 10 \quad <- \quad 10 \quad <- \quad 10 \quad <- \quad 165$$

Z A Š T I T A I N F O R M A C I J A

P = 3 7 11 29 61 155

1 1 0 0 0 1

$$(110001)_2 \rightarrow (49)_{10}$$

Polazni podatak je **M = 49**

3) M=? , C=519

$$TC = C * im \bmod n = 519 * 9 \bmod 386 = 39$$

3 <- 10 <- 10 <- 39 <- 39 <- 39

P = 3 7 11 29 61 155

1 1 0 1 0 0

$$(110100)_2 \rightarrow (52)_{10}$$

Polazni podatak je **M = 52**

9.5.4 Zadatak 4

Neka je dat privatni ključ po Knapsack algoritmu niz (2,3,7,15,31,63,127), vrednost m=51 i moduo n=305. Odrediti vrednost „inverzno m“(im) i javni ključ, i izračunat sledeće:

1) kriptovanu vrednost ,ako je polazni podatak M=31

2) polazni podatak, ako je kriptovana vrednost C=598

3) polazni podatak, ako je kriptovana vrednost C=470

P= (2,3,7,15,31,63,127)

m=51

n=305

$$J[i] = (P[i] * m) \bmod n$$

$$im * m = 1 \bmod n$$

$$J[0] = (2 * 51) \bmod 305 = 102$$

$$J[1] = (3 * 51) \bmod 305 = 153$$

$$J[2] = (7 * 51) \bmod 305 = 52$$

$$J[3] = (15 * 51) \bmod 305 = 155$$

$$J[4]=(31*51) \bmod 305 = 56$$

$$J[5]=(63 * 51) \bmod 305 = 163$$

$$J[6]=(127*51) \bmod 305 = 72$$

$$\mathbf{J=(102,153,52,155,56,163,72)}$$

$$im*51 = 1 \bmod 305$$

$$\mathbf{im = 6}$$

$$1) M=(113)_{10}=(01110001)_2$$

J	102	153	52	155	56	163	72
M	1	1	1	0	0	0	1

$$C=102*1+153*1+52*1+155*0+56*0+163*0+72*1 = \mathbf{379}$$

$$2) C=\mathbf{598}$$

$$TC = (C*im) \bmod n = (379 * 6) \bmod 305 = 233$$

TC	5-3=2	←12-7=5	←12	←43-31=12	←106-63=43	←233-127=106	←233
	2=2	5>3	12>7	12<15	43>31	106>63	233>127
P	2	3	7	15	31	63	127
M	1	1	1	0	1	1	1

$$M=(1110111)_2=(119)_{10}$$

$$3) C=\mathbf{470} \quad TC = (C*im) \bmod n = (470 * 6) \bmod 305 = 75$$

TC	5-3=2	←12-7=5	←12	←12	←75-63=12	←75	←75
	2=2	5>3	12>7	12<15	12<31	75>63	75<127
P	2	3	7	15	31	63	127
M	1	1	1	0	0	1	0

$$M=(1110010)_2=(114)_{10}$$

9.6 RSA

9.6.1 Zadatak 1

Date su vrednosti $p = 7$ i $q = 29$ koje učestvuju u kreiranju ključa po RSA algoritmu. Ako se zna da je komponenta javnog ključa $e = 5$, odrediti moduo N i komponentu privatnog ključa d (*odrediti minimalno d*),

a zatim izračunati:

1. kriptovani podatak, ako je polazna vrednost $M = 67$
2. kriptovani podatak, ako je polazna vrednost $M = 68$
3. polazni podatak, ako je kriptovana vrednost $C = 74$

Rešenje: Pre nego li počnemo da rešavamo zadatak treba da proverimo da li su p i q prosti brojevi. Kao što vidimo u našem zadatku p i q su prosti brojevi pa možemo da nastavimo sa rešavanjem zadatka.

Prvi korak u rešavanju zadatka je nalaženje vrednosti N (moduo kriptovanja i dekriptovanja) koja se računa po formuli $N = p * q$, to jest, za zadati zadatak će biti:

$$N = 7 * 29 = 203.$$

Sledeći korak je nalaženje $\varphi(N)$ po formuli $\varphi(N) = (p - 1) * (q - 1)$.

$$\varphi(N) = (7 - 1) * (29 - 1) = 6 * 28 = 168.$$

Zadata vrednost (*u zadatku*) e treba da zadovolji uslov $1 < e < \varphi(N)$ i pri tome e i $\varphi(N)$ treba da budu uzajamno prosti brojevi. Par $(e, \varphi(N))$ predstavlja javni ključ. Kada nađemo ove vrednosti onda upotreboom formule $d * e = 1 \bmod \varphi(N)$ nalazimo vrednost $d = \frac{\varphi(N)*k+1}{e}$. Brojilac mora da bude deljiv sa e . Za k uzimamo minimalnu vrednost zbog najlakšeg računanja pri čemu će i d biti minimalno. Iz prethodne jednačine vidimo da d može da ima beskonačno vrednosti za jedno e . Par (d, N) predstavlja privatni ključ.

$$d = \frac{168 * k + 1}{5} \rightarrow k = 3, d = 101.$$

1. Zamenjujući dobijene vrednosti u formulu $C = M^e \bmod N$ za zadatu vrednost $M = 67$, dobijamo:

$$C = 67^5 \bmod 203 = 121.$$

2. Za zadatu vrednost $M = 68$ dobijamo:

$$C = 68^5 \bmod 203 = 66.$$

3. M se računa po formuli $M = C^d \bmod N$, pa za zadatu vrednost u našem zadatku $C = 74$, dobijamo:

$$M = 74^{101} \bmod 203.$$

Sobzirom da je vrednost 74^{101} van opsega digitrona moramo da primenimo određeni algoritam da bi smo dobili rešenje za M . Prevodimo d iz dekadnog u binarni sistem, i dobijamo:

$$(101)_{10} = (1100101)_2$$

Zatim ispisujemo binarne cifre odozgo naniže počevši od najveće težine i računamo:

1 — — — $74 \bmod 203 = 74$ -- kvadrat rezultata koristimo kao prvi član u sledećem koraku

1 — — — $74^2 * 74 \bmod 203 = 36$ – kada radimo na mestu gde je bit = 1, prvi član množimo sa osnovom

$$0 — — — 36^2 \bmod 203 = 78$$

$$0 — — — 78^2 \bmod 203 = 197$$

1 — — — $197^2 * 74 \bmod 203 = 25$ -- bit = 1 pa prvi član množimo sa osnovom (74)

$$0 — — — 25^2 \bmod 203 = 16$$

1 — — — $16^2 * 74 \bmod 203 = 65$ -- bit = 1 pa prvi član množimo sa osnovom (74)

Zadnja vrednost koju smo dobili predstavlja vrednost jednačine $M = 74^{101} \bmod 203 = 65$.

