SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Diplomski studij

ANDROID APLIKACIJA ZA UČENJE O KLASIČNIM KRIPTOSUSTAVIMA

Diplomski rad

Marko Radanović

Osijek, 2018.

SADRŽAJ

1. UVOD

Danas su kriptografski sustavi u razmjeni podataka i poruka neizbježni dio osiguravanja njihove tajnosti i konzistentnosti. Svaki uređaj koji se spaja na vanjsku mrežu (internet) podležan je digitalnom napadu gdje postoji mogućnost krađe osobnih podataka te drugih informacija od visoke važnosti. Stoga je skrivanje sadržaja podataka iliti enkripcija nužna pri slanju kroz vanjsku mrežu. Djelotvorna i sigurna komunikacija bila je vrlo bitan faktor kraljevima, kraljicama, vojskovođama stoljećima, zbog svjesnosti o posljedicama koje bi mogle rezultirati ukoliko njihove poruke dođu u krive ruke.

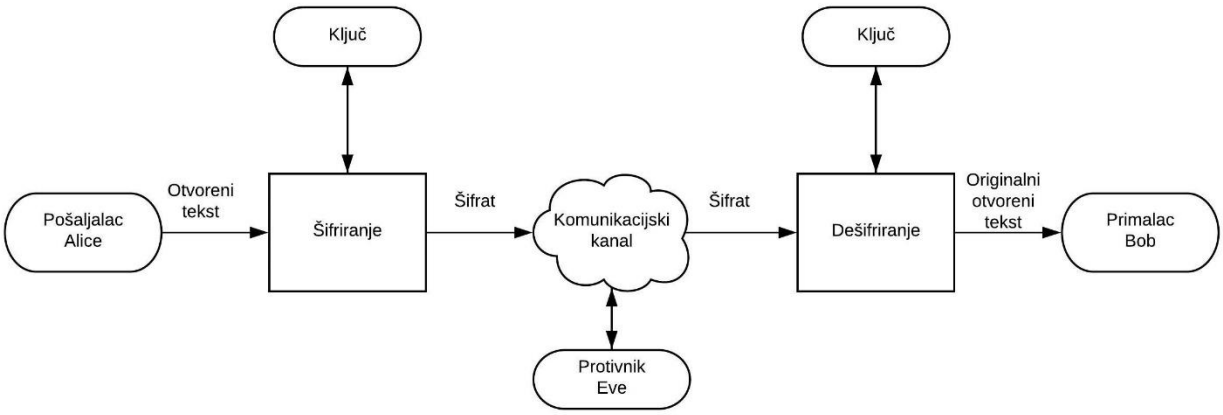
Uporaba enkripcije ima svoje tragove još od 5. stoljeća prije Krista kod starih Grka i od onda nije izašla iz uporabe. S vremenom se dosta stvari promjenilo, broj korisnika koji šalje svoje podatke preko nesigurnog kanala svakim danom se povećava te se isto tako povećava i broj korisnika koji žele doći do tih informacija i zlouporabiti ih. Radi tih činjenica uz razvoj tehnologije potreba za pronalaskom novih kriptosustava nikad nije prestala. Dobar temelj za razumijevanje današnjih kriptosustava su klasični kriptosustavi, kriptosustavi koji datiraju od 5. stoljeća prije Krista do pojave računala 70tih godina 20. stoljeća.

Ovaj diplomski rad daje uvid u klasične kriptosustave i njihovu primjenu te donosi aplikaciju za Android mobilni sustav gdje korisnik može učiti o klasičnim kriptosustavima te šifrirat i dešifrirati pomoću njih. Android sustav tvrtke Google spada u najkorištenije mobilne operacijske sustave, otvorenog je koda te besplatan za distribuciju što mu donijelo značajnu prednost u usporedbi s ostalim operacijskim sustavima. Aplikacije za Android operacijski sustav može pisati bilo tko, za početak razvoja korisniku je potrebno računalo s instaliranim Android studio razvojnim okruženjem i znanje programskog jezika Java, C++ ili Kotlin. Radi popularnosti sustava, njegovih mogućnosti i zahtjeva za razvoj početak razvoja je za zadatak ovog diplomskog rada upravo odabran Android sustav s Java programskim jezikom.

Diplomski rad podijeljen je na pet poglavlja, drugo poglavlje ovog diplomskog rada daje korisniku uvod u klasične kriptosustave i pojedine metode uz primjere. Sljedeće, treće poglavlje govori o Android sustavu, njegovom razvoju kroz vrijeme, uporabom te kratkom usporedbom s ostalim mobilnim operacijskim sustavima. 4. poglavlje donosi uvid u rješenje praktičnog dijela diplomskog rada, razvoj aplikacije dok zadnje, 5. poglavlje pogled na riješenje i uputstva za uporabu.

2. KRIPTOGRAFIJA

Kriptografija (engl. *Cryptography*) je znanost koja se bavi proučavanjem metoda za sigurno slanje poruka preko nesigurnog kanala tako da poruku može razumijeti samo osoba kojoj je poruka namijenjena. Riječ kriptografija grčkog je podrijetla te je spoj dviju riječi, pridjeva *kriptós* što znači "skriven" i glagola *gráfo* značenja "pisati". Svojstvo tajnosti poruke postiže se preoblikovanjem njenog sadržaja u na oko nerazumljiv oblik – šifrat (engl. *Cipher*), koji se uz pomoć metoda za dešifriranje može preoblikovati nazad u razumljivu poruku ili tzv. *otvoreni tekst (engl. Plain text).* Za šifriranje i dešifriranje poruka također je potreban ključ, unaprijed dogovorena *fraza* između pošaljilaca i primaoca uz pomoć kojeg strane mogu sigurno šifrirati i dešifrirati poruke. Od velike je važnosti da ključ znaju samo strane koje sudjeluju u komunikaciji tako da treća strana ne može iz šifrata izvući otvoreni tekst ili malverzirati komunikaciju slanjem poruka šifriranim istim ključem. U kriptografskoj terminologiji za pošaljilaca i primaoca rezervirana su imena Alice i Bob, dok je za treću stranu koja pokušava prisluškivati komunikaciju rezervirano Eve. Osnovni princip razmjene poruka glasi: Alice šifrira poruku te ih šalje Bobu, Eve ima pristup šifratu no ne zna njegov sadržaj te ju pokušava 'razbiti' dok Bob prima šifrat, dešifrira je i čita poruku. Za ovaj scenarij nužno je da Alice i Bob imaju isti ključ. Slikovni prikaz ovog principa vidljiv je na slici 2.1.



Slika 2.1. Osnovni princip razmjene poruka primjenom kriptografije

Uporaba kriptografije je svoje početke već imala za vrijeme starih Grka gdje su spartanci upotrebljavali Skital (engl. Skytale), drveni štap specifične debljine oko kojeg se namotavala vrpca te okomito na nju pisala poruka, odmotavanjem vrpce dobije se cedulja s šifratom i tek nakon ponovnog omotavanja cedulje oko skitala iste debljine mogla se poruka pročitati.

Rimski diktator Gaj Julije Cezar također je u komunikaciji upotrebljavao vrstu kriptografskog sustava kad je sadržaj poruke bio od vojne važnosti. Svako slovo unutar poruke zamjenio bi sa slovom koje 3 mjesta dalje od slova otvorenog teksta npr. slovo A zamjenio bi sa slovom D, B sa slovom E itd. Kriptografski algoritam kojim se koristio Cezar danas nazivamo "Cezarova šifra". Popularna Cezarova izreka "VENI VIDI VICI" šifrirana Cezarovom šifrom glasila bi "YHQL YLGL YLFL". Iz prijašnjeg opisa skitala da se zaključiti da je kod njega ključ za šifriranje i dešifriranje bila debljina skitala dok kod Cezarove šifre broj 3 te da navedene metode ne preoblikuju tekst na isti način, enkripcija skitalom mijenja poruci redosljed slova no sama slova ostaju ista dok Cezarova šifra očuva redosljed a mijenja slova. Općenito dijelimo kriptosustave prema sljedećim kriterijima[1]:

1. Tip operacija koje se koriste pri šifriranju

Pod ovim kriterijem razlikujemo *transpozicijske* i *supstitucijske* šifre; Transpozicijske šifre preoblikuju otvoren tekst tako da im se zamjeni redosljed elemenata (kao na primjer Skital), dok supstitucijske mjenjaju samo element poruke (Cezarova šifra)

2. Način na koji se obrađuje otvoreni tekst

Ovdje se razlikuju *blokovne* i *protočne* šifre, kod blokovnih se vremenom obrađuje jedan blok poruke uvijek istim ključem dok se kod protočnih obrađuje element po element poruke koristeći niz ključeva koji se u procesu dešifriranja generiraju.

3. Tajnost i javnost ključeva

Kod zadnjeg kriterija razlikujemo simetrične kriptosustave i kriptosustave s javnim ključem. Značajka simetričnih kriptosustava jest da se ključ za dešifriranje može izračunati poznavajući ključ za šifriranje i obrnuto, no najčešći je slučaj da se radi o istom ključu te je od velike važnosti da taj ključ ostane tajan. Za razliku od simetričnih kriptosustava, *kriptosustavi s javnim ključem* iliti *asimetrični kriptosustavi* temelje se na dvije vrste ključa, *javni* ključ s kojim se šifrira poruka te tajni s kojim se dešifrira poruka enkriptirana *javnim* ključem. Teoretski je moguće kao kod simetričnih kriptosustava izračunati ključ za dešifriranje iz ključa za šifriranje no nemoguće ga je

izračunati u razumnom vremenu stoga se generalno govori da ga u praksi *nije* moguće izračunati. Javnim ključem može bilo tko šifrirati poruku no dešifrirati ju može samo osoba koja posjeduje tajni ključ.

Kako se podrazumijeva da šifrirane poruke sadrže informacije od velike važnosti tako se može i zaključiti da su vremenom razvijene metode 'razbijanja' šifrata bez ključa, znanstvena disciplina koja se bavi razvojem takvih metoda zove se *kriptoanaliza* ili *dekriptiranje*. Unatoč tome što je cilj kriptoanalize ostao isti, metode i tehnike su se vremenom drastično promijenile u smislu povećavanja kompleksnosti matematičkog problema kojim bi se došlo do otvorenog teksta ili ključa. Za rane kriptografske sustave je za kriptoanalizu dovoljno bilo imati olovku i papir dok se za današnje kriptosustave koriste matematički napredne računalne algoritme. Kod kriptoanalize razlikujemo četiri osnovne razine *napada* – načina 'razbijanja'[1]:

1. **Samo šifrat** gdje kriptoanalitičar pomoću više šifrata šifriranih istim algoritmom pokušava otkriti otvoreni tekst što više šifrata ili u najboljem slučaju otkrivanjem ključa kojim su poruke šifrirane.
2. **Poznati otvoreni tekst** gdje kriptoanalitičar posjeduje šifriranu poruku i njegov otvoreni tekst te pomoću njih pokušava otkriti ključ ili algoritam s kojim će moći dešifrirati poruke tim ključem.
3. **Odabrani otvoreni tekst** kod koljeg kriptoanalitičar ima mogučnost odabira otvorenog teksta koji će se šifrirati te uz pomoć otvorenog teksta i šifrata izračunati ključ ili algoritam za šifriranje.
4. **Odabrani šifrat** gdje kriptoanalitičar ima pristup alatu za šifriranje stoga ima i mogučnost upisivanja proizvoljnog šifrata te dobivanjem otvorenog teksta. Zadatak kriptoanalitičara ovdje je otkrivanje ključa za dešifriranje (*tajnog ključa)*, ovaj napad tipičan je za asimetrične kriptosustave gdje postoje javni i tajni ključevi.
5. **Potkupljivanje, ucjena, krađa i slično,** ova vrsta napada ne ubraja se točno u kriptoanalizu li je vrlo efikasan i često primjenjivan u kombinaciji s "pravim" kriptoanalitičkim napadima.

Osnovna pretpostavka kod kriptoanalize jest da kriptoanalitičar zna s kojim kriptosustavom je šifrat šifriran. Ova pretpostavka zove se *Kerckhoffsovo načelo* i nazvana je prema nizozemcu Augustu Kerckhoffsu, autoru knjige "Vojna kriptografija". Ova pretpostavka u praksi ne mora biti uvijek točna

no nedopustivo je osloniti se na činjenicu da protivnik ne zna o kojem kriptosustavu se radi pogotovo ukoliko se radi o porukama od velike važnosti.

Ovaj diplomski rad se bavi klasičnim kriptosustavima, klasičnim kriptosustavima smatraju se kriptosustavi koji su se koristili kroz povijest no s vremenom su izašle iz uporabe radi pojave novijih i sigurnijih kriptosustava. Za razliku od modernih kriptosustava, većina šifrata kriptirano klasičnim kriptosustavima je lako za izračunati i dešifrirati, najčešće je dovoljna samo olovka i papir stoga daju dobru podlogu i uvod u kriptografiju i kriptoanalizu. Klasični kriptosustavi kao već napomenuto datiraju još iz vremena starih Grka i Rimljana no njihovi tragovi pronalaze se i u renesansnom dobu te prvom i drugom svjetskom ratu no razvoj i aktivna uporaba završava 70-tih godina 20. stoljeća gdje se kriptosustavi počinju oslanjati na nove algoritme i računala. Danas se osim u obrazovne svrhe klasični kriptosustavi koriste ponekad kao dijelovi većih modernih kriptosustava, primjer takve uporabe je *MixColumn* korak u *Advanced Encryption Standard (AES)* gdje se koristi *Hill* kriptosustav. Pod klasične kriptosustave ubrajamo:

- Supstitucijske šifre
- Vigenerova šifra
- Playfair šifra
- Hill šifra
- Jednokratna bilježnica
- Transpozicijske šifre

U nastavku se nalaze podpoglavlja s pojašnjenjem svake od navedenih klasičnih kriptosustava uz primjere enkripcije, dekripcije i kriptoanalize, prvih 5 podpoglavlja opisuju kriptosustave koji mjenjaju elemente poruke (supstitucijske šifre) dok zadnje, 6. podpoglavlje opisuje transpozicijske metode enkripcije.

2.1. Supstitucijske šifre

Supstitucijske šifre su kriptosustavi gdje poredak slova poruke ostaje isti, ali se za svako slovo koristi drugo slovo. Takve šifre zovu se supstitucijske jer nešto supstituira (mijenja) svako slovo poruke. Većina supstitucijskih šifri su monoalfabetske, što znači da svako slovo ima samo jedno supstitucijsko slovo ili znak. Ukoliko je šifra za slovo K slovo B, onda svaki puta kada se pojavi slovo B u šifri znamo da ono znači K i niti jedno drugo slovo osim B nema to značenje. Za primjer se može ponovo

spomenuti Cezarova šifra, supstitucijska šifra koja zamjenjuje svako slovo sa slovom koje je tri mjesta dalje od njega. Primjenom Cezarove šifre na englesku (međunarodnu) abecedu dobija se sljedeća kodna tablica:

Tablica 2.1. Zapis Cezarove šifre

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

Svako slovo odgovara točno onome slovu koje je ispod (ili iznad) njega. Tako će rečenica "IMAMO PREDSJEDNICU" šifrirana glasit:

LPDPR SUHGVMHGQLFX

I ukoliko se šifrat grupira još u grupe po četiri slova dobija se šifra:

LPDP RSUH GVMH GQLF X

Supstitucijske šifre ne moraju nužno mjenjati slovo za slovo već se često slova interpretiraju i kao brojevi, tako svako slovo u abecedi ima svoj pridodjeljen broj, što nam omogućuje pogled na supstitucijsku šifru s matematičke strane.

Neka je skup $\{0, 1, 2, \ldots, 26\}$ označen s $\mathbf{Z}_{26}$, te se pretpostavlja da su na njemu definirane operacije zbrajanja, oduzimanja i množenja na isti način kao u skupu cijelih brojeva. Ukoliko rezultat bude veći od 26 i na taj način van skupa, potrebno ga je zamjeniti s njegovim ostatkom pri dijeljenju s 26. Operacije zbrajanja i oduzimanja s modulo operacijom zadovoljava aksiome matematičke strukture zvanom *prsten*. Što znači da su operacije zbrajanja i oduzimanja zatvorene, komutativne i asocijativne te vrijedi distributivnost množenja prema zbrajanju. Potpuno analogno se definira skup $\mathbf{Z}_m$ i operacije na njemu za proizvoljan broj *m*. U tom tonu se Cezarova šifra može definirati na sljedeći način[1]:

Neka je $P = C = K = \mathbf{Z}_{26}$. Za $0 \leq K \leq 25$ definiramo

$$e_K(x) = (x + K) \bmod 26, \qquad d_K(y) = (y - K) \bmod 26 \qquad (2\text{-}1)$$

gdje je:

- P – konačan skup svih mogućih osnovnih elemenata konačnog skupa,

- C – konačan skup svih mogućih osnovnih elemenata šifre
- K – konačan skup svih mogućih ključeva
- e_K – enkripcijska funkcija
- d_K – dekripcijska funkcija

Kao primjer će se enkriptirati riječ "IMAMO" cezarovom šifrom sa ključem 3. Prvo se svako slovo zamjenjuje njegovom numeričkom vrijednošću:

I	M	A	O
8	12	0	14

Nakon dodijeljivanja vrijednosti svako slovo poruke enkriptira se enkripcijskom funkcijom iz (2-1) i ključem K=3.

$$e_K(I) = (8 + 3) \bmod 26 = 11 \text{ (J)}$$
$$e_K(M) = (12 + 3) \bmod 26 = 15 \text{ (P)}$$
$$e_K(A) = (0 + 3) \bmod 26 = 3 \text{ (D)}$$
$$e_K(O) = (14 + 3) \bmod 26 = 17 \text{ (R)}$$

Rezultat je kao i u prijašnjem primjeru šifrat "LPDPR". Dekripcija se vrši na isti način uporabom dekripcijske funkcije iz (2-1). Šifrat enkriptiran Cezarovom šifrom jednostavan je i za 'razbiti', dovoljno je *provrtiti* šifrat kroz sve ključeve dok se ne dobije smislena poruka, što je uz prostor ključeva od 26 moguće dekriptirati u kratkom vremenu olovkom i papirom. Kako se dobila sigurnija šifra, za proces šifriranja se mogu uzeti u obzir funkcije koje će uključivati više od jednog parametra. Jedna takva, također i najjednostavnija funkcija je afina funkcija $f(x) = ax + b$. Ovdje se pojavljuje novi problem, afina funkcija u skupu $\mathbf{Z}_{26}$ ne mora nužno imati inverz, stoga parametar a ne može biti proizvoljan nego mora biti relativno prost s modulom 26. Afina šifra definira se na sljedeći način[1]:

Neka je $P = C = \mathbf{Z}_{26}$, te neka je

$$K = \{(a,b) \in \mathbf{Z}_{26} \times \mathbf{Z}_{26} : (a,26) = 1\}.$$

Za $K = (a,b) \in K$ definiramo

$$e_K(x) = (ax + b) \bmod 26, \qquad d_K(y) = a^{-1}(y - b) \bmod 26 \qquad (2\text{-}2)$$

Šifra je dobila ime po istoimenoj funkciji. Kako broj 26 nije prost, nemaju svi elementi iz skupa $\mathbf{Z}_{26}$ multiplikativni inverz, već ih imaju brojevi koji su relativno prosti s 26 što znači da je najveći dijeljitelj od a i 26 jednak 1. Sljedeća tablica prikazuje te brojeve i njihove inverze.

Tablica 2.2. Relativno prosti brojevi s 26 iz skupa Z_{26} *i njihovi inverzi*

a	1	3	5	7	9	11	15	17	19	21	23	25
a^{-1}	1	9	21	15	3	19	7	23	11	5	17	25

Prvi korak kod enkripcije je kao i kod ostalih supstitucijskih šifri, pretvorba slova u ekvivalentne brojeve iz skupa $\mathbf{Z}_{26}$. Sljedeći korak je primjena afine funkcije za enkripciju (2-2) na svaki element otvorenog teksta kako bi se dobio šifrat. To se postiže množenjem brojčane vrijednosti slova s vrijednošću *a* zadanog ključa te dodavanjem broja *b* rezultatu umnoška. Rezultatu afine funkcije ostaje pronalazak ostatka dijeljenja s 26 koji se pretvara nazad u slovo.

Kao primjer šifrirati će se riječ „Osijek“ afinom šifrom s ključem K = (5,3). Brojevi unutar zagrade predstavljaju brojeve *a* i *b* u afinoj funkciji, treba još napomenuti da su oba broja relativno prosti s 26 kako bi postojala mogućnost dešifriranja šifrata. Riješenje primjera dano je u tablici 2.3.

Tablica 2.3. Primjer enkripcije riječi „Osijek“ afinom šifrom (5,3)

Otvoreni tekst	O	S	I	J	E	K
x	14	18	8	9	4	10
5x+3	73	93	43	48	23	53
(5x+3) mod 26	21	15	17	22	23	1
Šifrat	V	P	R	W	X	B

Proces dešifriranja analogan je procesu šifriranja s razlikom da se umjesto funkcije za šifriranje $e_K(x)$ iz (2-2) koristi funkcija za dešifriranje $d_K(x)$ iz (2-2). Pri dešifriranju se još umjesto broja *a* iz ključa koristi njegov inverz (Tablica 2.2).

