APLIKACIJA ZA UČENJE O KRIPTOSUSTAVIMA.

Vinicki, Patrik

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek, 16.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime studenta:	Patrik Vinicki
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4584, 16.09.2019.
OIB studenta:	70065917921
Mentor:	Izv. prof. dr. sc. Vanja Mandrić Radivojević
Sumentor:	Izv. prof. dr. sc. Krešimir Grgić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	Izv. prof. dr. sc. Vanja Mandrić-Radivojević
Član Povjerenstva 2:	Mr.sc. Anđelko Lišnjić
Naslov završnog rada:	APLIKACIJA ZA UČENJE O KRIPTOSUSTAVIMA.
Znanstvena grana rada:	**Telekomunikacije i informatika (zn. polje elektrotehnika)**
Zadatak završnog rada	Opisati kriptosustave i objasniti njihovu funkciju. Napraviti aplikaciju pomoću koje korisnik može vidjeti kako pojedini kriptosustav radi i njime šifrirati odnosno dešifrirati poruke. Implementirati supstitucijske i transpozicijske kriptosustave te DES kriptosustav. Odabrati program za izradu aplikacije.
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	16.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z2S: Zapisnik sa završnog ispita

Osijek, 28.9.2020.

Studentskoj službi

Zapisnik sa završnog ispita

Ime i prezime studenta:	Patrik Vinicki
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4584, 16.09.2019.
Mentor:	Izv. prof. dr. sc. Vanja Mandrić Radivojević
Sumentor:	Izv. prof. dr. sc. Krešimir Grgić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	Izv. prof. dr. sc. Vanja Mandrić-Radivojević
Član Povjerenstva 2:	Mr.sc. Anđelko Lišnjić
Naslov završnog rada:	APLIKACIJA ZA UČENJE O KRIPTOSUSTAVIMA.
Kratko obrazloženje prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina

Završni ispit održan je na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek dana 28.9.2020. u 10:15 sati

Pitanja članova Povjerenstva za završni ispit:	V. Mandrić Radivojević: 1. Zašto je odabran jezik Pyton? 2. Može li se koristiti neki drugi programski jezik? Anđelko Lišnjić: Asimterično kriptiranje. Krešimir Nenadić: Može li se upotrebom sirove sile razbiti poruka?
Mišljenje mentora o pismenom dijelu rada i Povjerenstva o tijeku završnog ispita:	Student je u potpunosti samostalno odradio zadani zadatak. Student je jasno izložio rezultate svoga rada i u potpunosti točno odgovorio na postavljena pitanja.

Ocjena pismenog dijela ispita (završni rad):	Izvrstan (5)
Ocjena usmenog dijela ispita (završni ispit):	Izvrstan (5)
Ukupna ocjena na završnom ispitu:	**Izvrstan (5)**

Predsjednik Povjerenstva:	
Član 1:	
Član 2:	
Zapisničar:	
Pristupnik:	

Uspješno položenim završnim ispitom Pristupnik stječe stručni naziv:

Stručni prvostupnik (baccalaureus) inženjer elektrotehnike smjer Informatika

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 06.10.2020.

Ime i prezime studenta:	Patrik Vinicki
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4584, 16.09.2019.
Turnitin podudaranje [%]:	11

Ovom izjavom izjavljujem da je rad pod nazivom**: APLIKACIJA ZA UČENJE O KRIPTOSUSTAVIMA.**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Vanja Mandrić Radivojević

i sumentora Izv. prof. dr. sc. Krešimir Grgić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

APLIKACIJA ZA UČENJE O KRIPTOSUSTAVIMA

Završni rad

Patrik Vinicki

Osijek, 2020.

Contents

1. UVOD

Kriptografija, korištenje šifri i kodova kako bi se osigurale tajne informacije upotrebljava se od rane povijesti čovječanstva. Prahistorijski Grci koristili su kriptografske metode kako bi šifrirali vojne poruke, Rimljani su za šifriranje koristili *„Cezarovu šifru"* , a srednjovjekovni narodi koristili su razne supstitucijske šifre. Komunikacija je neizbježna i sve dok ona postoji postojat će i potreba za zaštitom informacija koje se njome prenose.

Danas su ljudi međusobno povezani internetom i mogu u stvarnom vremenu globalno komunicirati jedni s drugima. *„Umreženost"* i povezanost ljudi dovela je do naglog rasta komunikacije i količine informacija koje se prenose mrežom. Porastom količine prenesenih informacija povećava se i broj osoba koje neovlašteno žele pristupiti tim informacijama i zlouporabiti ih. Kako bi se informacija koja se prenosi vanjskom mrežom zaštitila implementira se kriptosustav, skup kriptografskih tehnika i algoritama koji osigurava informaciju tokom prijenosa.

Ovaj završni rad opisuje kriptosustave i objašnjava kako oni funkcioniraju te donosi aplikaciju pomoću koje korisnik može vidjeti kako pojedini kriptosustav radi i njime šifrirati odnosno dešifrirati poruke. Aplikacija implementira supstitucijske i transpozicijske kriptosustave te DES kriptosustav. Izvedena je pomoću Python-a, interpretiranog programskog jezika opće namjene i visoke razine.

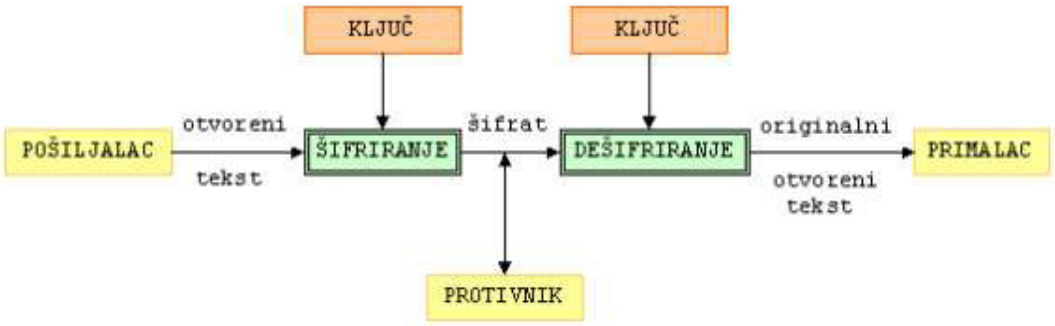
Završni rad podijeljen je na pet glavnih poglavlja. Drugo poglavlje opisuje klasične kriptosustave te pokazuje kako oni funkcioniraju uz primjere. Treće poglavlje detaljno opisuje moderni DES kriptosustav. Četvrto poglavlje opisuje razvoj aplikacije, objašnjava programski jezik Python i skripte koje implementiraju pojedine kriptosustave. Konačno, šesto poglavlje opisuje aplikaciju *„CryptoSys"*. Opisuje i prikazuje korisničko sučelje aplikacije te način pravilnog korištenja pojedine šifre.

2. KRIPTOSUSTAVI

Kriptosustav predstavlja skup kriptografskih algoritama koji se koriste za pružanje određenih usluga informacijske sigurnosti.

- **Autentifikacija** – osigurava autentičnost komunikacije.
- **Povjerljivost podataka** – osiguravanje podataka od neautoriziranog pristupa njihovom sadržaju.
- **Integritet podataka** – osigurava da su podaci primljeni u točno onakvom obliku u kakvom su poslani od strane autoriziranog subjekta (tj. da nisu modificirani te da nema naknadno umetnutih ili uklonjenih dijelova.

Jednostavan model kriptosustava (Slika 1.) može se prikazati pomoću pošiljaoca koji želi prenesti neke osjetljive podatke pimaocu na način da bilo koja treća osoba koja promatra komunikacijski kanal ne može pristupiti izvornim podacima. Cilj ovakvog jednostavnog kriptosustava jest da na posljetku procesa prijenosa informacija samo pošiljalac i primaoc imaju informacije o otvorenom tekstu.



Slika 2.1 *Jednostavan model kriptosustava*

Komponente od kojih se sastoji jednostavan kriptosustav su:

- **Otvoreni tekst** – podaci koje je potrebno zaštiti tokom prijenosa. Izvorna poruka koju pošiljalac želi poslati primaocu
- **Šifrat** – naizgled nečitljiva poruka dobivena kriptografskim postupkom. Šifrat nije zaštićen, bilo koja osoba s pristupom komunikacijskom kanalu može ga pročitati.
- **Algoritam za šifriranje** – kriptografski algoritam koji pomoću određenog otvorenog teksta i ključa za šifriranje proizvodi šifrat.
- **Algoritam za dešifriranje** – kriptografski algoritam koji pomoću određenog šifrata i ključa za dešifriranje proizvodi otvoreni tekst.
- **Ključ za šifriranje** – vrijednost poznata pošiljaocu koja se koristi u algoritmu za šifriranje prilikom generiranja šifrata
- **Ključ za dešifriranje** – vrijednost poznata primaocu koja se koristi u algoritmu za dešifriranje prilikom generiranja otvorenog teksta.

Algoritmi za šifriranje i dešifriranje matematičke su funkcije čija je uloga pretvorba elemenata otvorenog teksta (to mogu biti slova abedece, riječi, brojevi, bitovi itd.) u elemente šifrata i obrnuto.

Kriptosustav definiramo kao uređenu petorku $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ za koju vrijedi:

- $\mathcal{P}$ je konačan skup svih mogućih osnovnih elemenata otvorenog teksta
- $\mathcal{C}$ je konačan skup svih mogućih osnovnih elemenata šifrata
- $\mathcal{K}$ je prostor ključeva, tj. konačan skup svih mogućih ključeva
- $\mathcal{E}$ je skup svih funkcija šifriranja
- $\mathcal{D}$ je skup svih funkcija dešifriranja
- Za svaki ključ $k \in \mathcal{K}$ postoji funkcija šifriranja $E_k \in \mathcal{E}$ i odgovarajuća funkcija dešifriranja $D_k \in \mathcal{D}$

Kriptosustavi se klasificiraju prema sljedećim kriterijima:

1. **Vrsta operacije koja se koristi u postupku šifriranja**

Dijelimo ih na *supstitucijske* šifre, gdje se svaki element otvorenog teksta $\mathcal{P}$ zamjenjuje nekim elementom šifrata $\mathcal{C}$, i *transpozicijske* šifre, gdje se vrši permutacija elemenata otvorenog teksta, odnosno $\mathcal{P} = \mathcal{C}$.

2. **Način obrade otvorenog teksta**

Dijelimo ih na *blokovne šifre* (engl. *block-cipher*), gdje se otvoreni tekst $\mathcal{P}$ obrađuje u blokovima od N elemenata odjednom koristeći isti ključ, i *protočne šifre* (engl. *stream cipher*) gdje se ulazni elementi otvorenog teksta $\mathcal{P}$ obrađuju kontinuirano (jedan po jedan) koristeći paralelno generirani niz ključeva (engl. keystream).

3. **Vrsta ključa**

Razlikujemo *simetrične* (engl. *symmetric*) i *asimetrične* (engl. *asymmetric*) kriptosustave. Simetrični kriptosustavi koriste identičan ključ za šifriranje i dešifriranje, prema tome sigurnost ovakvih sustava ovisi o očuvanju tajnosti ključa pa se stoga često nazivaju kriptosustavi s tajnim ključem. Asimetrični kriptosustavi koriste različite ključeve za šifriranje (javni ključ) i dešifriranje (privatni ključ) te svoju sigurnost temelje na tajnosti ključa za dešifriranje, dok svatko može javnim ključem šifrirati poruku. Ključ za dešifriranje ne može izračunati iz ključa za šifriranje unutar razumnog vremenskog intervala.

2.1 Supstitucijski kriptosustavi

Supstitucijski kriptosustav elemente *p* otvorenog teksta zamjenjuje (supstituira) elementima šifrata *c* ovisno o ključu *k*. Element otvorenog teksta može biti slovo (znak) ili niz slova (znakova). Ukoliko ovakav kriptosustav elemente otvorenog teksta obrađuje znak po znak naziva se *jednostavan supstitucijski kriptosustav*, inače ako obrađuje dva ili više znakova naziva se *poligrafski supstitucijski kriptosustav*. Supstitucijski kriptosustavi funkcioniraju na sljedeći način:

$$E_k(p) = E(k,p) = (p + k)\ mod\ \mathcal{C}, \tag{2-1}$$

$$E_k(c) = D(k,p) = (p - k)\ mod\ \mathcal{C} \tag{2-2}$$

Gdje je:

- $p \in \mathcal{P}$ element otvorenog teksta,
- $c \in \mathcal{C}$ element šifrata,
- $E_k: \mathcal{P} \rightarrow \mathcal{C}$ funkcija šifriranja,
- $D_k: \mathcal{C} \rightarrow \mathcal{P}$ funkcija dešifriranja,
- $k \in \mathcal{K}$ ključ šifriranja

Kod supstitucijskih kriptosustava vrijedi $\mathcal{C} = \mathcal{P} = \mathcal{K}$.

Supstitucijski kriptosustavi općenito se dijele na :

Monoabecedne: monoabecedni supstitucijski kriptosustav koristi istu supstituciju za sve elemente otvorenog teksta, tj. ključ šifriranja je konstantan. Klasičan primjer ovog kriptosustava je *Cezarova šifra* koja pri supstituciji koristi ključ k = 3 , odnosno svaki element otvorenog teksta ciklički pomiče po prostoru elemenata šifrata za tri mjesta.

Poliabecedne: poliabecedni supstitucijski kriptosustav koristi nekoliko različitih supstitucija tokom šifriranja poruke, tj. ključ šifriranja različit je za različite elemente otvorenog teksta. Primjer ovog kriptosustava je *Vigenèreova šifra* koju ćemo objasniti u nastavku.

Prostor elemenata šifrata $\mathcal{C}$ naziva se supstitucijska abeceda i najčešće se sastoji od istih elemenata kao i prostor otvorenog teksta $\mathcal{P}$. Pretpostavimo da za šifriranje koristimo englesku abecedu koja se

sastoji od 26 znakova, tada supstitucijska abeceda može biti bilo koja permutacija engleske abecede, to znači da postoji 26! mogućih ključeva.

Bez obzira na velik prostor mogućih ključeva supstitucijska šifra vrlo je nesigurna i relativno lagano se može razbiti frekvencijskom analizom. Pod pretpostavkom da je šifrirana poruka razumne duljine (neki tekst) moguće je odrediti frekvenciju pojavljivanja znakova u šifratu. Frekvencijska distribucija znakova šifrata uspoređuje se sa frekvencijom pojavljivanja slova u jeziku te se podudaranjem frekvencija pojavljivanja slovo abecede asocira s odgovarajućim znakom šifrata. Na ovaj način mogu se otkriti najfrekventnija slova i dobiti polovične riječi koje sa zatim jednostavno dopune kako bi se dobio otvoreni tekst.

Uzmimo kao primjer otvoreni tekst *„SUPSTITUCIJA"* koji želimo šifrirati koristeći *Cezarovu šifru*, prema tome je ključ šifriranja k = 3. Kao supstitucijsku abecedu koristit ćemo englesku abecedu (bez permutacije). Elementi otvorenog teksta ne moraju se nužno prikazivati znakovima već se mogu zamijeniti brojevima, tj. svako slovo poprima vrijednost svog rednog broja u abecedi. Tako definiramo skup $\{0, 1, \ldots, 25\}$ označen sa $\mathbf{Z_{26}}$ te pretpostavljamo da su na njemu definirane operacije zbrajanja, oduzimanja i množenja na isti način kao u skupu cijelih brojeva.

Na svaki znak otvorenog teksta primjenjuje se funkcija šifriranja (2-1)

$$E_k(S) = (19 + 3)\ mod\ 26 = 22,$$

$$E_k(U) = (21 + 3)\ mod\ 26 = 24,$$

$$E_k(P) = (16 + 3)\ mod\ 26 = 19$$

Itd.

Na kraju dobivamo:

Otvoreni tekst	S	U	P	S	T	I	T	U	C	I	J	A
Šifrat	V	X	S	V	W	L	W	X	F	L	M	D

Primjećujemo kako se ista slova otvorenog teksta mijenjaju istim slovima šifrata zbog čega je šifra vrlo podložna napadu frekvencijskom analizom. Ovu manu imaju svi monoabecedni supstitucijski kriptosustavi.