9.6.2 Zadatak 2

Date su vrednosti $p = 7$ i $q = 23$, koje učestvuju u kreiranju ključa po RSA algoritmu. Ako se zna da je komponenta javnog ključa $e = 13$, odrediti moduo N i komponentu privatnog ključa d (odrediti minimalno d), a zatim izračunati:

- Kriptovani podatak, ako je polazna vrednost $M = 67$,
- Kriptovani podatak, ako je polazna vrednost $M = 120$,
- Polazni podatak, ako je kriptovana vrednost $C = 50$.

Rešenje:

$$P = 7$$

$$q = 23$$

$$e = 13$$

$$N, d = ?$$

Modulo kriptovanja/dekriptovanja: $N = p * q = 7 * 23 = 161$.

Izračunamo: $\phi(N) = (p-1) * (q-1) = 6 * 22 = 132$.

Komponenta privatnog ključa: $d * e = 1 \text{ mod } \phi(N)$

$$d * 13 = 1 \text{ mod } 132$$

$$d = (132 * k + 1) / 13 \Rightarrow k = 6, d = 61.$$

Kriptovanje:

$$M = 67$$

$$C = M^e \text{ mod } N = 67^{13} \text{ mod } 161 = 88$$

- Pošto stepenovanje prevazilazi kapacitet kalkulatora potrebno je primeniti algoritam da bi dobili traženu vrednos. Algoritam je sledeći: Broj kojim stepenujemo (13) prevedemo u binarni oblik, a svaku binarnu cifru pišemo jednu ispod druge, pri čemu je cifra najveće težine na vrhu.

$$13 = 1101$$

$$1 - 67 \text{ mod } 161 = 67$$

$$1 - 67^2 * 67 \text{ mod } 161 = 15$$

$$0 - 15^2 \text{ mod } 161 = 64$$

$$1 - 64^2 * 67 \text{ mod } 161 = 88 \Rightarrow C = 88$$

- Na poziciji prve jedinice izračunamo ostatak pri deljenju broja M(67) brojem N(161). Na svakoj sledećoj poziciji gde je 1 računamo ostatak deljenja proizvoda kvadrata ostatka

9. ZADACI SA VEŽBI

prethodnog računanja (642) i broja M(67), i broja N(161). Dok na svakoj poziciji gde je 0 računamo ostatak deljenja kvadrata ostatka prethodnog računanja (152) i broja N (161).

Kriptovanje:

$$M = 120$$

$$C = M^e \bmod N = 120^{13} \bmod 161 = 113$$

$$13 = 1101$$

$$1 - 120 \bmod 161 = 120$$

$$1 - 120^2 * 120 \bmod 161 = 148$$

$$0 - 148^2 \bmod 161 = 8$$

$$1 - 8^2 * 120 \bmod 161 = 113 \Rightarrow C = 113$$

Dekriptovanje:

$$C = 50$$

$$M = C^d \bmod N = 50^{61} \bmod 161 = 71$$

$$61 = 111101$$

$$1 - 50 \bmod 161 = 50$$

$$1 - 50^2 * 50 \bmod 161 = 64$$

$$1 - 64^2 * 50 \bmod 161 = 8$$

$$1 - 8^2 * 50 \bmod 161 = 141$$

$$0 - 141^2 \bmod 161 = 78$$

$$1 - 78^2 * 50 \bmod 161 = 71 \Rightarrow M = 71$$

9.6.3 Zadatak 3

Date su vrednosti $p = 7$ i $q = 37$ koje učestvuju u kreiranju ključa po RSA algoritmu. Ako se zna da je komponenta javnog ključa $e = 5$, odrediti moduo N i komponentu privatnog ključa d (odrediti minimalno d), a zatim izračunati

- kriptovani podatak, ako je polazna vrednost $M = 77$
- kriptovani podatak, ako je polazna vrednost $M = 88$
- polazni podatak, ako je kriptovana vrednost $C = 171$

Rešenje:

$$N = p * q = 7 * 37 = 259$$

$$f_i(N) = (p - 1)(q - 1) = 6 * 36 = 216$$

$$1 < e < f_i(N)$$

$$1 < 5 < 216$$

$$d * e = 1 \pmod{f_i(N)}$$

$$d * 5 = 1 \pmod{216} \Rightarrow d = \frac{216 * k + 1}{5}$$

$$k = 4 \Rightarrow d = 173$$

- $M = 77 \rightarrow C = ?$

$$C = M^e \pmod{N} = 77^5 \pmod{259} = 21$$

- $M = 88 \rightarrow C = ?$

$$C = M^e \pmod{N} = 88^5 \pmod{259} = 177$$

- $C = 171 \rightarrow M = ?$

$$M = C^d \pmod{N} = 171^{173} \pmod{259}$$

$$173_{10} = 10101101_2$$

```

1    171 mod 259 = 171
0    1712 mod 259 = 233
1    2332 * 171 mod 259 = 82
0    822 mod 259 = 249
1    2492 * 171 mod 259 = 6
1    62 * 171 mod 259 = 199
0    1992 mod 259 = 233
1    2332 * 171 mod 259 = 82  => M = 82

```

9.6.4 Zadatak 4

Date su vrednosti $p = 3$ i $q = 73$ koje učestvuju u kreiranju ključa po RSA algoritmu. Ako se zna da je komponenta javnog ključa $e = 11$, odrediti moduo N i komponentu privatnog ključa d (odrediti minimalno d), a zatim izračunati:

- kriptovani podatak, ako je polazna vrednost $M = 102$
- kriptovani podatak, ako je polazna vrednost $M = 121$
- polazni podatak, ako je kriptovana vrednost $C = 135$

Rešenje

$$p = 3, q = 73$$

$$\underline{e = 11}$$

$$n = p \cdot q = 3 \cdot 73 = \mathbf{219}$$

$$F_i(n) = (p-1) \cdot (q-1) = 2 \cdot 72 = 144$$

$$d \cdot e = 1 \pmod{F_i(n)} \Rightarrow d = \frac{F_i(n) \cdot k + 1}{e} = \frac{144 \cdot k + 1}{11}$$

Zamenom vrednosti za k u gornjem izrazu, metodom grube sile, počevši od $k = 0$, dobijamo da je gornja jednačina tačna za $k = 10$, i **minimalno $d = 131$** .