Kako je afina šifra monoabecedna supstitucijska šifra, nasljeđuje slabosti te vrste šifriranja. Uzimajući u obzir da se koristi univerzalna (engleska) abeceda gdje je *m* jednak 26, ukupno postoji 12 brojeva koji su relativno prosti s 26 i koji su manji od 26 (broj *a* u funkciji) te uz to svaki broj *a* može imati

26 različitih dodatnih pomicanja (broj *b*). Prema tome, ukupno 12 * 26, odnosno ukupno 312 mogućih ključeva što se prema Kerckhoffsovom načelu smatra vrlo nesigurnom šifrom.

Primarna slabost afine šifre jest činjenica da ukoliko kriptoanalitičar uspije pronaći otvoreni tekst dvaju znakova šifrata, lako može doći do ključa riješavajući istodobnu jednadžbu. Pošto je poznato da su broj *a i m* relatino prosti, odma se pri početku dekripcije šifrata mogu odbaciti mnogo „netočnih" ključeva u automatiziranom sustavu.

Cezarova i afina šifra su slučajevi supstitucijskih šifri koji se mogu definirati na matematički način[1]:

Neka je $P = C = \boldsymbol{Z_{26}}$. Prostor ključeva K se sastoji od svih permutacija skupa {0, 1, 2, ... , 25}. Za svaku permutaciju $\pi \in K$ definiramo

$$e\pi(x) = \pi(x), \quad d\pi(y) = \pi\text{-}1(y) \qquad (2\text{-}3)$$

gdje je π^{-1} inverzna permutacija od π.

Osim slaganja abecede šifrata Cezarovom ili afinom šifrom, abeceda šifrata se može sastaviti i nasumično gdje se za svako slovo abecede otvorenog teksta izabire permutacija slova A, B, ... , Z. Te prema tako sastavljenoj tablici šifrira otvoreni tekst. Takva vrsta šifriranja naziva se *Mixed Alphabet Cipher* te za razliku od Cezarove i afine šifre, umjesto broja za ključ, često koristi ključnu riječ. Ovakvo šifriranje ima čak 26! odnosno otprilike $4 * 10^{26}$ mogućih ključeva što otkrivanje ključa ispitivanjem iliti „grubom silom" čini praktički nemogućim. Ovom se šifrom umjesto primjene matematičkih operacija nad svakim slovom ili samim pomicanjem abecede, stvara slučajni niz slova za abecedu šifrata. Sljedeća tablica prikazuje primjer sastavljene abecede ovim principom s nasumično odabranim šifratom,

Tablica 2.4. Abeceda šifrata s nasumično odabranim šifratom

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

Očito je da je od velike važnosti da se svako slovo u abecedi šifrata pojavi jednom i samo jednom kako se nebi narušila jednoznačnost abecede i izbjegla situacija da jedno slovo iz abecede otvorenog teksta ima dva ista slova u abecedi šifrata.

Nakon stvaranja abecede, proces šifriranja poruke jednak je kao kod svih ostalih monoabecednih supstitucijskih šifri. Svako se slovo otvorenog teksta zamjenjuje odgovarajućim slovom iz šifrata.

Iako je moguće napraviti i primjeniti abecedu generiranu slučajnim odabirom redosljeda slova kao u tablici 2.4. U praksi je to nepraktično jer bi pošaljilac i primaoc trebali zapamtiti slučajni niz od 26 slova što nije lak zadatak. Stoga se kao kod većine šifri upotrebljava ključna riječ. Takva se abeceda stvara na način da se na prva mjesta u nizu zapisuje dogovorena ključna riječ, zanemarujući pri tome ponavljajuća slova. Nakon ključne riječi se zapisuju sva ostala ne zauzeta slova po abecednom redu. Primjer takve abecede šifrata nalazi se u tablici 2.5. gdje smo kao ključnu riječ uzeli frazu „MONOALFABETSKA“.

Tablica 2.5. Abeceda šifrata s ključnom riječi „MONOALFABETSKA“

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

Iz gornjeg primjera vidljivo je kako su slova „A“ i „O“ zanemarena nakon prvog pojavljivanja i kako se svako slovo abecede pojavljuje samo jednom. Također je vidljiva i slabost ove metode, pri kraju abecede šifrata, od slova „U“ pa do kraja, slova otvorenog teksta i šifrata su ista. Ovaj problem se pojavljuje kada ključna riječ ne sadrži slova koja se nalaze na zadnjem kraju abecede otvorenog teksta. Kako bi se problem spriječio poželjno je izabrati ključnu riječ koja sadrži slova sa kraja abecede.

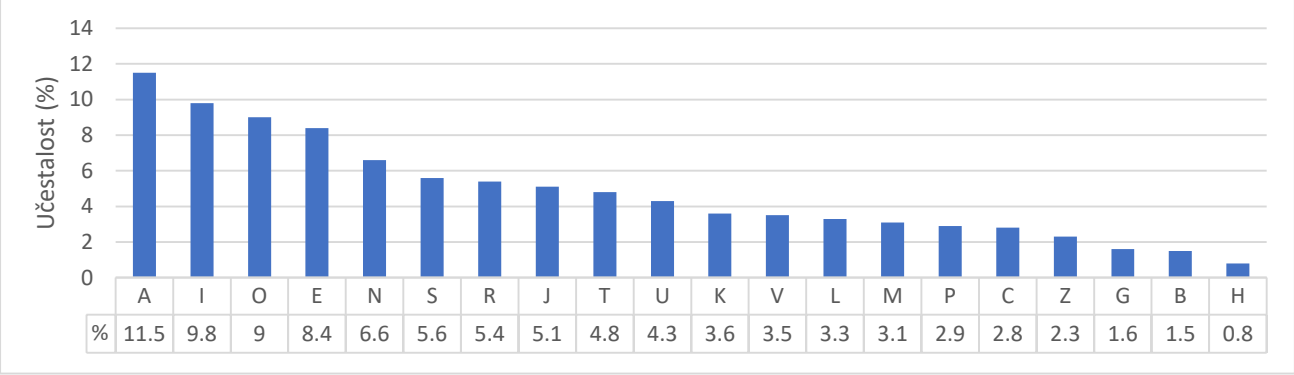
Iako je do sada spomenuta samo ključna riječ za stvaranje abecede, također se može upotrijebiti i cijela rečenica prateći ista pravila kao kod ključne riječi uz dodatno izbacivanje svakog znaka koji nije dio abecede (npr. razmaci, točke i zarezi).

Dešifriranje ovakve šifre slično je procesu šifriranja, prvo je potrebno generirati abecedu šifrata na isti način kao što je to kod procesa šifriranja, nakon toga se slovo šifre mjenja odgovarajućim slovom otvorenog teksta.

Kao što je spomenuto ranije, ponekad je potrebno dešifrirati šifru bez ključa. Dok je to kod Cezarove i afine šifre relativno lako za učiniti, *Mixed Alphabet Cipher* vrsta šifrata zahtjeva drugačiju metodu dekripcije. Dekripcija ovakve metode enkripcije ne vrši se na temelju matematičkih operacija nego se ovdje koriste statistička svojstva jezika na kojem je kod napisan, ova metoda dekripcije naziva se *analiza frekvencija slova*. Analiza frekvencija slova primjenjiva je na sve vrste supstitucijskih šifri, no radi malog broja mogućih ključeva kod Cezarove i afine šifre, nije ju potrebno primjenjivati.

Metodologija iza analize frekvencija slova temelji se na činjenici da na svakom jeziku, svako slovo ima svoju *osobnost*. Najočitija takva osobnost jest učestalost kojim se slovo pojavljuje na nekom jeziku. Jasno je da ce se u hrvatskom jeziku slovo „E“ pojaviti češće nego slovo „P“. Učestalost pojave slova i ostale osobnosti nekog jezika moguće je saznati analizom duljeg teksta, potrebno je samo brojati pojavu svakog slova. Danas se uz pomoć računalnih programa mogu u kratkom vremenu saznati učestalosti slova velikog broja teksta uz pretpostavku da je svaki tekst koji je uzet u obzir na istom jeziku. Ponekad je moguće preskočiti i taj korak, razlog tome jest to da za većinu jezika postoje već baze podataka na internetu sa statističkim informacijama prikupljenih iz tisuće tekstova. Osim podatka o učestalosti pojavljivanja pojedinog slova, za dekripciju analizom frekvencija slova također su od velike koristi podaci o učestalosti pojave bigrama i trigrama određenog jezika.

Za primjer analizirati će se frekvencije slova, bigrama i trigrama u hrvatskom i engleskom jeziku, statistika o hrvatskom jeziku preuzeta je iz knjige „Kriptografija“ autora Andrej Dujella [1] dok su podaci za engleski jezik preuzeti sa internet stranice *Oxford Math Center* [2]. Treba još napomenuti da su se dijakritički zakovi hrvatskog jezika zamjenili s odgovarajućim slovom iz međunarodne abecede. Prvo slijedi analiza hrvatskoga jezika, učestalosti pojedinih slova prikazani su na grafu naslici 2.2.



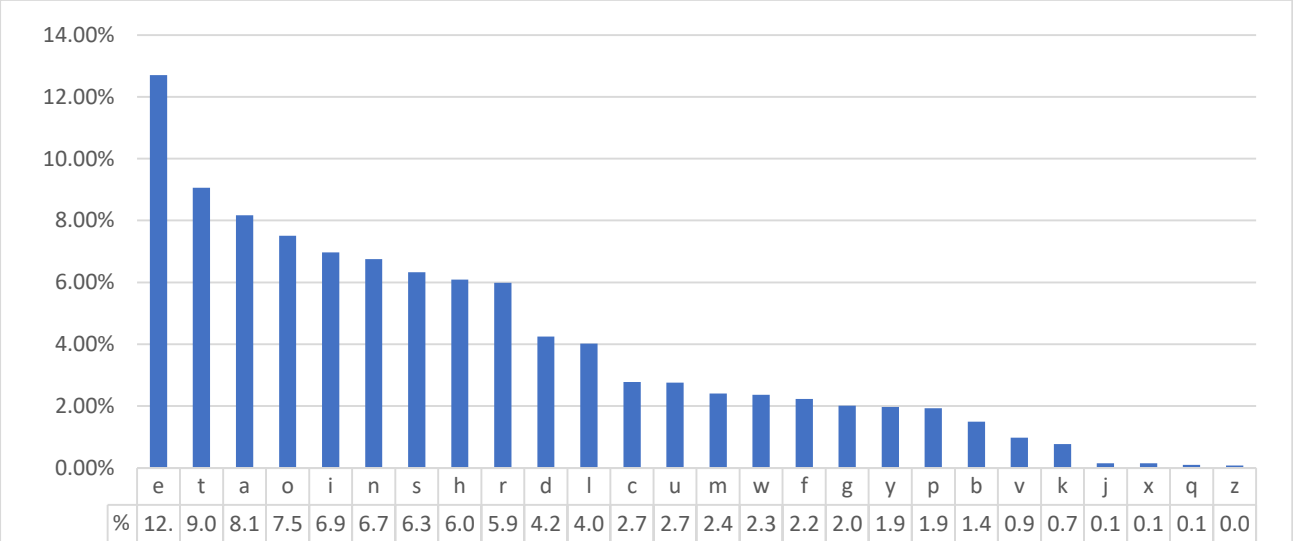
	A	I	O	E	N	S	R	J	T	U	K	V	L	M	P	C	Z	G	B	H
%	11.5	9.8	9	8.4	6.6	5.6	5.4	5.1	4.8	4.3	3.6	3.5	3.3	3.1	2.9	2.8	2.3	1.6	1.5	0.8

Slika 2.2. Učestalost pojedinih slova u hrvatskom jeziku

Iz grafa se da zaključiti da se slovo A najčešće pojavljuje u hrvatskom jeziku s postotkom učestalosti od skoro 12%, iz čega još možemo zaključiti da je otprilike svako deseto slovo, slovo A. Drugo po redu slovo je slovo I s postotkom od 9.8%. Učestalosti ostalih slova vidljivi su u tablici unutar grafa 2.1. Najčešći bigrami, kombinacije dvaju slova su JE (2,7%), NA (1.5%), RA, ST, AN, NI, KO, OS, TI, IJ, NO, EN, PR (1.0%). Od koristi je ovdje uočiti kako su svi bigrami kombinacija suglasnika i samoglasnika ili obrnuto, osim posebnih slučaja „ST“ i „PR“. Najčešći recipročni bigrami, bigrami obrnutog redosljeda u odnosu na prije napomenute su NA i AN s učestalosti od 1.5% i 1.4% te NI i IN s postotcima 1.3% i 0.9%. Jedino su kod ova dva para učestalosti pojave oba bigrama minimalno 0.9%.

Najćešći trigrami, kombinacija triju slova u hrvatskom jeziku su IJE s frekvencijom 0.6% te STA, OST, JED, KOJ, OJE, JEN s frekvencijama između 0.3% i 0.4%. U svrhu kriptoanalize metodom frekvencije slova, informacije o učestalosti pojedinih slova, bigrama i trigrama dovoljne su kako bi se uspješno mogla dekriptirati poruka šifrirana supstitucijskom šifrom.

U slučaju engleskog jezika, najfrekventnija pojedinačna slova nalaze se na grafu na slici 2.3.



Slika 2.3. Učestalost pojedinih slova u engleskom jeziku

Ovdje su najfrekventniji bigrami (poredani po učestalosti): TH, HE, IN, EN, NT, RE, ER, AN, TI, ES, ON, AT, SE, ND, OR, AR, AL, TE, CO, DE, TO, RA, ET, ED, IT, SA, EM i RO.

Dok su najfrekventniji trigrami, trigrami THE, AND, THA, ENT, ING, ION, TIO, FOR, NDE, HAS, NCE, EDT, TIS, OFT, STH i MEN.

Metoda analize frekvencije slova funkcionira, jer ukoliko je slovo A enkriptirano kao X, na svakom mjestu u cijeloj šifri gdje se nalazi X zna se da je ono slovo A, te ga tako možemo zapisati na svakom mjestu u šifri. Ovo se može pretpostaviti ukoliko se slovo „X“ najviše puta pojavi u šifri. Treba imati još na umu da ne mora svaki tekst ili šifra poštovati točnost grafova 2.1. i 2.2; Ukoliko bi se abeceda šifrata sastavila prema učestalosti slova u šifri i informacijama iz grafova, najvjerojatnije bi se dešifriranjem dobio netočni i nečitljivi otvoreni tekst. Točnost kod ovakvih statistika postiže se što duljim tekstovima. Osim pronalaska slova otvorenog teksta putem najučestalijih slova, kriptoanalitičar se može poslužiti i bigramima, trigramima te logikom pronalaska nedostajućih slova iz konteksta teksta već pronađenih slova.

Kao primjer uporabe analize frekvencija slova upotrijebit će se primjer 1.4. iz knjige „Kriptografija“ autora Andrej Dujella [1]. U primjeru potrebno je dekriptirati šifrat dobiven supstitucijskom šifrom ako je poznato da je šifrat napisan na hrvatskom jeziku, tekst glasi

TQCWT QCKIQ RWNOQ OBCEW OQVKB UKAPK OQOQB CQPQA JGDUQ
EQORW TSJGR WEQKY WGTWC JKRBI KZGVO GBQ

Prvi korak jest brojanje učestalosti slova, usput je korisno još izbrojati i bigrame šifrata. Bigrami se broje tako da za svako slovo šifrata napišemo jos njegove sljedbenike. Brojanjem slova i bigrama dane šifre dobija se tablica:

Tablica 2.6. Učestalost pojedinih slova i bigrama

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

Iz tablice se mogu isčitati najfrekventnija slova i bigrami te koliko se puta koji pojavljuje. Brojanjem učestalosti vidiljivo je da je slovo Q najučestalije slovo s 13 ponavljanja, nakon njega slijede slova K, O i W sa 7 ponavljanja. Ostala pojedina slova koja se više od tri puta pojavljuju su B, C, G, R, T, E i J

Od bigrama se najčešće pojavljuju bigrami OQ s 4 ponavljanja, nakon njega bigrami QO i RW s 3 ponavljanja. Ostali učestaliji bigrami koji se mogu uzeti u obzir su BC, EQ, JG, QC, TQ i WT.

Iz razloga što je Q najučestalije slovo, može se pretpostaviti da se kod njega radi o slovu A otvorenog teksta. Odma je moguće pretpostaviti i drugo slovo pomoću bigrama OQ i QO, kombinacija bigrama navodi da je slovo O šifrat za slovo N. Ako se detaljnije analizira treći bigram, RW, može se primjetiti da se slovo R najčešće pojavljuje uz W, isto tako da je W jedno od naučestalijih slova u šifri. Što govori da bi slovo R šifrata moglo biti slovo J otvorenog teksta, a slovo W, slovo E otvorenog teksta. Bigrami OQ, QO i RW sada imaju vrijednost, jedini još neotkriveni česti bigram koji se pojavljuje u hrvatskom jeziku jest ST. Za ST se može pretpostaviti da se krije iza bigrama BC ili JG. Iz razloga što u šifri slova B i C imaju učestalost kao i slova S i T po statistici, pretpostavlja se da je točan bigram, bigram BC. Od najučestalijih slova u hrvatskom jeziku, slova I i O nemaju još svoje značenje. Radi učestalosti slova K i G, isprobat će se pretpostavka da se slovo I krije iza K a iza G da se nalazi O. Radi pregleda zadatka se u sljedećoj tablici nalaze slova abecede otvorenog teksta i slova koja su do sada otkrivena ili pretpostavljena.

Tablica 2.7. Abeceda šifrata

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

Ako se po ovoj abecedi prevede šifrat, dobija se tekst (mala slova su slova otvorenog teksta, velika šifrata)

TateT atiIa jeNna nstEe naVis UiAPi nanas taPaA JoDUa Eanje TSJoj eEaiY eoTet JijsI iZoVn osa

Sad kada je dovoljno elemenata otvorenog teksta otkriveno, iz konteksta se mogu postupno otkrivati ostala slova šifrata. Da se zaključiti da je prva riječ šifrata „matematika“ a zadnja „odnosa“ zapisivanjem otkrivenih slova u abecedu šifrata i ponavljanjem postupka dobijamo otvoreni tekst

Matematika je znanstvena disciplina nastala proučavanjem brojeva i geometrijskih odnosa.

Potpuna abeceda šifrata izgleda ovako:

Tablica 2.8. Abeceda šifrata

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

Pogledom na tablicu uočava se riječ „kriptografija“. Što govori da je šifra enkriptirana Cezarovom šifrom s ključnom riječi. Kod ove metode enkripcije ključ predstavljaju ključna riječ i broj koji označava poziciju od koje se kreće pisati ključna riječ.

Navedena metoda dekripcije funkcionira za svaki šifrat šifriran monoalfabetskom supstitucijskom šifrom i za svaki jezik uz pretpostavku da kriptoanalitičar posjeduje frekvencije slova tog jezika. No i ona ima svoje slabosti. Jedna od slabosti jest da se iz kraćih šifrata ne može prikupiti dovoljno podataka o frekvenciji slova da bi se mogli usporediti s frekvencijom slova tog jezika. Drugi primjer jest ukoliko šifra sadrži tekst s ne uobičajenom temom, lako je moguća pojava riječi s slovima koje se inače ne pojavljuju često. Na primjer, ukoliko se u tekstu radi o zebrama, postoji velika mogućnost da će se slovo Z pojaviti češće nego uobičajeno za taj jezik što bi moglo rezultirati poremećajem analize frekvencija, ali je ovaj problem lako prevladati drugim tehnikama.

Navedeni kriptosustavi, kao najraniji dokumentirani sustavi, danas ne predstavljaju problem za dešifriranje, no za vrijeme aktivne uporabe služile su za prijenos poruka vojskama i diplomatima s instrukcijama za daljnji napredak, stoga je i od velike važnosti bilo da ostaju tajne jer su nerijetko o njima ovisili ishodi ratova i diplomatskih odnosa. S vremenom su opisani kriptosustavi postajali sve jednostavniji i poznatiji, stoga su se vremenom razvijali noviji i sigurniji kriptosustavi, jedna od njih je Vigenèrova šifra čiji opis slijedi u nastavku.

2.2. Vigenèrova šifra

Šifra koja je danas poznata kao Vigenèrova šifra prvi puta je opisana 1533. godine u knjizi *La cifra del. Sig. Giovan Battista Bellaso* autora Giovan Battista Bellaso. Međutim, u 19. stoljeću pogrešno je pripisano francuskom duplomatu Blaise de Vigenère, koji je 1586. godine predstavio sličnu šifru (Vigenèrova šifra s autoključem)[3].