Postoje dvije osnovne metode koje se primjenjuju kod supstitucijskih šifri kako bi se poboljšala sigurnost šifre:

Uporaba blokova slova: umjesto pojedinačnih slova kao osnovni elementi otvorenog teksta koriste se blokovi slova kojima se vrši supstitucija. Ovakav način šifriranja koristi Playfairova šifra koju ćemo detaljnije obraditi u nastavku

Uporaba različitih abeceda za šifriranje: slovo otvorenog teksta može se preslikati u više različitih slova šifrata. Ovakav način rada čini šifru otpornijom na napad frekvencijskom analizom i primjenjuje je Vigenereova šifra.

2.2 Vigenèreova šifra

Vigenèreova šifra je poliabecedna supstitucijska šifra, tj. svako slovo otvorenog teksta *p* može se preslikati u jedno od *m* mogućih slova (gdje je *m* duljina ključa) šifrata *c* ovisno o svom položaju unutar otvorenog teksta. Vigenèreova šifra elemente p_i otvorenog teksta duljine *n* pomoću ključa duljine *m* pretvara u elemente šifrata c_i, definiramo ju na sljedeći način:

$$E_k(p_i) = E(k_i, p_i) = (p_i + k_{i \bmod m}) \bmod \mathcal{C}, \tag{2-3}$$

$$D_k(c_i) = E(k_i, c_i) = (c_i + k_{i \bmod m}) \bmod \mathcal{C} \tag{2-4}$$

Gdje je:

- $p_i \in \mathcal{P}$ element otvorenog teksta,
- $c_i \in \mathcal{C}$ element šifrata,
- $E_k: \mathcal{P} \rightarrow \mathcal{C}$ funkcija šifriranja,
- $D_k: \mathcal{C} \rightarrow \mathcal{P}$ funkcija dešifriranja,
- $k_i \in \mathcal{K}$ element ključa šifriranja
- n duljina otvorenog teksta, a m duljina ključa pri čemu je najčešće $m < n$

Vigenèreova šifre općenito se dijele prema vrsti ključa koja se koristi prilikom šifriranja:

Klasična Vigenèreova šifra: kao ključ šifriranja koristi se proizvoljno odabrana ključna riječ pri čemu je duljina ključne riječi manja ili jednaka duljini otvorenog teksta. Ukoliko je $m < n$ ključna riječ se ponavlja kako bi svaki element otvorenog teksta imao odgovarajući element ključa za šifriranje.

Vigenèreova šifra s autoključem: kao i kod klasične šifre kao ključ šifriranja se koristi proizvoljno odabrana ključna riječ, međutim, nakon ključne riječi naredni elementi ključa su elementi otvorenog teksta. .

Kao što je prethodno spomenuto *Vigenèreova šifra* je poliabecedna, tj. isti element otvorenog teksta može preslikati u više elemenata šifrata, što ju čini značajno otpornijom na napad frekvencijskom analizom.

Uzmimo kao primjer otvoreni tekst *„VIGENERE“* koji želimo šifrirati ključem *„TAJNA“*. Kao i kod *Cezarove šifre* elemente otvorenog teksta i ključa možemo definirati kao skup $\{0, 1, \ldots, 25\}$ označen sa $\mathbf{Z_{26}}$**.** Zatim na svaki element otvorenog teksta primjenjujemo funkciju šifriranja (2-3):

$$E_k(V) = (21 + 19)\ mod\ 26 = 14,$$

$$E_k(I) = (8 + 0)\ mod\ 26 = 18$$

$$E_k(G) = (6 + 9)\ mod\ 26 = 11$$

Itd.

Primjenom funkcije šifriranja na svaki element otvorenog teksta dobiva se sljedeći šifrat:

Otvoreni tekst	V	I	G	E	N	E	R	E
Ključ	T	A	J	N	A	T	A	J
Šifrat	O	I	P	R	N	X	R	N

Korištenjem *Vigenèreove šifre* s autoključem dobivamo sljedeći šifrat:

Otvoreni tekst	V	I	G	E	N	E	R	E
Ključ	T	A	J	N	A	V	I	G
Šifrat	O	I	P	R	N	Z	Z	K

Šifrirati se također može pomoću tablice koju nazivamo *Vigenèreova tablica* ili *Vigenèreov kvadrat*. *Vigenèreov kvadrat* prestavlja matricu s $\mathcal{P}\ x\ \mathcal{C}$ elemenata. Indeksi redaka tablice prestavljaju prostor mogućih elemanata otvorenog teksta, a indeksi stupaca prostor mogućih elemanata šifrata, tj. element otvorenog teksta indeksira stupac, a element tajnog ključa redak *Vigenèreovog kvadrata* . Ukoliko za šifriranje koristimo englesku abecedu imamo *Vigenèreov kvadrat* s $26\ x\ 26$, odnosno 676 elemenata (tablica 2.1). Element otvorenog teksta u tom slučaju teoretski može poprimiti 26 različitih vrijednosti ovisno o odgovarajućem elementu ključa te se šifra može shvatiti kao 26 međusobno isprepletenih *Cezarovih šifri*. Šifrirajmo ponovno otvoreni tekst *„VIGENERE“* pomoću ključne riječi *„TAJNA“* koristeći *Vigenèreov kvadrat*. Određivanje prvog elementa šifrata prikazano je tablicom 2.1

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Tablica 2.1 *Vigenèreov kvadrat*

Postupak se ponavlja za svaki element otvorenog teksta te na kraju dobivamo šifrat *„OIPRNXRN“*, jednako kao i primjenom funkcije šifriranja.

2.3 Playfairova šifra

Playfairova šifra je supstitucijski kriptosustav koji supsituciju vrši nad parovima slova (bigramima) umjesto da supstituira svako slovo pojedinačno. Ova šifra značajno je otpornija na kriptoanalizu korištenjem frekvencijske analize jer šifriranjem bigrama postoji 26^2 odnosno 676 mogućih bigrama umjesto 26 monograma koje dobivamo klasičnom supstitucijom.

Uzmimo kao primjer ključnu riječ *„PRIMJER PLAYFAIR“* koji želimo šifrirati. Algoritam se temelji na matrici slova dimenzija 5 x 5 koja se konstruira pomoću ključne riječi. Matrica se formira tako da se ključna riječ upisuje na početak (slijeva na desno, odozgo prema dolje) bez ponavljanja slova. Ostatak matrice popunjava se preostalim slovima abecede, pri čemu se slova I i J poistovjećuju (šifriraju se jednako), ili se neko slovo izbacuje (najčešće se izbacuje slovo Q).

P	R	I	M	J
E	L	A	Y	F
B	C	D	G	H
K	N	O	S	T
U	V	W	X	Z

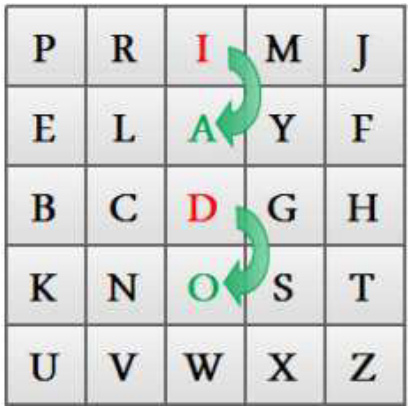
Slika 2.2 *Matrica slova*

Prilikom šifriranja, otvoreni tekst se najprije podijeli na blokove od po dva slova (bigrame), uzimajući u obzir da se nijedan blok ne sastoji od dva jednaka slova, kao i da je duljina teksta paran broj (prema potrebi umeće se slovo X). Imamo bigrame *„PR IM JE RP LA YF AI RX“*, kako nemamo paran broj elemenata ključne riječi na kranje slovo *„R“* nadodajemo slovo *„X“* kako bi formirali bigram. Dobiveni bigrami šifriraju se prema sljedeća tri pravila (ovisno o položaju slova u matrici):

Slova se nalaze u istom retku: zamjenjujemo ih ciklički slovima koja se nalaze za jedno mjesto udesno odgovarajućih slova bigrama.

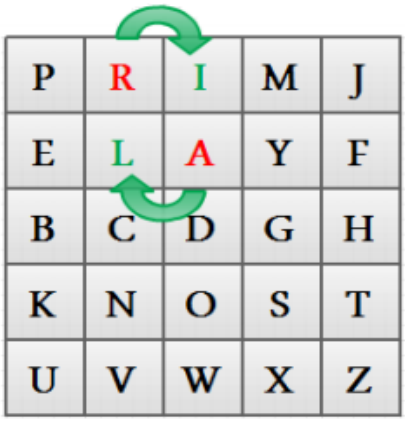
Slika 2.3 *Zamjena slova u istom retku*

Slova se nalaze u istom stupcu: zamjenjujemo ih ciklički slovima koja se nalaze za jedno mjesto ispod odgovarajućih slova bigrama.



Slika 2.4 *Zamjena slova u istom stupcu*

Slova nisu u istom retku ili stupcu: promatramo pravokutnik kojeg određuju ta dva slova te ih zamjenjujemo s preostala dva vrha tog pravokutnika tako što se zamjenjuju slova vrhova istog retka.



Slika 2.5 *Zamjena slova koja formiraju pravokutnik*

2.4 Hillov kriptosustav

Hillov kriptosustav je poligrafski supstitucijski kriptosustav utemeljen na linearnoj algebri. Zamjenjuje *m* uzastopnih slova otvorenog teksta s *m* slova u šifratu. Ukoliko broj slova otvorenog teksta nije djeljiv sa *m* poruku treba nadopuniti kako bi se mogla podijeliti na blokove od *m* slova. Hillov kriptosustav definira se na sljedeći način:

$$E_k(P) = P\ x\ K\ mod\ \mathcal{C}, \tag{2-5}$$

$$E_k(C) = P\ x\ K\ mod\ \mathcal{C} \tag{2-6}$$

Gdje je:

- $P \in \mathcal{P}$ vektor m elemenata otvorenog teksta,
- $C \in \mathcal{C}$ vektor m elemenata šifrata,
- $E_k: \mathcal{P} \rightarrow \mathcal{C}$ funkcija šifriranja,
- $D_k: \mathcal{C} \rightarrow \mathcal{P}$ funkcija dešifriranja,
- $K \in \mathcal{K}$ invertibilna matrica koja predstavlja ključ šifriranja

Matrica K koja se koristi kao ključ mora biti invertibilna budući da se prilikom dešifriranja koristi njezin inverz. Elemente otvorenog teksta možemo definirati kao skup $\{0, 1, \ldots, 25\}$ označen sa $\mathbf{Z_{26}}$,

šifriranje se zatim vrši rješavanjem *m* linearnih jednadžbi, tj. množenjem svakog bloka od *m* slova (vektor P) s invertibilnom matricom veličine *m x m* koja predstavlja ključ. Naprimjer, za m = 3 imamo:

$$c_1 = (k_{11}p_1 + k_{21}p_2 + k_{31}p_3)\ mod\ 26,$$

$$c_2 = (k_{12}p_1 + k_{22}p_2 + k_{32}p_3)\ mod\ 26,$$

$$c_3 = (k_{13}p_1 + k_{23}p_2 + k_{33}p_3)\ mod\ 26$$

Što možemo prikazati u matričnom obliku kao:

$$[c_1\ \ c_2\ \ c_3] = [p_1\ \ p_2\ \ p_3]\begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}$$

Uzmimo kao primjer otvoreni tekst *„OSIJEK“* koji ćemo šifrirati koristeći Hillovu šifru sa ključem:

$$K = \begin{bmatrix} 5 & 8 & 22 \\ 2 & 5 & 24 \\ 10 & 20 & 17 \end{bmatrix}$$

Slova otvorenog teksta zamjenjujemo brojevnim ekvivalentima te definiramo niz Z = {14, 18, 8, 9, 4, 10}. Kako imamo matricu 3 x 3 elemenata, dobiveni niz Z dijelimo na blokove s 3 elementa po bloku. Tako dobivamo blok p1 = {14, 18, 8}, i blok p2 = {8, 9, 10} koje šifriramo množenjem s matricom K:

$$[14\ \ 18\ \ 8]\begin{bmatrix} 5 & 8 & 22 \\ 2 & 5 & 24 \\ 10 & 20 & 17 \end{bmatrix} = [186\ \ 362\ \ 876]\ mod\ 26 = [4\ \ 24\ \ 18] = E\ Y\ S,$$

$$[8\ \ 9\ \ 10]\begin{bmatrix} 5 & 8 & 22 \\ 2 & 5 & 24 \\ 10 & 20 & 17 \end{bmatrix} = [158\ \ 309\ \ 562]\ mod\ 26 = [2\ \ 23\ \ 16] = C\ X\ Q$$

Tako dobivamo šifrat *„EYSCXQ“*.

Hillov kriptosustav značajno je otporniji na razbijanje frekvencijskom analizom jer šifrira odnosno supstituira *m* elemenata otvorenog teksta odjednom.

2.5 Transpozicijski kriptosustavi

Transpozicijski kriptosustavi permutiraju elemente otvorenog teksta, tj. mijenja se međusobni položaj elemenata otvorenog teksta, za razliku od supstitucijskih kriptosustava koji elemente otvorenog teksta preslikavaju u elemente šifrata. Prema tome šifrat može poprimiti vrijednost određenih permutacija otvorenog teksta. Neka je p skup elemenata otvorenog teksta duljine *n* elemenata , tada su funkcije šifriranja (dešifriranja) permutacije nad p odnosno transpozicijske kriptosustave definiramo kao:

$$E_k\colon\ p^n \rightarrow\ p^n,$$

$$D_k\colon\ p^n \rightarrow\ p^n$$

Gdje je:

- $p \in \mathcal{P}$ otvoreni tekst,
- n duljina ključa šifriranja
- $E_k\colon \mathcal{P}\ \rightarrow\ \mathcal{C}$ funkcija šifriranja,
- $D_k\colon\ \mathcal{C}\ \rightarrow \mathcal{P}$ funkcija dešifriranja,
- $k \in \mathcal{K}$ ključ šifriranja

Najčešće upotrebljavan transpozicijski kriptosustav je tzv. stupčana transpozicija. Otvoreni tekst se po recima upisuje u pravokutnik $n\ x\ (p/n)$ elemenata, a zatim se poruka iščitava po stupcima ali sa izmijenjenim redoslijedom stupaca (ovisno o ključu). Ukoliko posljednji redak nije ispunjen do kraja, on se popunjava proizvoljnim slovima koja neće promijeniti sadržaj poruke (npr. X). Uzmimo kao primjer otvoreni tekst *„TRANSPOZICIJA“* koji želimo šifrirati stupčanom transpozicijom koristeći ključ $k = \{4, 1, 3, 2, 5\}$. Najprije stvaramo pravokutnik u koji upisujemo otvoreni tekst.

Ključ	4	1	3	2	5
Otvoreni tekst	T	R	A	N	S
	P	O	Z	I	C
	I	J	A	X	X

Zatim iščitavamo stupce uzlaznim redoslijedom brojeva ključa. Prema tome dobivamo šifrat: *„ROJ NIX AZA TPI SCX“*.

Čistu stupčanu transpozicijsku šifru vrlo je lako dekriptirati, budući da su frekvencije slova u šifratu i u otvorenom tekstu jednake. Prilikom kriptoanalize ovakve šifre prvi korak je odrediti dimenzije pravokutnika faktoriziranjem broja slova u šifratu. Ukoliko postoji više mogućnosti slova se upisuju u pravotkutnike pretpostavljenih dimenzija po stupcima te se promatra odnos samoglasnika i suglasnika u svakom retku. Ako je odabrana dimenzija pravokutnika točna ovaj odnos neće puno odstupati od njihovog odnosa u jeziku otvorenog teksta (npr. Za hrvatski jezik ovaj omjer je 43% : 57%). Preostaje još odrediti redoslijed stupaca što se u slučaju manjeg broja stupaca može izvesti i „ručnim“ anagramiranjem pri čemu od dodatne koristi mogu biti podaci o frekvencijama bigrama i trigrama.