- 1) **M = 102**

$$C = M^e \pmod{n} = 102^{11} \pmod{219} = \mathbf{186}$$

$$11_{10} = 1011_2$$

$$\begin{array}{ll} \mathbf{1} & 102 \bmod 219 = 102 \\ \mathbf{0} & (102)^2 \bmod 219 = 111 \\ \mathbf{1} & (111)^2 * 102 \bmod 219 = 120 \\ \mathbf{1} & (120)^2 * 102 \bmod 219 = 186 \end{array}$$

2) **M = 121**

$$C = M^e \bmod n = 121^{11} \bmod 219 = \mathbf{61}$$

$$11_{10} = 1011_2$$

$$\begin{array}{ll} \mathbf{1} & 121 \bmod 219 = 121 \\ \mathbf{0} & (121)^2 \bmod 219 = 187 \\ \mathbf{1} & (187)^2 * 121 \bmod 219 = 169 \\ \mathbf{1} & (169)^2 * 121 \bmod 219 = 61 \end{array}$$

3) **C = 135**

$$M = C^d \bmod n = 135^{131} \bmod 219 = \mathbf{87}$$

$$131_{10} = 10000011_2$$

$$\begin{array}{ll} \mathbf{1} & 135 \bmod 219 = 135 \\ \mathbf{0} & (135)^2 \bmod 219 = 48 \\ \mathbf{0} & (48)^2 \bmod 219 = 114 \\ \mathbf{0} & (114)^2 \bmod 219 = 75 \end{array}$$

- 0** $(75)^2 \bmod 219 = 150$
- 0** $(150)^2 \bmod 219 = 162$
- 1** $(162)^2 * 135 \bmod 219 = 177$
- 1** $(177)^2 * 135 \bmod 219 = 87$

9.7 Modovi kodera

9.7.1 Output feedback mod

Kod ovog moda, kodirani tekst se dobija, korišćenjem bloka kodiranja koji se zatim XOR-uje sa originalnim tekstrom. Zamena pozicije bita u kodiranom tekstu, automatski povlači zamenu pozicije bita u originalnom tekstu na istoj poziciji. Ovo osobina omogućava da kodovi za korekciju rade normalno iako su primjenjeni pre enkripcije.

Treba napomenuti da je dekodiranje, isto kao i kodiranje, s obzirom na simetričnost xor-operacije

$$C_j = P_j \oplus O_j$$

$$P_j = C_j \oplus O_j$$

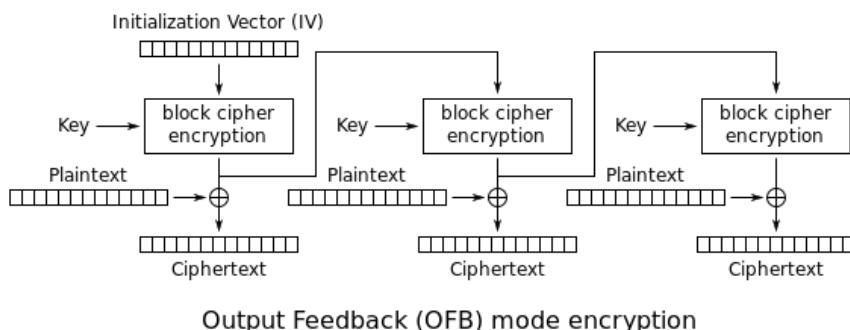
$$O_j = E_K(I_j)$$

$$I_j = O_{j-1}$$

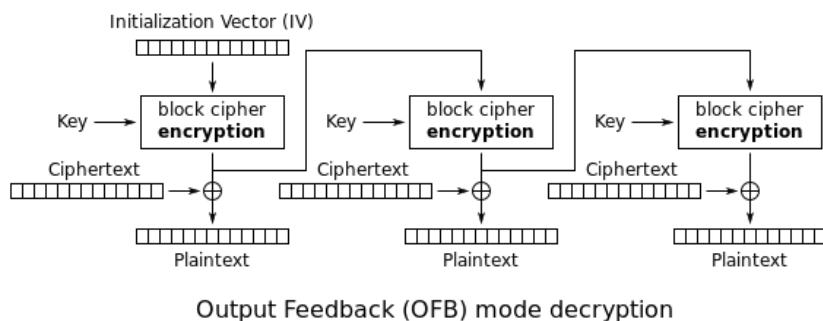
$$I_0 = \text{IV}$$

Svaki novi kodirani blok zavisi od prethodnog, odnosno nove operacije zavise od prethodne, te se ove operacije ne mogu izvršavati paralelno.

Ilustracija algoritma za kodiranje:



Ilustracija algoritma za dekodiranje:



Princip rada:

Pretpostavimo da se u inicijalizacionom vektoru nalazi neka slučajna vrednost. Vrednost se prosleđuje u deo za kodiranje (block cipher encryption-BCE), koji primenom nekih od algoritama, generise kodirani izlaz, koji se kasnije XOR-uje sa originalnim tekstom. U slučaju dekodiranja XOR-uje se sa kodiranim tekstom.

Treba napomenuti da se kodirani izlaz iz BCE, u prvom koraku, prosleđuje sledećem bloku, tj. ta vrednost se koristi kao IV za sledeći blok, i tako dok ima potrebe za kodovanjem.

Pogledajmo na primeru :

Tekst koji treba kodirati:

LALAVOLIRUZE

Slovo	L	A	V	O	I	R	U	Z	E
Kod	0000	0001	0010	0011	0100	0101	0110	0111	1000

Uzećemo proizvoljnu vrednost za inicijalizacioni vektor (16bit npr.)

IV =1000 1010 0110 1100

Razdvojićemo tekst na 3 dela:

t1=LALA = 0000 0001 0000 0001

t2=VOLI = 0010 0011 0000 0100

t3=RUZE = 0101 0110 0111 1000

9. ZADACI SA VEŽBI

Kao sto je navedeno, u delu za šifrovanje, koristi se neki od BCE algoritama. Ovde će biti primenjen one time pad. Neka je ključ:

key=1001 1101 0010 1101

1. iteracija

IV XOR Key -->

1	0	0	0	1	0	1	0	0	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

XOR

1	0	0	1	1	1	0	1	0	0	1	0	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

=

0	0	0	1	0	1	1	1	0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Dobijamo nIV= 0001 0111 0100 0001, ova vrednost se vodi u novi block cipher encryption, a I koristi se za dobijanje sifrovaniog teksta, odnosno xor-uje se sa originalni tekstrom.

LALA = 0000 0001 0000 0001

xor

nIV = 0001 0111 0100 0001

c1 = 0001 0110 0100 0000 ->sifrirani tekst

2. iteracija

nVI xor Key

0	0	0	1	0	1	1	1	0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

XOR

1	0	0	1	1	1	0	1	0	0	1	0	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

=

1	0	0	0	1	0	1	0	0	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ZAŠTITA INFORMACIJA

Dobijamo nnIV= 1000 1010 0110 1100.

VOLI= 0010 0011 0000 0100

Xor

nnIV= 1000 1010 0110 1100

c2= 1010 1001 0110 1000 --> sifrirani tekst

3. iteracija

1	0	0	0	1	0	1	0	0	1	1	0	1	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

XOR

1	0	0	1	1	1	0	1	0	0	1	0	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

=

0	0	0	1	0	1	1	1	0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

nnnIV = 0001 0111 0100 0001, a sifrirani tekst je:

xor

RUZE= 0101 0110 0111 1000

=

c3 = 0100 0001 0011 1001

Ceo šifrirani tekst je C= 0001 0110 0100 0000 1010 1001 0110 1000 0100 0001 0011
1001

Dekodiranje:

Key= 1001 1101 0010 1101

Xor

IV= 1000 1010 0110 1100

= 0001 0111 0100 0001 -> nIV

Xor

= 0001 0110 0100 0000

= 0000 0001 0000 0001 = LALA

nIV=0001 0111 0100 0001

xor

key=1001 1101 0010 1101

= 1000 1010 0110 1100 -> nnIV

Xor

C2= 1010 1001 0110 1000

= 0010 0011 0000 0100 -> VOLI

nnIV=1000 1010 0110 1100

xor

key =1001 1101 0010 1101

= 0001 0111 0100 0001

xor

c3= 0100 0001 0011 1001

= 0101 0110 0111 1000 -> RUZE

Dobili smo početni tekst, tj. LALAVOLIRUZE.