U to vrijeme, ali i mnogo stoljeća nakon njenog izuma, Vigenèrova šifra smatrala se vrlo sigurnom šifrom i dugo se vremena mislilo da se nemože „razbiti“. Radi te misli dobila je i nadimak „le chiffre indéchiffrable“ (fran. Neprobojna šifra). Iako to nije istinito (probijena je 1863. od strane Friedrich Kasiskia), vrlo je sigurna po pitanju metoda za olovku i papir.

Vigenèrova šifra spada pod substitucijske šifre gdje se pri šifriranju slovo otvorenog teksta zamjenjuje slovom dobivenim šifriranjem. No za razliku od prijašnjih substitucijskih šifri (podpoglavlje 2.1), Vigenèrova šifra je *poliabecedna* šifra. Što znači da se svako slovo otvorenog teksta može šifrirati u jedno od *m* mogućih slova (gdje je *m* duljina ključa). Vigenèrova šifra može se opisati i algebarski. Pretvorimo li slova abecede u odgovarajuće brojeve iz $\mathbf{Z}_{26}$ (A = 1, B = 1, ..., Z = 26) i primjenimo modulo 26 operaciju, funkcija za šifriranje može se napisati kao[1]

$$c_i = e_k(x_i) = (x_i + k_i) \bmod 26 \qquad (2\text{-}4)$$

dok se funkcija za dešifriranje može napisati kao

$$x_i = d_k(c_i) = (c_i - k_i) \bmod 26 \qquad (2\text{-}5)$$

gdje su $x = x_1, ..., x_i$ poruka, $c = c_1, ..., c_i$ šifrat a $k = k_1, ..., k_i$ ključ koji je dobiven ponavljanjem ključne riječi.

Iz funkcija se da zaključiti da se svako slovo otvorenog teksta pomiče za *k* mjesta, ovisno o tome gdje se slovo nalazi u tekstu. Kao primjer šifrirati će se tekst „SLAVONIJA I BARANJA“ s ključem „OSIJEK“ duljine m = 6. Pretvaranjem otvorenog teksta i ključa u njihove odgovarajuće numeričke ekvivalente, dobijaju se sljedeći nizovi brojeva;

$$K = (14, 18, 8, 9, 4, 10)$$

$$X = (18, 11, 0, 21, 14, 13, 8, 9, 0, 8, 1, 0, 17, 0, 13, 9, 0)$$

Primjenom enkripcijske funkcije (2-4) dobija se sljedeći šifrat,

	14	18	8	9	4	10	14	18	8	9	4	10	14	18	8	9	4
	18	11	0	21	14	13	8	9	0	8	1	0	17	0	13	9	0
$+_{26}$	6	3	8	4	18	23	22	1	8	17	5	10	5	18	21	18	4

Bez razmaka i pretvoreno nazad u slova, šifrat glasi „GDIESXWBIRFKFSVSE“. Kako ključ i otvoreni tekst nemaju istu duljinu, opća praksa je ponavljati ključ dok se ne dode do kraja teksta. Može se i otvoreni tekst nadopuniti do kraja „bloka“ ključa no to nije nužno iz razloga što se šifriranje svodi na „slovo po slovo“. Iz rezultata se može uočiti kako je prvo N (numerički 13), u šifratu pretvoreno u X dok drugo u V.

Osim matematičke metode za šifriranje i dešifriranje može se koristi i tzv. *Vigenèrov kvadrat*. Postupak šifriranja Vigenèrovim kvadratom odvija se prvo pronalaženjem slova ključa u prvom redu tablice i odgovarajućeg slova otvorenog teksta u prvom stupcu. Mjesto gdje se red i stupac preklapaju označava šifrat. Sljedeća tablica prikazuje Vigenerov kvadrat s postupkom pronalaska prvog šifrata iz prijašnjeg primjera.

Tablica 2.9. Vigenèrov kvadrat

	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

Dešifriranje šifrata uz pomoć kvadrata slično je šifriranju, prvo se traži slovo ključa u prvom retku kvadrata, nakon pronalaska, u istom stupcu se traži slovo šifrata, slovo u prvom stupcu istog retka gdje je slovo šifrata označava njegovo značenje u otvorenom tekstu. Postoje razne varijacije Vigenèrove šifre, jedna od njih je ona s autoključem (engl. autokey cipher). Posebnost ove šifre jest to što otvoreni tekst generira ključ, u početku dogovoreni ključ služi samo za šifriranje prvog bloka otvorenog teksta. Sljedeći primjer prikazuje ovu varijaciju šifriranja, zadatak u primjeru je isti kao i prethodni primjer.

	O	S	I	J	E	K	S	L	A	V	O	N	I	J	A	I	B
	S	L	A	V	O	N	I	J	A	I	B	A	R	A	N	J	A
$+_{26}$	G	D	I	E	S	X	A	U	A	D	P	N	Z	J	N	R	B

Kao rezultat dobijen je šifrat GDIESXAUADPNZJNRB, ako se dobivena šifra usporedi s šifrom iz prošlog primjera, da se primjetiti da su šifrati isti samo u prvih šest slova (kolika je i duljina ključa *m*), nakon ključa šifrat se razlikuje od onog iz prošlog primjera.

Vigenèrova šifra bila je najveći korak u kriptografiji u preko 1000 godina. Ideja mijenjanja slova abecede šifrata pri samoj enkripciji bila je revolucionarna, ideja koja se i danas koristi za poboljšavanje šifrata. Najpoznatiji primjer kodova i šifrata u povijesti, naprava za šifriranje i dešifriranje poruka ENIGMA, svoju logiku temelji na izmjenjenoj metodi poliabecedne supstitucijske šifre.

Jedna od posebnosti Vigenerove šifre i ostalih poliabecednih šifri jest raspodjela frekvencija slova. Kod monoabecednih šifri gdje jedno slovo otvorenog teksta ima jedno i samo jedno slovo u abecedi šifrata, lako je doći do otvorenog teksta frekvencijskom analizom jer unatoč promjeni slova, njegova učestalost ostaje ista. Dok kod poliabecednih šifri jedno slovo otvorenog teksta može poprimiti jedno od *m* mogućih slova iz $\mathbf{Z}_{26}$. Kao dokaz šifrirat će se tekst[3] monoabecednom i poliabecednom substitucijskom šifrom s ključem „SIFRIRANJE“ te prikazati i usporediti frekvenciju slova otvorenog teksta s dobivenim šiframa, tekst glasi (za točnije rezultate su iz teksta pri šifriranju maknuti dijaktirički znakovi):

U dobi od dvadeset šest godina, Vigènere je poslan u Rim na diplomatsku misiju. Ovdje se upoznao sa spisima Albertija, Trithemiusa i Porta, a njegov interes za kriptografijom je planuo. Kriptografija je dugi niz godina bila samo alat koji mu je pomogao u diplomatskom radu, ali u dobi od trideset devet godina, Vigènere je odlučio da je prikupio dovoljno novca kako bi mogao napustiti karijeru i usredotočiti se na život studija. Tek je tada započeo istraživanje novog šifrata.

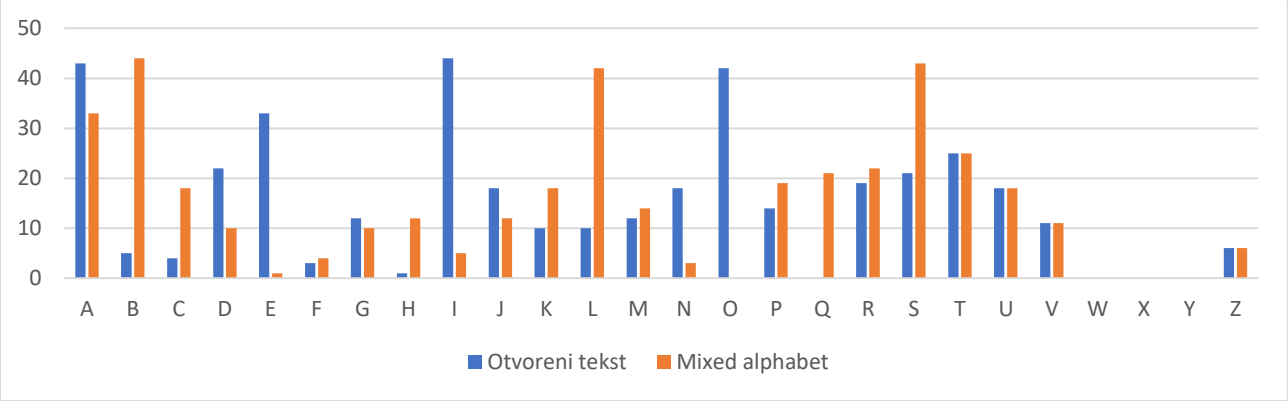
Šifriranjem teksta *Mixed alphabet* metodom šifra glasi,

U RLIB LR RVSRAQAT QAQT JLRBKS, VBJAKAPA CA MLQGSK U PBH KS RBMGLHSTQDU HBQBCU. LVRCA QA UMLZKSL QS QMBQBHS SGIAPTBCS, TPBTEAHBUQS B MLPTS, S KCAJLV BKTAPAQ ZS DPBMTLJPSNBCLH CA MGSKUL. DPBMTLJPSNBCS CA RUJB KBZ JLRBKS IBGS QSHL SGST DLCB HU CA MLHLJSL U RBMGLHSTQDLH PSRU, SGB U RLIB LR TPBRAQAT RAVAT JLRBKS, VBJAKAPA CA LRGUFBL RS CA MPBDUMBL RLVLGCKL KLVFS DSDL IB HLJSL KSMUQTBTB DSPBCAPU B UQPARLTLFBTB QA KS ZBVLT QTURBCS. TAD CA TSRS ZSMLFAL BQTPSZBVSKCA KLVLJ QBNPSTS.

dok sa Vigenerovom šifrom glasi,

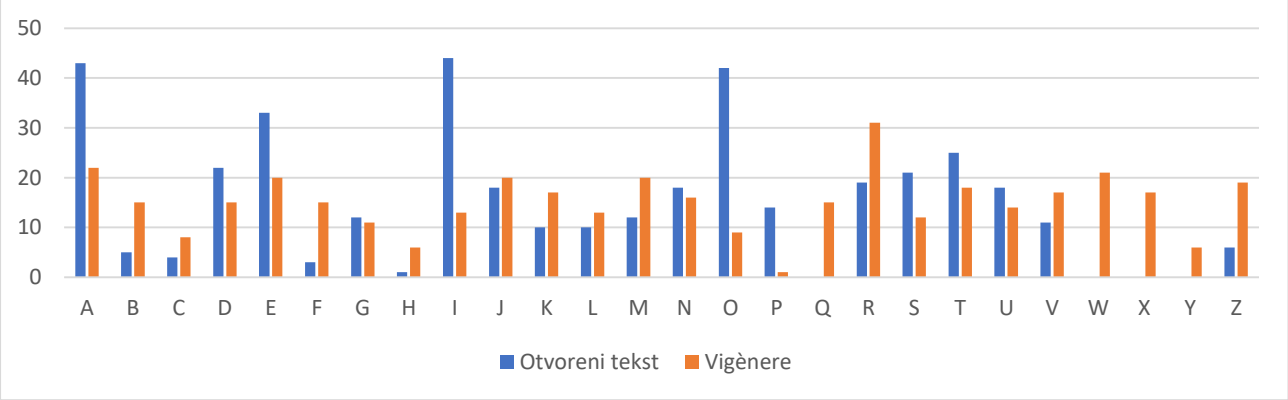
M LTSQ FD QEEVMXVB JEFC KGLNEI, MITNRWZJ AM GOFUEF C WZU EA QRTDWRRBJKH VMKQOL. WMDWN WW CUFHEAB BE KXNJQDA NUFWZYZRR, TERXZMRZCJA V YSJBF, R VAETXZ AVYVZVS MJ OJQUKWXRNOMBWR AM GLNWYG. SWZXKOTAEXQOR RV DHPM FQE XWUIAJ FATF JIDO NUEL STAQ DU WN TGUTXIF U QRTDWRRBJKBV VSLZ, RTZ U QXFA WI KZZDRBIL LJMMK GBMMFI, AZOVNRAI BM TUTLCVX HS RJ GZZKHYMG LTMWCJAX RGDHR SRKB KM EWLRW EACDWLQYZ SRRVSIJC N LAIEQXXGKNKQ JE AJ DADTK AKUQRNS. BJB RV TNME RIUFKVO VBXJIEZDRNWN RGDTX AZFEJXS.

Na oko oba dva šifrata izgledaju nerazumljivo i sigurno, no frekvencijskom analizom može se uočiti razlika, usporedbom frekvencije slova otvorenog teksta i šifrata šifriranog monoabecednom supstitucijskom šifrom dobija se graf 2.3. iz kojeg je vidljivo da su minimumi i maksimumi frekvencija slova isti samo su na drugim mjestima u abecedi.



Slika 2.4. Usporedba frekvencije slova

Vigènerovom šifrom su frekvencije slova drugačije, pogledom na graf 2.4. odma se uoči kako se nakon šifriranja svako slovo pojavljuje barem jednom u šifratu i da su frekvencije slova „izravnatije“ što značajno otežava dekriptiranje analizom frekvencije slova.



Slika 2.5. Usporedba frekvencije slova

Iako je dosta teže, nije nemoguće razbiti Vigènerovu šifru (unatoč njenom nadimku). 1854. godine Charles Babbage je uspio u tome, no njegov rad nije objavljen za vrijeme njegovog života. U otprilike isto vrijeme, 1863. godine Friedrich Kasiski samostalno je razbio Vigènerovu šifru te je objavio svoj rad. Metoda se ne oslanja na poznavanju ključa ili otvorenog teksta te je u njegovu čast nazvana Kasiskijev test. Test se temelji na pronalasku moguće duljine ključa uočavanjem ponavljajućih schemi u šifratu, koje se tada koriste za razdvajanje šifrata u odjele te naknado primjenom analize frekvencija na svaki dio šifrata koji koristi isto slovo iz ključa (ukoliko je poznata duljina ključa poznato je i gdje se nalaze k_1, k_2, ..., k_m). Metoda je moguća radi prirode ponavljanja ključa u Vigènerovoj šifri.

2.3. Playfairova šifra

Playfairovu šifru prvi puta je opisao Charles Wheatstone 1854. godine, te je bila prva bigramska supstitucijska šifra. Naziv je dobila po barunu Playfairu od St Andrewsa, Wtheatstoneovom prijatelju koji je popularizirao šifru za uporabu u vojne svrhe.

Šifra šifrira parove slova (bigrame) umjesto pojedinih slova kao što je slučaj kod jednostavnijih supstitucijskih šifri kao npr. Cezarova šifra. Što je tada predstavljalo napredak supstitucijskih šifri jer se su se umjesto pojedinih slova odma šifrirali blokovi otvorenog teksta čiji rezultat ovisi i o jednom i drugom slovu.

Radi njene sigurnosti, razumno brze uporabe i činjenice da ne zahtjeva posebnu opremu, prisvojena je od strane britanske vojske i postala standardni način enkripcije za vrijeme prvog i drugog svjetskog rata, koristila ju je čak i američka vojska za šifriranje poruka manje važnosti za vrijeme drugog svjetskog rata.

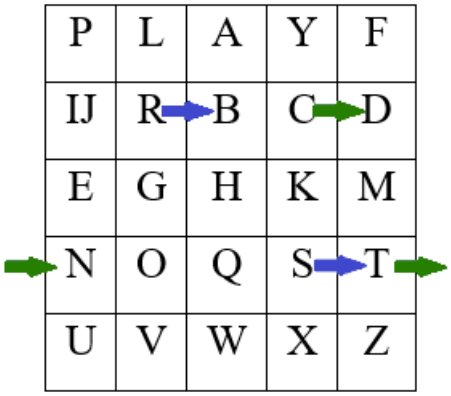
Enkripcija započinje prvo stvaranjem Polibijevog kvadrata, matrice veličine 5 × 5 (brojevi se izostave), obično se to radi uporabom ključne riječi na prvim mjestima te nadopunom ostalim slovima abecednim redom. Kako u matrici ima 25 mjesta za slova dogovor je da se slova „I“ i „J“ poistovjete ili da se iz tablice izbaci slovo Q. U slučaju hrvatskog jezika, ponekad je praksa da se poistovjete slova V i W kako bi se izbjegli nesporazumi kod dešifriranja.

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

Slika 2.6. Polibijev kvadrat s ključnom riječi PLAYFAIR

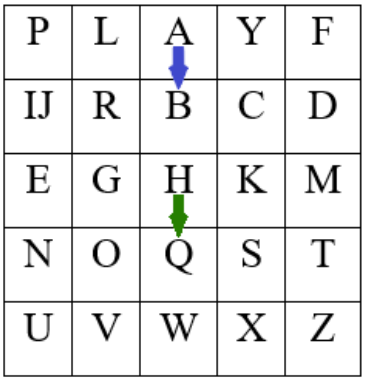
Sljledeći korak je rastavljanje otvorenog teksta u bigrame i primjena jednog od sljedećih pravila na svaki bigram.

1. Ukoliko su oba slova bigrama u istom redu, slova se zamjenjuju s slovima koji su desno od pojedinog slova otvorenog teksta (ukoliko se slovo nalazi na rubu matrice, ciklikči se uzima najlijevije slovo iz istog retka). Dva slučaja pretvorbe prikazani su na slici, pretvorba bigrama RC u BD i ST u TN.



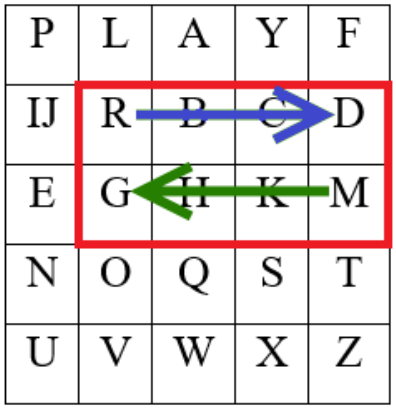
Slika 2.7. Pretvorba bigrama, RC → BD i ST → TN

2. Ukoliko su slova bigrama u istom stupcu, zamjenjuju se slovima koji su ispod pojedinog slova bigrama, ukoliko se slovo nalazi u zadnjem redu vrijedi cikličko pravilo da se uzima slovo iz prvog reda (slično kao kod prvog slučaja).



Slika 2.8. Pretvorba bigrama, AH → BQ

3. Ukoliko se slova bigrama ne nalaze u istom redu ili stupcu, unutar tablice stvaramo kvadrat gdje slova bigrama predstavljaju suprotne kuteve kvadrata, slovo bigrama otvorenog teksta zamjenjujemo s slovom koje se nalazi sa suprotne strane u istom retku kvadrata (kao kod proša dva slučaja, i ovdje vrijedi pravilo cikličnosti).



Slika 2.9. Pretvorba bigrama, RM → DG

Treba još napomenuti da se uvijek kriptiraju parovi slova, ukoliko pri šifriranju ostane jedno slovo bez para, dodaje mu se slovo X na kraj kako bi se cijela poruka šifrirala.

Prema navedenim pravilima enkriptirat će se riječ ELEKTROTEHNIKA s ključnom riječi PLAYFAIR. Prvi korak je razdvajanje riječ na bigrame, tako se dobijaju sljedeći bigrami:

EL EK TR OT EH NI KA

Praćenjem navedenih pravila za pojedine bigrame dobijaju se zamjene:

EL → GP, EK → GM, TR → OD, OT → QN, EH → GK, NI → UE, KA → HY

Rješenje:

GP GM OD QN GK UE HY

Dekripcija je slična procesu enkripcije, razlika je u tome da se u prvom i drugom pravilu obrnu strane zamjene slova, kod drugog pravila se slova zamjenjuju s onima s njihove lijeve i kod trečeg pravila s slovom iznad slova šifrata. Također jos treba iz dešifriranog teksta maknuti sva slova X koja su višak.

Iz razloga što se enkriptiraju parovi slova, dekriptiranje analizom frekvencija slova je znatno teže (bigramsko šifriranje smanjuje na polovicu broj elemenata dostupnih analizi frekvencije). Kod monoabecednih šifri postoji 26 mogućih slova za provjeriti, dok za bigramske imamo 26 × 26 = 676 mogućih parova koje treba analizirati frekvencijski. U slučaju Playfairove šifre broj mogućih parova je nešto manja jer nije moguće šifrirati par koji se sastoji od dva ista slova, nakon oduzimanja takvih parova (ukupno 26) ostaju 650 mogućih parova za analizu. Što je prevelik napor za dektiptiranje „papirom i olovkom“, pomoću računala ipak, moguće je dekriptirati za nekoliko sekundi.

Jedna od korisnih slabosti Playfairove šifre koje se mogu iskoristiti za dekriptiranje je činjenica da isti parovi slova ali obrnuti daju isti par šifrata samo obrnuto, na primjer, ukoliko bigram AN šifriran kao PQ, recipročni bigram NA bit će šifriran kao QP.

Playfairova šifra je za vrijeme prvog i drugog svjetskog rata bila standardni način enkripcije poruka, od korisnika nije zahtjevala znanje matematike ili dodatni uređaj za uporabu te uz činjenicu da je prva bigramska šifra, predstavljala je napredak u smislu jednostavnosti i efikasnosti šifre. Sljedeća šifra je također bigramska (poligramska) no za razliku od Playfairove šifre oslanja se na napredniju uporabu matematike.