Kako bi se povećala sigurnost kriptosustava i njegova otpornost na kriptoanalizu najčešće se primjenjuje transpozicija kroz više koraka (stupnjeva), tj. na dobiveni šifrat ponovno se primjenjuje transpozicija te se postupak ponavlja proizvoljan broj puta.

2.6 Jednokratna bilježnica

Jednokratna bilježnica je teoretski apsolutno siguran kriptosustav odnosno nemoguće ga je razbiti kriptoanalizom. Za svaki otvoreni tekst generira se novi potpuno nasumičan ključ jednake duljine. Otvoreni tekst, ključ i šifrat mogu biti niz bitova (znakova) duljine *n*, pri čemu se šifrat dobiva primjenom logičke operacije „isključivo ili (XOR)“ ili modularnim zbrajanjem između odgovarajućih bitova (znakova) otvorenog teksta i ključa, tj. ovakav kriptosustav definira se kao:

$$E_k(p) = p \oplus k, \tag{2-7}$$

$$D_k(c) = c \oplus k \tag{2-8}$$

,

Gdje je:

- $p \in \mathcal{P}$ otvoreni tekst,
- $c \in \mathcal{C}$ šifrat,
- $E_k: \mathcal{P} \rightarrow \mathcal{C}$ funkcija šifriranja,
- $D_k: \mathcal{C} \rightarrow \mathcal{P}$ funkcija dešifriranja,
- $k \in \mathcal{K}$ ključ šifriranja

Ovakav kriptosustav potpuno je siguran, tj. ne postoji nikakva statistička veza između otvorenog teksta i šifrata, ako su zadovoljeni sljedeći uvjeti:

1. Ključ mora biti u potpunosti nasumičan.
2. Ključ mora biti jednake duljine ili veći od otvorenog teksta.
3. Za šifriranje dva različita otvorena teksta ne smije se koristiti isti ključ niti dio prethodno iskorištenog ključa.
4. Ključ mora biti u potpunosti tajan.

Iako je jednokratna bilježnica u teoriji neslomljiva, u praktičnoj uporabi njena implementacija je vrlo problematična. Računalno generirani ključevi najčešće nisu potpuno nasumični već pseudo nasumični, tj. stvaraju se primjenom algoritama na neku fiksnu početnu vrijednost. Dobiveni ključ tada u potpunosti ovisi o početnoj vrijednosti i neko drugo računalo može ponoviti proces i dobiti isti ključ.

Potreba za jedinstvenim ključem prilikom svakog šifriranja u kombinaciji sa zahtjevom ključa koji je jednake duljine kao otvoreni tekst stvara logističke probleme tokom distribucije, održavanja te generiranja ključeva. Potrebno je održavati bazu podataka potencijalno vrlo velikih ključeva te tokom generiranja novog ključa osigurati kako je on uistinu jedinstven i niti jedan njegov dio ne odgovara dijelu nekog prethodnog ključa što je vrlo neefikasno i neprikladno za uporabu. Također dobivene ključeve potrebno je na siguran način proslijediti sudionicima u komunikaciji.

2.7 Simetrični kriptosustavi

Simetričan kriptosustav koristi jedan ključ (tajni ključ) za šifriranje (dešifriranje) otvorenog teksta (šifrata) koji posjeduju i pošiljatelj i primatelj. Otvoreni tekst se obrađuje jedinicu po jedinicu. Jedinica može biti jedan element (broj, slovo, bit) ili više elemenata (niz brojeva, riječ, više bitova). Simetrični kriptosustavi funkcioniraju na sljedeći način:

$$c = e_k(p),$$

$$p = e_k(c)$$

Gdje je:

- $p \in \mathcal{P}$ otvoreni tekst,
- $c \in \mathcal{C}$ šifrat,
- $e_k: \mathcal{P} \rightarrow \mathcal{C}$ funkcija šifriranja,
- $e_k: \mathcal{C} \rightarrow \mathcal{P}$ funkcija dešifriranja,
- $k \in \mathcal{K}$ tajni ključ

2.8 Blokovni kriptosustavi

Glavno obilježje blokovnih sustava je način na koji se vrši šifriranje. Blokovni kriptosustavi rade nad skupinama podataka određene veličina, tzv. blokovima. Ulazne podatke varijabilne duljine dijele na blokove, zatim procesiraju blok po blok te iz njih tvore blokove šifrata jednake duljine. Blokovni kriptosustavi funkcioniraju na sljedeći način:

$$c = e_k(p),$$

$$p = e_k(c)$$

Gdje je:

- $p \in \mathcal{P}$ blok otvorenog teksta,
- $c \in \mathcal{C}$ blok šifrata,
- $e_k: \mathcal{P} \rightarrow \mathcal{C}$ funkcija šifriranja,
- $e_k: \mathcal{C} \rightarrow \mathcal{P}$ funkcija dešifriranja,

- $k \in \mathcal{K}$ tajni ključ

Blokovna šifra sastoji se od dva algoritma, jednog za šifriranje, E, i drugog za dešifriranje, D koji šifriraju (dešifriraju) blokove otvorenog teksta (šifrata). Algoritam za šifriranje E kao ulaz prima blok veličine *n* bitova i ključ veličine *k* bitova i opisuje se funkcijom šifriranja :

$$E_k(P) := E(K,P) : \{0,1\}^k \, x \, \{0,1\}^n \rightarrow \{0,1\}^n \tag{2-9}$$

,

Funkcija dešifriranja D inverzna je funkciji šifriranja i kao ulaz prima ključ K veličine k i šifrat C :

$$E_k^{-1}(C) := D_k(C) = D(K,C) : \{0,1\}^k \, x \, \{0,1\}^n \rightarrow \{0,1\}^n, \tag{2-10}$$

$$D = E^{-1}$$

Primjećujemo kako su rezultati funkcije šifriranja (dešifriranja) permutacije nad $\{0,1\}^n$, prema tome u blokovnim sustavima je $\mathcal{P} = \mathcal{C} = \{0,1\}^n$.

Najčešće veličine blokova su 64 bita (DES) i 128 bitova u modernijim kriptosustavima. Kako bi se povećala sigurnost sustava i njegova otpornost na kriptoanalizu koriste se „*načini djelovanja*“ (engl. *modes of operation*) gdje se blokovi ulančavaju tj. izlazni blok šifrata koristi se u šifriranju sljedećeg bloka otvorenog teksta. Ovakav način rada čini blokove zavisnima i u slučaju istog otvorenog teksta proizvest će se potpuno drugačiji šifrat.

Najčešće i naprednije metode razbijanja blokovnih šifri su:

Linearna kriptoanaliza: šifra se pokušava razbiti linearnom aproksimacijom. Stvaraju se linearne jednadžbe koje povezuju otvoreni tekst, šifrat i ključ, zatim se jednadžbe primjenjuju na poznate parove otvorenog teksta i šifrata kako bi se dobili određeni bitovi ključa.

Diferencijalna kriptoanaliza: temelji se na uočavanju razlika u šifratu s obzirom na različite otvorene tekstove. Ovaj napad je najčešće „izabrani otvoreni tekst“ odnosno napadač mora imati mogućnost dobiti šifrate za neki niz proizvoljno odabranih otvorenih tekstova. Nakon šifriranja otvorenih tekstova računa se razlika između odgovarajućih šifrata pomoću isključivo ili logičke operacije. Analizom izračunatih razlika pokušava se pronaći statistički značajan uzorak koji će dati korisne informacije o sustavu.

2.9 Sigurnost kriptosustava

Poželjno je da funkcije šifriranja i dešifriranja budu javne i da sigurnost sustava ovisi samo o tajnosti ključa, ovo načelo nazivamo *Kerckhoffsovo načelo.* Kerckhoffsovo načelo kasnije je preformulirano u Claude Shannonovo „*neprijatelj poznaje sustav*“ koje govori da kriptosustav treba razvijati s pretpostavkom kako će neprijatelj (napadač) u vrlo kratkom roku saznati sve o njemu. Kriptografi prihvaćaju ovaj koncept u odnosu na „*sigurnost kroz tajenje*“ čija sigurnost ovisi isključivo o tajnosti funkcija i algoritama šifriranja (dešifriranja) te ukoliko oni postanu komprimitirani sustav postaje nesiguran. Također, pouzdani i vjerodostojni sustavi su oni koji su javno objavljeni i analizirani.

Mogući napadi na kriptosustav općenito se dijele na dvije osnovne skupine, *pasivne* i *aktivne* :

Pasivni napadi: napadač pokušava prikupiti informacije o napadnutom sustavu i/ili pristupiti informacijama iz napadnutog sustava bez utjecaja na njegove resurse. Napadač može samo presretati poruke u komunikacijskom kanalu te na osnovu pribavljenih informacija pokušava odrediti otvoreni tekst ili ključ. Ovakve napade vrlo je teško otkriti budući da ne mijenjaju podatke niti na bilo koji drugi način utječu na komunikaciju koju prate. Pasivne napade nadalje razlikujemo u ovisnosti o količini poznatih informacija:

> **Poznat šifrat:** napadač ima pristup određenom broju šifriranih poruka te na temelju njih pokušava odrediti otvoreni tekst ili ključ. Sustav koji je podložan ovakvom napadu vrlo je nesiguran.
>
> **Poznat otvoreni tekst:** napadač ima pristup šifratu i odgovarajućem otvorenom tekstu te na temelju njih pokušava odrediti tajni ključ ili razviti algoritam koji će mu omogućiti dešifriranje narednih poruka.
>
> **Izabrani otvoreni tekst:** napadač ima pristup funkciji šifriranja te svojevoljno odabire neki otvoreni tekst koji zatim šifrira i dobiva odgovarajući šifrat. Analizom odgovarajućeg para otvorenog teksta i šifrata pokušava otkriti tajni ključ ili razviti algoritam pomoću kojeg bi mogao dešifrirati naredne poruke. Kriptosustavi koji jednake elemente otvorenog teksta pretvaraju u jednake elemente šifrata iznimno su slabi na napad „izabrani otvoreni tekst“.
>
> **Izabrani šifrat:** napadač ima pristup funkciji dešifriranja i može dešifrirati bilo koju šifriranu poruku. Analizom odgovarajućeg para otvorenog teksta i šifrata pokušava otkriti tajni ključ ili razviti algoritam pomoću kojeg bi mogao dešifrirati naredne poruke

Aktivni napadi: napadač pokušava modificirati sustavne resurse ili njihov rad ubacivanjem, brisanjem ili mijenjanjem podataka. Također može replicirati slanje podataka kako bi simulirao nekog legitimnog korisnika.

3. DES KRIPTOSUSTAV

DES ili Data Encryption standard je simetričan blokovni kriptosustav koji je prije bio najrašireniji i najpouzdaniji sustav za šifriranje podataka.

3.1 Povijest DES-a

Početkom 70-ih godina prošlog stoljeća, razvojem računala i povećanjem broja i važnosti financijskih transakcija u svijetu počela se stvarati sve veća potreba za skrivanjem podataka u realnom sektoru. Do tada je najveća primjena kriptografije bila u vojnom i državnom sektoru. Američki NBS (National Bureau of Standards) raspisao je 1972. godine natječaj za kriptosustav koji će biti dovoljno dobar i siguran za korištenje u javnosti i koji će se uvesti kao standard u kriptografiji. Postavili su uvjete koje taj kriptosustav mora zadovoljavati:

1. Visok stupanj sigurnosti.
2. U potpunosti specificiran i lako razumljiv.
3. Sigurnost sustava leži u ključu, a ne tajnosti algoritma (Kerckhoffsov princip).
4. Dostupan svim korisnicima.
5. Prilagodljiv raznim vrstama primjene.
6. Jednostavna implementacija na elektroničke uređaje.
7. Mora biti učinovit.
8. Moguća je provjera tog kriptosustava.
9. Moguć izvoz.

Nakon prvog kruga odabira 1973. NBS u dogovoru s NSA odlučuje da nijedan od ponuđenih kriptosustava ne zadovoljava njihove stroge zahtjeve. Na ponovljenom natječaju 1974. godine tvrtka IBM izlaže svoj kriptosustav razvijen u periodu od 1973-1974 godine temeljen na Feistelovom „Lucifer kriptosustavu“. IBM-ov DES objavljen je u Federal registru 1975. Tražilo se mišljenje javnosti te su tijekom sljedeće dvije godine otvorene radionice gdje se raspravljalo o njemu. Kriptografski stručnjaci Martin Hellman i Whitfield Diffie kritizirali su sustav zbog skraćene veličine

ključa i misterioznih „S-kutija“ smatrajući kako te činjenice dokazuju nepriličan utjecaj NSA u razvoju algoritma. Sumnjalo se kako je NSA oslabila algoritam kako bi imala pristup šifriranim porukama. Na kraju se ispostavilo kako je NSA uvjerila IBM da je smanjena veličina ključa dovoljna za sigurnost algoritma te nije imala nikakvu ulogu u razvoju algoritma. IBM je osmislio i razvio algoritam te donosio sve bitne odluke tokom razvoja. Unatoč kritikama, DES je prihvaćen kao federalni standard 1976. godine i objavljen 1977. godine u FIPS PUB 46.

3.2 Feistelova šifra

Feistelova šifra dobiva ima po svom tvorcu, Horst-u Feistel-u, Njemačko-Američkom kriptografu koji se bavio razvojem šifri i kriptosustava u IBM-u. Algoritam DES-a temelji se na Feistelovoj šifri stoga ćemo ju obraditi prije opisivanja samog DES-a. Feistelova šifra je blokovna šifra koja ima specifičnu strukturu u kojoj su operacije šifriranja i dešifriranja vrlo slične, ponekad identične, tzv. Feistelovu mrežu. Šifra se temelji na naizmjeničnoj uporabi supstitucija i transpozicija kroz više iteracija (rundi).

Osnovni elementi otvorenog teksta Feistelove šifre su $\mathcal{P} = \{0,1\}$, a ulaz u algoritam je blok duljine *2n* elemenata koji se sastoji od elemenata otvorenog teksta. Za ključ *k* generira se *r* novih ključeva k_1, . . . , k_n koji se koriste u odgovarajućoj rundi i koje nazivamo *ključevima runde*. U svakoj rundi se uz ključ i dio otvorenog teksta računa funkcije runde F.

Šifriranje počinje podjelom ulaznog bloka na dva jednaka dijela (L_o, R_o). Na desnu polovicu se primjenjuja funkcija runde uz odgovarajući ključ runde, zatim se vrši logička operacija isključivo ili između rezultata funkcije runde i lijeve polovice ulaznog bloka. Na posljetku se vrši zamjena lijeve i desne polovice ulaznog bloka te se proces ponavlja onoliko puta koliko ima rundi. Parove šifrata (L_{i+1}, R_{i+1}) kroz runde i = 0,1,. . . ., n dobivamo formulom:

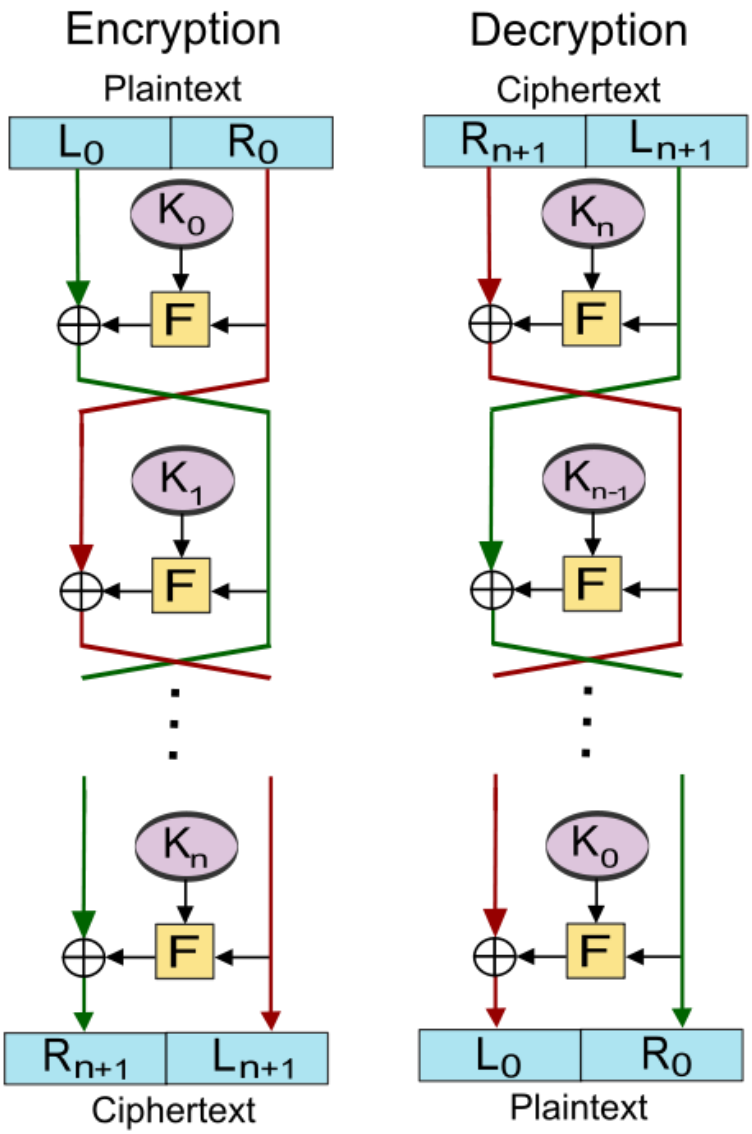
$$(L_i, R_i) = (R_i \oplus \mathrm{F}(R_i, K_i),\ L_i) \tag{3-1}$$

Gdje je $\oplus$ isključivo ili logička operacija, a F funkcija runde.