9.7.2 COUNTER MODE (CTR)

Counter mode koristi kod-blokove (blokove ključa), fiksne veličine, koje XOR-uje sa blokovima originalnog teksta kako bi dobio kodirani tekst. Odnosno, blokove ključa XOR-uje sa kodiranim tekstrom da bi dobio blok originalnog teksta. Šifrovani blokovi se razlikuju jer se svaki sastoji iz bitova brojača, a brojač se svaki naredni put uvećava za 1 (ukoliko se koristi funkcija inkrementa za brojač). Brojač može biti bilo koja funkcija koja generiše neku sekvencu, ali pod uslovom da se ista sekvenca ne generiše duži vremenski period. Najkorišćeniji brojač je vrlo jednostavna funkcija – inkrement-za-jedan (increment-by-one). Iako ima dosta sličnosti sa output feedback-om, za razliku od njega, dozvoljava nasumičan pristup tokom dešifrovanja. CTR je prilagođen više procesorskim mašinama, gde se više blokova originalnog teksta može šifrovati paralelno. Isto važi i za dešifrovanje. Takođe rešen je i problem kratkog ciklusa, koji postoji kod OFB-a. Ako se inicijalni vektor (IV ili nonce) bira kao neka proizvoljna vrednost, onda se on kombinuje sa brojačem. Koriste se operacije spajanja (konkateniranja), dodavanja ili XOR operacija. Na taj način se dobija kodirani blok koji se dalje putem dodatnih operacija, kombinuje sa blokom originalnog ili kodiranog teksta. U slučaju da se nonce vrednost ne bira slučajno, vrši se konkatovanje te vrednosti i brojača (nonce vrednost se smešta u viši deo 64-bit a brojač u niži deo). Ukoliko bi umesto

ovoga izvršili dodavanje ili XOR-ovanje nonce-a i brojača u neku vrednost, potpuno bismo narušili bezbednost originalnog teksta (plain text-a) od mogućih nada.

Ove šeme dokazuju da ovaj mod ima osobinu paralelnosti, koja se odnosi na šifrovanje i dešifrovanje. Sa slike 1, gde je prikazano šifrovanje originalnog teksta, vidimo da se blokovi originalnog teksta (plaintext-a) šifruju šifrovanim blokovima (block cipher), čija vrednost se stalno menja, promenom vrednosti brojača (counter-a). Pritom, niti jedan korak ne zahteva rezultat ili neku povratnu vrednost prethodnog koraka. Nezavisni su, te se mogu izvršavati paralelno. Analogno ovome, dokazuje se i paralelnost prilikom dešifrovanja.

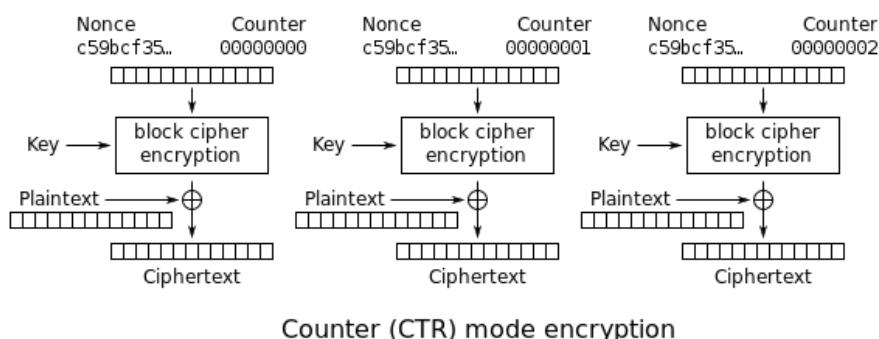
Primer:

Neka je originalan tekst $P = \text{venividivici}$. Ovaj tekst možemo podeliti u blokove od po 4 slova:

$$P_0 = \text{veni}$$

$$P_1 = \text{vidi}$$

$$P_2 = \text{vici}$$



Binarna reprezentacija slova							
Slovo:	v	e	n	i	d	c	s
Binarni kod:	000	001	010	011	100	101	110

Ovi blokovi u binarnom obliku su:

$$P_0 = 000001010011$$

$$P_1 = 000011100011$$

$$P_2 = 000011101011$$

Neka je nonce neki proizvoljan broj:

Nonce= 1100

Šifrovani blok dobijamo konkatovanjem ovog broja i trenutne vrednosti brojača. Brojač će menjati vrednost inkrementacijom.