2.4. Hillova šifra

Hillova šifra je bigramska šifra izumljena je 1929. godine od strane američkog matematičara Lester S. Hilla. No za razliku od drugih bigramskih šifri, Hillova šifra ima mogućnost rada sa različitim duljinama blokova slova. Što ju tehnički čini poligramskom supstitucijskom šifrom jer radi sa bigramima, trigramima i u teoriji sa proizvoljnom duljinom bloka.

Šifriranje i dešifriranje Hillovom šifrom zahtjeva poznavanje linearne algebre, točnije, zahtjeva da korisnik šifre ima osnovno razumijevanje matrica. Osim linearne algebre koristi se još i modulo aritmetika, zajedno sa potrebom znanja linearne algebre može se zaključiti da je Hill šifra značajno veće matematičke prirode nego druge bigramske šifre. Ta priroda dozvoljava da se šifra (relativno) lako koristi na većim blokovima slova.

Prema [1] Hillova šifra definirana je:

Neka je m fiksan prirodan broj. Neka je $P = C = (\mathbf{Z}_{26})^m$, te

$$K = \{m \times m \text{ invertibilne matrice nad } \mathbf{Z}_{26}\}$$

Za $K \in K$ definiramo

$$e_K(x) = xK, \qquad d_K(x) = yK^{-1}, \qquad (2\text{-}6)$$

Gdje su sve operacije u prstenu $\mathbf{Z}_{26}$.

Prema definiciji šifrirati će se riječ OSIJEK s ključem

$$K = \begin{bmatrix} 1 & 0 & 2 \\ 10 & 20 & 15 \\ 0 & 1 & 2 \end{bmatrix}$$

Prvo je potrebno pretvoriti otvoreni tekst u njegov numerički ekvivalent, pretvorbom dobija se niz (14, 18, 8, 9, 4, 10) koji se dijeli u blokove po 3 broja kako bi se u sljedećem koraku mogli množiti s matricom K. Za kraj potrebno je na rezultat primjenit modulo 26 operaciju i pretvorit rezultat u tekstualni oblik.

$$[14\ 18\ 8]\begin{bmatrix} 1 & 0 & 2 \\ 10 & 20 & 15 \\ 0 & 1 & 2 \end{bmatrix} = [194 \quad 368 \quad 314]\ mod\ 26 = [12 \quad 4 \quad 2] = MEC$$

$$[9\ 4\ 10]\begin{bmatrix} 1 & 0 & 2 \\ 10 & 20 & 15 \\ 0 & 1 & 2 \end{bmatrix} = [49 \quad 90 \quad 98]\ mod\ 26 = [23 \quad 12 \quad 20] = WLT$$

Šifriranjem dobija se šifra MECWLT. Za dešifriranje šifrata potrebno je imati inverz matrice ključa kako bi se mogao dobiti isti tekst nazad. Hill je preporučio uporabu involutornih matrica, matrice gdje vrijedi $K^{-1} = K$. Uporaba takvih matrica olakšava proces šifriranja i dešifriranja ali također smanjuje prostor ključeva što na kraju rezultira manjom sigurnošću šifre.

Posebnost Hillovog kriptosustava jest da sakriva frekvencijske karakteristike slova, za razliku od prijašnje šifre (2.3. Playfairova šifra), Hillov kriptosustav uspješno sakriva i informacije o frekvencijama bigrama. Iz tog razloga šifre šifrirane s matricom ključa veličine od 5 × 5 pa na dalje smatraju sigurnim na napad "samo šifrat".

Hillova šifra ima i svojih mana, radi svoje matematičke prirode, obrtanjem funkcija i matematičkih postupaka, lako je „razbiti" šifru napadom „poznati otvoreni tekst" i „odabrani otvoreni tekst", što je i razlog zašto ova šifra nikad nije doživjela aktivnu uporabu. Još jedna slabost jest činjenica da nemože svaka matrica biti matrica ključa. Kako bi se poruka mogla dešifrirati, prema (2-6) korisniku je potrebna inverzna matrica ključa što nema svaka matrica, odabrana matrica mora imati determinantu koja nije jednaka nuli i koja je relativno prosta s duljinom abecede kako bi mogla služiti kao matrica ključa.

Sljedeći kriptosustav postupkom sliči jednom od prijašnjih spomenutih kriptosustava ali ima bitnu razliku kod stvaranja i uporabe ključa. Radi tih svojstava ključa se šifra smatra (i danas) *savršeno sigurnom* i kriptografskim snom.

2.5. Jednokratna bilježnica

Prvi puta opisana 1882. godine od strane američkog kriptografa Frank Millera, no tek 1917. patentirana, jednokratna bilježnica smatra se kriptografskim snom. Njena posebnost leži u tome da nudi savršenu sigurnost. Pojam *savršene sigurnosti* uveo je 1949. Claude Shannon, utemeljitelj teorije informacija kao znanstvene discipline. Što za kriptosustav znači da ne daje nikakvu informaciju od otvorenom tekstu osim najveće moguće duljine ključa.

Temeljna premisa jednokratne bilježnice jest da je ključ istinski slučajan. Jednostavan način postizanja slučajnosti jest izvlačenje papira sa slovom iz nekakve crne kutije, njegovim dodavanjem u ključ, te vraćanjem u kutiju i ponavljanjem postupka. Važan dio stvaranja ključa kod jednokratne bilježnice jest da ključ mora imati jednaku duljinu kao i otvoreni tekst tj. da se ključ ne ponavlja kao u slučaju Vigenèrove šifre.

S takvim slučajnim slijedom može se uz pomoć tabule recte (Vigenèrove tablice) šifrirati tekst. Sam postupak šifriranja i dešifriranja jednak je Vigenèrovoj šifri, slova otvorenog teksa (ili šifre) i ključa pretvore se odgovarajuće brojevne vrijednosti te se zbrajaju i dijele s duljinom abecede (modulo operacija).

Jakost kriptosustava izlazi na svjetlo tek kad ju se pokuša razbiti. Imali li mi beskonačno računalne snaga za dekripciju, presretnemo li kod FBHQGIB, ona može imati značenje POZDRAV s ključem KNSNLSU, SIGURAN s ključem NHZELSM te čak ABECEDA s ključem VAXMYVZ. Činjenica je da se iz šifrata FBHQGIB može izvući bilo koja riječ s 7 slova odabirajući određeni niz slova iste duljine. Savršena sigurnost se ne oslanja na to da presretač poruke nema mogućnost dobijanja točne poruke, nego da može dobiti beskonačno mnogo mogućih točnih poruka ne znajući koja je od njih prava.

Jednokratna bilježnica je u praksi jedino savršeno sigurna ukoliko se ispune sljedeći kriteriji:

- Ključ mora biti *istinski* slučajan
- Isti ključ se smije samo jednom upotrijebiti

- Ključ mora biti iste duljine kao i otvoreni tekst
- Ključ mora ostati tajan

Neki od ovih kriterija izazivaju logističke probleme za uporabu jednokratne bilježnice. Stvaranje istinski slučajnog ključa nije trivijalan zadatak. Kad računalo stvori *slučajni* niz slova, niz nije istinski slučajan nego *pseudo slučajan*. Što znači da iako na oko izgleda slučajno, drugo računalo može ponavljati proces kako bi dobijalo ponovo isti niz. Slučajan niz najlakše je napraviti uz pomoć crne kutije na način kako je opisan prethodno.

Jednokratna uporaba ključa je lagana za poštovati u teoriji, ali u praktičnoj uporabi to znači da svaki puta kada osoba želi poslati poruku mora stvoriti novi slučajni niz slova (odavdje dolazi djelomično i naziv kriptosustava)

Treči kriterij govori da ključ mora biti iste duljine kao i otvoreni tekst. Nepoštivanjem kriterija, ukoliko je ključ kraći od otvorenog teksta, šifrat se neće moći do kraja dešifrirati ili će korisnik morat ponavljati ključ što smanjuje njenu sigurnost. A ukoliko je duži onda će korisnik morati pamtiti dulji niz slova što je svakako već zahtjevan zadatak. Ukoliko korisnik želi šifrirati čitavu stranicu otvorenog teksta, mora imati i niz slova duljine čitave stranice, što je skoro nemoguće za zapamtiti i što dovodi do zadnjeg problema.

Čuvanje niza nije lak zadatak. Radi njegove slučajnosti i duljine, niz je potrebno negdje zapisati ili pohraniti. Ukoliko korisnik pohrani ključ negdje, mora ga nekako i poslati drugoj osobi u komunikaciji, što predstavlja velik sigurnosni rizik jer bilo tko da dobije niz u ruke, može lagano dešifrirati šifrat. Kako bi se sigurno mogao prenijeti niz, potrebno je imati siguran oblik komunikacije. Te ukoliko on postoji, opet bi lakše bilo poslati otvoreni tekst nego enkriptirati ga prvo. Stoga se u ovom koraku uvijek pojavljuje oblik steganografije.

Radi navedenih razloga, iako je teoretski neslomljiva, jednokratna bilježnica ima svojih mana u praktičnoj uporabi. Ukoliko bi bila upotrebljena, sudionici u komunikaciji bi imali samo dvije kopije bilježnice (odakle i naziv) popunjene nizom slova. Dogovorili bi se osobno kako koristiti bilježnicu (koja stranica, dali je uporaba povezana s nekom informacijom, datumom i slično) i nakon što bi enkriptirali ili dekriptirali poruku, isčupali bi stranicu i zapalili je.

Iako jednokratna bilježnica nudi savršenu sigurnost uz jednostavnu enkripciju i dekripciju, može prouzročiti probleme i propuste za vrijeme prijenosa ključa i uporabe bilježnice što polako dovodi do

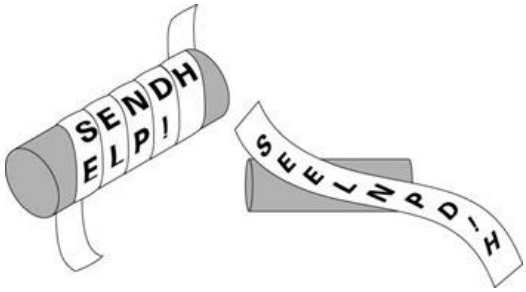
zaključka da ni jedna šifra nije savršena u smislu lake uporabe, visoke efikasnosti te jednostavne terenske uporabe. Sljedeći podnaslov donosi tanspozicijske šifre, za razliku od prijašnjih šifri ova vrsta šifri ostavlja elemente slova iste ali im mjenja mjesto u šifri prema određenim pravilima.

2.6. Transpozicijske šifre

Transpozicijske šifre su nešto drugačije od supstitucijskih šifri. Gdje supstitucijske šifre mjenjaju elemente otvorenog teksta s elementima iz abecede šifrata, transpozicijske šifre mjenjaju raspored elemenata prema danim pravilima (ključ), no sami elementi poruke ostaju isti.

Najjednostavniji primjer transpozicijske šifre jest obrtanje redosljeda slova otvorenog teksta. Tako će rečenica „jednostavan primjer“ postati „rejmirp navatsondej“. Drugi, sličan primjer jest obrtanje redosljeda slova riječi dok same riječi u rečenici ostaju na istom mjestu, na taj način bi prijašni primjer glasio „navatsondej rejmirp“.

Praktični primjer transpozicijske šifre jest skital, naprava za šifriranje i dešifriranje poruka kojeg su koristili stari Grci i Spartanci. Radi se o štapu određene debljine oko kojeg se omotavao papir te dužinski pisala poruka, odmotavanjem papira dobila bi se cedulja s nizom pomješanih slova. Ponovnim omotavanjem cedulje oko skitala jednake debljine omogučavalo je isčitavanje poruke. Dekripcija ovakve poruke je očita, presretač bi samo trebao cedulju omotati oko skitala iste debljine ili isprobati nekoliko skitala različite debljine dok nebi razbio šifru. No važnost skitala danas više leži u činjenici da je jedan od prvih korištenih uređaja u kriptografiji.



Slika 2.6. Skital

Bitna prednost transpozicijskih šifri jest da nisu osjetljive na napad analizu frekvencije slova. To je iz razloga što se simbol slova ne mjenja pri enkripciji, ukoliko otvoreni tekst sadrži 12 slova „a“, šifrat će također sadržavati 12 slova „a“, samo na drugim mjestima.

Jedna od najupotrebljenijih transpozicijskih šifri je tzv. stupčana transpozicija (engl. Columnar transposition ciper). Gdje pošaljilac u pravokutnik po redcima piše otvoreni tekst te ga isčitava po stupcima, redosljed stupaca po kojem se isčitava sadržaj ovisi o ključu.

Prvi korak enkripcije stupčanom transpozicijom jest odabir ključa. Ključ može biti niz brojeva ili riječ. Ukoliko se upotrebljava ključna riječ, redosljed stupaca nakon popunjavanja pravokutnika ide po abecednom redu ključne riječi, ako ključna riječ ima više istih slova, prvo se uzima slovo koje je prvo na redu u ključnoj riječi, zatim drugo i tako dalje. Sljedeće je stvaranje pravokutnika, broj stupaca pravokutnika određuje duljina ključa dok se broj redaka saznaje popunjavanjem pravokutnika otvorenim tekstom. Na vrh svakog stupca se piše element ključa, prema indeksima na vrhovima stupaca se kasnije isti transponiraju. Zatim se upisuje poruka u pravokutnik te isčitava po redosljedu numeričke vrijednosti na vrhu stupca.

Kao primjer šifrirat će se poruka FAKULTET ELEKTROTEHNIKE RACUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA s ključem OSIJEK.

Prateći gore navedene korake dobija se tablica (ukoliko poruka ne popunjava tablicu, na prazna mjesta upisuju se proizvoljna slova koja neće utjecati na sadržaj poruke):

O(14)	S(18)	I(8)	J(9)	E(4)	K(10)
5	6	2	3	1	4
F	A	K	U	L	T
E	T	E	L	E	K
T	R	O	T	E	H
N	I	K	E	R	A
C	U	N	A	R	S
T	V	A	I	I	N
F	O	R	M	A	C
I	J	S	K	I	H
T	E	H	N	O	L
O	G	I	J	A	X

Isčitavanjem po stupcima dobija se šifra (radi čitljivosti šifra je podijeljena u blokove po 10 slova što je jednako broju slova po stupcu):

LEERRIAIOA KEOKNARSHI ULTEAIMKNJ TKHASNCHLX FETNCTFITO ATRIUVOJEG

Dešifriranje ovakve poruke uz pomoć ključa ima 4 koraka. Prvo je potrebno stvoriti tablicu prema dimenzijama koje određuju duljina ključa i duljuna šifre. Nakon toga se upisuju brojevi na vrhove svakog stupca redosljedom od manje vrijednosti prema većoj, popunjava se tablica šifrom te se zamjenjuju stupci kako bi redosljed brojeva na vrhovima stupaca bio isti kao i kod ključa. Zamjenom stupaca dobija se razumljiva poruka.

Unatoč ne osjetljivosti na analizu frekvencija slova, postoje metode koje kriptoanalitičari mogu iskoristiti kako bi slomili presrećenu poruku bez uporabe ključa. Pogledom na šifrat lako je prepoznati da li je poruka šifrirana supstitucijskom ili transpozicijskom šifrom jer kod transpozicijske šifre slova imaju iste frekvencije kao u otvorenom tekstu. Uz pomoć statističkih informacija o bigramima nekog jezika, matematikom i promatranjem šifrata, stupčana transpozicija se može u par koraka dekriptirati.

Sljedeći primjer prikazuje kriptoanalizu šifrata dobivenog stupčanom transpozicijom, primjer je preuzet iz [1]

Potrebno je dekriptirati šifrat

> AAOSA JRTIE EAAAU SCEAE IANPI JAJJN FVSOI RZVPH NIONA RNZIJ INUSS NTRZU TIC

dobiven stupčanom transpozicijom.

Prvi korak pri kriptoanalizi je određivanje veličine pravokutnika, to se radi faktorizacijom broja slova u šifratu. Ukoliko se faktorizacijom dobije više mogućnosti, šifrom se popunjavaju mogući pravokutnici po stupcima nakon čega se promatra odnos samoglasnika i suglasnika u svim redovima. Moguće točan pravokutnik bi trebao imati odnos samoglasnika i suglasnika oko 43% : 57% po retku (ovaj odnos vrijedi za otvoreni tekst pisan na hrvatskom jeziku). Šifrat iz primjera ima ukupno 63 slova te se faktorizacijom dobijaju dvije najvjerojatnije dimenzije, 7×9 i 9×7. Popunjavanjem pravokutnika i računanjem odnosa dobijaju se sljedeća dva pravokutnika.

A	E	A	J	Z	R	S	3:4
A	E	E	J	V	N	N	3:4
O	A	I	N	P	Z	T	3:4
S	A	A	F	H	I	R	3:4
A	A	N	V	N	J	Z	2:5
J	U	P	S	I	I	U	3:4
R	S	I	O	O	N	T	3:4
T	C	J	I	N	U	I	3:4
I	E	A	R	A	S	C	4:3

A	T	U	A	J	R	O	J	T	4:5
A	I	S	N	N	Z	N	I	R	3:6
O	E	C	P	F	V	A	N	Z	3:6
S	E	E	I	V	P	I	U	U	5:4
A	A	A	J	S	H	N	S	T	3:6
J	A	E	A	O	N	Z	S	I	5:4
R	A	I	J	I	I	I	N	C	5:4

Odnos samoglasnika i suglasnika navodi na to da je lijevi pravokutnik (43:57) bolji odabir nego desni (44:56). Stoga se pretpostavlja da je lijevi pravokutnik točan. Ukoliko je pretpostavka netočna, sljedeće korake biti će potrebno ponoviti za drugu moguću dimenziju.

Nakon određivanja dimenzije, dobivene stupce potrebno je još posložiti kako bi se dobio smisleni tekst. Za pravokutnike s manjim brojem stupaca, za ovaj je korak dovoljno isprobavati različite redosljede stupaca dok se ne dobije smisleni tekst. Drugi je način uporaba analize frekvencije bigrama. Za svaki od parova redaka provjerava se dali se bigram njihovih slova nalazi unutar skupa najfrekventnijih bigrama odabranog jezika. Hrvatski jezik ima 36 bigrama koijma je frekvencija veća od 0.8% što je dovoljno da se mogu uzeti u obzir ukoliko se pojave zajedno u šifratu, poredano abecednim redom su to:

AK, AN, AS, AT, AV, CI, DA, ED, EN, IC, IJ, IN, IS, JA, JE, KA, KO, LI, NA, NE,
NI, NO, OD, OJ, OS, OV, PO, PR, RA, RE, RI, ST, TA, TI, VA, ZA

Uspoređivajući kombinacije stupaca s najfrekventnijim bigramima, dobije se da se između 4. i 2. stupca nalazi najveći broj navedenih bigrama, ukupno 6 i s time najveći broj u tablici. Radi toga se može pretpostaviti da su stupci 4 i 2 jedan pored drugoga. Ostale vrijednosti nalaze se u sljedećoj tablici:

	1	2	3	4	5	6	7
1		0	2	2	2	2	4
2	1		1	3	1	1	4
3	2	2		1	0	3	2
4	5	6	5		2	0	0
5	4	2	1	2		1	1
6	5	4	2	2	1		0
7	3	5	4	0	1	1	

Kod pravokutnika ovih dimenzija, podatak da su stupci 4 i 2 jedan pored drugoga, dovoljan je kako bi se isprobavanjem, u kratkom vremenu dobio točan otvoreni tekst. No tablica se može još iskoristiti u svrhu pronalaska korisnih informacija, eliminiranjem parova kojima je frekvencija 0 ili 1 i parova koji su u kontradikciji s pretpostavkom 4 2 (parovi kojima je 4 na prvom mjestu ili 2 na drugom), moguće parove predstavljaju još stupci 7 3, 6 1 i 2 7. Rezimiranjem poznatih informacija dobijaju se mogući ključevi 4273615, 5164273, 6154273.

Preostalo je samo poredanje stupaca prema ključevima, ključem 5 1 6 4 2 7 3 dobija se otvoreni tekst i s time riješenje primjera „ZA RJEŠAVANJE NEPOZNATIH ŠIFARA NAJVAŽNIJI SU UPORNOST, INTUICIJA I SREĆA“.

Druga, slična šifra stupčanoj, tzv. *Route Cipher* je transpozicijska šifra gdje je ključ poznavanje putanje koju treba pratiti dok se čita šifra stvorena iz bloka s otvorenim tekstom. Slično kao kod stupčane transpozicije, stvara se kvadrat u kojeg se otvoreni tekst upisuje po redcima, no za razliku od nje, ovdje se ne iščitava šifrat po stupcima nego prema putanji koju korisnik odabere. Iščitavanje može početi iz gornjeg desnog ugla kvadrata pa pratiti spiralno u smjeru kazaljke na satu, obrnuto od nje i iz drugog ugla ili čak „cik-cak“ putem od gore prema dole. Tako se izraz „Transpozicijska šifra“ može šifrirati na više načina,

T	R	A	N
S	P	O	Z
I	C	I	J
S	K	A	S
I	F	R	A

Počevši iz gornjeg desnog ugla, spiralno u obrnutom smjeru od kazaljke na satu, šifrat iz kvadrata glasit će NARTS ISIFR ASJZO PCKAI.