Dešifriranje para šifrata (L_{i+1}, R_{i+1}) vrši se računanjem uz i= i, i-1, ... , 0:

$$(L_i, R_i) = (R_{i+1} \oplus \mathrm{F}(L_{i+1}, K_i),\ L_{i+1})$$

Tokom dešifriranja ključevi runde koriste se u obrnutom redoslijedu. Funkcije šifriranja i dešifriranje su iste. Također, ovo svojstvo ne ovisi o izboru funkcije F; runda će uvijek biti invertibilna.



Slika 3.1 *Šifriranje i dešifriranje Feistelovom šifrom*

Za realizaciju Feistelove šifre potrebno je definirati sljedeće parametre:

1. **Veličina bloka** – s povećanjem veličine bloka povećava se razina sigurnosti, ali se ujedno smanjuje brzina šifriranja/dešifriranja za neki zadani algoritam (najčešće se koriste blokovi od 64 ili 128 bita)
2. **Duljina ključa** – povećanjem duljine ključa raste razina sigurnosti, ali se smanjuje brzina šifriranja/dešifriranja
3. **Broj iteracija (rundi)** – veći broj iteracija povećava razinu sigurnosti
4. **Algoritam kreiranja potključeva (ključeva rundi)** – veća kompleksnost ovog algoritma povećava otpornost šifre na kriptoanalizu
5. **Funkcija runde F** – složenija funkcija F povećava otpornost i sigurnost šifre.

3.3 DES funkcija šifriranja

Funkcija šifriranja DES-a je Feistelova šifra kojemu je duljina ulaznog bloka otvorenog teksta 64 bita, odnosno n = 32 i broj rundi r = 16. Prostor svih mogućih blokova otvorenog teksta i šifrata jednak je svim mogućim permutacijama 0 i 1 na 64 mjesta. Ključ K koji se koristi kod DES algoritma sastoji se od 64 bita od kojih 56 predstavljaju ključ, dok su preostalih 8 paritetni bitovi koji se nalaze na kraju svakog bajta ključa, a definirani su tako da svaki bajt sadrži neparan broj jedinica. Paritetni bitovi ignoriraju se prilikom izračunavanja tablice ključeva. Koraci fukcije šifriranja su sljedeći:

1. **Inicijalna permutacija**

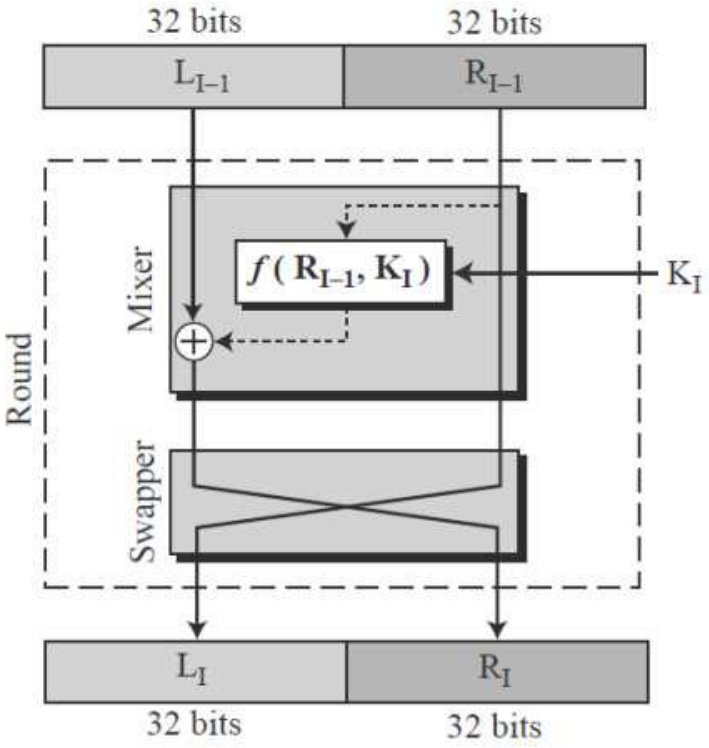
Na otvoreni tekst primjenjuje se inicijalna permutacija, tj. elementi bloka otvorenog teksta p1, p2, ... , p64 permutiraju u p58, p50, ..., p7 (tablica 3.1).

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tablica 3.1 Inicijalna permutacija IP

2. **Feistelove runde**

Prolazimo kroz 16 rundi iste funkcije F. Svaka runda zapravo predstavlja Feistelovu šifru.



Slika 3.2 *Feistelova runda*

3. **Inverzna permutacija**

Konačno se na izlaznom bloku šifrata vrši inverzna permutacija, element p1 koji je u inicijalnoj permutaciji postao p58 sada prema tablici inverzne permutacije opet postaje p1 itd. (tablica 3.2).

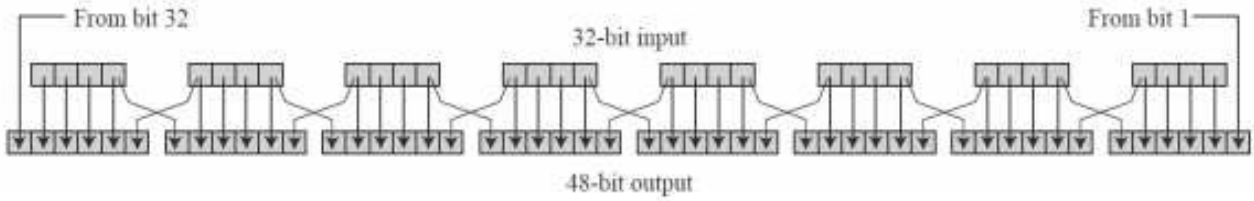
40	08	48	16	56	24	64	32
39	07	47	15	55	23	63	32
38	06	46	14	54	22	62	30
37	05	45	13	53	21	61	29
36	04	44	12	52	20	60	28
35	03	43	11	51	19	59	27
34	02	42	10	50	18	58	26
33	01	41	09	49	17	57	25

Tablica 3.2 Inverzna permutacija IP^{-1}

3.3.1 Funkcija runde

Funkcija runde sastoji se od 4 operacije:

Ekspanzija: desna polovica ulaznog bloka od 32 bita proširuje se na 48 bitova koristeći proširujuću permutaciju (tablica 3.3) tako da se pola ulaznih bitova kopira. Izlazni blok od 48 bitova sastoji se od osam 6-bitnih dijelova od kojih svaki sadrži kopiju 4 odgovarajuća ulazna bita i kopiju susjednog bita najbližeg lijevog i desnog ulaznog dijela. Ova operacija odabrana je kako bi jedan bit s ulaza utjecao na dvije zamjene u izlazu putem S-kutija čime se povećava difuzija sustava, tj. mala promjena u otvorenom tekstu uzrokovat će jako veliku promjenu u šifratu (efekt lavine).



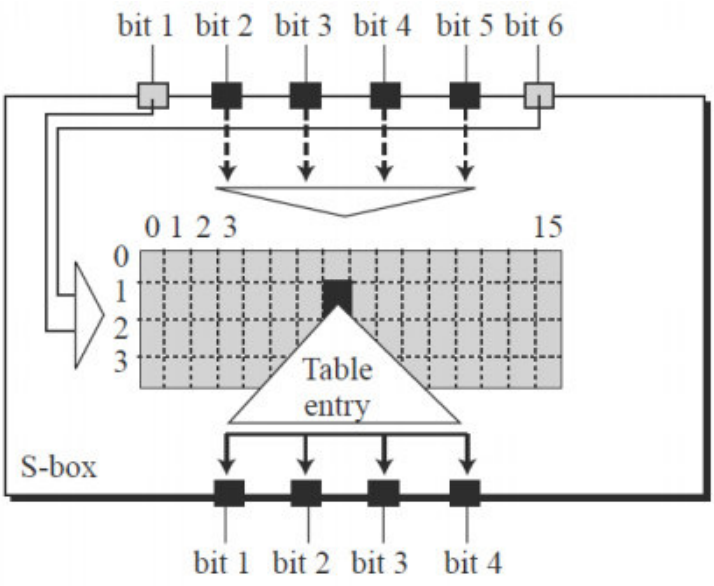
Slika 3.3 *Shema proračuna izlaznog bloka*

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	01

Tablica 3.3 Ekspanzijska D-box tablica

Dodavanje ključa runde: nakon proširenja desnog podniza u prethodnom koraku, dobiveni niz od 48 bitova zbraja se pomoću XOR logičke operacije sa ključem runde. Ovo je jedini korak u kojem se koristi ključ.

Supstitucija: nakon dodavanja ključa runde 48-bitni blok se dijeli na skupine od 6 bitova te se nad svakom skupinom vrši supstitucija pomoću osam tzv. S-kutija (tablica 3.4). Svaka S-kutija je fiksna matrica dimenzija 4x16 čiji su elementi cijeli brojevi između 0 i 15 tj. 4 bita. Ulazni skup od 6 bitova indeksira određeno polje u matrici tako da 2 bita dodana prilikom ekspanzije (prvi i šesti bit) indeksiraju redak, a preostala 4 bita stupac matrice. Indeksirani element predstavlja izlaz iz S-kutije. Spajanjem po 4 bita iz 8 S-kutija na izlazu dobivamo 32 bita. Bitno je napomenuti kako je ovaj dio funkcije šifriranja nelinearan i značajno pridonosi sigurnosti sustava, bez njega šifra bi bila linearna a samim time i vrlo nesigurna.



Slika 3.4 *Shema indeksiranja S-kutije*

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S1	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1	0	15	7	4	14	12	13	1	10	6	12	11	9	5	3	8
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S2	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S3	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S4	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S5	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S6	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	2	9	14	5	5	2	8	12	3	7	0	4	10	1	13	11	6
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S7	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S8	0	13	2	8	4	6	15	11	1	10	9	3	14	15	0	12	7
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Tablica 3.4 S-kutije

3.3.2 Generator međuključeva

Generator međuključeva od osnovnog ključa k generira 16 ključeva (po jedan za svaku iteraciju) duljine 48 bita. Najprije se odbacuju paritetni bitovi prema tablici 3.5

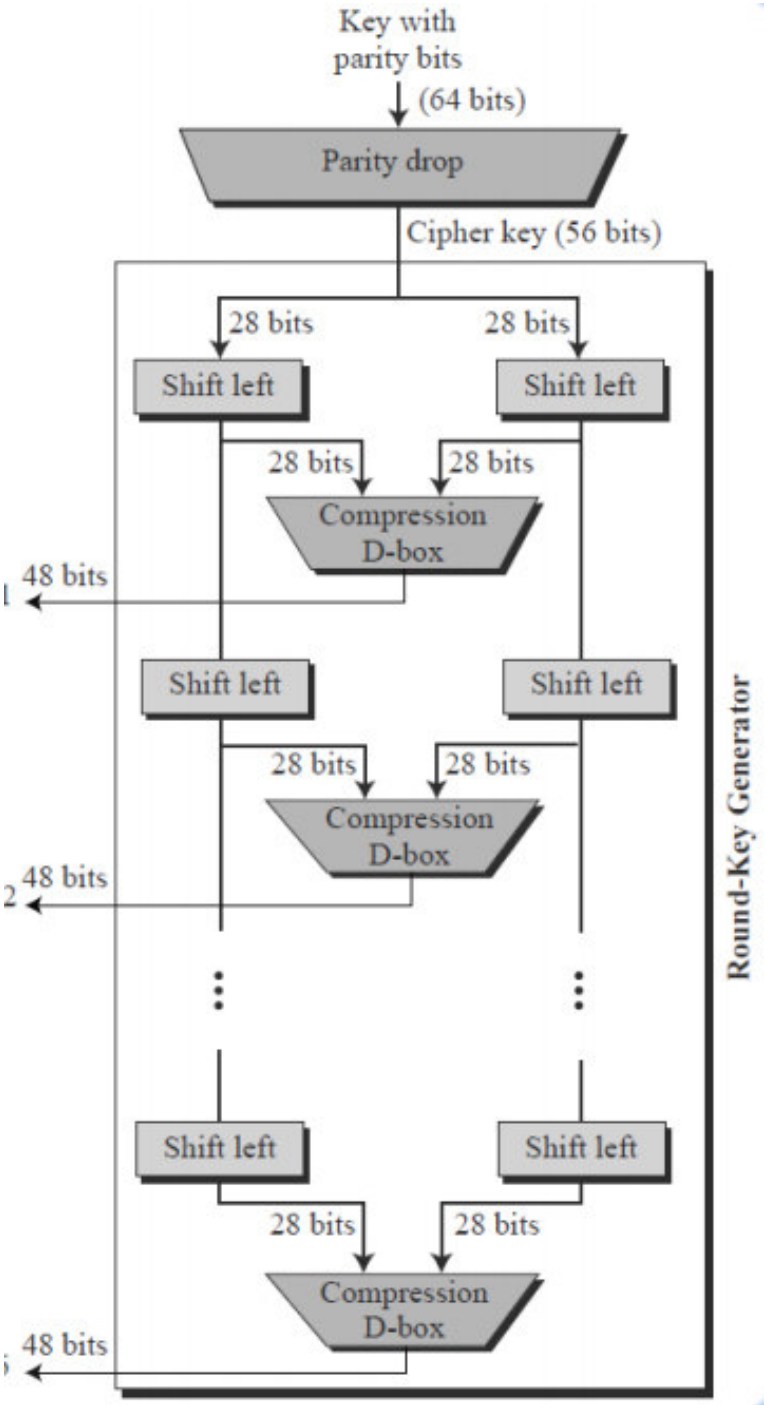
57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Tablica 3.5 Odbacivanje paritetnih bitova

Ključ se zatim podijeli na dva dijela od 28 bitova, te se za runde 1, 2, 9 i 16 vrši posmak od jednog bita, a za ostale runde posmak od 2 bita na svakoj polovici zasebno. Rezultati posmaka ulaze u kompresijsku D-kutiju (tablica 3.6) koja sažima 56 bitova na 48 bitova i čiji izlaz predstavlja ključ runde. Ovaj postupak ponavlja se 16 puta kako bi se generirao ključ za svaku rundu.