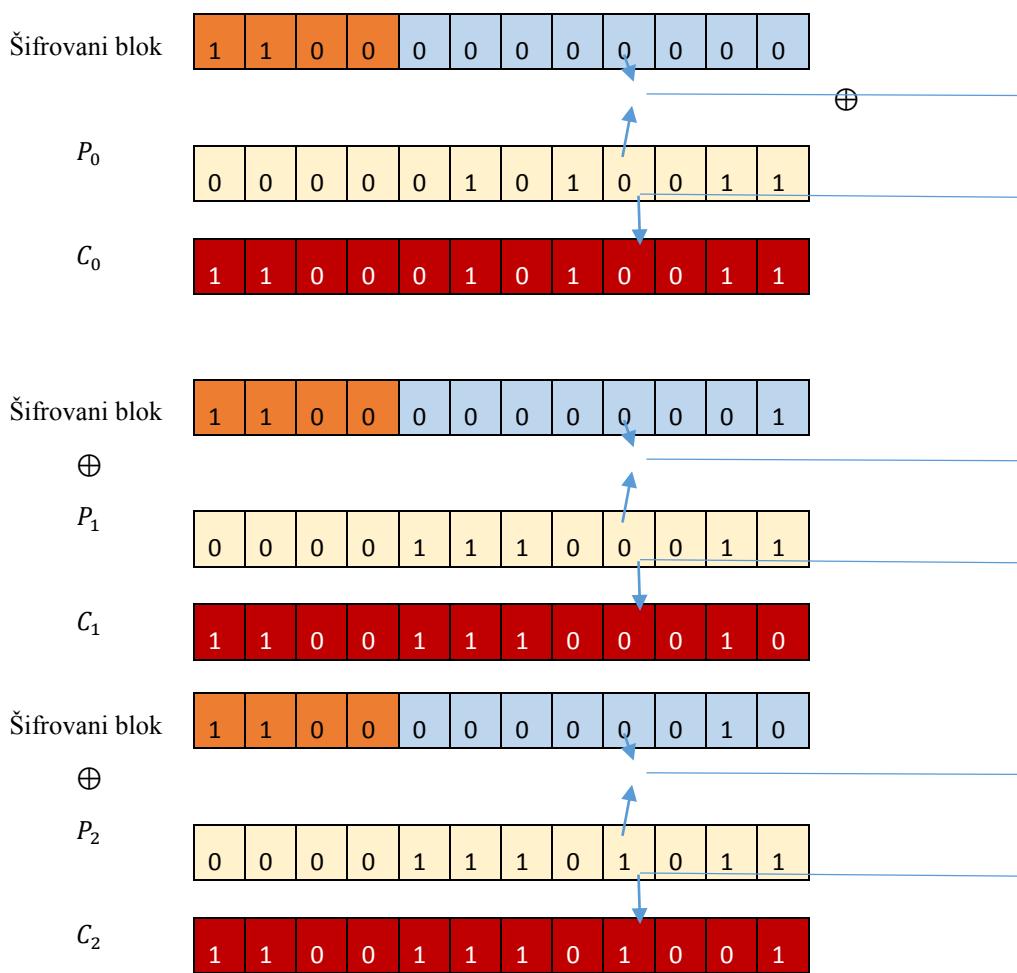
Nonce

Brojač

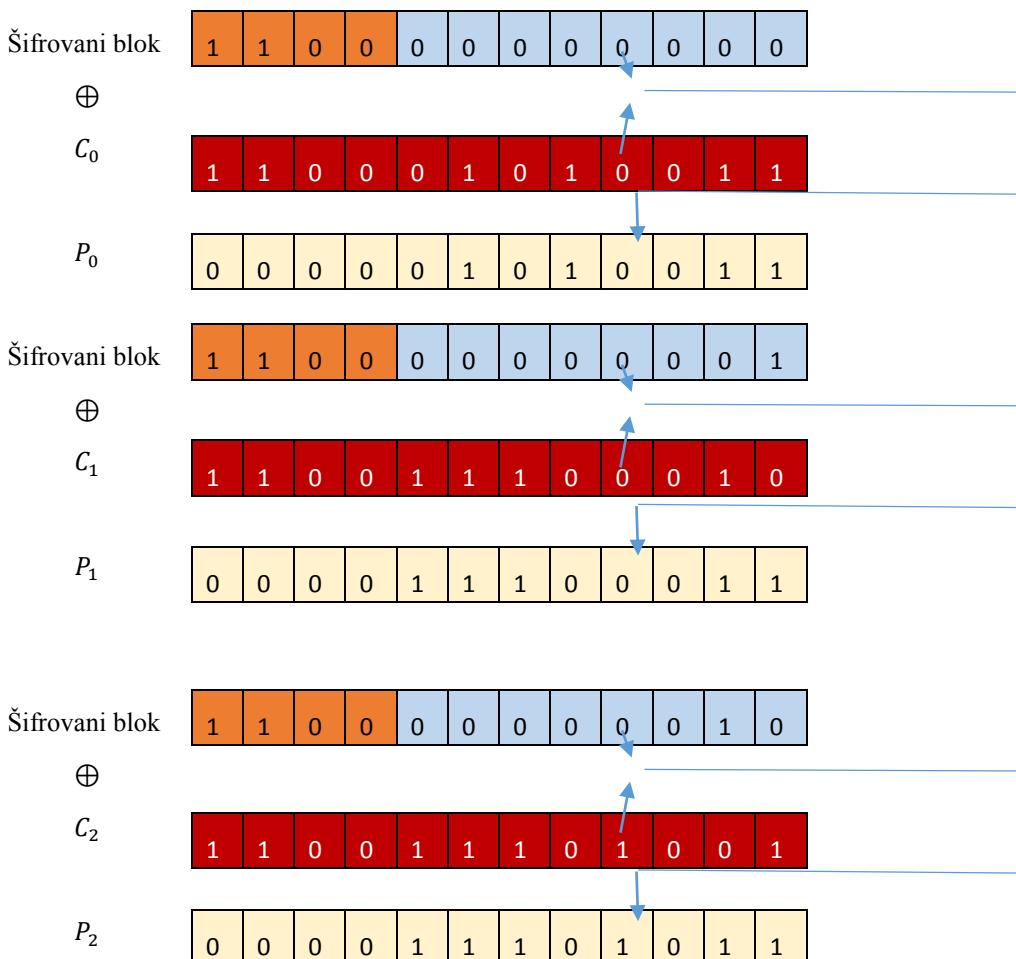
Plain text

Cipher text

XOR-ujemo šifrovani blok i blok P_0 i dobijamo blok šifrovanog teksta C_0 . Postupak ponavljamo i za druge blokove teksta ali se brojač inkrementira.



Dobili smo blokove šifrovanog teksta C_0, C_1, C_2 . Odnosno celokupan šifrovan tekst bio bi: $C=110001010011110011100010110011101001$, tj. $C = \text{senisidnsice}$. Dešifrovanje vršimo XOR-ovanjem određenog šifrovanog bloka i šifrovanog teksta, kako bismo dobili početni tekst.



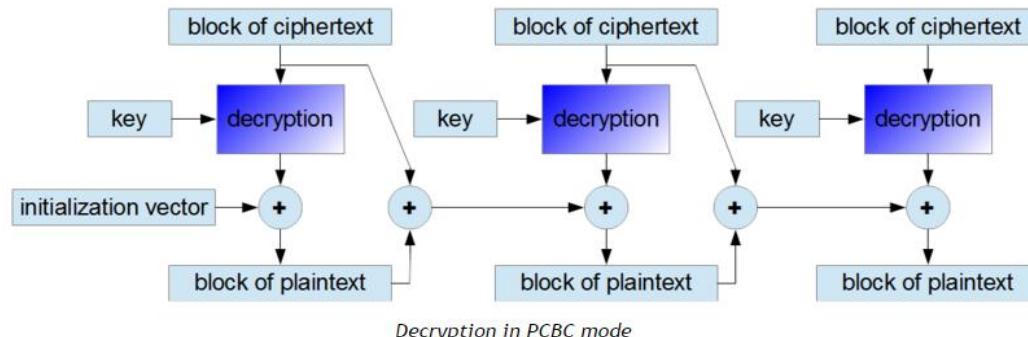
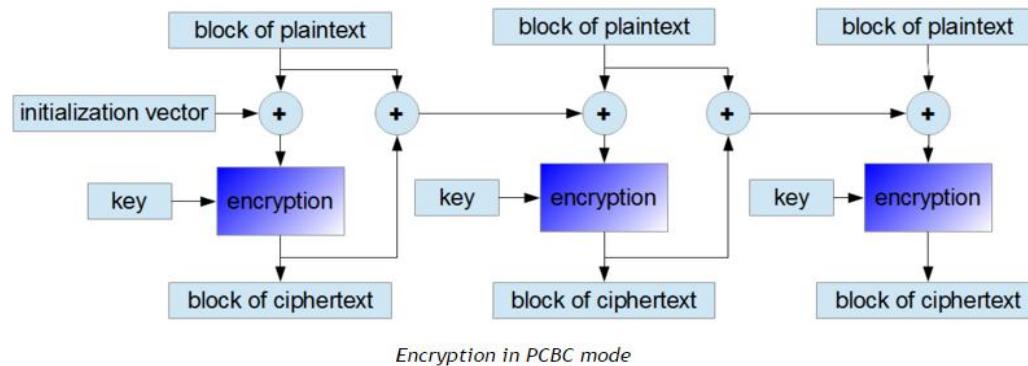
Dešifrovanjem dobijamo blokove originalnog teksta P_0, P_1, P_2 , odnosno početni tekst: *venividivici*

9.7.3 Propagating Cipher Block Chaining PCBC mod

PCBC ili Propagating Cipher Block Chaining predstavlja jedan od modova rada algoritama za kodiranje baziranih na kodiranju blokova podataka.