Dok „cik-cak“ putem od dole prema gore počevši iz donjeg lijevog ugla glasi ISIST RPCKF RAIOA NZJSA.

Za dešifriranje ovakve šifre korisnik treba znati put te dimenzije kvadrata, ukoliko to zna, samo je potrebno da po putu upisuje redom šifru kako bi dobio otvoreni tekst.

Ovim kriptosustavom moguće je lako enkriptiranje poruke u kratkom vremenu ali korisnik treba biti oprezan koji će put odabrati kako bi se spriječili blokovi otvorenog teksta u šifratu i s time odale informacije o dimenzijama korištenog kvadrata. Kod poruka s duljim tekstom i većim kvadratom ovaj problem uobičajeno ne predstavlja opasnost ili prijetnju iz razloga što postoji potencijalno beskonačan broj puteva kroz kvadrat.

Još jedna poznatija transpozicijska šifra je tzv. *Railfence* šifra. Ona radi tako da korisnik preko stranice svoju poruku piše u različite redove te ju nakon toga isčitava po redovima od gore prema dolje, ključ ovdje jest broj redova u koliko je pisan otvoreni tekst. Tako će prijašnji primjer napisano s ključem 3 izgledati

T				S				I				S				I				X
	R		N		P		Z		C		J		K		S		F		A	
		A				O				I				A				R		

Isčitavanjem šifre po redovima dobija se šifra TSISIXRNPZCJKSFAAOIAR.

Dešifriranje ovakve šifre uključuje rekonstrukciju kvadrata koji je upotrebljen za šifriranje. Znajući ključ (broj redova), korisnik može odma početi upisivati šifrat u kvadrat red po red, počevši pisati slova šifre samo u prvi red dok u ostale redove na mjesta slova crtice. Crtice se s vremenom zamjenjuju slovima sve dok se ne ispuni kvadrat (prepiše cijela šifra).

Transpozicijske šifre donose raznovrsnost pri šifriranju, dešifriranju i kriptoanalizi što uz prethodne supstitucijske šifre predstavlja dobar temelj za razumijevanje kriptografskih sustava. Upotrebljene metode kao što su modulo operacija, Polibijev kvadrat, analiza frekvencija i uporaba statističkih informacija o nekom jeziku u svrhu kriptoanalize, uz klasične metode enkcipcije i dekripcije potiču dublje razmijevanje mehanika pojedinih šifri koje i kod praktičnog dijela rada neće biti izostavljene. Sljedeće poglavlje donosi uvod te opće informacije i statistike o operacijskom sustavu Android i razvojnom okruženju Android studio pomoću kojeg je izrađen praktični dio rada.

3. ANDROID

Android je mobilni operativni sustav razvijen od strane Google-a koji se temelji na izmjenjenoj verziji Linux kernela i drugih aplikacija otvorenog koda. Prvenstveno se razvija za uređaje s ekranima osjetljivim na dodir kao što su pametni telefoni i tableti, no s vremenom razvijene su još inačice za televizore, automobile i ručne satove te se još varijacija androida pojavljuju na igraćim konzolama, digitalnim fotoaparatima, računalima pa čak i hladnjacima. Brzu popularnost i uporabu Android zahvaljuje činjenici da je jedan od prvih sustava svoje vrste i da je sam sustav otvorenog koda, što znači da je besplatan za distribuciju i modifikaciju. Otvorenost koda omogućuje tvrtkama prilagodbu sustava svojim uređajima i instalaciju na bezbroj uređaja bez dodatnih troškova za licencu. Prvi android uređaj (HTC Dream) pušten je u prodaju u rujnu 2008. godine te je od tada sustav dobio nekoliko većih nadogradnji s novim mogućnostima, izmjenama postojećih te sigurnosnim zakrpama. Od 2011. godine je android najprodavaniji operativni sustav za mobilne telefone na svijetu te najprodavaniji za teblete od 2013. godine. Od lipnja 2017 evidentirano je da ima mjesečno preko dva trilijuna aktivnih korisnika što je najveća baza korisnika od svih operativnih sustava.

3.1. Povijest androida

Android Inc. osnovan je u Palo Alto u Kaliforniji u listopadu 2003. Projekt android opisan je kao „*ogroman potencijal za razvoj mobilnih uređaja koji će biti svjesni o svojoj lokaciji i postavkama korisnika"*. U početku je namjera tvrtke bila razviti napredni operativni sustav za digitalne fotoaparate no iz razloga što tržište za njihov cilj nije dovoljno veliko bilo, preusmjerili su se prema mobilnim telefonima kako bi konkuriralo Symbianu i Microsoftovom Windows Mobileu.

U srpnju 2005. Google je preuzeo Android Inc. za 50 milijuna dolara zajedno s njegovim ključnim zaposlenicima. Nakon preuzimanja, sve do Prosinca 2006. se nije puno znalo o tajanstvenom projektu Android osim detalja da se radi o programu za mobilne telefone. Tek se u prosincu 2006. pojavio rani prototip telefona koji je sličio BlackBerry telefonu, s cijelom fizičkom QWERTY tipkovnicom i bez ekrana osjetljivog na dodir što pojavom Apple iPhonea 2007. godine nije predstavljalo veliki napredak. Ubrzo nakon toga Google je u specifikaciji Androida napomenuo da će ekrani osjetljivi na dodir biti podržani ali da je produkt dizajniran s prisutnošću fizičke tipkovnice jer ekrani osjetljivi na dodir ne mogu potpuno zamjeniti tipkovnicu. Do 2008. godine su BlackBerry i Nokia već najavili

svoje uređaje kako bi parirali iPhoneu 3G te se na kraju i Androidov fokus usmjerio potpuno na dodirnike. Prvi komercijalno dostupni Android telefon HTC Dream najavljen je 23. rujna 2008.

Slika 3.1. HTC Dream

3.2. Razvoj sustava

Android sustav razvija Google sve dok posljednje promjene i zakrpe nisu spremne za puštanje u javnost, zatim se izvorni kod postavlja na internet stranice Android Open Source Projekta (AOSP) gdje se besplatno može preuzeti. Proizvođači uređaja i komponenti tad preuzimaju sustav kako bi ga oblikovali i prilagodili za pokretanje na svom hardveru. Izvorni kod ne sadrži pokretačke programe za pokretanje određenih hardverskih komponenti što ima za posljedicu da ovisno o proizvođaču zakrpe često znaju kasniti ili ih proizvođač preskoči.

Svaku novu iteraciju Androida Google najavljuje i predstavlja na godišnjoj bazi na svoj konferenciji „Google I/O“ nakon čega bude dostupan zajednici na stranicama AOSP-a. Zakrpe i nove iteracije nakon proizvođačevih prilagodbi budu dostupne korisnicima *over-the-air* (preuzima se i instalira na samom uređaju preko bežične mreže ili mobilnog interneta). Prve inačice 1.0 i 1.1 nisu imali svoja službena kodna imena, iako je inačica 1.1 imala neslužbeno, *Petit four*. Od 2009. s inačicom 1.5, sve do danas, Android dobija svoja kodna imena po različitim vrstama poslastica po abecednom redu. S prvim kodnim imenom *Cupcake* za inačicu 1.5 do posljednje inačice 9.0 s nazivom *Pie*. Sljedeća tablica prikazuje sve inačice androida s kodnim imenima, API inačicama te datumima izdavanja.

Tablica 3.1. Kodna imena, inačice, datum izdavanja te API Androida

Kodno ime	Inačica	Datum izdavanja	API
-	1	23. rujna 2008	1
Petit Four	1.1	9. veljače 2009	2
Cupcake	1.5	27. travnja 2009	3
Donut	1.6	15. rujna 2009	4
Éclair	2.0 – 2.1	26. listopada 2009	5 – 7
Froyo	2.2 – 2.2.3	20. svibnja 2010	8
Gingerbread	2.3 – 2.3.7	6. prosinca, 2010	9 – 10
Honeycomb	3.0 – 3.2.6	22. veljače 2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.4	18. listopada 2011	14 – 15
Jelly Bean	4.1 – 4.3.1	9. srpnja 2012	16 – 18
KitKat	4.4 – 4.4.4	31. listopada 2013	19 – 20
Lollipop	5.0 – 5.1.1	12. studenog 2014	21 – 22
Marshmallow	6.0 – 6.0.1	5. listopada 2015	23
Nougat	7.0 – 7.1.2	22. kolovoza 2016	24 – 25
Oreo	8.0 – 8.1	21. kolovoza 2017	26 – 27
Pie	9	5. kolovoza 2018	28

Iz tablice se vidi trend koji se drži od 2013. da se godišnje na tržište izbaci nova, velika iteracija s novim kodnim imenom. Ali taj trend ima i svoje negativne strane, uz trend, zbog velikog broja uređaja i potrebe za prilagodbom hardverskih i softverskih pokretača, proizvođači često zanemaruju zakrpe za uređaje te ih često izbacuju redovno samo za *flagship* modele (najviše kvalitete, ali i cijene) tokom nekoliko mjeseci od izlaska novog modela. Sljedeće potpoglavlje donosi detalje o raširenosti pojedinih inačica, zastupljenost na svjetskom držištu i druge uspredbe i statističke podatke.

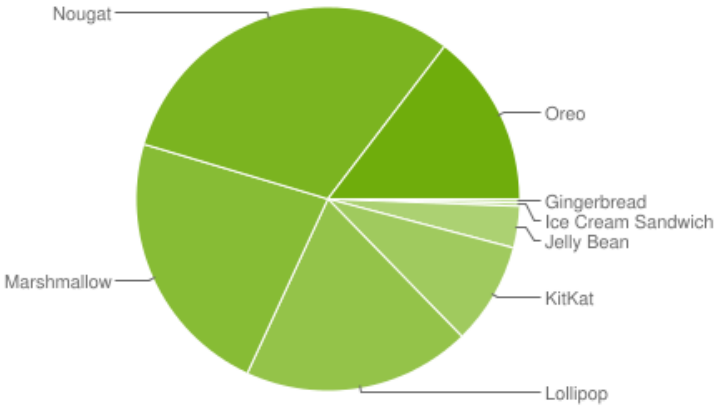
3.3. Statistički podaci

Kao ranije opisano, radi velike količine uređaja na tržištu i problemom s prilagodbom novih inačica došlo je do fragmentacije globalne uporabe Androida, u uporabi su još uređaji koji pokreću inačice stare po 7 godina te najveću zastupljenost nema najnovija inačica nego 3 godine starija (Marshmallow) iteracija. Sljedeća tablica i grafika (3.2) prikazuju u postocima zastupljenost pojedinih inačica. Inačice

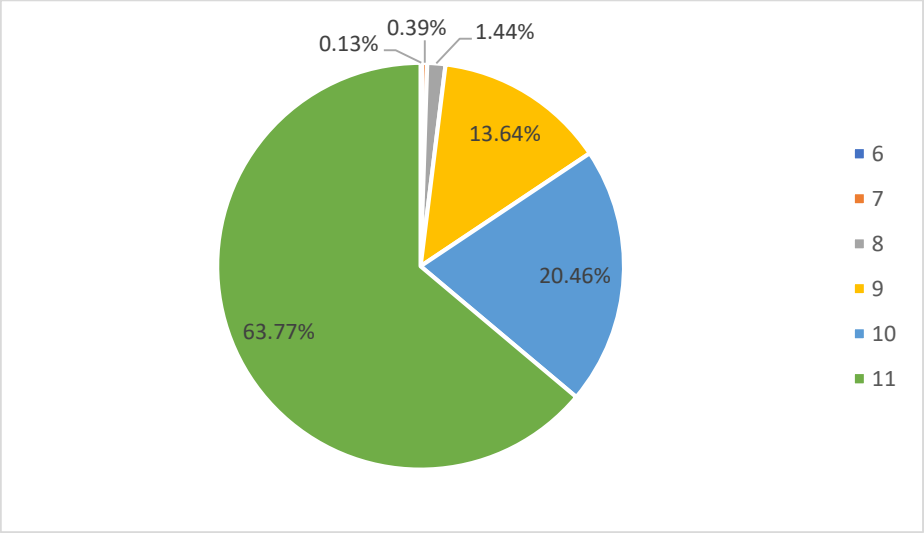
kojima je zastupljenost manja od 0.1% su izostavljene a podaci su prikupljani u periodu trajanja od 7 dana završno s 31. kolovozom 2018[4].

Tablica 3.2. Zastupljenost inačica Androida

Inačica	Kodno ime	Distribution
2.3.3 - 2.3.7	Gingerbread	0.30%
4.0.3 - 4.0.4	Ice Cream Sandwich	0.30%
4.1.x	Jelly Bean	1.20%
4.2.x		1.80%
4.3		0.50%
4.4	KitKat	8.60%
5	Lollipop	3.80%
5.1		15.40%
6	Marshmallow	22.70%
7	Nougat	20.30%
7.1		10.50%
8	Oreo	11.40%
8.1		3.20%

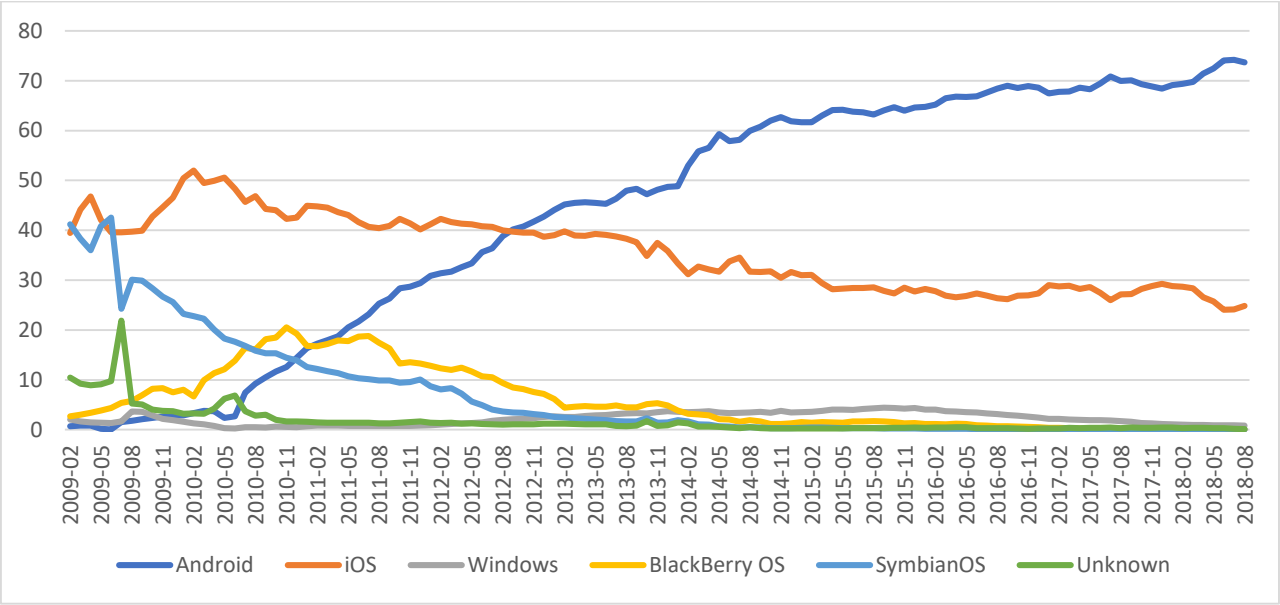


Dok je kod Androidovog suparnika, Apple s operativnim sustavom iOS najzastupljenija najnovija inačica iOS 11 s 63.77%[5].



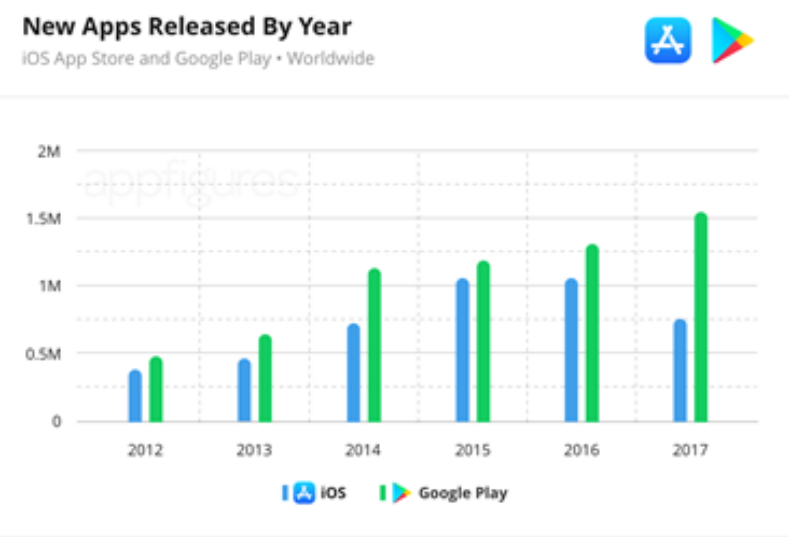
Slika 3.2. Zastupljenost inačica iOS

Razlog toliko velike zastupljenosti jest manji broj modela uređaja koji su uz to još svi uz Apple-ove tvornice što prilagodbu operacijskog sustava uređajima znatno olakšava. No mali broj uređaja također ima svoje posljedice. Gledajući zastupljenost operacijskih sustava na svjetskom tržištu, Android je najzastupljeniji operacijski sustav s 73% na svjetskom tržištu dok je iOS na drugom mjestu s 25%. Zastupljenosti Androida, iOS-a i drugih operacijskih sustava na svjetskom tržištu prikazane su na grafu 3.3. kroz vremensku liniju od veljače 2009 kad je izašao Android 1.1 do kolovoza 2018[6].



Slika 3.3. Zastupljenost mobilnih operacijskih sustava na svjetskom tržištu

Još jedna brojka koja korisnicima bitna jest broj aplikacija u trgovini, Google-ov Play store broji otprilike 3.8 milijuna aplikacija a Appleov App Store oko 2.2 milijuna. No ti brojevi nisu najbolji za uzeti u obzir iz razloga što su skoro sve popularne aplikacije dostupne za oba operacijska sustava. Poznato je da je Appleov App Store lukrativnija trgovina stoga postoji tendencija da nove aplikacije budu dostupne prvo za iOS pa tek kasnije za Android, no radi kontinuiranog rasta zastupljenosti Androida na svjetskom tržištu taj trend nestaje. Preko godina su obje platforme imale stabilan rast broja novih aplikacija godišnje, no prvi puta se 2017. godine dogodilo da je AppStore imao manji broj novih aplikacija nego nekoliko godina prije dok je broj novih aplikacija na Play Storeu cvjetao[7]. Usporedba brojeva novih aplikacija godišnje nalaze se na slici 3.2. Brz rast broja novih aplikacija Android nažalost zahvaljuje svom velikom broju *besplatnih* i loše napravljenih aplikacija te pronalazak dobrih i kvalitetnih postaje sve teži.



Slika 3.4. Rast broja novih aplikacija godišnje[7]

Appleov AppStore je moža lukrativnija trgovina za razvijače, no pristup njoj je skuplji. Cijena pristupa razvijačima na mogućnost izdavanja aplikacija na AppStoreu iznosi $99 godišnje dok za Googleov Play Store korisnik plaća jednokratni iznos od $25. Osim službenih trgovina postoje još i razne alternativne trgovine koje nemaju pretplatničku cijenu za izdavanje. Zarada od prodaje aplikacije ne dolazi razvijaču odmah na račun, mnoge trgovine aplikacija ostvaruju veći dio prihoda od plaćenih aplikacija stoga ga još zadržavaju neko vrijeme. Googleov Playstore je po tom pitanju najbolji. Zarada razvijaču dolazi na račun par dana nakon što mjesec završi i ne postoji minimalni iznos koji je potreban za podizanje na bankovni račun. Appleov AppStore za usporedbu ima minimalni iznos od $10 ukoliko je valuta gdje je pretplatnik podržana od strane Applea. Ukoliko nije, minimalna zarada iznosi $150. Isplata se vrši unutar 45 dana od završetka mjeseca. Alternativne trgovine aplikacija svoje pretplatnike isplaćuju oko 30 dana nakon što mjesec završi uz minimalni iznos zarade. Nekim trgovinama taj iznos zna biti i do $250 što je problem manjim razvijačima[8].