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Tablica 3.6 Kompresijska D-kutija

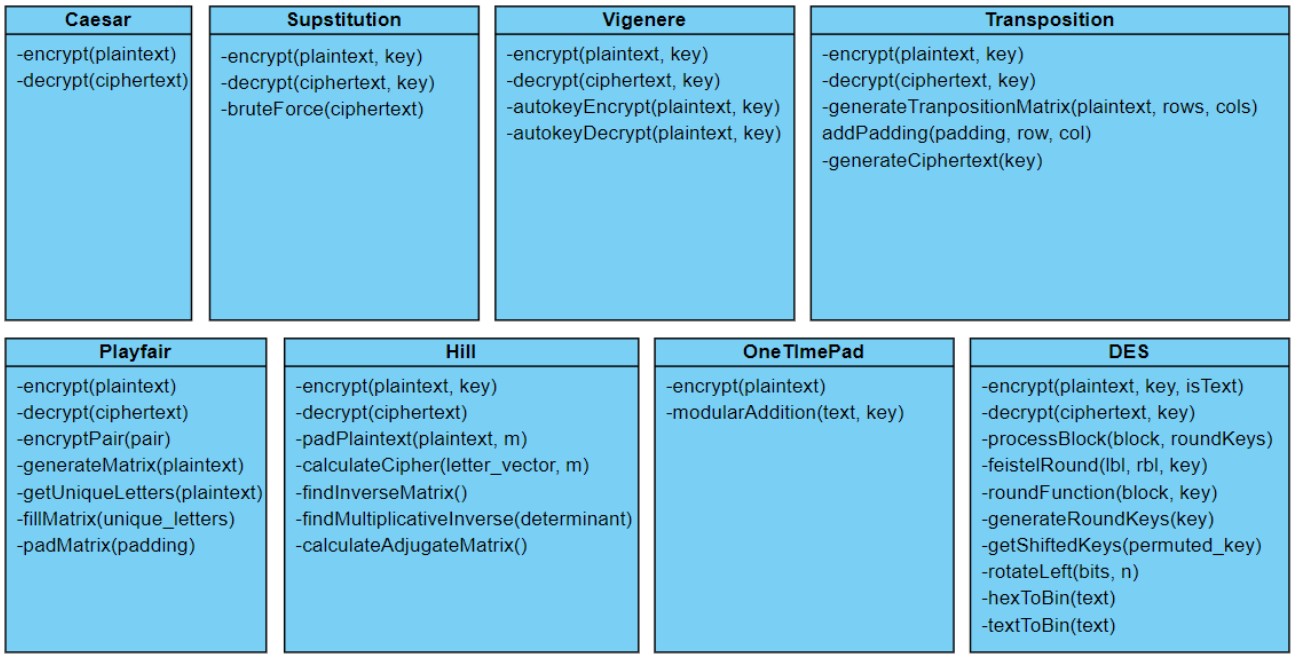


Slika 3.5 *Shema generiranja međuključeva*

4. RAZVOJ APLIKACIJE „CRYPTOSYS“

Praktični dio završnog rada izrađen je pomoću Pythona i PyQt5 modula. Korištena je Python 3.8.2 verzija pomoću koje su programski implementirani odabrani kriptosustavi i šifre. PyQt5 je aplikacijsko sučelje za Qt skup alata koji pruža pristup raznim svojstvima modernih desktop sustava te omogućava razvoj korisničkog sučelja u pythonu. Korištenjem PyQt5 aplikacijskog sučelja izvedeno je korisničko sučelje i logika aplikacije.

Na slici 4.1 prikazane su klase šifri i njihove metode, te argumenti pojedinih metoda.



Slika 4.1 *Klase i metode implementiranih šifri*

Naredna podpoglavlja ukratko objašnjavaju strukturu i funkcionalnosti pojedine klase. Programski kod implementiranih klasa nalazi se u prilogu.

4.1 Python

Python je interpretirani, opće namjenski programski jezik visoke razine. Osmislio ga je Guido van Rossum kasnih 1980-ih godina u „Centrum Wiskunde & Informatica“ u Norveškoj. Njegova implementacija započela je 1989. godine, a prvi put je objavljen 1991. godine. Python 2.0 objavljen je 2000. godine te donosi nova svojstva poput obuhvaćenih nizova i sustava za automatsko upravljanje memorijom. Python 3.0 objavljen je 2008. godine i predstavlja značajnu promjenu programskog jezika koja nije u potpunosti kompatibilna unatrag te se većina koda napisanog u Pythonu 2 ne može izvršiti u Pythonu 3 bez dodatnih modifikacija.

Python naglašava čitljivost koda korištenjem značajnog uvlačenja kako bi ograničio blokove koda, što je ujedno i jedna od njegovih glavnih značajki. Dijelovi koda na istoj razini uvučenosti smatraju se logičkom cijelinom, tako naprimjer tijelo funkcije ili petlje nije potrebno obuhvatiti vitičastim zagradama nego dio koda koji pripada funkciji/petlji uvući na istu razinu (unutar funkcije/petlje). Osim značajnog uvlačenja, python je dinamičan i implementira automatsko upravljanje memorijom. Memorijski prostor koji program više ne koristi automatski će se osloboditi.

Python podržava nekoliko programskih paradigmi, uključujući proceduralno, objektno orijentirano i funkcionalno programiranje.

Kao sustav tipova python koristi „duck typing“ odnosno postoje tipovi objekata ali ne postoje tipovi imena varijabli. Varijablu nije potrebno eksplicitno odrediti već će ona poprimiti tip podatka koji je u nju spremljen. Ograničenja koja postavljaju tipovi ne provjeravaju se tokom kompajliranja već tokom izvođenja, točnije operacije nad objektom mogu biti neuspješne što označava da objekt nije prikladnog tipa te će doći do pogreške i prekida izvršavanja programa.

Za razvijanje aplikacije koja je tema ovog završnog rada koristi se Python 3.8.2 .

4.2 Caesar.py

Klasa Caesar implementira cezarovu šifru i sastoji se od dvije metode. Metode za šifriranje i metode za dešifriranje. Šifrat odnosno otvoreni tekst dobiva se modularnim zbrajanjem odnosno oduzimanjem.

4.3 Supstitution.py

Klasa Supstitution implementira supstitucijsku šifru. Sastoji se od metode za šifriranje, dešifriranje i dešifriranje grubom silom. Metode šifriranja i dešifriranja kao ulazne argumente primaju tekst i ključ šifriranja pomoću kojih se vrši modularno zbrajanje ukoliko se tekst šifrira, odnosno oduzimanje ukoliko se tekst dešifrira. Metoda dešifriranja grubom silom kao ulaz prima tekst koji se zatim dešifrira koristeći sve moguće ključeve.

4.4 Vigenere.py

Klasa Vigenere implementira Vigenerovu šifru. Sastoji se od metoda za šifriranje, dešifriranje, te šifriranje i dešifriranje korištenjem autoključa. Sve metode kao argumente primaju tekst i ključ šifriranja. Šifriranje i dešifriranje vrši se korištenjem Vigenereovog kvadrata. Metode šifriranja i dešifriranja obrađuju tekst ponavljanjem ključa, dok metode šifriranja i dešifriranja uz autoključ nakon predanog ključa koriste znakove ulaznog teksta.

4.5 Playfair.py

Klasa Playfair implementira playfairovu šifru. Sastoji se od metoda za šifriranje, dešifriranje, šifriranje bigrama teksta, generiranje matrice, popunjavanje matrice, nadopunjavanje matrice i pronalaženje jedinstvenih slova ulaznog teksta. Metode šifriranja i dešifriranja najprije generiraju matricu pozivanjem metode *generateMatrix()*. Matrica se prvo popunjava jedinstvenim slovima ulaznog teksta pozivanjem metoda *getUniqueLetters()* i *fillMatrix()*, a zatim se poziva metoda *padMatrix()* kako bi se matrica nadopunila preostalim slovima abecede. Nakon stvaranja matrice ulazni tekst dijeli se u bigrame te se poziva metoda *encryptPair()* koja šifrira predani bigram. Nakon šifriranja/dešifriranja svih bigrama dobiva se šifrat/otvoreni tekst.

4.6 Hill.py

Klasa Hill implementira hillovu šifru. Sastoji se od metoda za šifriranje, dešifriranje, izračun dijela šifrata, nadopune predanog teksta (ukoliko je potrebno) te pronalazak inverzne matrice koja se koristi za dešifriranje. Metoda šifriranja kao ulazne argumente prima tekst i ključ šifriranja (3x3 matrica). Ulazni tekst dijeli se na trigrame te se sa svakim dobivenim trigramom poziva metoda *calculateCipher()* koja množi predani trigram s matricom ključa. Nakon šifriranja dobiveni šifrat moguće je dešifrirati množenjem trigrama šifrata s inverznom matricom ključa. Metode *findInverseMatrix()*, *findMultiplicativeInverse()* i *calculateAdjugateMatrix()* odgovorne su za proračun inverzne matrice.

4.7 Transposition.py

Klasa Transposition implementira transpozicijsku šifru. Sastoji se od metoda za šifriranje, dešifriranje, generiranje matrice te nadopune matrice. Metode šifriranje i dešifriranja kao ulazne argumente primaju tekst i ključ šifriranja. Najprije se poziva metoda *generateTranspositionMatrix()* koja stvara matricu šifriranja odgovarajućih dimenzija te ju popunjava znakovima teksta i po potrebi nadopunjava znakom *'X'*. Šifrat se dobiva isčitavanjem stupaca dobivene matrice redoslijedom određenim ključem šifriranja. Metoda dešifriranja generira matricu s predanim šifratom te isčitava stupce prema ključu šifriranja, zatim stvara matricu s pravilnim redoslijedom stupaca te isčitavanjem njenih redaka dobiva otvoreni tekst.

4.8 OneTimePad.py

Klasa OneTimePad implementira jednokratnu bilježnicu. Sastoji se od metoda za šifriranje i modularno zbrajanje. Metoda šifriranja kao ulazni argument prima tekst. Metoda šifriranja najprije nasumično generira ključ šifriranja jednake duljine kao ulazni tekst, a zatim poziva metodu *modularAddition()*. Metoda *modularAddition()* modularno zbraja odgovarajuće elemente otvorenog teksta i nasumično generiranog ključa. Rezultat modularnog zbrajanja predstavlja šifrat.

4.9 DES.py

Klasa DES implementira DES kriptosustav. Sastoji se od metoda za šifriranje, dešifriranje, procesiranje bloka bitova, feistelove runde, funkcije runde, generiranje ključeva runde, posmak ključa, rotacije bitova ključa ulijevo te pretvorbe heksadecimalnog ili tekstualnog zapisa u binarni. Metode šifriranja i dešifriranja kao ulazne argumente primaju otvoreni tekst i ključ šifriranja. Metoda šifriranja prima dodatni argument isText tipa boolean koji omogućuje unos otvorenog teksta u obliku znakova ili kao heksadecimalni zapis. Prilikom šifriranja najprije se poziva metoda *generateRoundKeys()* kojoj se predaje ulazni ključ šifriranja.

Metoda *generateRoundKeys()* permutira ključ te ga dijeli na dvije jednake polovice, a zatim pozivanjem metoda *getShiftedKeys()* i *rotateLeft()* generira 16 ključeva koji se koriste u odgovarajućim rundama tokom šifriranja. Metoda šifriranja zatim dijeli ulazni tekst na 64-bitne blokove koji se predaju metodi *processBlock()*.

Metoda *processBlock()* dijeli 64-bitni ulazni blok na dvije jednake polovice te odgovarajući broj puta poziva metodu *feistelRound()* s predanim polovicama bloka i odgovarajućim ključem runde. Metoda *feistelRound()* vrši operaciju isključivo ili između dviju polovica bloka i zamjenjuje ih. Također poziva metodu *roundFunction()* s desnom polovicom ulaznog bloka.

Metoda *roundFunction()* permutira ulazni blok, vrši operaciju isključivo ILI između bloka i odgovarajućeg ključa runde te supstituira elemente bloka koristeći S-kutije.

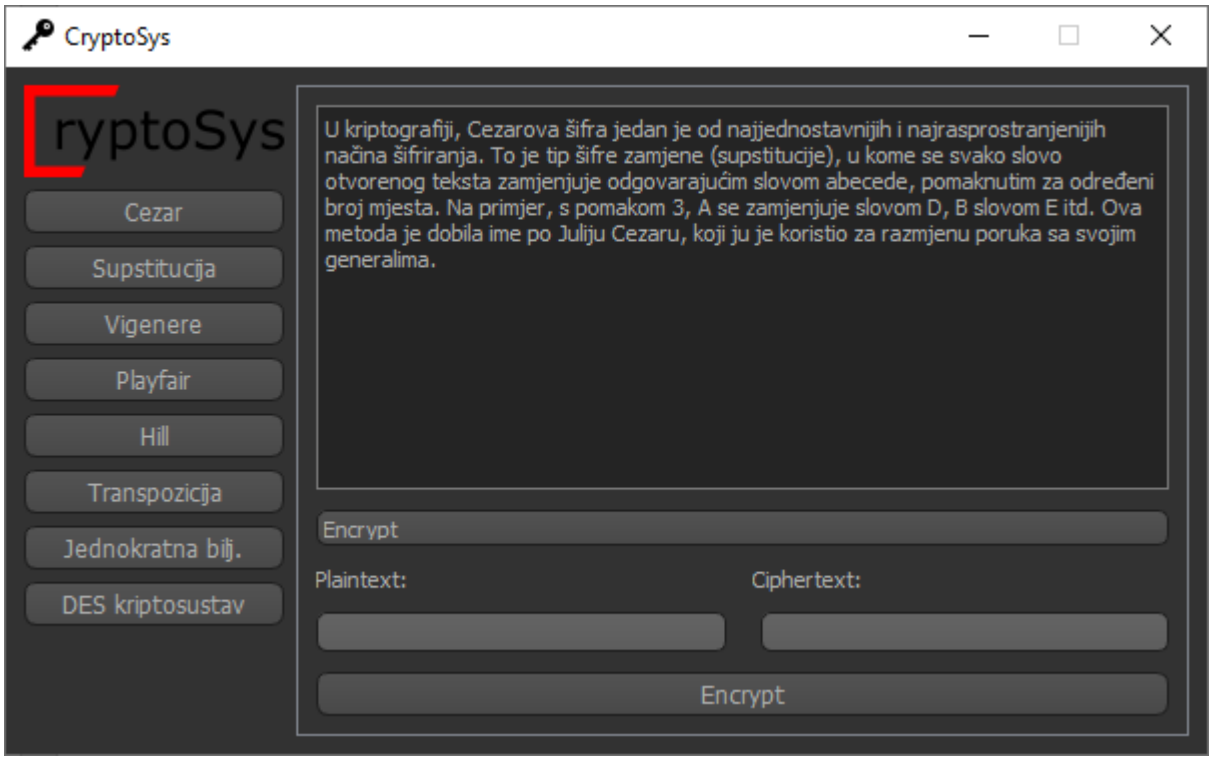
Nakon izvršavanja svih rundi metoda *processBlock()* prije vraćanja šifriranog/dešifriranog bloka vrši krajnju permutaciju elemenata bloka. Nakon obrade svih blokova dobiva se šifrat/otvoreni tekst.

5. APLIKACIJA CRYPTOSYS

Aplikacija *„CryptoSys“* implementira osam različitih kriptosustava kojima je moguće šifrirati odnosno dešifrirati poruke. Također objašnjava funkcioniranje pojedinog kriptosustava.

Na slici 6.1 prikazano je korisničko sučelje aplikacije. Izbornik se nalazi uz lijevi rub glavnog prozora i sastoji se od osam gumbova. Ostatak glavnog prozora prekriva okvir koji prikazuje odgovarajuće sučelje za šifriranje odnosno dešifriranje poruka. Sadržaj okvira mijenja se prema trenutno odabranoj šifri.

Općenito, svaki okvir se sastoji od tekstualne kutije koja sadrži opis šifre, izbornika, polja za unos otvorenog teksta (šifrata), polja za prikaz rezultata i gumba za šifriranje (dešifriranje). Pojedini okviri zahtjevaju dodatne unose korisnika (najčešće je to ključ šifriranja) te pružaju dodatne mogućnosti šifriranja (dešifriranja).