Većina modova zahtevaju jedinstven početni binarni niz, često pod **nazivom inicijalizacioni vektor** (IV), za svaku operaciju kodiranja.

PCBC je režim je dizajniran da izazove da male promene u kodiranom tekstu se propagiraju beskonačno dugo kada se radi dekodiranje , kao i kada radi šifriranje, drugim rečima ako je jedan bit u kodiranom tekstu oštećen sledeći blok plain text-a i ostali će biti nemoguće pročitati.



U PCBC modu , svaki blok plain text-a je XOR-ovan

sa XOR-vanim prethodnim blokom i prethodnog bloka šifriranog text-a pre nego što je kodiran . Inicijalizacija vektor koristi u prvom bloku .

Na sledećoj strani je data šema koja opisuje rad ovog algoritma kao i matematička formula koja ga opisuje.

Matematički zapis algoritma za šifrovanje i dešifrovanje je :

$$C_i = E_K(P_i \oplus P_{i-1} \oplus C_{i-1}), P_0 \oplus C_0 = IV$$

$$P_i = D_K(C_i) \oplus P_{i-1} \oplus C_{i-1}, P_0 \oplus C_0 = IV$$

Gde su:

IV – inicijalizacioni vektor

Pi – i-ti block Plain Text-a

Ci – i-ti block Cipher Texta-a

Ek – operacija ekripotvanja

Dk – operacija dekripotovanja

Tabela koja opisuje karakteristike ovog moda kodera:

PCBC	
Propagating Cipher Block Chaining	
Encryption parallelizable:	No
Decryption parallelizable:	No
Random read access:	No

PRIMER :

Neka je Plain Text rasporedjen u tri blocka po 8 bita

$P_0 = [00111100]$

$P_1 = [10101010]$

$P_2 = [11100110]$

I neka je IV zadat:

$IV = [00110101]$

Neka su Encrypt i Decrypt operacija definisana sledecim pravilom:

$E[0] \Rightarrow [1]$

$E[1] \Rightarrow [0]$

$D[0] \Rightarrow [1]$

$E[1] \Rightarrow [0]$

Sada primenimo algoritam:

Korak 0

$P_0 \text{ xor } IV = [00001001] = U_0$

9. ZADACI SA VEŽBI

$E[U_0] = [11110110] = C_0$

$C_0 = [11110110]$

Uvedimo Tmp kao pomocnu promenljivo koja se koristi u koracima 0> umesto IV.

$\text{Tmp} = P_0 \text{ xor } C_0 = [11001010]$

Korak 1

$P_1 \text{ xor } \text{Tmp} = [01100000] = U_1$

$E[U_1] = [10011111] = C_1$

$C_1 = [10011111]$

$\text{Tmp} = P_1 \text{ xor } C_1 = [00110101]$

Korak 2

$P_2 \text{ xor } \text{Tmp} = [11010011] = U_2$

$E[U_2] = [00101100] = C_2$

$C_2 = [00101100]$

Dakle dobili smo kripotvanje blokovi C_0, C_1 i C_2

$[11110110], [10011111], [00101100]$ respektivno.

Izvrsimo sada dekripciju kako bismo dobili izvorni(Plain Text).

Korak 0

$D[C_0] = [00001001] = U_0$

$IV = [00110101]$

$U_0 \text{ xor } IV = [00111100] = P_0$

$P_0 \text{ xor } C_0 = [11001010] = \text{Tmp}$

$P_0 = [00111100]$, dakle kao sto vidimo P_0 je isto kao i u izvornom tekstu idemo dalje...

Korak 1

$D[C_1] = D[10011111] = [01100000] = U_1$

$[11001010] = \text{Tmp}$

$U_1 \text{ xor } \text{Tmp} = [10101010] = P_1$

$\text{Tmp} = U_1 \text{ xor } P_1 = [00110101]$

$P_1 = [10101010]$, dakle i P_1 je isti kao i u izvornom text-u

Korak 2

$$D[C2] = D[00101100] = [11010011] = U2$$

$$Tmp = [00110101]$$

$$U2 \text{ xor } Tmp = [11100110] = P2$$

$$P2 = [11100110]$$

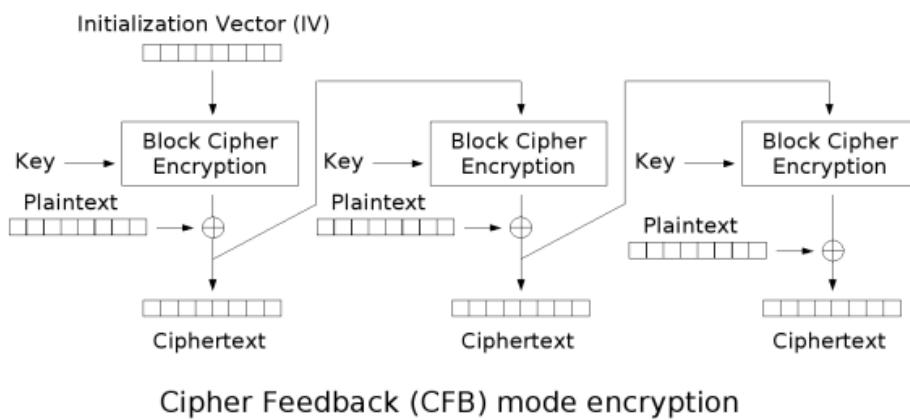
P0,P1,P2 su [00111100] , [10101010], [11100110] respektivno sto smo i zadali kao Plain Text.

$$P0 = [00111100]$$

$$P1 = [10101010]$$

$$P2 = [11100110]$$

CFB – Cipher Feedback kodiranje teksta uz korišćenje “One-time-pad” algoritma za kodiranje



Tekst koji treba kodirati je: „**Ugovor o poverljivosti podataka je pisani pravni ugovor.**“ Iz teksta se uklanjuju blanko i specijalni karakteri, a sva slova se pretvaraju u mala, radi lakšeg kodiranja. Za kodiranje teksta koristimo “One-time-pad” algoritam.