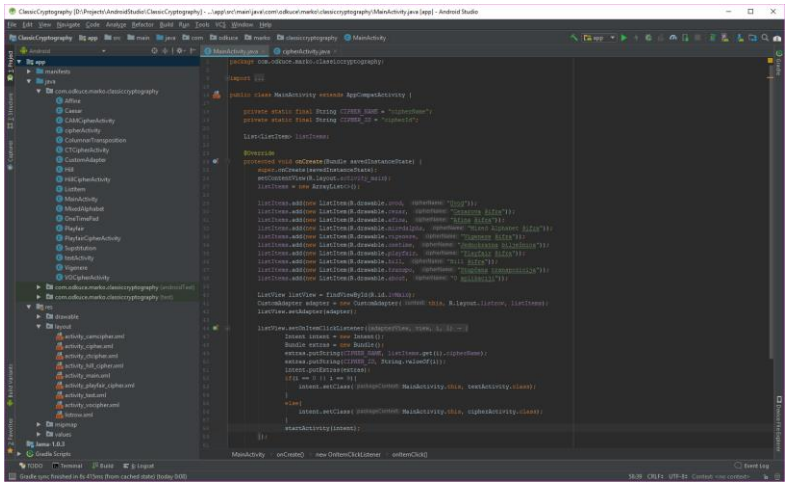
Zaključuje se da je oba operacijska sustava i tvrtke iza njih imaju svoje prednosti i mane. Apple sa svojim mobilnim uređajima, operacijskim sustavom i trgovinom razvijaču nudi ekskluzivnu i lukrativnu platformu za zaradu. Android sa svojim velikim iznosom fragmentacije distribucija predstavlja manje stabilnu platformu no porastom popularnosti i zastupljenošću na svjetskom tržištu uz mali startni kapital i redovnu isplatu postaje sve veći favorit malim razvijačima te početnicima. Sljedeće poglavlje se radi o razvoju aplikacije „Classic Cryptography“ koja predstavlja praktični dio ovog diplomskog rada.

4. RAZVOJ APLIKACIJE „CLASSIC CRYPTOGRAPHY“

Praktični dio diplomskog rada izrađen je u Androidovom službenom razvojnom sučelju Android Studio koji se temelji na integriranom razvojnom sučelju Intelij IDEA. Okruženje nudi mnogobrojne funkcionalnosti kao što su

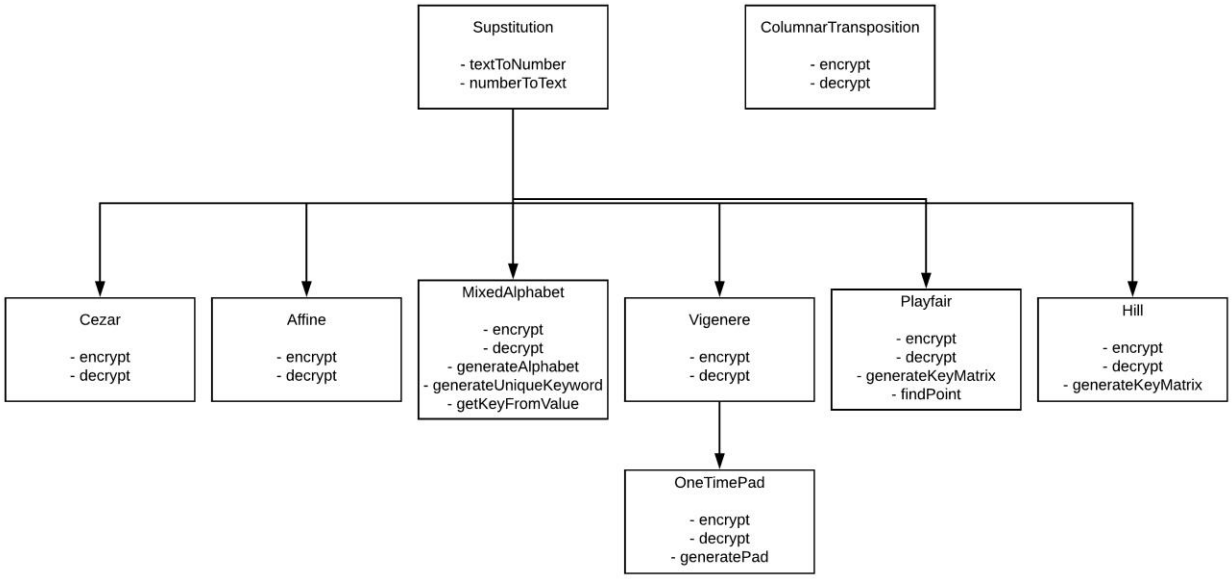
- Podrška za Java i Kotlin programski jezik i XML jezik za uređivanje
- Android emulator za isprobavanje projekta
- Jedinstveno razvojno okruženje za sve Android uređaje (mobilne telefone, satove, televiore...)
- Integracija programa za verzioniranje koda unutar sučelja (GitHub i BitBucket)
- Debuger za ispravljanje sintatičkih, semantičkih i logičkih pogrešaka te kontrolu toka naredbi

Preuzimanje razvojnog okruženja moguće je na službenoj stranici bez naknade[9]. Za razvoj aplikacija Android Studio zahtijeva Java Developement Kit, skup alata za razvoj Java aplikacija kojeg automatski preuzima i instalira ukoliko već ne postoji na računalu korisnika. Nakon početnih postavki korisnik može stvoriti novi projekt, definira ime projekta, svoju domenu te podršku za C++ i Kotlin programski jezik. Sljedeći korak je definiranje minimalne verzije Androida koju aplikacija zahtjeva (novije verzije nude i novije mogućnosti) i vrste uređaja za koje je aplikacija namjenjena. Zadnji korak u procesu stvaranja projekta je odabir izgleda početne *Aktivnosti* – dizajna zaslona. Potvrdom razvojno okruženje priprema početne datoteke, direktorije i resurse za početak rada. Korisniku se tako generira osnovna *Hello World* aplikacija koja je odmah pokretljiva na uređaju. Izgled razvojnog okruženja prikazano je na slici 4.1.



Slika 4.1. Android Studio

Svaka aktivnost u aplikaciji sastoji se od dva dijela, njegovog izgleda koji se izrađuje u XML jeziku za naznačavanje i pozadinske Java klase u kojoj se definira ponašanje elemenata aktivnosti i same aktivnosti. U aktivnosti se mogu implementirati dodatne klase koje mogu sadržavati metode za obradu podataka, tako je i u ovom projektu za svaki kriptosustav stvorena posebna klasa u kojima su definirane metode za šifriranje, dešifriranje i dodatne međukorake. Sljedeća slika prikazuje dijagram klasa kriptosustava u aplikaciji.



Slika 4.2. Dijagram klasa

Dijagram prikazuje implementirane kriptosustave u aplikaciji te metode koja svaka klasa sadrži. Radi zajedničkih svojstava, većina supstitucijskih kriptosustava nasljeđuje klasu *Supstitution* koja sadrži metode *textToNumber* i *numberToText*. Metode služe za pretvorbu slova u njihove ekvivalentne numeričke vrijednosti i obrnuto. Klasa ColumnarTransposition kao transpozicijska šifra ne zahtjeva te metode. Poneki kriptosustavi osim metoda za šifriranje i dešifriranje sadrže metode koje služe kao međukoraci. Tako na primjer klasa MixedAlphabet sadrži dodatne metode za generiranje abecede šifrata i izbacivanje dupliciranih slova u ključu.

Postupci šifriranja i dešifriranja razvijeni prema stvarnom odvijanju procesa. Sljedeća potpoglavlja opisuju pojedine klase i načine na koji obrađuju upisani sadržaj, napisani kod unutar klasa nalazi se u prilogu.

4.1. Supstitution.java

Prva klasa sadrži metode *textToNumber* i *numberToText*. Metoda *textToNumber* kao parametar prima znakovni niz kojeg je potrebno pretvoriti u numerički ekvivalent. Numerička vrijednost izračuna se tako da se slovu A oduzme slovo za koje se traži vrijednost. Kao rezultat dobija se numerička vrijednost s negativnim predznakom. Dobivenom rezultatu se mijenja predznak i ubacuje u listu koju funkcija vraća nakon što obradi sva slova u nizu.

NumberToText kao parametar prima listu (niz) brojeva nad kojim računa ostatak dijeljenja s 26 svih njegovih elemenata, rezultat jednak je poziciji slova unutar znakovnog niza *alphabet* s kojim se broj treba zamjeniti, kao rezultat metoda vraća znakovni niz.

4.2. Caesar.java

Za realizaciju Cezarove šifre potrebne su bile samo dvije metode, *encrypt* i *decrypt*. Obje metode kao parameter primaju znakovni niz koji treba pretvoriti i numeričku vrijednost kao ključ. Metode prvo pretvaraju znakovni niz u numerički preko nasljeđene metode, zatim se svakom dobivenom elementu poveća ili smanji vrijednost za iznos ključa (ovisno da li se radi o šifriranju ili dešifriranju) nakon čega se lista pretvara nazad u znakovni niz što ujedno metoda vraća kao rezultat.

4.3. Affine.java

Klasa Affine.java osim metoda za šifriranje i dešifriranje sadrži mapu čiji su elementi brojevi relativno prosti brojem 26 i njihovi inverzi u obliku para ključ-vrijednost.

Metode za obradu ovdje primaju kao argument znakovni niz za pretvorbu te dvije numeričke vrijednosti koje predstavljaju varijable a i b u afinoj funkciji (potpoglavlje 2.1. formula (2-2)). Metode prvo provjeravaju dali je vrijednost *a* unutar mape relativno prostih brojeva, ukoliko jest, znakovni niz se slovo po slovo pretvara prema (2-2) ovisno o metodi te dodaje u numerički niz, za kraj se niz pretvori nazad u tekstualni oblik kojeg funkcija vraća kao rezultat. Ukoliko broj *a* nije unutar mape s relativno prostim brojevima, funkcija vraća znakovni niz „ERROR“ i prekida izvršavanje.

4.4. MixedAlphabet.java

MixedAlphabet klasa sadrži najviše metoda od svih klasa kriptosustava u projektu. Općenito se kod Mixed alphabet šifre abeceda može složiti prema nasumičnom izboru te uz pomoć ključa koji može

započeti na bilo kojem mjestu unutar abecede. Metoda *generateUniqueKeyword* ulaznom znakovnom nizu provjerava dali sadrži duplicirana slova te ih izbacuje iz niza. Ukoliko se slovo prvi puta pojavljuje dodaje ga u poseban znakovni niz kojeg kao listu vraća u glavni program.

Metoda *generateAlphabet* generira ukupnu abecedu, ovisno o korisnikovom odabiru metoda slaže slova po korisnikovom redosljedu ili prema ključnoj riječi s definiranim početkom. Ukoliko se slaže prema korisnikovom odabiru funkcija vraća abecedu kako ju je definirao. U slučaju ključne riječi metoda putem ranije opisane metode izbacuje duplikate iz riječi te ih kreće ubacivati u konačnu abecedu prema redosljedu nakon čega slovima koji nemaju svoj šifrat pridodijeljuje ostala ne zauzeta slova abecede.

Metode za preoblikovanje unosa prema odabiru korisnika prvo stvaraju abecedu te nakon toga zamjenjuju slova otvorenog teksta s slovom iz abecede šifrata. Pri procesu zamjene slova metoda *getKeyFromValue* pronalazi slovo za koje se treba element ulaznog teksta zamijeniti.

4.5. Vigenere.java

Ova klasa se sastoji samo od metoda za šifriranje i dešifriranje. Metode kao parametre primaju tekst koji treba obraditi, ključ te brojčanu vrijednost kojom se definira da li se radi o Vigenerovoj šifri gdje se ključna riječ pri pretvorbi ponavlja ili se radi o autoključu. U početku se kod obje metode prvo ulazna riječ i ključ pretvaraju u numeričke vrijednosti. Nakon toga se o korisnikovom odabiru izvršava jedan od bloka kodova. Za ponavljajući ključ program nakon prevođenja s zadnjim elementom ključa postavlja pokazivač nazad na prvi element ključa dok za *autokey* nakon zadnjeg elementa postavlja pokazivač na prvi element unesenog otvorenog teksta. Sama pretvorba odvija se zbrajanjem numeričkih vrijednosti ulaznog niza i ključa za šifriranje a oduzimanjem za dešifriranje. Rezultat se sprema u listu koja se na kraju šifriranja pretvara u znakovni niz.

4.6. OneTimePad.java

Postupak kriptiranja jednokratnom bilježnicom jednak je postupku u Vigenerovoj šifri, stoga OneTimePad nasljeđuje Vigenere.java klasu kako bi se mogle preuzeti metode za preoblikovanje. Razlika između Jednokratne bilježnice i Vigenerove šifre leži u ključu, što je u ovoj klasi dodatno implementirano. metoda *generatePad* generira slučajni niz slova duljine jednake kao i tekst koji se preoblikuje. Iz poglavlja 2.5. poznato je da se putem računala nemože na siguran način stvoriti slučajan niz slova koji se neće određenim metodama moći reproducirati. Posebnost ovdje jest da se

pri stvaranju niza koristi u Javu ugrađena klasa *SecureRandom* koja proizvodi kriptografski sigurne pseudo-slučajne brojeve koji se mogu uporabiti u sigurnosno osjetljivim aplikacijama[10]. Pretvorba iz tekstualnog u numerički oblik i obrnuto izvodi se preko nasljeđene *encrypt* i *decrypt* metode.

4.7. Playfair.java

Playfair klasa sadrži četiri metode, *generateKeyMatrix* koja stvara tablicu šifrata, *findPoint* koja vraća objekt klase *Point* koja sadrži x i y koordinatu za pronalazak traženog slova u tablici te metode za šifriranje.

Metoda za šifriranje prvo stvara tablicu šifrata putem *generateKeyMatrix* metode nakon koje računa broj bigrama, sljedeće što radi jest provjerava dali se neki od bigrama sastoji od dva ista slova te ukoliko je to istinito rastavlja ih i ispred prvog slova bigrama ubacuje slovo X. Prije zamjene elemenata se još provjerava dali se preuređeni otvoreni tekst sastoji od neparnog broja slova te se na kraj dodaje slovo X ukoliko je to slučaj. Zatim se za svako slovo pojedinog bigrama pronalazi koordinata unutar kvadrata *findPoint* metodom. Odnosi koordinate se tad uspoređuju prema pravilima u poglavlju 2.3 te primjenjuje zamjena.

Metoda za dešifriranje je nešto jednostavnija, prvo se kao kod šifriranja stvara tablica šifrata nakon čega se ulazni niz (šifra) rastavlja na bigrame kojima se metodom *findPoint* pronalaze koordinate pojedinog slova. Odnos koordinati se prema pravilima uspoređuje te primjenjuje određena zamjena. Bigram se nakon zamjene sprema u znakovni niz koji funkcija nakon prolaska kroz cijeli ulazni tekst vraća kao rezultat.

4.8. Hill.java

Hill šifra se dosta oslanja na linearnu algebru stoga je ovdje bilo potrebno iskoristi *Jama* (engl. A Java Matrix Package) biblioteku za operacije nad matricama (množenje, inverz i determinanta)[11].

Enkripcija započinje pretvorbom ulaznog teksta i ključa u numerički oblik, zatim se iz dobivenog numeričkog oblika ključa stvara matrica ključa K *generateKeyMatrix* metodom. U praktičnom radu je veličina matrice za ključ K fiksiran na dimenziju 3 × 3 stoga će se i ulazni tekst obrađivat u trigramima. Svakim korakom petlje se varijablama plainText pridodijeljuju elementi trigrama iz kojih se svakim prolazom kroz petlju stvara nova matrica P. Matrice P i K se zatim pomnože što rezultira

novom matricom umnoška R. Elementi matrice R se redom isčitavaju te dodavaju u niz koji se na kraju metode pretvara u tekstualni oblik.

Decrypt metoda identična je metodi za šifriranje izuzev dvije razlike. Prva razlika jest da se odmah nakon deklarizacije i inicijalizacije matrice K računa njen inverz, te je druga razlika da se nakon množenja matrice C i K, elementi matrice R zaokruživaju s obzirom na to dali im je predznak negativan ili ne. Ovaj korak je potreban iz razloga što Java operatorom „%“ računa ostatak dijeljenja umjesto modulo operacije, što u slučaju negativnog broja može dati ne tražen rezultat.

4.1. ColumnarTransposition.java

Klasa stupčane transpozicije sastoji se samo od metoda za šifriranje i dešifriranje.

Metoda encrypt na osnovu duljine ključa određuje broj stupaca kvadrata i na osnovu broja stupaca i duljine ulaznog teksta broj redaka kvadrata. Nakon računanja dimenzija se na osnovu dobivenih vrijednosti deklarira dvodimenzionalni niz unutar kojeg se upisuje ulazni tekst (ukoliko ostane praznih mjesta u kvadratu, oni se popunjuju slovom X). Elementi dobivene tablice se tako razdvajaju po stupcima te preraspoređuju prema ključu. Preraspodiljenu matricu metoda isčitava po redovima te elemente dodaje u znakovni niz kojeg metoda vraća kao rezultat.

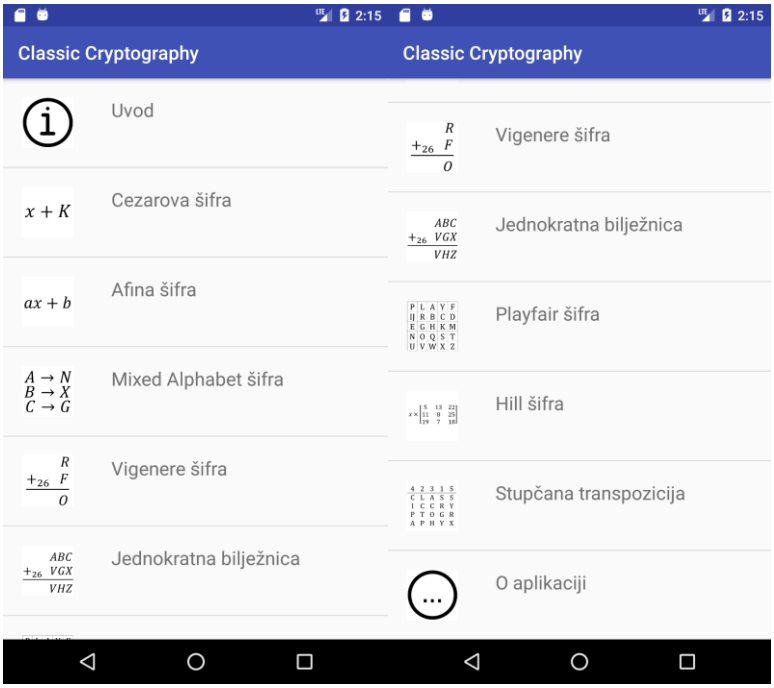
Dešifriranje se odvija slično kao šifriranje, na osnovu ulaznih parametara se stvara nova tablica unutar koje se upisuje ulazna riječ po stupcima, stupci se zatim u sljedećoj petlji preraspoređuju prema ključu te opet isčitavaju po redovima kako bi se dobio otvoreni tekst.

Stvaranjem klasa i metoda za svaki kriptosustav olakšava njihovu daljnju implementaciju u aplikaciju. Dovoljno je u pozadinskoj .java datoteci neke aktivnosti stvoriti novi objekt pomoću klase odabranog kriptosustava te ulazne parametre i rezultat koje metode vraćaju povezati s elementima korisničkog sučelja kako bi bili vidljivi korisniku i kako bi mogao upisivati svoje parametre za šifriranje.

Nadolazeće poglavlje daje pogled na završenu aplikaciju te opisuje navigaciju kroz istu.

5. CLASSIC CRYPTOGRAPHY

Stvaranjem korisničkog sučelja i njegovog spajanja s pozadinskim metodama kriptografskih sustava, korisnik može putem izbornika odabrati kriptosustav koji ga zanima te pročitati detalje o odabranom sustavu. Izbornik kojim korisnik navigira do pojedinih sustava prikazan je na slici 5.1.

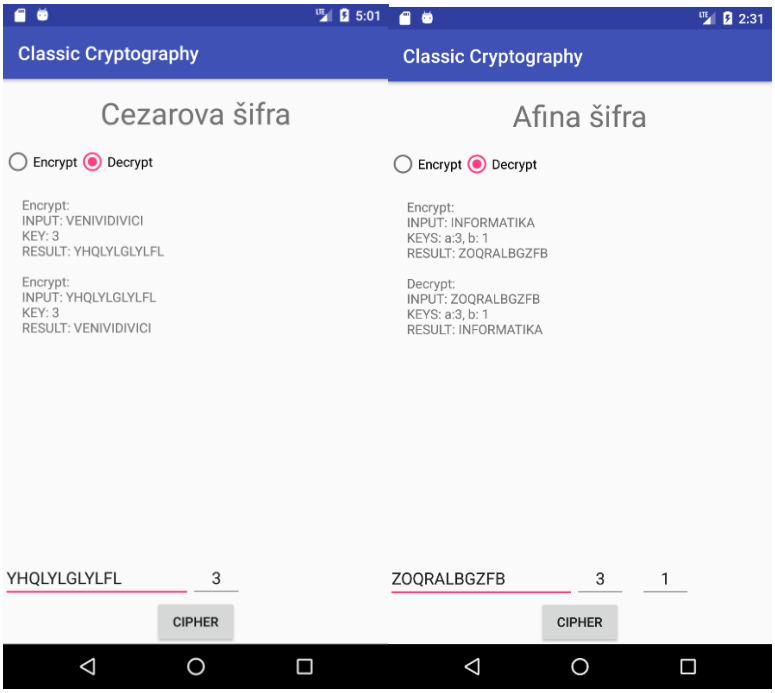


Slika 5.1. Izbornik aplikacije

Prva stavka izbornika *Uvod* daje osnovne informacije o kriptografiji i kriptoanalizi te pojašnjava termine koji se spominju dalje u opisima kriptosustava. Pritiskom na temu, korisniku se otvara novi prozor s informacijama o odabranom sustavu te na dnu prozora ima tipku koja ga dalje vodi do alata za šifriranje i dešifriranje. Alati za svaki sustav su posebno stvoreni tako da ovisno o kriptosustavu korisnik može osim otvorenog teksta definirati i parametre ključa karakteristične za odabrani sustav. Ukoliko kriptosustav kao parametar ne zahtjeva samo ključnu riječ ili broj, pored polja za unos ubačena je tipka koja korisnika vodi do izbornika gdje ih može definirati.