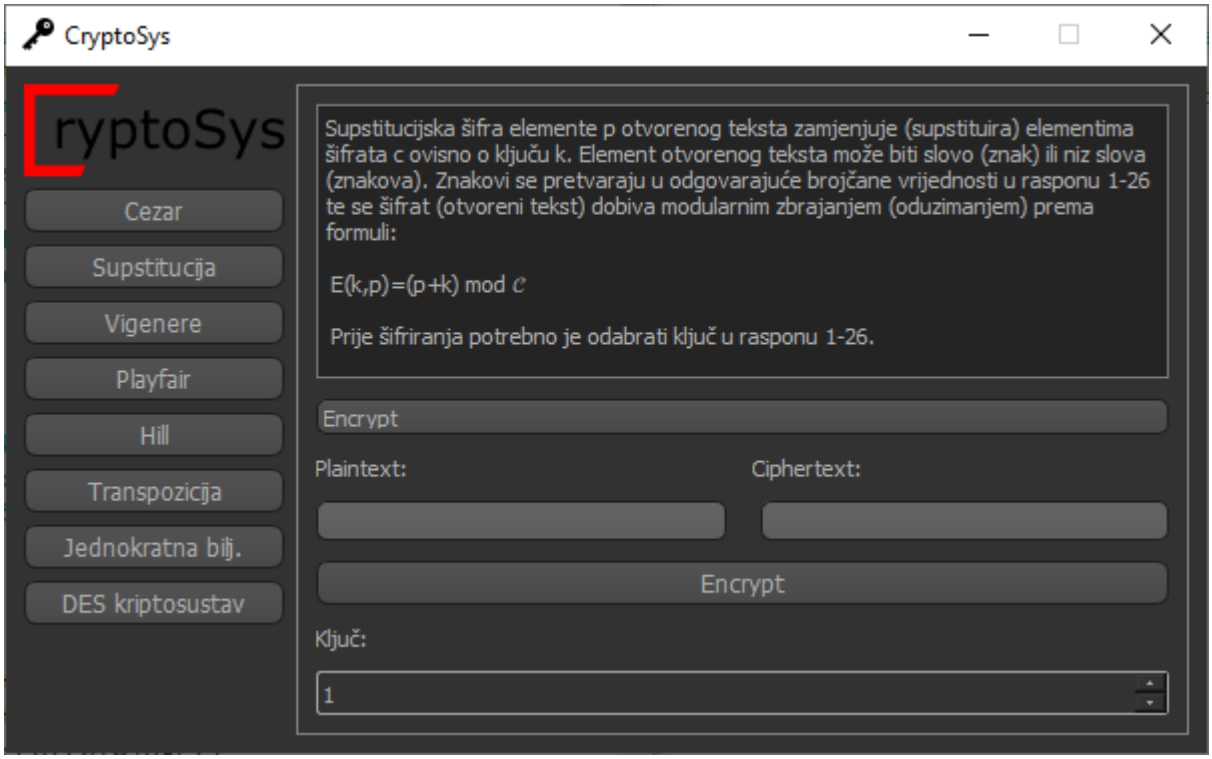


Slika 5.1 *Korisničko sučelje aplikacije*

Zadani okvir prilikom pokretanja aplikacije je okvir cezarove šifre (slika 6.1). U narednim podpoglavljima razraditi ćemo ostale okvire.

5.1 Supstitucija

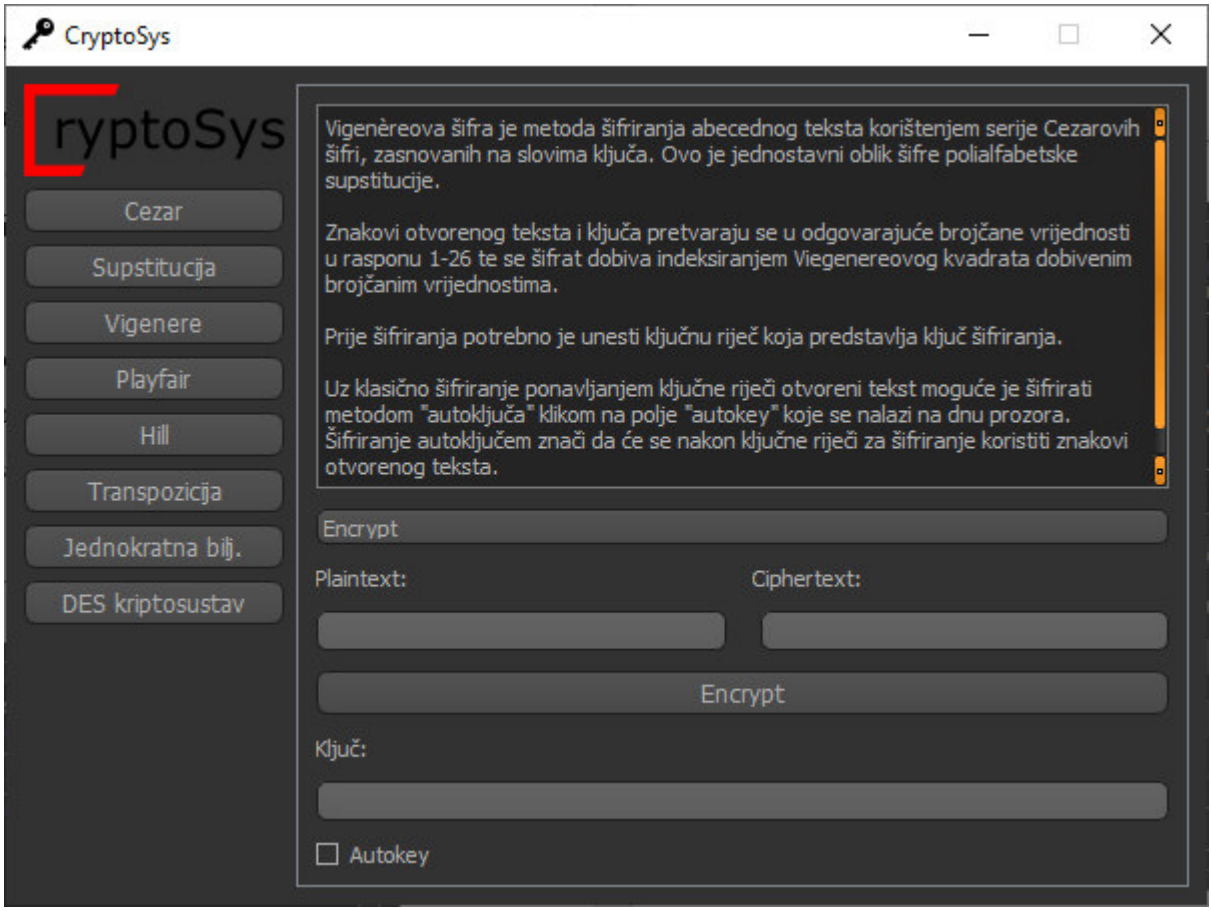
Okvir Supstitucijske šifre (Slika 6.2) osim klasičnih elemenata sadrži brojčani izbornik na dnu okvira kojim se odabire ključ šifriranja u rasponu od 1 do 26. Moguće je šifriranje, dešifriranje, te dešifriranje grubom silom.



Slika 5.2 *Okvir supstitucijske šifre*

5.2 Vigenere

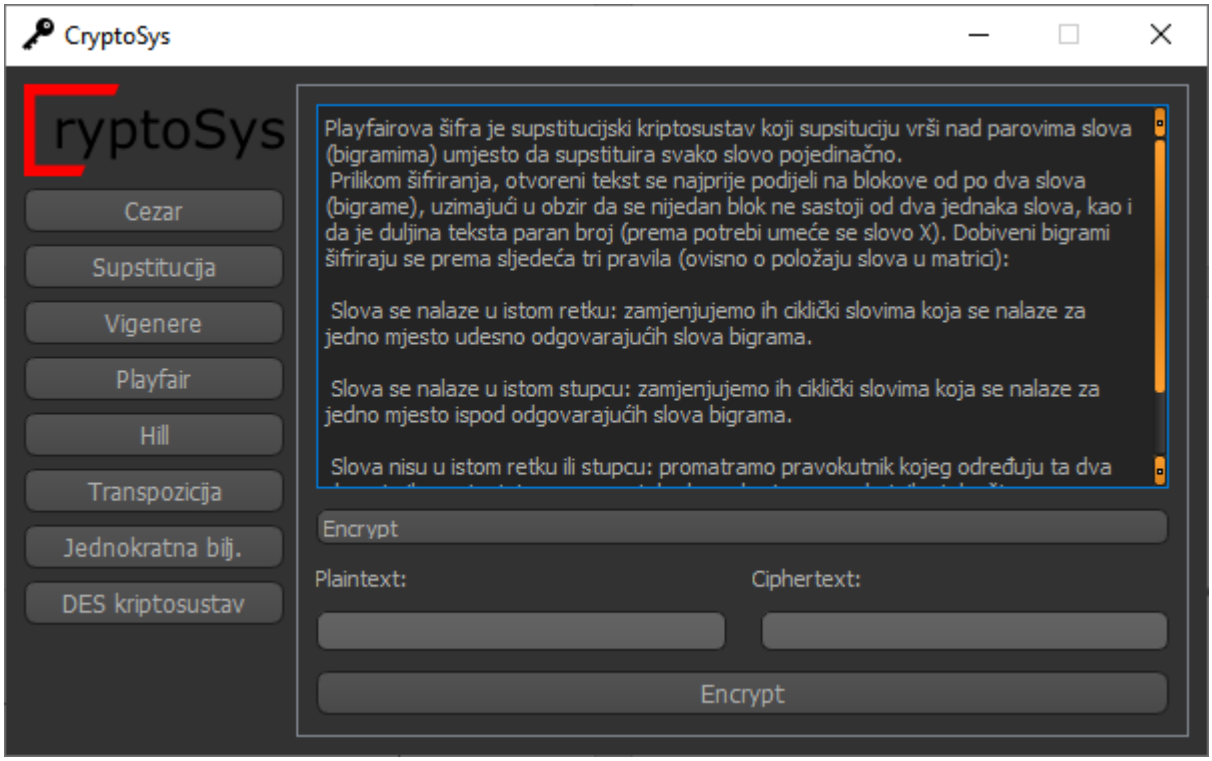
Okvir Vigenereove šifre (Slika 6.3) kao dodatna polja sadrži polje za unos ključa u tekstualnom obliku te odabirni okvir za šifriranje (dešifriranje) autoključem. Moguće su operacije šifriranja i dešifriranja uz ili bez primjene autoključa.



Slika 5.3 *Okvir vigenereove šifre*

5.3 Playfair

Okvir Playfairove šifre (Slika 6.4) sastoji se isključivo od klasičnih elemenata sučelja. Moguće su operacije šifriranja i dešifriranja.



Slika 5.4 *Okvir playfairove šifre*

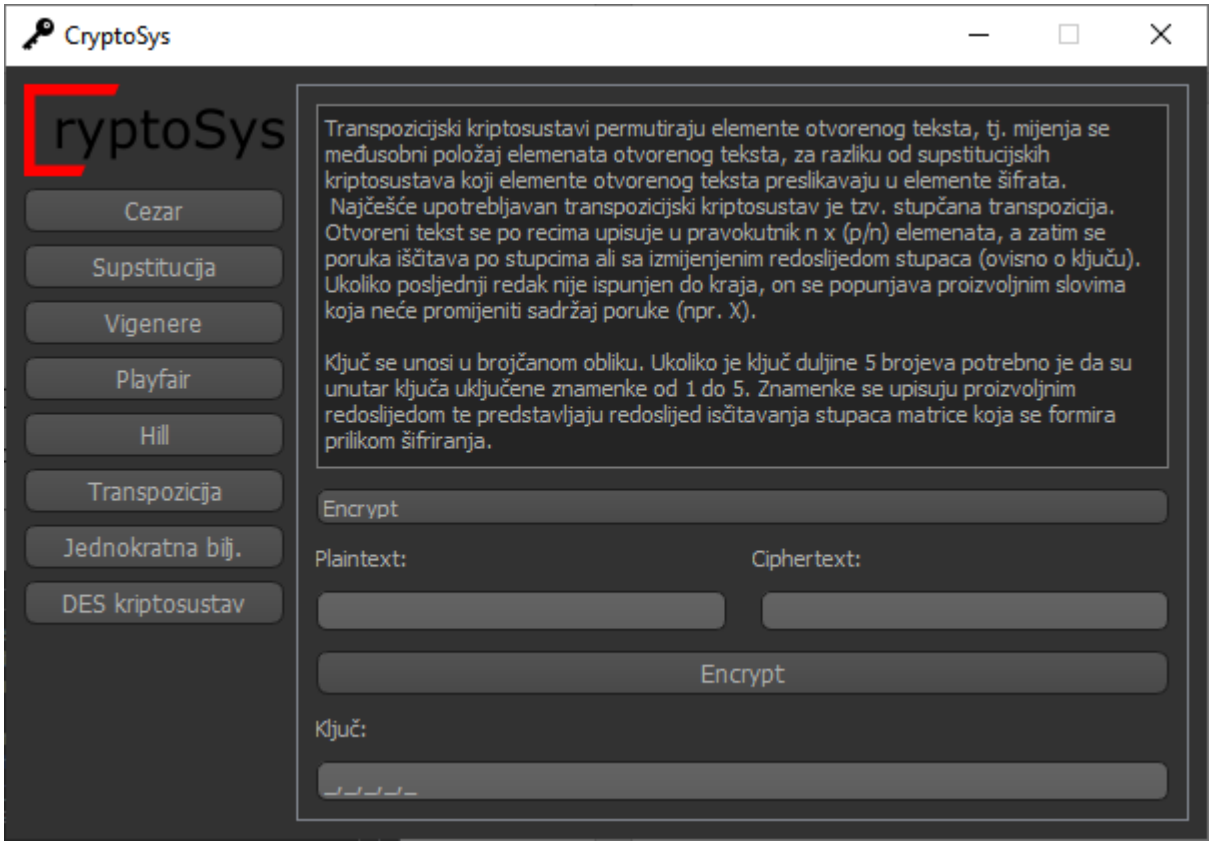
5.4 Hill

Okvir Hillove šifre (Slika 6.5) osim klasičnih elemenata sučelja zahtjeva unos ključa u obliku 3x3 matrice. Predana matrica mora biti invertibilna kako bi se šifrat mogao dešifrirati. Moguće su operacije šifriranja i dešifriranja.