Tablica kodova za svaki od karaktera:

u	0000	j	1000
g	0001	i	1001
o	0010	s	1010
v	0011	t	1011
r	0100	d	1100
p	0101	a	1101
e	0110	k	1110
l	0111	n	1111

Izaberimo za ključ $k = 1011000101001110$, a za inicijalizujući vektor $IV = \text{"riakogpi"} = 01001001110111100010000101011000$. Na osnovu algoritma One-time-pad kodiramo IV . Pošto svaki karakter kodiramo sa 4 bita, a treba nam 32 bita za primenu XOR operacije, grupisaćemo tekst u grupe od po 8 karaktera. Dakle, delimo ulazni tekst u 6 blokova, odvojenih vertikalnim crtama: **ugovorop|overljiv|ostipoda|takajepi|saniprav|niugovor**. Prvi blok P_1 je predstavljen kao: 00000001001000110010010000100101, drugi kao: 0010001101100100011100010010011, itd. Sa C_1, C_2, \dots, C_6 označeni su kodirani blokovi koji, kada se nadovežu, čine kodirani ulazni tekst.

$C_1 = (IV \text{ xor } K) \text{ xor } P_1 = (0100100111011100010000101011000 \text{ xor}$
 $10110001010011101011000101001110) \text{ xor } 00000001001000110010010000100101 =$
 $1111100010010000100100000010110 \text{ xor } 00000001001000110010010000100101 =$
 $11111001101100111011010000110011.$

C_1 koristimo umesto IV za kodiranje drugog bloka i tako redom.

$C_2 = (C_1 \text{ xor } K) \text{ xor } P_2 = (11111001101100111011010000110011 \text{ xor}$
 $10110001010011101011000101001110) \text{ xor } 0010001101100100011100010010011 =$
 $0100100011111010000010101111101 \text{ xor } 0010001101100100011100010010011 =$
 $0110101110011001011110111101110.$

$C_3 = (C_2 \text{ xor } K) \text{ xor } P_3 = (0110101110011001011110111101110 \text{ xor}$
 $10110001010011101011000101001110) \text{ xor } 00101010101110010101001011001101 =$
 $1101101011010111110011001010000 \text{ xor } 00101010101110010101001011001101 =$
 $11110000011011101001111001101101.$

$C_4 = (C_3 \text{ xor } K) \text{ xor } P_4 = (11110000011011101001111001101101 \text{ xor}$
 $10110001010011101011000101001110) \text{ xor } 1011110111011011000011001011001 =$
 $01000001001000000010111100100011 \text{ xor } 1011110111011011000011001011001 =$
 $1111100110011011010100101111010.$

ZAŠTITA INFORMACIJA

$C5 = (C4 \text{ xor } K) \text{ xor } P5 = (11111001100110110100101111010 \text{ xor}$
 $10110001010011101011000101001110) \text{ xor } 10101101111100101010011010011 =$
 $01001101100000110001100000110100 \text{ xor } 10101101111100101010011010011 =$
 $0001110010110111110010110011101.$

$C6 = (C5 \text{ xor } K) \text{ xor } P6 = (0001110010110111110010110011101 \text{ xor}$
 $10110001010011101011000101001110) \text{ xor } 11111001000000010010001100100100 =$
 $10101101111100101010011010011 \text{ xor } 11111001000000010010001100100100 =$
 $0101010011111000011101111110111.$

Dakle, rezultat kodiranja je $C = C1C2C3C4C5C6 =$
 $111110011011001110110100001100110110101110011001011110111101110111100000$
 $110111010011110011011011111100110011011010100101111010000111001011011111$
100101100111010101010011111000011101111110111, a to je tekst:
“nitvrvv|etilakk|nuekikea|nddasils|gdtlkpia|prnjllnl”.

10 ZAKLJUČAK

Ovaj udžbenik je proistekao iz praktičnog rada i iskustva koji su autori stekli držeći teoretska predavanja i računske vežbe iz istoimenog predmeta na IV godini osnovnih studija na Elektronskom fakultetu u Nišu. Autori su u prvom redu namenili ovaj rukopis studentima koji su polaznici ovog kursa.

Imajući u vidu da na našem jeziku ne postoji dovoljno adekvatne literature, pogotovo za ovaj predmet koji se tek od pre nekoliko godina drži kao osnovni kurs, autori smatraju da je ovo izdanje konkretan doprinos nastavi i efikasnom usvajanju znanja iz ove oblasti kao i pripremama studenata za ispite ili kasniji rad u ovoj veoma dinamičnoj i interesantnoj oblasti. Sa druge strane, način na koji je izložena materija omogućava i znatno širu primenu i podršku inžinjerskoj praksi u raznim oblicima i vidovima sistema zaštite informacija, koji su danas veoma aktuelni.

U toku izrade rukopisa autori su se oslanjali na obimnu literaturu, koja je navedena u rukopisu, a naročito na dobro poznatu knjigu Marka Stampa *Information Security: Principles and Practises* iz 2005 godine, koja je i poslužila kao osnova za predloženu klasifikaciju. Udžbenik sadrži veliki broj praktičnih algoritama koji ilustruju brojne teoretske navode, što predstavlja verovatno i njegovu najveću praktičnu vrednost. U skladu sa ovim, autori smatraju da se udžbenik može koristiti kao pomoćna literatura, kako u teoretskom tako i u praktičnom delu iz istoimenog predmeta. Suština zaštite informacija, pogotovo u njenom digitalnom domenu je upravo u upotrebi tih algoritama, tako da se autori nadaju da njihov pristup olakšava savladavanje obimnog gradiva iz ove oblasti i upućuje čitaoca na dalja istraživanja.

ZAŠTITA INFORMACIJA

LISTA SLIKA

Sistem za šifriranje predstavljen kao crna kutija	4
Učestalost pojavljivanja slova u rečima engleskog jezika	8
Osobine funkcije eksluzivno ili	10
Deo rečnika koja je Nemačka koristila pre i za vreme Prvog svetskog rata	14
Čuvani "Cimermanov telegram" kofiran rečnikom	14
Deo rečnika korišćenog za izbore 1876.	15
Ilustracija rada LFRS registra kod A5/1 algoritma	19
LFRS registri za A5/2	20
Generisanje ključa za A5/2	21
Grafički prikaz kriptovanja algoritmom RC6	36
Osnovna šema fajstelovog kodera	40
Fajstelova mreža kod DES algoritma	41
Detaljni prikaz jedne runde u DES-u	42
Fajstelova funkcija u DES algoritmu	45
Distribucija ključa kod DES algoritma	46
Tri suksesivna poziva DES-a u 3DES-u	48
SubBytes korak u AES-u	50
ShiftRows operacija u AES-u	51
MixColumns operacija u AES-u	52
AddRoundKey operacija u kod AES algoritma	54
Dve Fajstelove runde u TEA algoritmu	61
Mreža XXTEA algoritma	65
HMAC - blok šema	78
MD5 algoritma - struktura jedne runde	79
Šema jedne runde u okviru SHA-1	82
Šema jedne runde algoritma SHA-2	85
Enigma (Ljubaznošću TB Perera i Enigma muzeja)	94
Alice i ECB režim	95
Alisa u CBC režimu kodiranja	95
Ključ Diffie-Hellman razmena	96
Diffie-Hellman man-in-the-middle napad	96
Eliptička kriva	97
Tiger spoljašnji ciklus	99
Tiger unutrašnji ciklus za Fm	100
Novčanica sa vodenim žigom	101
Priča o dve Alice	101
Primeri Galton-ovih Minutia	105
Automatsko određivanje Minutia	105
Poređenje Minutia	106
Geometrija šake – merenja	106
Iris skeniranje	107
Histogram irisa - rezultati skeniranja	108
Čitač pametnih kartica. (Ljubaznošću Athena, Inc.)	109