Cezarova šifra od korisnika zahtjeva tekst koji treba pretvoriti i brojevnu vrijednost za pomak slova kao ključ. Odabirom načina pretvorbe (šifriranje ili dešifriranje), upisom teksta i ključa, korisnik pritiskom na tipku *Cipher* šalje parametre u metodu koja na zaslon vraća upisane podatke i rezultat šifriranja.

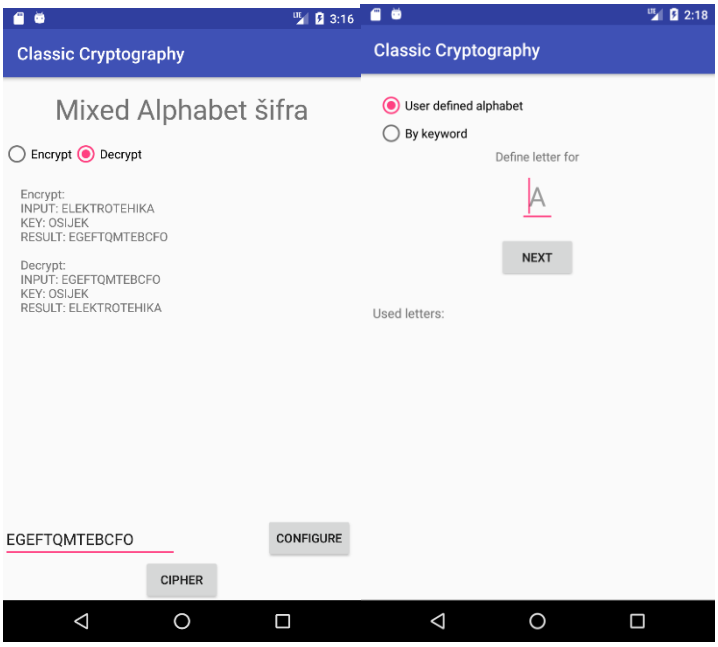
Kod Afine šifre korisnik ima iste mogućnosti kao i kod Cezarove šifre, uz dodatak da je umjesto jednog parametra kao ključ potrebno upisati dva, varijable a i b iz (2-2). Izgled alata za pretvorbu teksta Cezarovom i Afinom šifrom vidljivi su na slici 5.2.



Slika 5.2. a) Pretvorba Cezarovom šifrom b) Pretvorba Afinom šifrom

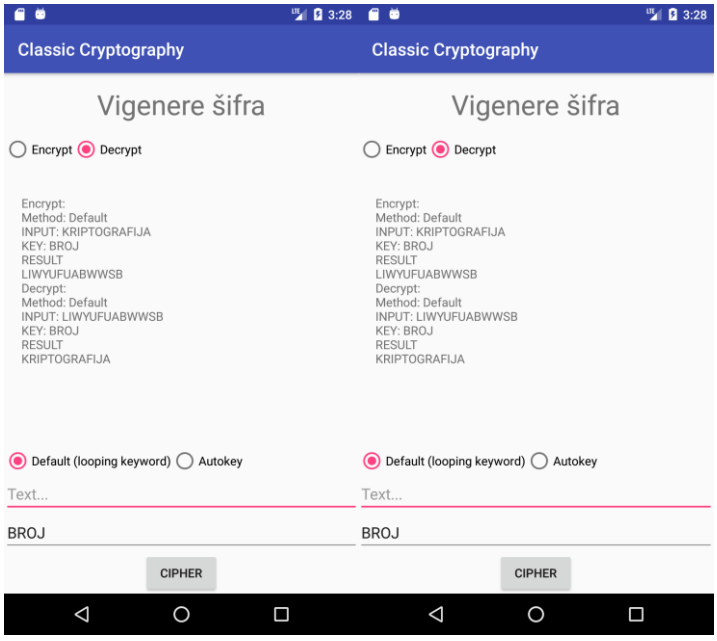
Mixed Alphabet šifra prije početka šifriranja zahtjeva od korisnika da definira abecedu putem koje će se pretvarati upisani tekst. U tu svrhu je pored polja za upis tipka koja vodi do postavki gdje se abeceda može definirati na dva načina, slovo po slovo ili putem ključne riječi. Prvi način traži od korisnika da za slovo koje se nalazi u polju za upis definira slovo abecede šifrata, dok drugi cijelu ključnu riječ, iz ključne riječi se nakon potvrde brišu duplicirana slova te stavlja na početak abecede (ostatak abecede se puni prema abecednom redu). Izbornik gdje korisnik definira svoje podatke prikazan je na slici 5.2.b.

Nakon definiranja abecede, aplikacija se vraća nazad u izbornik za pretvorbu gdje korisnik pritiskom na tipku *Cipher* unesene parametre šalje u metodu koja će vratiti zajedno sa upisanim parametrima rezultat te prikazati na zaslonu zajedno sa rezultatima prethodnih pretvorbi (slika 5.2.a).



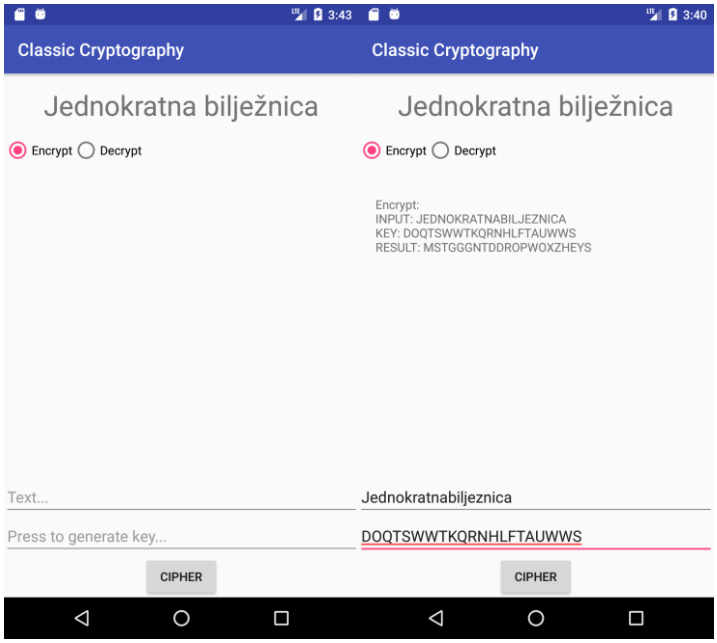
Slika 5.3. a) Alat za pretvaranje b) Dodatni izbornik za prilagođavanje

Vigenèrova šifra prije šifriranja od korisnika zahtjeva da definira kojom metodom želi pretvoriti tekst. Kao izbori postoje uobičajeni način s ponavljajućim ključem i autokey gdje se ključ u postupku šifriranja generira iz otvorenog teksta. Nakon odabira metode, upisa teksta i ključa, pritiskom na tipku cipher korisniku se rezultat prikazuje na zaslonu.



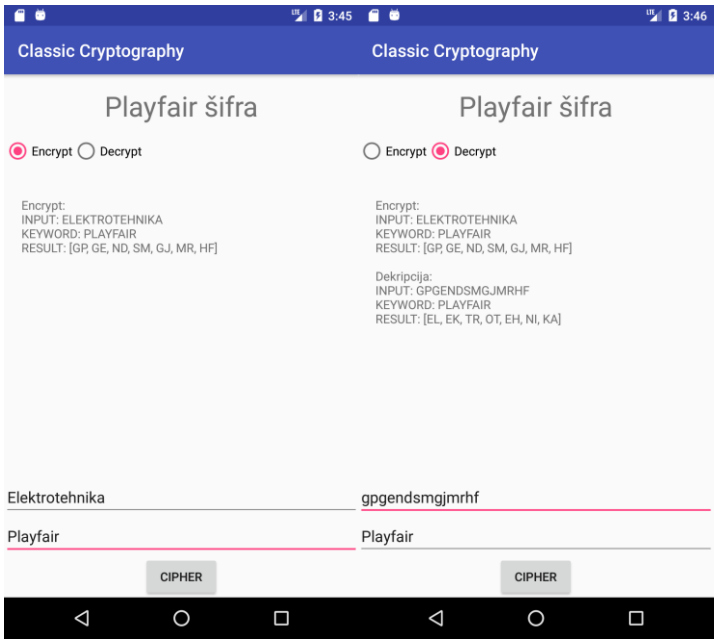
Slika 5.4. a) Alat za pretvaranje b) Dodatni izbornik za prilagođavanje

Za šifriranje jednokratnom bilježnicom korisnik prvo upisuje tekst koji želi šifrirati te mu se nakon toga pritiskom na polje za ključ generira pseudo slučajni niz duljine kao i tekst koji je upisan.



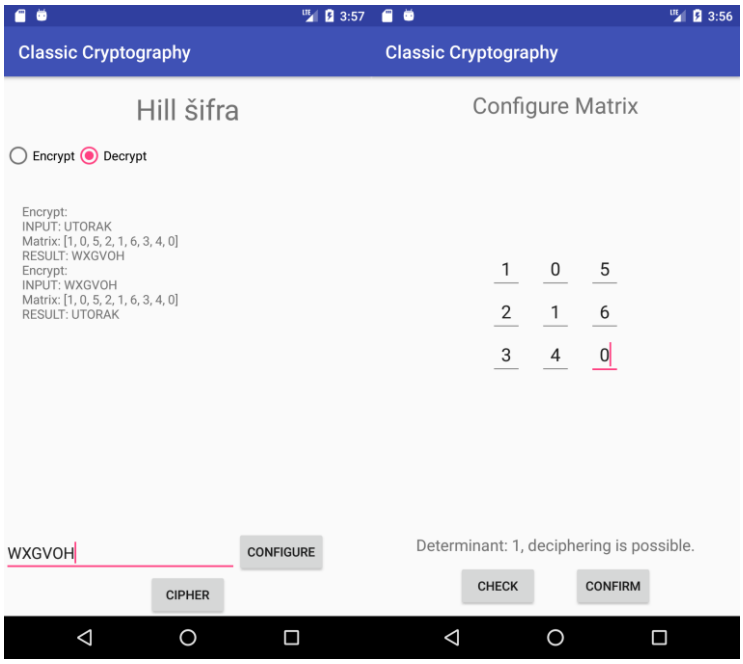
Slika 5.5. Šifriranje jednokratnom bilježnicom

Za šifriranje Playfair šifrom korisnik treba upisati tekst koji treba šifrirati te ključnu riječ. Aplikacija nakon pritiska na *cipher* generira Polibijev kvadrat i započinje pretvorbu upisanog teksta.



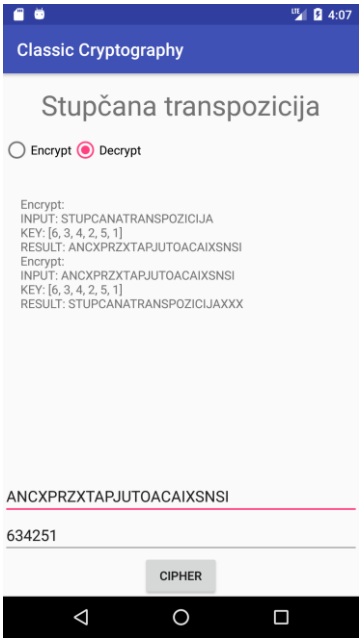
Slika 5.6. Šifriranje Playfair kriptosustavom

Hill kriptosustav je za ovaj projekt ograničen na matrice dimenzija 3 × 3 te duljinu riječi koja mora biti višekratnik broja 3. Prije početka pretvaranja korisnik prvo treba u dodatnom izborniku definirati matricu. Nakon upisa, pritiskom na tipku *Check* može provjeriti determinantu matrice i tako mogućnost dešifriranja (slika 5.7.b), ukoliko je determinanta različita od 1, dešifriranje šifrata neće biti moguće. Potvrdom na tipku *Confirm* aplikacija se vraća u prethodni prozor gdje se može upisani tekst šifrirati definiranom matricom.



Slika 5.7. a) Šifriranje Hil kriptosustavom b) Postavke matrice

Zadnji kriptosustav, stupčana transpozicija za ispravan rad zahtjeva da korisnik u polje za ključ upiše brojeve od 0 – 9 tako da se svaki broj ne pojavi više od jednom i da upisani brojevi poredano čine niz bez preskočenog broja. Ukoliko je upisani ključ napisan prema pravilima korisnik će pritiskom na *Cipher* dobiti rezultat na zaslonu, inače će se pokazati *Toast* poruka s razlogom zašto se nije rezultat prikazao na zaslonu. Izgled alata za stupčanu transpoziciju nalazi se na slici 5.8.



Slika 5.8. Stupčana transpozicija

Zadnja stavka na glavnom izborniku „O aplikaciji“ sadrži osnovne informacije i sažetak o samoj aplikaciji.

Slika 5.9. Opis aplikacije

6. ZAKLJUČAK

Cilj ovog diplomskog rada bio je razvoj aplikacije putem koje korisnik može nešto naučiti o klasičnim kriptosustavima. Klasični kriptosustavi kao početak i temelj kriptografije daju dobar uvid u svijet kriptografije što je u ovom radu i aplikaciji naznačeno. Opisani su svi kriptosustavi koji su bili značajni za svoje vrijeme te je za svaki opisan primjer i način razbijanja – kriptoanalitike.

Osim kriptosustava ukratko je spomenuta i odabrana platforma za razvoj *Android*. Opisani su počeci kada je bio operacijski sustav namijenjen za digitalne fotoaparate, preko vremena pretjecanja najvećeg konkurenta do danas kada sustav slavi svoj 10. rođendan i kada je praktički na svakom uređaju s mikroračunalom. Uspoređeni su još i statistički podataci Androida s svojim najvećim konkurentom što se tiće fragmentacije distribucija, zastupljenosti te pogodnosti i nedostaci za razvijače.

Razvoj aplikacije na način da proces pretvaranja teksta u drugi oblik poštuje izvorna pravila i tok kriptosustava ostvareni su u ovom radu no jedna od mogućih unaprijeđenja bi mogao biti detaljniji prikaz koraka pretvorbe. Način na koji su razvijene mehanike pretvorbe ne dopuštaju ispisivanje svakog koraka na zaslon korisniku što bi korisniku bilo od pomoći pri učenju. Još jedno od unaprijeđenja za sustav je mogućnost uporabe brojeva u otvorenom tekstu, metoda za pretvaranje teksta u brojeve i obratno ograničena je samo na relaciju tekst – broj, pri pretorbi iz brojeva u slova aplikacija umjesto broja prikazuje slovo koje je na mjestu broja unutar abecede.

Aplikacija ukratko nudi sve što je potrebno za početak ulaska u svijet kriptografije te ju je najbolje primjeniti tako da se pročitaju informacije i koraci potrebni za pretvorbu pa *klasičnom* metodom olovke i papira razrađivati sustav, alat za pretvorbu u tu svrhu omogućava brzu provjeru rezultata ili slanja drugoj osobi kao izazov za početak kriptoanalitike.

7. LITERATURA

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

[2] Letter Frequencies in English - The Oxford Math Center, dostupno na www.oxfordmathcenter.com/drupal7/node/353, kolovoz 2018

[3] The Vigenère Cipher: Introduction, dostupno na pages.mtu.edu/~shene/NSF-4/Tutorial/VIG/Vig-Intro.htmll, kolovoz 2018

[4] Distribution Dashboard | Android Developers, dostupno na https://developer.android.com/about/dashboards/ , rujan 2018.

[5] iOS Distribution and iOS Market Share, dostupno na https://data.apteligent.com/ios, rujan 2018

[6] Mobile Operating System Market Share Worldwide, dostupno na http://gs.statcounter.com/os-market-share/mobile/worldwide, rujan 2018

[7] Some interesting stats about Google Play Store vs. Apple App Store apps and development, dostupno na https://www.androidpolice.com/2018/04/05/interesting-stats-google-play-store-vs-apple-app-store-apps-development/

[8] Tim Mackenzie: Android Ad Network Primer: How to Monetize Android Apps with Advertisements, CreateSpace Independent Publishing Platform, 2012

[9] Download Android Studio, dostupno na https://developer.android.com/studio/, rujan 2018

[10] SecureRandom (Java Platform SE 8), dostupno na https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html, rujan 2018

[11] JAMA: Java Matrix Package, dostupno na https://math.nist.gov/javanumerics/jama/, rujan 2018

8. SAŽETAK

Cilj ovog diplomskog rada izrada je aplikacije za učenje o klasičnim kriptosustavima. Klasični kriptosustavi predstavljaju temelj za razvoj modernih kriptosustava, te je stoga prilikom proučavanja modernih kriptosustava gotovo neizostavno (radi njihovog lakšeg razumijevanja) upoznati se s klasičnim kriptosustavima.

Aplikacija izrađena za Android operativni sustav nudi uvid u pojedine kriptosustave te za svaki opisani sustav nudi mogućnost šifriranja i dešifriranja prema proizvoljnim parametrima.

Ključne riječi: Kriptografija, klasični kriptosustavi, Android, Java

9. ABSTRACT

The goal of this paper is to develop an application for learning about classic cryptosystems. Classic cryptosystems are the basis for the developement of modern cryptosystems, it is almost indispensable (for their easier understanding) to become familiar with classical cryptosystems.

The developed application for the Android operating system offers insight into each classic cryptograpic system, it also offers the ability to encrypt and decrypt by arbitrary parameters.

Keywords: Cryptography, classic cryptosystems, Android, Java

10. ŽIVOTOPIS

Marko Radanović rođen je 20.12.1991 u Ludinghausenu, Njemačka. Osnovnu školu „OŠ Kneževi Vinogradi“ završava 2006. godine te upisuje „Prvu srednju školu“ u Belom Manastiru, smjer tehničar za računarstvo. 2010. godine upisuje po prvi puta Elektrotehnički fakultet u Osijeku gdje završava stručni studij informatike 2013. godine. Razlikovnu godinu upisuje 2015. godine te odmah nakon završetka razlikovne godine 2016. upisuje diplomski studij mrežnih tehnologija na Fakultetu elektrotehnike, računarstva i informacijskih tehnilogija u Osijeku.