Slika 5.5 *Okvir hillove šifre*

5.5 Transpozicija

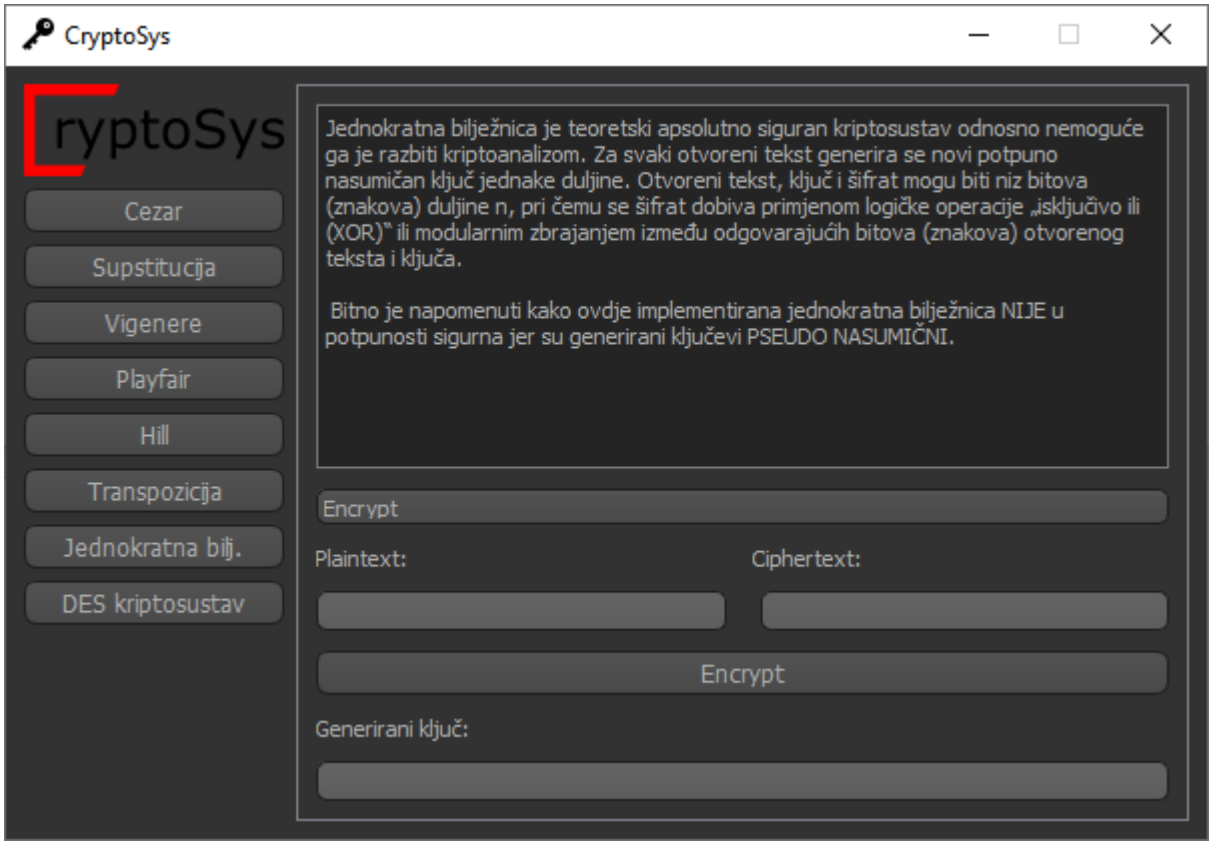
Okvir Transpozicijske šifre (Slika 6.6) osim klasičnih elemenata sadrži polje za unos ključa. Ključ se unosi u brojčanom obliku. Moguće su operacije šifriranja i dešifriranja.



Slika 5.6 *Okvir transpozicijske šifre*

5.6 Jednokratna bilježnica

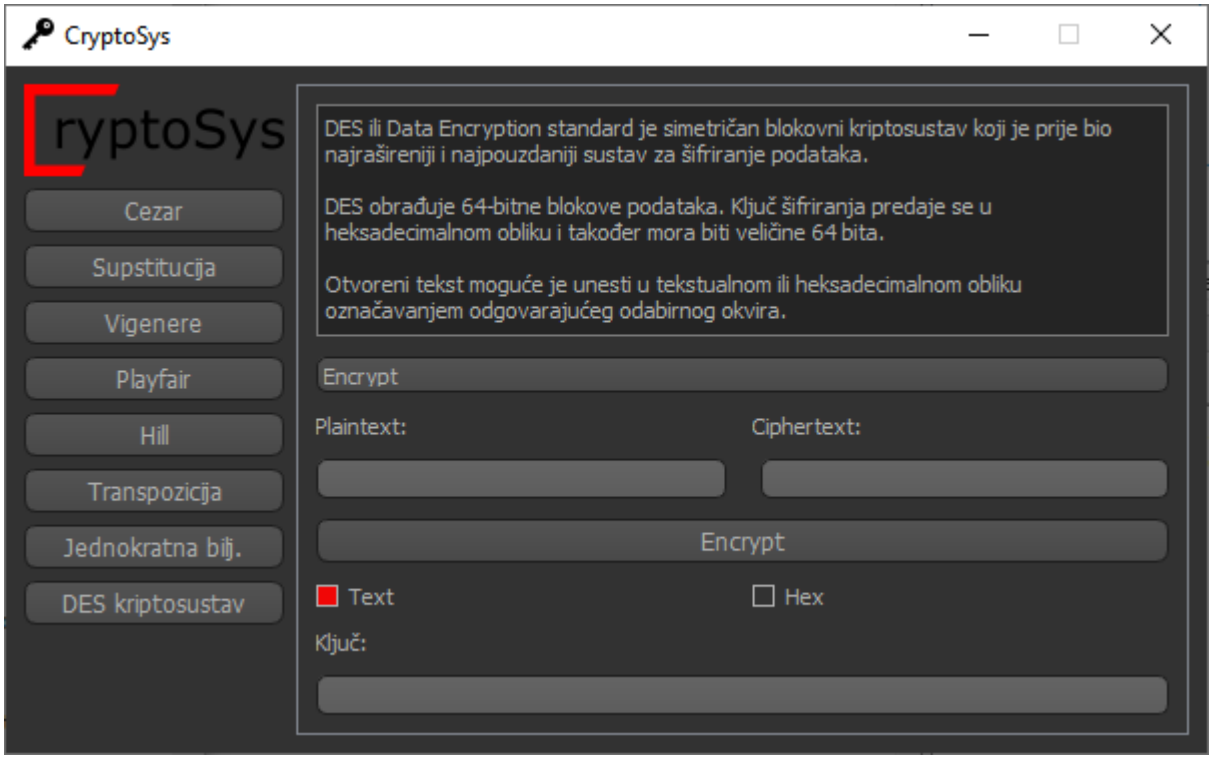
Okvir jednokratne bilježnice (Slika 6.7) osim klasičnih elemenata sučelja sadrži polje koje prikazuje nasumično generirani ključ koji se koristio za šifriranje otvorenog teksta. Moguća je isključivo operacija šifriranja.



Slika 5.7 *Okvir jednokratne bilježnice*

5.7 DES kriptosustav

Okvir DES kriptosustava (Slika 6.8) osim klasičnih elemenata sučelja sadrži odabirne okvire *„Text“* i *„Hex“* te polje za unos ključa u heksadecimalnom obliku. Odabirni okviri određuju vrstu ulaznih podataka, odnosno, otvoreni tekst moguće je unesti u tekstualnom ili heksadecimalnom obliku.



Slika 5.8 *Okvir DES kriptosustava*

6. ZAKLJUČAK

Zadatak ovog završnog rada jest implementacija nekoliko različitih kriptosustava u obliku aplikacije edukacijskog značaja. Odabrani kriptosustavi najprije su teorijski obrađeni. Prikazano je kako pojedini kriptosustav funkcionira te je dana matematička podloga koja opisuje proces šifriranja (dešifriranja) pojedinog sustava. Način rada pojedinog sustava prikazan je i odgovarajućim primjerom šifriranja kako bi se njegova funkcionalnost dodatno razjasnila. Aplikacija „CryptoSys“, kao i klase kriptosustava od kojih se ona sastoji, izvedena je pomoću Python programskog jezika. Aplikacija implementira Cezarovu, Supstitucijsku, Vigenereovu, Playfairovu, Hillovu i Transpozicijsku šifru te jednokratnu bilježnicu i DES kriptosustav. Za svaku od navedenih šifri pruža kratak opis, pojašnjenje funkcionalnosti te način pravilnog korištenja unutar aplikacije. Korisnik može šifrirati (dešifrirati) proizvoljne otvorene tekstove (šifrate) s proizvoljno odabranim ključevima šifriranja. Ova aplikacija korisna je za sticanje osnovnih znanja o načinu rada kriptosustava. Osim što pruža teorijske podloge o pojedinim kriptosustavima također omogućava njihovo praktično korištenje.

7. LITERATURA

[1] A. Dujella, M. Maretic: Kriptografija, Element, Zagreb, 2007.

[2] M. Summerfield: Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming, Prentice Hall, 2009.

[3] Security concepts, IBM, dostupno na: https://www.ibm.com/support/knowledgecenter/SSB23S_1.1.0.2020/gtps7/seccon.html [16.8.2020]

[4] National Institute of Standards and Technology, Data Encryption Standard(DES), Federal Information Processing Standards Publication 46-3, 25.10.1999

[5] Python dokumentacija, dostupno na: https://www.python.org/doc/ [16.8.2020]

[6] Qt dokumentacija, dostupno na: https://doc.qt.io/ [16.8.2020]

8. SAŽETAK

Tema završnog rada je aplikacija za učenje o kriptosustavima. Aplikacija implementira Cezarovu, Supstitucijsku, Vigenereovu, Playfairovu, Hillovu i Transpozicijsku šifru te jednokratnu bilježnicu i DES kriptosustav. Navedeni kriptosustavi implementirani su pomoću Python programskog jezika. Korisničko sučelje aplikacije izvedeno je korištenjem PyQt5 modula. Aplikacija sadrži kratak opis funkcionalnosti pojedinog kriptosustava, te omogućava njihovu praktičnu upotrebu.

Ključne riječi: aplikacija, python, učenje, kriptosustav

9. ABSTRACT

The subject of this graduate work is an application for learning about cryptosystems. The application implements Caesar's, Substitution, Vigenere's, Playfair's, Hill's and Transposition cipher as well as One-time pad and DES cryptosystem. Previously mentioned cryptosystems are implemented using the Python programming language. The application's user interface is realized using the PyQt5 module. The application contains a short description of the funcionalities of each cryptosystem and enables their practical usage.

Key words: application, cryptosystem, python, learning

10. ŽIVOTOPIS

Patrik Vinicki rođen je 25.6.1997 u Varaždinu, Hrvatska. Osnovnu školu „OŠ Mate Lovraka“ završava 2012. godine te upisuje „Tehničku školu Kutina“ u Kutini, smjer tehničar za računarstvo. Srednju školu završava 2017. godine te iste godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, stručni smjer Informatika.

11. PRILOZI

caesar.py

```
import converter as cv


class Caesar:
    def __init__(self):
        self.key = 3

    #funkcija šifriranja
    def encrypt(self, plaintext):
        global key
        ciphertext = []
        plaintext = plaintext.lower()
        plaintext = cv.textToNum(plaintext)

        for letter in plaintext:
            ciphertext.append((letter + self.key) % 26)

        return cv.numToText(ciphertext)

    #funkcija dešifriranja
    def decrypt(self, ciphertext):
        plaintext = []
        ciphertext = ciphertext.lower()
        ciphertext = cv.textToNum(ciphertext)

        for letter in ciphertext:
            plaintext.append((letter - self.key) % 26)

        return cv.numToText(plaintext)
```

supstitution.py

```python
import converter as cv

class Supstitution:
    def __init__(self):
        pass

    #funkcija šifriranja
    def encrypt(self, plaintext, key):
        ciphertext = []
        plaintext = plaintext.lower()
        plaintext = cv.textToNum(plaintext)

        for letter in plaintext:
            ciphertext.append((letter + key) % 26)

        return cv.numToText(ciphertext)

    #funkcija dešifriranja
    def decrypt(self, ciphertext, key):
        plaintext = []
        ciphertext = ciphertext.lower()
        ciphertext = cv.textToNum(ciphertext)

        for letter in ciphertext:
            plaintext.append((letter - key) % 26)

        return cv.numToText(plaintext)

    #funkcija za dešifriranje "grubom silom"
    def bruteForce(self, ciphertext):
        plaintext = []
        buffer = []
        ciphertext = cv.textToNum(ciphertext)

        for key in range(1,26):
            for letter in ciphertext:
                buffer.append((letter - key) % 26)

            plaintext.append(buffer)
            buffer = []

        return cv.numArrayToText(plaintext)
```

vigenere.py

```
import converter as cv


class Vigenere():
    def __init__(self):
        self.alphabet = 'abcdefghijklmnopqrstuvwxyz'
        self.letter_matrix = [[0 for x in range(len(self.alphabet))] for y in range(len(self.a
lphabet))]

    #funkcija šifriranja
    def encrypt(self, plaintext, key):
        ciphertext = ''
        numKey = []
        self.generateLetterMatrix()

        plaintext = cv.textToNum(plaintext)
        numKey    = cv.textToNum(key)
        counter = 0

        for letter in plaintext:
            if(counter >= len(key)):
                counter = 0

            ciphertext += self.letter_matrix[letter][numKey[counter]]
            counter += 1

        return ciphertext

    #funkcija šifriranja s autoključem
    def autokeyEncrypt(self, plaintext, key):
        ciphertext = ''
        numKey = []
        self.generateLetterMatrix()

        plaintext = cv.textToNum(plaintext)
        numKey    = cv.textToNum(key)

        counter = 0

        for letter in numKey:
            ciphertext += self.letter_matrix[plaintext[counter]][letter]
            counter += 1

        position = counter
        counter = 0

        while(position < len(plaintext)):
            ciphertext += self.letter_matrix[plaintext[position]][plaintext[counter]]
            counter += 1
            position += 1

        return ciphertext
```

```
def autokeyDecrypt(self, ciphertext, key):
        plaintext = ''
        numKey    = []

        self.generateLetterMatrix()
        ciphertext = cv.textToNum(ciphertext)
        numKey     = cv.textToNum(key)

        counter = 0

        for index in numKey:
            for letter in self.alphabet:
                if(ciphertext[counter] == self.alphabet.index(self.letter_matrix[self.alphabet
.index(letter)][index])):
                    plaintext += letter

            counter += 1

        position = counter
        counter = 0

        while(position < len(ciphertext)):
            for letter in self.alphabet:
                if(ciphertext[position] == self.alphabet.index(self.letter_matrix[self.alphabe
t.index(letter)][self.alphabet.index(plaintext[counter])])):
                    plaintext += letter

            position += 1
            counter += 1

        return plaintext

    #funkcija dešifriranja
    def decrypt(self, ciphertext, key):
        plaintext = ''
        numKey    = []

        self.generateLetterMatrix()

        ciphertext = cv.textToNum(ciphertext)
        numKey     = cv.textToNum(key)
        counter = 0

        #za svaki znak u šifratu
        for cipherletter in ciphertext:
            if(counter >= len(key)):
                counter = 0

            #pronađi slovo koje uz ključ indeksira polje vigenereovog kvadrata
            #čiji je element odgovarajući znak šifrata.
            for letter in self.alphabet:
                if(cipherletter == self.alphabet.index(self.letter_matrix[self.alphabet.index(
letter)][numKey[counter]])):
```

```
                plaintext += letter

            counter += 1

        return plaintext

    #funkcija koja generira Vigenereov kvadrat s predanom abecedom
    def generateLetterMatrix(self):
        starting_position = 0

        for i in range(len(self.letter_matrix)):
            value = starting_position

            for j in range(len(self.letter_matrix[i])):

                if(value >= len(self.alphabet)):
                    value = 0

                self.letter_matrix[i][j] = self.alphabet[value]
                value += 1

            starting_position += 1

        #ispis abecede
        for i in range(len(self.letter_matrix)):
            for j in range(len(self.letter_matrix[i])):
                print(self.letter_matrix[i][j], end=" ")

            print('\n')
```