Z A Š T I T A I N F O R M A C I J A

Password generator	109
ACL naspram mogućnosti	111
CAPTCHA (Ljubaznošću Luis von Ahn)	113
Firewall	114
Paket filter	114
Identificuje se prijatelj ili neprijatelj	115
MIG-in-the-middle protokoli za autentifikaciju	116
Prosta autorizacija	117
Autentifikacija sa simetričnim ključem	117
Autentifikacija sa javnim ključem šifrovanje	117
Logičke promenjive	119
Stek – primer	119
Prekoračenje steak izaziva problem	120
Proces kodiranja i dekodiranja kod One-Time-Pad algoritma	122

Literatura

- [1] Mark Stamp: Information Security – Principles and Practices, Wiley-Interscience (October 28, 2005), ISBN-10: 0471738484
- [2] Jean-Luc Chabert, C. Weeks, E. Barbin, and J. Borowczyk: A History of Algorithms: From the Pebble to the Microchip, Springer; 1 edition (August 1, 1999), ISBN-10: 3540633693
- [3] Andrew Jaquith: Security Metrics: Replacing Fear, Uncertainty, and Doubt, Addison-Wesley Professional; 1 edition (April 5, 2007), ISBN-10: 0321349989
- [4] Kyle Loudon: Mastering Algorithms with C, O'Reilly Media; Pap/Dis/CD edition (August 5, 1999), ISBN-10: 1565924533
- [5] Bruce Schneier: Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition, Wiley; 2nd edition (October 18, 1996), ISBN-10: 0471117099
- [6] Bruce Schneier: Secrets and Lies: Digital Security in a Networked World, Wiley; 1 edition (January 30, 2004), ISBN-10: 0471453803
- [7] Man Young Rhee: Internet Security: Cryptographic Principles, Algorithms and Protocols, Wiley (March 31, 2003), ISBN-10: 0470852852
- [8] Thomas W. Cusick and Pantelimon Stanica: Cryptographic Boolean Functions and Applications, Academic Press (March 26, 2009), ISBN-10: 0123748909
- [9] Christopher Swenson: Modern Cryptanalysis: Techniques for Advanced Code Breaking, Wiley (March 17, 2008), ISBN-10: 047013593X
- [10] Gregory V. Bard: Algebraic Cryptoanalysis, Springer; 1 edition (August 24, 2009), ISBN-10: 0387887563
- [11] Harald Niederreiter, Chaoping Xing: Algebraic Geometry in Coding Theory and Cryptography, Princeton University Press (October 11, 2009), ISBN-10: 0691102880
- [12] Henk C.A. van Tilborg: Encyclopedia of Cryptography and Security, Springer; 1 edition (August 10, 2005), ISBN-10: 038723473X
- [13] Serge Vaudenay: A Classical Introduction to Cryptography: Applications for Communications Security, Springer; 1 edition (September 16, 2005), ISBN-10: 0387254641
- [14] Günter Schäfer: Security in Fixed and Wireless Networks: An Introduction to Securing Data Communications, Wiley; 1 edition (March 1, 2004), ISBN-10: 0470863706
- [15] Rolf Oppliger: Contemporary Cryptography (Artech House Computer Security503), Artech House Publishers (April 30, 2005), ISBN-10: 1580
- [16] Shanmugam, Gayathri, Richard M. Low, and Mark Stamp. "Simple substitution distance and metamorphic detection." *Journal of Computer Virology and Hacking*

Techniques 9.3 (2013): 159-170.

[17] Wong, W. H. Timing attacks on RSA: revealing your secrets through the fourth dimension. *Crossroads*, 11(3), 5-5. (2005).

[18] Moon, I., Yi, F., Han, M., & Lee, J. Efficient asymmetric image authentication schemes based on photon counting-double random phase encoding and RSA algorithms. *Applied Optics*, 55(16), 4328-4335. (2016).

[19] Micheli, G., Rosenthal, J., & Schnyder, R. An Information Rate Improvement for a Polynomial Variant of the Naccache-Stern Knapsack Cryptosystem. In *Physical and Data-Link Security Techniques for Future Communication Systems* (pp. 173-180). Springer International Publishing. (2016).

[20] Cilardo, A., & Mazzocca, N. Exploiting vulnerabilities in cryptographic hash functions based on reconfigurable hardware. *IEEE Transactions on Information Forensics and Security*, 8(5), 810-820. (2013).

[21] Bogachkova, I. A., Zaikin, O. S., Kochemazov, S. E. E., Otpushchennikov, I. Y. V., Semenov, A. A. E., & Khamisov, O. O. Problems of search for collisions of cryptographic hash functions of the MD family as variants of Boolean satisfiability problem. *Vychislitel'nye Metody i Programmirovaniye*, 16(1), 61-77. (2015).

[22] Biham, E., Chen, R., & Joux, A. Cryptanalysis of SHA-0 and reduced SHA-1. *Journal of Cryptology*, 28(1), 110-160. (2015).

[23] Shah, K., Kaul, S., & Dhande, M. S. Image Steganography using DWT and Data Encryption Standard (DES). *International oster Science and research*, 3(5). (2015).

[24] Holden, J. Demitasse: A “Small” Version of the Tiny Encryption Algorithm and Its Use in a Classroom Setting. *Cryptologia*, 37(1), 74-83. (2013).

[25] ALabaichi, A., Ahmad, F., & Mahmud, R. Security analysis of blowfish algorithm. In *Informatics and Applications (ICIA), 2013 Second International Conference on* (pp. 12-18). IEEE. (2013, September).

[26] Dhavare, Amrapali, Richard M. Low, and Mark Stamp. "Efficient cryptanalysis of homophonic substitution ciphers." *Cryptologia* 37.3 250-281. (2013):

[27] Biham, E., & Shamir, A. *Differential cryptanalysis of the data encryption standard*. Springer Science & Business Media. (2012).

[28] I. Petković, *A new program for Web protection and compression*. Novi Sad Journal of Mathematics, Vol. 32, No. 1, 167-178. (2002).

[29] I. Petković, Programska paket “Web Encrypt” za zaštitu i enkripciju Web strana, Elektronski fakultet u Nišu, (2003).