_____

Marko Radanović

11. PRILOG

Supstitution.java

```
public class Supstitution {

    private static final String TTN_OUTPUT = "M/O textToNumber";
    private static final String NTT_OUTPUT = "M/O numberToText";

    char[] alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".toCharArray();

    public List textToNumber(String inputText){         //Converts input string to array
of numbers
        List textToNum = new ArrayList();

        Log.i(TTN_OUTPUT, "textNumbers cleared, length: " + textToNum.size());
        char[] chars = inputText.toCharArray();
        for(int i = 0; i < chars.length; i++){
            if(chars[i] != ' ' ) {
                textToNum.add(('A' - chars[i]) * -1);
            }
        }
        Log.i(TTN_OUTPUT, textToNum.toString());
        return textToNum;
    }
    public String numberToText(List inputNumbers){        //Converts input list of
numbers to one textual string
        String output = "";
        Integer letter;
        for(int i = 0; i < inputNumbers.size(); i++){
            letter = Integer.parseInt(inputNumbers.get(i).toString());
            output = output + alphabet[letter % 26];

        }
        Log.i(NTT_OUTPUT, output);
        return output;
    }
}
```

Caesar.java

```
public class Caesar extends Supstitution {

        private static final String ENC_OUTPUT = "M/O CaesarEncrypt";
        private static final String DEC_OUTPUT = "M/O CaesarDecrypt";

        public String encrypt(String inputText, int key){
            List textNumbers = new ArrayList();

            textNumbers = textToNumber(inputText.toUpperCase());
            for(int i = 0; i<textNumbers.size(); i++){
                Integer temp = Integer.parseInt(textNumbers.get(i).toString());
                textNumbers.set(i, temp+key);
            }
            Log.i(ENC_OUTPUT, textNumbers.toString());
            return numberToText(textNumbers);
        }
```

```
        public String decrypt(String inputText, int key){
            List textNumbers = new ArrayList();

            textNumbers = textToNumber(inputText.toUpperCase());
            for(int i = 0; i<textNumbers.size(); i++){
                Integer temp = Integer.parseInt(textNumbers.get(i).toString());
                textNumbers.set(i, temp-key);
            }
            Log.i(DEC_OUTPUT, textNumbers.toString());
            return numberToText(textNumbers);
        }
}
```

Affine.java

```
public class Affine extends Supstitution {

    private static final String ENC_OUTPUT = "M/O AffineEncrypt";
    private static final String DEC_OUTPUT = "M/O AffineDecrypt";

    private List textNumbers = new ArrayList();
    Map<Integer, Integer> aS = new HashMap<Integer, Integer>(){{
     put(1, 1);
     put(3, 9);
     put(5, 21);
     put(7, 15);
     put(9, 3);
     put(11, 19);
     put(15, 7);
     put(17, 23);
     put(19, 11);
     put(21, 5);
     put(23, 17);
     put(25, 25);
    }};


    public String encrypt(String inputText, int a, int b){
        if(aS.containsKey(a)){
            Supstitution supstitution = new Supstitution();
            textNumbers = supstitution.textToNumber(inputText);
            for(int i = 0; i<textNumbers.size(); i++){
                Integer temp = Integer.parseInt(textNumbers.get(i).toString());
                textNumbers.set(i, (a*temp)+b);
            }
            Log.i(ENC_OUTPUT, textNumbers.toString());
            return supstitution.numberToText(textNumbers);
        }
        else {
            Log.i(ENC_OUTPUT, "Key 'a' is not invertable");
            return "ERROR";
        }
    }

    public String decrypt(String inputText, int a, int b){
        if(aS.containsKey(a)){
            Supstitution supstitution = new Supstitution();
            textNumbers = supstitution.textToNumber(inputText);
            for(int i = 0; i<textNumbers.size(); i++){
                Integer temp = Integer.parseInt(textNumbers.get(i).toString());
                textNumbers.set(i, aS.get(a)*(temp-b));
```

```
            }
            Log.i(ENC_OUTPUT, textNumbers.toString());
            return supstitution.numberToText(textNumbers);
        }
        else {
            Log.i(DEC_OUTPUT, "Key 'a' is not invertable");
            return "ERROR";
        }
    }
}
```

MixedAlphabet.java

```
public class MixedAlphabet extends Supstitution {


    Map<String, String> mixedAlphabet = new HashMap<>();
    List<String> uniqueKeyword = new ArrayList<>();
    List<String> tempKeyword = new ArrayList<>();
    List<String> tempResult = new ArrayList<>();
    Map<String, String> tempAlphabet = new HashMap<>();
    private static final String ENC_OUTPUT = "M/O MixedEncrypt";
    private static final String DEC_OUTPUT = "M/O MixedDecrypt";
    private static final String GA_OUTPUT = "M/O generateAlphabet";
    private static final String GUK_OUTPUT = "M/O uniqueKeyword";


    public Map<String, String> generateAlphabet(int choice, String keyword, int start){
        switch (choice){
            case 0:         //manual without keyword
                mixedAlphabet.clear();
                char[] chars = keyword.toCharArray();
                for(int i = 0; i < keyword.length();i++){
                    mixedAlphabet.put(String.valueOf(alphabet[i]),
String.valueOf(chars[i]));
                }
                return mixedAlphabet;
            case 1:         //with keyword at start position
                tempKeyword = generateUniqueKeyword(keyword);
                int br = 0;
                for (int i = 0; i < tempKeyword.size(); i++){
                    try {
                        mixedAlphabet.put(String.valueOf(alphabet[start+i]),
tempKeyword.get(i));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                Log.i(GUK_OUTPUT, mixedAlphabet.toString());
                for (int i = 0; i <= 25; i++){
                    if(mixedAlphabet.containsKey(String.valueOf(alphabet[i]))){  //2nd
argument
                        continue;
                    }
                    else {
                        while(mixedAlphabet.containsValue(String.valueOf(alphabet[br])))
{
                            br++;
                        }
                        mixedAlphabet.put(String.valueOf(alphabet[i]),
String.valueOf(alphabet[br]));
```

```
                        br++;
                    }
                }
                Log.i(GUK_OUTPUT, mixedAlphabet.toString());
                return mixedAlphabet;
            }
            return null;
        }

        public List<String> generateUniqueKeyword(String keyword){
            tempKeyword.clear();
            char[] tempKeyword = keyword.toCharArray();
            for(int i = 0; i < tempKeyword.length; i++){
                if(uniqueKeyword.contains(String.valueOf(tempKeyword[i]))){
                    Log.i(GUK_OUTPUT, "uniqueKeyword conains " + tempKeyword[i]);
                }
                else {

                    uniqueKeyword.add(String.valueOf(tempKeyword[i]));
                    Log.i(GUK_OUTPUT, tempKeyword[i] + " added to uniqueKeyword");
                }
            }
            Log.i(GA_OUTPUT, uniqueKeyword.toString());
            return uniqueKeyword;
        }

        public String encrypt (String inputText, int choice, String keyword, int start){
            tempResult.clear();
            char[] tempInput = inputText.toCharArray();
            tempAlphabet = generateAlphabet(choice, keyword, start);
            for(int i = 0; i<inputText.length(); i++){
                tempResult.add(tempAlphabet.get(String.valueOf(tempInput[i])));
            }
            Log.i(ENC_OUTPUT, tempResult.toString());
            return tempResult.toString();
        }

        public String decrypt (String inputText, int choice, String keyword, int start){
            tempResult.clear();
            char[] tempInput = inputText.toCharArray();
            tempAlphabet = generateAlphabet(choice, keyword, start);
            for(int i = 0; i<inputText.length(); i++){
                tempResult.add(getKeyFromValue(tempAlphabet,
    String.valueOf(tempInput[i])).toString());
            }
            Log.i(DEC_OUTPUT, tempResult.toString());
            return tempResult.toString();

        }

        public static Object getKeyFromValue(Map hm, Object value) {
            for (Object o : hm.keySet()) {
                if (hm.get(o).equals(value)) {
                    return o;
                }
            }
            return null;
        }
    }
```

Vigenere.java

```
public class Vigenere extends Supstitution {


    private List tempInput = new ArrayList<>();
    private List tempKeyword = new ArrayList<>();
    private List tempResult = new ArrayList();

    int keyCount;


    private static final String ENC_OUTPUT = "M/O vigenereEncrypt";
    private static final String DEC_OUTPUT = "M/O vigenereDecrypt";

    public String encrypt(String plainText, String keyword, int choice){

        tempInput = textToNumber(plainText);
        tempKeyword = textToNumber(keyword);

        switch(choice){
            case 0:  // Normal
                keyCount = 0;
                tempResult.clear();
                for(int i = 0; i< tempInput.size();i++){
                    tempResult.add(Integer.valueOf(tempInput.get(i).toString()) +
Integer.valueOf(tempKeyword.get(keyCount).toString()));
                    keyCount++;
                    if(keyCount == tempKeyword.size()){
                        keyCount = 0;
                    }
                }
                Log.i(ENC_OUTPUT, numberToText(tempResult));
                return numberToText(tempResult);
            case 1:     // Autokey
                keyCount = 0;
                tempResult.clear();
                for(int i = 0; i< tempInput.size();i++){
                    tempResult.add(Integer.valueOf(tempInput.get(i).toString()) +
Integer.valueOf(tempKeyword.get(keyCount).toString()));
                    keyCount++;
                    if(keyCount == tempKeyword.size()){
                        keyCount = 0;
                        tempKeyword.clear();
                        tempKeyword = tempInput;
                    }
                }
                Log.i(ENC_OUTPUT, numberToText(tempResult));
                return numberToText(tempResult);
        }
        return null;
    }

    public String decrypt(String cipher, String keyword, int choice){

        tempInput = textToNumber(cipher);
        tempKeyword = textToNumber(keyword);

        switch(choice){
            case 0:  // Normal
                int x;
                keyCount = 0;
```

```
                tempResult.clear();
                for(int i = 0; i< tempInput.size();i++){
                    x = Integer.valueOf(tempInput.get(i).toString()) -
Integer.valueOf(tempKeyword.get(keyCount).toString());
                    if(x < 0){
                        x = 26 - (x*-1);
                        tempResult.add(x);
                    }
                    else{
                        tempResult.add(x);
                    }
                    keyCount++;
                    if(keyCount == tempKeyword.size()){
                        keyCount = 0;
                    }
                }
                Log.i(DEC_OUTPUT, numberToText(tempResult));
                return numberToText(tempResult);

            case 1:     // Autokey
                keyCount = 0;
                tempResult.clear();
                for(int i = 0; i< tempInput.size();i++){
                    x = Integer.valueOf(tempInput.get(i).toString()) -
Integer.valueOf(tempKeyword.get(keyCount).toString());
                    if(x < 0){
                        x = 26 - (x*-1);
                        tempResult.add(x);
                    }
                    else{
                        tempResult.add(x);
                    }
                    keyCount++;
                    if(keyCount == tempKeyword.size()){
                        keyCount = 0;
                        tempKeyword = tempResult;
                    }
                }
                Log.i(ENC_OUTPUT, numberToText(tempResult));
                return numberToText(tempResult);
        }
        return null;
    }
}
```

OneTimePad.java

```
public class OneTimePad extends Vigenere {

    private static final String ENC_OUTPUT = "M/O oneTimePadEncrypt";
    private static final String DEC_OUTPUT = "M/O oneTimePadDecrypt";
    List pad = new ArrayList();
    List randomPad = new ArrayList();

    public List generatePad(int length) {
        SecureRandom secureRandom = new SecureRandom();

        for (int i = 0; i < length; i++) {
            pad.add(alphabet[secureRandom.nextInt(alphabet.length)]);
        }
```

```
        Log.i("generatePad", pad.toString());
        return pad;
    }

    public String encrypt(String inputText) {
        randomPad = generatePad(inputText.length());
        Log.i(ENC_OUTPUT, super.encrypt(inputText,
generatePad(inputText.length()).toString(), 0));
        return super.encrypt(inputText, randomPad.toString(), 0);
    }

    public String decrypt(String inputText, String keyword) {
        Log.i(DEC_OUTPUT, super.decrypt(inputText, keyword, 0));
        return super.decrypt(inputText, keyword, 0);
    }
}
```

Playfair.java

```
public class Playfair extends Supstitution {

    String[][] tempKeyMatrix = new String[5][5];
    String[][] keyMatrix = new String[5][5];
    List<String> uniqueKeyword = new ArrayList<>();

    List<String> tempKeyword = new ArrayList<>();
    List<String> tempResult = new ArrayList<>();

    private static final String ENC_OUTPUT = "M/O MixedEncrypt";
    private static final String DEC_OUTPUT = "M/O MixedDecrypt";
    private static final String GUK_OUTPUT = "M/O generateKeyMatrix";


    public String[][] generateKeyMatrix(String keyword) {
        int x, y, i, z = 0;
        tempKeyword.clear();
        char[] tempKeyword = keyword.toCharArray();
        for (i = 0; i < tempKeyword.length; i++) {
            if (uniqueKeyword.contains(String.valueOf(tempKeyword[i]))) {
            } else {
                uniqueKeyword.add(String.valueOf(tempKeyword[i]));
            }
        }
        i = 0;
        for (x = 0; x < 5; x++) {
            for (y = 0; y < 5; y++) {
                try {
                    tempKeyMatrix[x][y] = uniqueKeyword.get(i);
                    i++;
                } catch (Exception e) {
                    while (uniqueKeyword.contains(String.valueOf(alphabet[z])) ||
alphabet[z] == 'Q') {
                        z++;
                    }
                    tempKeyMatrix[x][y] = String.valueOf(alphabet[z]);
                    z++;
                }
                //e.printStackTrace();
            }
        }
```

```
        Log.i(GUK_OUTPUT, Arrays.deepToString(tempKeyMatrix));
        return tempKeyMatrix;
    }
    public Point findPoint(char e){
        Point point = new Point(0,0);
        for(int x = 0; x < 5; x++){
            for(int y = 0; y < 5; y++){
                if(e == keyMatrix[x][y].charAt(0)){
                    point = new Point(x,y);
                }
            }
        }
        return point;
    }

    public List<String> encrypt(String inputText, String keyword){
        tempResult.clear();
        keyMatrix = generateKeyMatrix(keyword);
        int digraphNumber = inputText.length() / 2 + inputText.length() % 2;

        for(int i = 0; i < (digraphNumber -1); i++){
            if(inputText.charAt(2 * i) == inputText.charAt(2 * i + 1)){
                inputText = new StringBuffer(inputText).insert(2 * i + 1,
"X").toString();
                digraphNumber = inputText.length() / 2 + inputText.length() % 2;
            }
        }
        String[] digraph = new String[digraphNumber];
        for(int j = 0; j < digraphNumber; j++){
            if(j == (digraphNumber-1) && inputText.length() / 2 == (digraphNumber-1)){
                inputText = inputText + "X";

            }
            digraph[j] = inputText.charAt(2*j) + ""+ inputText.charAt(2*j+1);
        }
        for(int i = 0; i < digraphNumber; i++){
            char a = digraph[i].charAt(0);
            char b = digraph[i].charAt(1);
            int r1 = findPoint(a).x;
            int c1 = findPoint(a).y;
            int r2 = findPoint(b).x;
            int c2 = findPoint(b).y;

            if(r1 == r2) {
                c1 = (c1 + 1) % 5;
                c2 = (c2 + 1) % 5;
            }else if(c1 == c2) {
                r1 = (r1 + 1) % 5;
                r2 = (r2 + 1) % 5;
            }else{
                int temp = c1;
                c1 = c2;
                c2 = temp;
            }
            tempResult.add(keyMatrix[r1][c1]+""+keyMatrix[r2][c2]);

        }

        Log.i(ENC_OUTPUT, String.valueOf(digraphNumber));
        Log.i(ENC_OUTPUT, Arrays.deepToString(digraph));
        Log.i(ENC_OUTPUT, tempResult.toString());

        return tempResult;
```

```
    }

    public String decrypt(String inputText, String keyword){
        keyMatrix = generateKeyMatrix(keyword);
        tempResult.clear();
        for(int i = 0; i < inputText.length()/2; i++){
            char a = inputText.charAt(2*i);
            char b = inputText.charAt(2*i+1);
            int r1 = findPoint(a).x;
            int c1 = findPoint(a).y;
            int r2 = findPoint(b).x;
            int c2 = findPoint(b).y;
            if(r1 == r2){
                c1 = (c1 + 4) % 5;
                c2 = (c2 + 4) % 5;
            }else if(c1 == c2){
                r1 = (r1 + 4) % 5;
                r2 = (r2 + 4) % 5;
            }else{
                int temp = c1;
                c1 = c2;
                c2 = temp;
            }
            tempResult.add(keyMatrix[r1][c1] + keyMatrix[r2][c2]);

        }
        Log.i(DEC_OUTPUT, tempResult.toString());
        return tempResult.toString();
    }
}
```

Hill.java

```
public class Hill extends Supstitution {

    private static final String ENC_OUTPUT = "M/O hillEncrypt";
    private static final String DEC_OUTPUT = "M/O hillDecrypt";

    double[][] keyword = new double[3][3];
    double[] plainText = new double[3];
    List tempInputNumbers = new ArrayList();
    List tempKeyNumbers = new ArrayList();
    List tempResultList = new ArrayList();

    public double[][] generateKeyMatrix(List inputKeyword) {          // add for numbers
        int i = 0;
        for (int x = 0; x <= 3; x++) {
            for (int y = 0; y <= 3; y++) {
                try {
                    keyword[x][y] = Double.valueOf(inputKeyword.get(i).toString());
                    i++;
                } catch (Exception e) {
                    Log.i("GKM", "Something went wrong");
                    //e.printStackTrace();
                }
            }
        }
        Log.i("GKM", Arrays.deepToString(keyword));
        return keyword;
    }
```

```
    public String encrypt(String inputText, String key) {
        tempResultList.clear();
        tempInputNumbers = textToNumber(inputText);
        tempKeyNumbers = textToNumber(key);
        int trigraphs = tempInputNumbers.size() / 3 + tempInputNumbers.size() % 3;
        Matrix K = new Matrix(generateKeyMatrix(tempKeyNumbers));

        for (int i = 0; i < trigraphs; i++) {
            plainText[0] = Double.valueOf(tempInputNumbers.get(3 * i).toString());
            plainText[1] = Double.valueOf(tempInputNumbers.get(3 * i + 1).toString());
            plainText[2] = Double.valueOf(tempInputNumbers.get(3 * i + 2).toString());
            Matrix P = new Matrix(plainText, 1);
            Matrix R = P.times(K);
            double[][] tempResult = R.getArrayCopy();
            for (int br = 0; br <= 2; br++) {
                int t = (int) round(tempResult[0][br]) % 26;
                tempResultList.add(t);
                Log.i(ENC_OUTPUT, t + " added to temp result list");
            }
        }
        return numberToText(tempResultList);
    }

    public String decrypt(String cipher, String keyword) {
        tempResultList.clear();
        int t = 0;
        tempInputNumbers = textToNumber(cipher);
        tempKeyNumbers = textToNumber(keyword);
        int trigraphs = tempInputNumbers.size() / 3 + tempInputNumbers.size() % 3;
        Matrix K = new Matrix(generateKeyMatrix(tempKeyNumbers));
        Matrix iK = K.inverse();
        for (int i = 0; i < trigraphs; i++) {
            plainText[0] = Double.valueOf(tempInputNumbers.get(3 * i).toString());
            plainText[1] = Double.valueOf(tempInputNumbers.get(3 * i + 1).toString());
            plainText[2] = Double.valueOf(tempInputNumbers.get(3 * i + 2).toString());
            Matrix C = new Matrix(plainText, 1);
            Matrix R = C.times(iK);
            double[][] tempResult = R.getArrayCopy();
            for (int br = 0; br <= 2; br++) {
                if (tempResult[0][br] > 0) {
                    t = (int) round(tempResult[0][br]) % 26;
                } else {
                    t = (int) (((tempResult[0][br] % 26) + 26) % 26) + 1;
                }
                tempResultList.add(t);

                Log.i(DEC_OUTPUT, tempResultList.get(br) + " added to temp result list");
            }
        }
        return numberToText(tempResultList);
    }

}
```

ColumnarTransposition.java

```
public class ColumnarTransposition {

    private static final String ENC_OUTPUT = "M/O cTEncrypt";
    private static final String DEC_OUTPUT = "M/O cTDecrypt";

    String[][] table;
    Map<Integer, String> tempResultMap = new HashMap<>();
    Map<Integer, String> sortedMap;

    public String encrypt(String inputText, List key){
        tempResultMap.clear();
        Integer br = 0;
        String tempResult = "";
        Integer columns = key.size();
        Integer rows = (int) ceil(Double.valueOf(inputText.length())/columns);

        table = new String[rows][columns];
        for(int x = 0; x<rows;x++){
            for(int y = 0; y<columns;y++){
                try {
                    table[x][y] = String.valueOf(inputText.charAt(br));
                    br++;
                } catch (Exception e) {
                    table[x][y] = "X";
                    e.printStackTrace();
                }
            }
        }
        br=0;
        for(int y = 0;y<columns;y++){
            for(int x = 0; x<rows;x++){
                tempResult = tempResult + table[x][y];
            }
            tempResultMap.put(Integer.valueOf(key.get(br).toString()), tempResult);
            tempResult = "";
            br++;
        }
        sortedMap = new TreeMap<>(tempResultMap);
        for (Map.Entry<Integer, String> entry : sortedMap.entrySet()) {
            tempResult = tempResult + entry.getValue();
        }
        Log.i(ENC_OUTPUT, tempResult);
        return tempResult;
    }

    public String decrypt(String cipher, List key) {

        Integer br = 0;
        String tempResult = "";
        Integer columns = key.size();
        Integer rows = (int) ceil(Double.valueOf(cipher.length()) / columns);

        table = new String[rows][columns];
        tempResultMap.clear();

        for (int y = 0; y < columns; y++) {
            for (int x = 0; x < rows; x++) {
                try {
                    table[x][y] = String.valueOf(cipher.charAt(br));
                    tempResult = tempResult + table[x][y];
```

```
                    br++;
                } catch (Exception e) {
                    table[x][y] = "X";
                    e.printStackTrace();
                }
            }
            tempResultMap.put(y + 1, tempResult);
            tempResult = "";
        }
        br = 0;
        for (int i = 0; i < tempResultMap.size(); i++) {
            tempResult = tempResult + tempResultMap.get(key.get(i));
        }
        for (int y = 0; y < columns; y++) {
            for (int x = 0; x < rows; x++) {
                table[x][y] = String.valueOf(tempResult.charAt(br));
                br++;
            }
        }
        tempResult = "";
        for(int x = 0; x<rows;x++){
            for(int y = 0; y<columns;y++){
                tempResult = tempResult + table[x][y];
            }
        }

        Log.i(DEC_OUTPUT, Arrays.deepToString(table));
        Log.i(DEC_OUTPUT, tempResult);

        return tempResult;
    }
}
```