playfair.py

```
import math

class Playfair:
    def __init__(self):
        self.alphabet = 'abcdefghijklmnopqrstuvwxyz'
        self.matrix = [[0 for x in range(5)] for y in range(5)]
        self.current_row = 0
        self.current_col = 0
        self.flag = True

    #funkcija za šifriranja
    def encrypt(self, plaintext):
        ciphertext = ''
        self.flag = True

        if(len(plaintext) % 2 != 0):
            plaintext += 'x'

        self.generateMatrix(plaintext)

        buffer = []
        for letter in plaintext:
            for row in range(5):
                for col in range(5):
                    if (self.matrix[row][col] == letter):
                        buffer.append([row, col])

                        if(len(buffer) == 2):
                            ciphertext += self.encryptPair(buffer)
                            buffer = []


        print(self.matrix)

        return ciphertext

    def decrypt(self, ciphertext):
        plaintext = ''
        self.flag = False

        buffer = []
        for letter in ciphertext:
            for row in range(5):
                for col in range(5):
                    if (self.matrix[row][col] == letter):
                        buffer.append([row, col])

                        if(len(buffer) == 2):
                            plaintext += self.encryptPair(buffer)
                            buffer = []

        return plaintext
```

```
def encryptPair(self, pair):
        ciphertext = ""
        Arow = pair[0][0]
        Acol = pair[0][1]
        Brow = pair[1][0]
        Bcol = pair[1][1]

        if(Arow == Brow):
            #zamijeni sa sljedecim u retku
            #vjerojatno dodati mod 5 za pomicanje stupaca i redaka kako bi se osigurala ciklic
nost
            if(self.flag):
                ciphertext += self.matrix[Arow][(Acol + 1) % 5]
                ciphertext += self.matrix[Brow][(Bcol + 1) % 5]

            else:
                ciphertext += self.matrix[Arow][(Acol - 1) % 5]
                ciphertext += self.matrix[Brow][(Bcol - 1) % 5]

        elif(Acol == Bcol):
            #zamijeni sa sljedecim u stupcu
            if(self.flag):
                ciphertext += self.matrix[(Arow + 1) % 5][Acol]
                ciphertext += self.matrix[(Brow + 1) % 5][Bcol]

            else:
                ciphertext += self.matrix[(Arow - 1) % 5][Acol]
                ciphertext += self.matrix[(Brow - 1) % 5][Bcol]

        else:
            #zamijeni vrhovima kvadrata kojeg formiraju elementi
            if(Arow < Brow):
                ciphertext += self.matrix[Arow][Bcol]
                ciphertext += self.matrix[Brow][Acol]
            else:
                ciphertext += self.matrix[Brow][Acol]
                ciphertext += self.matrix[Arow][Bcol]

        return ciphertext

    #funkcija koja stvara matricu šifriranja
    def generateMatrix(self, plaintext):
        padding = [x for x in self.alphabet if x not in plaintext]   #slova abecede koja nisu
u otvorenom tekstu
        plaintext = plaintext.lower()

        unique_letters = self.getUniqueLetters(plaintext)
        self.fillMatrix(unique_letters)
        self.padMatrix(padding)


    #funkcija koja stvara listu jedinstvenih slova otvorenog teksta
    def getUniqueLetters(self, plaintext):
```

```
unique_letters = [];

        for letter in plaintext:
            if (letter in self.alphabet):
                if letter not in unique_letters:
                    unique_letters.append(letter)

        return unique_letters

    #funkcija koja popunjava matricu jedinstvenim slovima otvorenog teksta
    def fillMatrix(self, unique_letters):
        flag = True
        counter = 0

        for row in range(5):
            for col in range(5):
                if (flag != True):
                    self.current_row = row
                    self.current_col = col
                    break

                self.matrix[row][col] = unique_letters[counter]
                counter += 1

                if (counter == len(unique_letters)):
                    flag = False

            if(flag != True):
                break

    #funkcija koja dopunjuje matricu slovima abecede
    def padMatrix(self, padding):
        counter = 0
        startPadding = False

        for row in range(5):
            for col in range(5):
                if(row == self.current_row and col == self.current_col):
                    startPadding = True

                if (startPadding == True):
                    self.matrix[row][col] = padding[counter]
                    counter += 1
```

hill.py

```
import math
import converter as cv
import numpy as np
from numpy.linalg import inv, det

class Hill:
    def __init__(self):
        self.key = []
        pass

    #funkcija šifriranja
    def encrypt(self, plaintext, key, m = 3):
        self.key = key
        ciphertext = ''
        letter_vector = []
        buffer = []

        #pretvori otvoreni tekst u niz brojeva
        plaintext = cv.textToNum(plaintext)

        plaintext = self.padPlaintext(plaintext, m)

        for value in plaintext:
            buffer.append(value)

            #Stvori 2D polje koje se sastoji od m-elementnih polja
            if(len(buffer) == m):
                letter_vector.append(buffer)
                buffer = []

        ciphertext = self.calculateCipher(letter_vector, m)

        return ciphertext

    def decrypt(self, ciphertext, m = 3):
        plaintext = ''
        self.invkey = self.findInverseMatrix()
        plaintext = self.encrypt(ciphertext, self.invkey)

        return plaintext

    def findInverseMatrix(self):
        #pronadji determinantu
        determinant = round(det(self.key))
        determinant = determinant % 26

        #pronadji inverz determinante
        inv_determinant = self.findMultiplicativeInverse(determinant)

        #izracunaj matricu
        adjugate_matrix = self.calculateAdjugateMatrix()

        #adjugate_matrix = inv(self.key)
```

```
#adjugate_matrix = np.dot(inv(self.key), inv_determinant)

        #promijeni predznake dobivene matrice
        adjugate_matrix[0][1] = -1 * adjugate_matrix[0][1]
        adjugate_matrix[1][0] = -1 * adjugate_matrix[1][0]
        adjugate_matrix[1][2] = -1 * adjugate_matrix[1][2]
        adjugate_matrix[2][1] = -1 * adjugate_matrix[2][1]

        #mod 26 dobivenu matricu
        for row in range(3):
            for col in range(3):
                adjugate_matrix[row][col] = round(adjugate_matrix[row][col] % 26)
                print(round(adjugate_matrix[row][col] % 26))

        #pomnozi matricu i inverznu determinantu
        adjugate_matrix = np.dot(adjugate_matrix, inv_determinant)

        #mod 26
        for row in range(3):
            for col in range(3):
                adjugate_matrix[row][col] = round(adjugate_matrix[row][col] % 26)

        return adjugate_matrix

    def calculateAdjugateMatrix(self):
        adjugate_matrix = [[0 for x in range(3)] for y in range(3)]
        adjugate_matrix[0][0] = (self.key[1][1] * self.key[2][2]) - (self.key[1][2] * self.key
[2][1])
        adjugate_matrix[0][1] = (self.key[0][1] * self.key[2][2]) - (self.key[0][2] * self.key
[2][1])
        adjugate_matrix[0][2] = (self.key[0][1] * self.key[1][2]) - (self.key[0][2] * self.key
[1][1])
        adjugate_matrix[1][0] = (self.key[1][0] * self.key[2][2]) - (self.key[1][2] * self.key
[2][0])
        adjugate_matrix[1][1] = (self.key[0][0] * self.key[2][2]) - (self.key[0][2] * self.key
[2][0])
        adjugate_matrix[1][2] = (self.key[0][0] * self.key[1][2]) - (self.key[0][2] * self.key
[1][0])
        adjugate_matrix[2][0] = (self.key[1][0] * self.key[2][1]) - (self.key[1][1] * self.key
[2][0])
        adjugate_matrix[2][1] = (self.key[0][0] * self.key[2][1]) - (self.key[0][1] * self.key
[2][0])
        adjugate_matrix[2][2] = (self.key[0][0] * self.key[1][1]) - (self.key[0][1] * self.key
[1][0])

        return adjugate_matrix

    def findMultiplicativeInverse(self, determinant):
        inv_determinant = -1

        for x in range(26):
            if(((determinant * x) % 26) == 1):
                inv_determinant = x
```

```
return inv_determinant

    #funkcija za izračun šifrata
    def calculateCipher(self, letter_vector, m):
        ciphertext = ''
        vector = []
        buffer = 0

        for value in range(len(letter_vector)):
            for col in range(m):
                for row in range(m):
                    #pomnoži blok od m slova s matricom ključa
                    buffer += int((letter_vector[value][row] * self.key[col][row]))

                buffer %= 26
                vector.append(buffer)
                buffer = 0

            #pretvori izračunate vrijednosti bloka u tekst
            print(vector)
            ciphertext += cv.numToText(vector)
            print(ciphertext)
            vector = []

        return ciphertext

    def padPlaintext(self, plaintext, m):
        remainder = len(plaintext) % m
        if(remainder != 0):
            for value in range (m -  remainder):
                plaintext.append(23)

        return plaintext
```

transposition.py

```
import math

class Transposition:
    def __init__(self):
        self.alphabet = 'abcdefghijklmnopqrstuvwxyz'
        self.transposition_matrix = []

    def encrypt(self, plaintext, key):
        #if(math.ceil(len(plaintext)/len(key)) == 1 ):
        #    return 1
        text = ''

        for letter in plaintext:
            if letter not in self.alphabet:
                continue
            else:
                text += letter

        rows = math.ceil(len(text)/len(key))

        self.generateTranspositonMatrix(text, rows, len(key))

        return self.generateCiphertext(key)

    def decrypt(self, ciphertext, key):
        padding = len(ciphertext) % len(key)

        if(padding != 0):
            for x in range(len(key) - padding):
                ciphertext += 'x'

        print(ciphertext)

        rows = math.ceil(len(ciphertext)/len(key))
        matrix = [[0 for x in range(len(key))] for y in range(rows)]


        plaintext = ""
        buffer = ""
        column_letters = []
        ordered_letters = []

        #stvori listu slova ciji su elementi slova stupaca
        for letter in ciphertext:
            buffer += letter

            if (len(buffer) == rows):
                column_letters.append(buffer)
                buffer = "";

        #poredaj stupce slova prema kljucu
        for value in key:
            ordered_letters.append(column_letters[value - 1])
```

```
#popuni matricu s ispravnim poretkom slova
        for col in range(len(key)):
            for row in range(rows):
                matrix[row][col] = ordered_letters[col][row]

        #iscitaj redove popunjene matrice
        for row in range(rows):
            for col in range(len(key)):
                plaintext += matrix[row][col]

        return plaintext

    def generateTranspositonMatrix(self, plaintext, rows, cols):
        self.transposition_matrix = [[0 for x in range(cols)] for y in range(rows)]
        print(self.transposition_matrix)
        padding = (cols * rows) - len(plaintext)
        counter = 0

        for row in range(rows):
            for col in range(cols):
                self.transposition_matrix[row][col] = plaintext[counter]

                counter += 1

                if(counter >= len(plaintext)):
                    self.addPadding(padding, row, col)
                    break


    def addPadding(self,  padding, row, col):
        counter = 0
        col += 1

        while(padding > 0):
            self.transposition_matrix[row][col + counter] = 'x'

            padding -= 1
            counter += 1


    def generateCiphertext(self, key):
        ciphertext = ''
        for num in range(1, len(key)+1):
            index = key.index(num)

            for row in self.transposition_matrix:
                ciphertext += str(row[index])

        return ciphertext
```

onetimepad.py

```python
import random as rand
import converter as cv

class OneTimePad:
    def __init__(self):
        pass

    #funkcija šifriranja
    def encrypt(self, plaintext):
        ciphertext = []
        result = []
        key = []
        plaintext = cv.textToNum(plaintext)

        #za svaku vrijednost otvorenog teksta dodaj pseudonasumican broj u rasponu 0-25
        for value in range(len(plaintext)):
            key.append(rand.randrange(0, 26))

        #modularno zbroji otvoreni tekst i kljuc
        ciphertext = self.modularAddition(plaintext, key)
        ciphertext = cv.numToText(ciphertext)

        result.append(ciphertext)
        result.append(cv.numToText(key))

        return result

    def modularAddition(self, text, key):
        ciphertext = []
        for value in range(len(text)):
            ciphertext.append((text[value] + key[value]) % 26)

        return ciphertext
```

DES.py

```
def encrypt(plaintext, key, isText, alphabet = 'abcdefghijklmnopqrstuvwxyz'):
    flag = True
    ciphertext = ''
    buffer = ''
    roundKeys = generateRoundKeys(key)
    print('roundkeys encrypt: ')
    print(roundKeys)
    print('\n')

    #napravi listu blocks u koju dodas sve blokove i onda procesirat
    if(isText):
        for letter in plaintext:
            if letter not in alphabet:
                continue
            else:
                buffer += letter

            if(len(buffer) == 8):
                block = textToBin(buffer)

                block = permuteBlock(block)
                ciphertext += processBlock(block, roundKeys, flag)
                buffer = ''
    else:

        for letter in plaintext:

            buffer += letter

            if(len(buffer) == 16):
                print(buffer)
                block = hexToBin(buffer)

                block = permuteBlock(block)
                ciphertext += processBlock(block, roundKeys, flag)
                buffer = ''

    print("encrypted block: " + ciphertext)
    ciphertext = hex(int(ciphertext, 2))[2:]
    return ciphertext

def permuteBlock(block):
    buffer = ''

    for element in initialPerm:
        buffer += block[element - 1]

    return buffer

def decrypt(ciphertext, key, alphabet = 'abcdefghijklmnopqrstuvwxyz'):
    flag = False
    plaintext = ''
    buffer = ''
```

```
    blocks = []
    roundKeys = generateRoundKeys(key)
    roundKeys = list(reversed(roundKeys))
    print('roundkeys decrypt: ')
    print(roundKeys)
    print('\n')

    ciphertext = bin(int(ciphertext, 16))[2:].zfill(64)

    for element in ciphertext:
        buffer += element

        if(len(buffer) == 64):
            blocks.append(buffer)
            buffer = ''

    for element in blocks:
        element = permuteBlock(element)
        plaintext += processBlock(element, roundKeys, flag)
        print("decrypted block: " + plaintext)

    plaintext= hex(int(plaintext, 2))[2:]
    return plaintext


def processBlock(block, roundKeys, flag):
    lbl = ''
    rbl = ''
    temp = ''
    encrypted_block = ''

    for n in range(32):
        lbl += block[n]

    for n in range(32, 64):
        rbl += block[n]

    for n in range(16):
        result = feistelRound(lbl, rbl, roundKeys[n])
        lbl = result[0]
        rbl = result[1]

    temp = rbl + lbl

    #inverzna permutacija
    for element in finalPermutation:
        encrypted_block += temp[element-1]

    return encrypted_block

def feistelRound(lbl, rbl, key):
    temp = rbl
    result = []
```

```
rbl = roundFunction(rbl, key)

    #xor lijeve i desne polovice
    lbl = int(lbl, 2)
    rbl = int(rbl, 2)

    rbl = rbl ^ lbl
    rbl = bin(rbl)[2:].zfill(32)


    #zamjena
    lbl = temp

    result.append(lbl)
    result.append(rbl)

    return result

def roundFunction(block, key):
    expblock = ''
    sblocks = []
    sresult  = ''
    rbl = ''

    #ekspanzija
    for element in expansionDBox:
        expblock += block[element-1]

    #XOR s ključem
    expblock = int(expblock, 2)
    key      = int(key, 2)

    expblock = expblock ^ key
    expblock = bin(expblock)[2:].zfill(48)

    #razdvajanje na 6 bitne blokove
    buffer = ''
    for element in expblock:
        buffer += element

        if(len(buffer) == 6):
            sblocks.append(buffer)
            buffer = ''


    #supstitucija
    for n in range(8):
        row = int((sblocks[n][0] + sblocks[n][5]), 2)
        col = int((sblocks[n][1] + sblocks[n][2] + sblocks[n][3] + sblocks[n][4]), 2)
        sresult += bin(sboxes[n][row][col])[2:].zfill(4)


    for element in straightDBox:
```

```
        rbl += sresult[element-1]

    return rbl



def hexToBin(text):
    bin_block = bin(int(text, 16))[2:].zfill(64)
    #bin_block = bin_block.split(' ')

    return bin_block

def textToBin(text):
    bin_block = ''.join(format(ord(x), 'b') for x in text)[2:].zfill(64)


    return bin_block


def generateRoundKeys(key):
    roundKeys  = []
    temp = []
    permuted_key = ''

    key = bin(int(key, 16))[2:].zfill(64)

    #inicijalna permutacija
    for element in initialPermutation:
        permuted_key += key[element-1]

    temp = getShiftedKeys(permuted_key)

    #stvaranje kljuceva za pojedinu rundu
    for n in range(16):
        buffer = ''

        for element in compressionDBox:
            buffer += temp[n][element - 1]

        roundKeys.append(buffer)

    return roundKeys

def getShiftedKeys(permuted_key):
    temp = []
    lh = ''
    rh = ''

    #lijeva polovica kljuca
    for n in range(28):
        lh += permuted_key[n]

    #desna polovica kljuca
```

```
    for n in range(27, 56):
        rh += permuted_key[n]


    #pomicanje bitova kljuceva ulijevo
    for element in key_shift:
        lh = int(lh, 2)
        rh = int(rh, 2)
        lh = rotateLeft(lh, element)
        rh = rotateLeft(rh, element)

        temp.append(lh + rh)

    return temp


def rotateLeft(bits, n):
    temp = '11111111111111111111111111'
    temp = int(temp, 2)

    fillBits  = int(bin(bits >> (28 - n))[2:].zfill(28),2)
    leftShift = int(bin(temp & bits << n)[2:].zfill(28),2)

    return bin(leftShift | fillBits)[2:].zfill(28)
```