Računalna aplikacija za sigurnu razmjenu tekstualnih poruka u stvarnom vremenu

Vujnovac, Matko

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

RAČUNALNA APLIKACIJA ZA SIGURNU RAZMJENU TEKSTUALNIH PORUKA U STVARNOM VREMENU

Diplomski rad

Matko Vujnovac

Osijek, 2017.

Sadržaj

1. UVOD

„Kriptografija je znanstvena disciplina koja se bavi proučavanjem metoda za slanje poruka u takvom obliku da ih samo onaj kome su namijenjene može pročitati. Sama riječ kriptografija je grčkog podrijetla i mogla bi se doslovno prevesti kao tajnopis.“[1] Glavni je zadatak kriptografije omogućiti sigurno komuniciranje preko nesigurnog kanala. Kriptografija kao disciplina pojavljuje se još u doba starih Rimljana i Grka. U početku su se za kriptiranje koristile jednostavne šifre odnosno algoritmi. Ti algoritmi temeljili su se na jednostavnoj supstituciji znakova u poruci. Kod nekih šifri nije bilo potrebno koristiti ključ što je samu enkripciju činilo lako probojnom u slučaju da netko sazna ili otkrije algoritam pomoću kojega se ona izvodi. Zbog toga se u proces enkripcije uvodi ključ. Postoje dvije vrste enkripcije pomoću ključa, a to su simetrična i asimetrična enkripcija. Kod simetrične enkripcije poruke se šifriraju i dešifriraju pomoću istog ključa dok se kod asimetričnih koriste različiti ključevi za šifriranje i dešifriranje. Enkripcija pomoću ključeva znatno je sigurnija od one koja se temelji na algoritmu bez upotrebe ključa. Međutim, kod enkripcije pomoću ključa glavni problem predstavlja razmjena tih ključeva među sudionicima u razgovoru te u slučaju otkrivanja ključa od treće strane algoritam gubi sigurnost bez obzira na njegovu složenost. U današnje doba velik dio komunikacije odvija se preko računalnih i informacijskih sustava što ju čini bržom i efikasnijom, ali isto tako pruža više mogućnosti za zlouporabu podataka koji se šalju. Bilo kakva razmjena podataka u računalnim sustavima, bila ona lokalna ili preko mreže, predstavlja potencijalnu mogućnost da do tih podataka dođe netko kome oni nisu namjenjeni. Svaki prosječni korisnik računala zasigurno se barem jednom susreo s aplikacijama za razmjenu poruka u stvarnom vremenu. Neke od tih aplikacija se mogu smatrati sigurnijima od drugih s obzirom na vrstu zaštite i enkripcije koju koriste kako bi zaštitili privatnost korisnika. Postoje razne metode enkripcije poruka i podataka, a jedna od njih je i AES algoritam koji danas predstavlja standard u enkripciji. U ovom radu bit će obrađen AES algoritam kao metoda enkripcije poruka i podataka. U početnom dijelu rada bit će objašnjeni osnovni pojmovi vezani uz AES algoritam, njegove specifikacije te načini rada. Drugi dio rada bavit će se postupkom izrade aplikacije za sigurnu razmjenu poruka u stvarnom vremenu rađenu u programskom jeziku C# te implementacijom AES-a u programskom jeziku C# te mogućnostima koje ona pruža pri izradi aplikacija. Također, biti će pokazani rezultati testiranja ispravnosti aplikacije u stvarnom okruženju.

1.1. Zadatak diplomskog rada

U praksi se često javlja potreba za sigurnim komuniciranjem putem tekstualnih poruka u stvarnom vremenu. Potrebno je razviti i implementirati računalnu aplikaciju koja će omogućiti razmjenu tekstualnih poruka u stvarnom vremenu, kao i prijenos datoteka među sugovornicima, pri čemu je komunikaciju potrebno osigurati primjenom kriptografskih metoda. Aplikaciju je potrebno testirati unutar stvarnog mrežnog okruženja.

2. ADVANCED ENCRYPTION STANDARD (AES)

AES ili napredni enkripcijski standard je simetrična blok šifra namijenjena za šifriranje i dešifriranje podataka. S obzirom da se radi o simetričnoj šifri, postupci šifriranja i dešifriranja odvijaju se korištenjem istog ključa za oba procesa. Šifriranje se radi na blokovima veličine 128 bita.

2.1. Povijest AES-a

Prethodnik AES-a bio je DES (eng. *Data Encryption Standard*) algoritam koji je s vremenom izgubio na sigurnosti i pouzdanosti zbog razvoja računalne moći. S obzirom da je DES koristio 64-bitni ključ (od čega je 8 bitova bilo paritetno) na 64-bitnim blokovima bilo je potrebno osmisliti algoritam koji će koristiti dulji ključ te veće blokove kako bi se onemogućilo probijanje čak i najjačim računalima. Natječaj za nasljednika DES algoritma objavljen je početkom 1997. godine od američkog Nacionalnog instituta za standarde i tehnologiju (*NIST*). U sljedećih devet mjeseci na natječaj je prijavljeno petnaest različitih algoritama koji će se natjecati u izboru. Nakon analize svih algoritama neki su odmah odbačeni zbog raznih propusta ili problema u performansama pri izvođenju. Nakon dvije sjednice NIST je u kolovozu 1999. godine suzio izbor na pet algoritama. Među njima je bio i Rijndael algoritam. Rijndael algoritam osmislili su dva belgijska kriptografa, Joan Daemen i Vincent Rijmen. Rijndael je proglašen pobjednikom natječaja u listopadu 2000. godine, a službeno je prihvaćen i standardiziran kao AES u studenom 2001. godine.

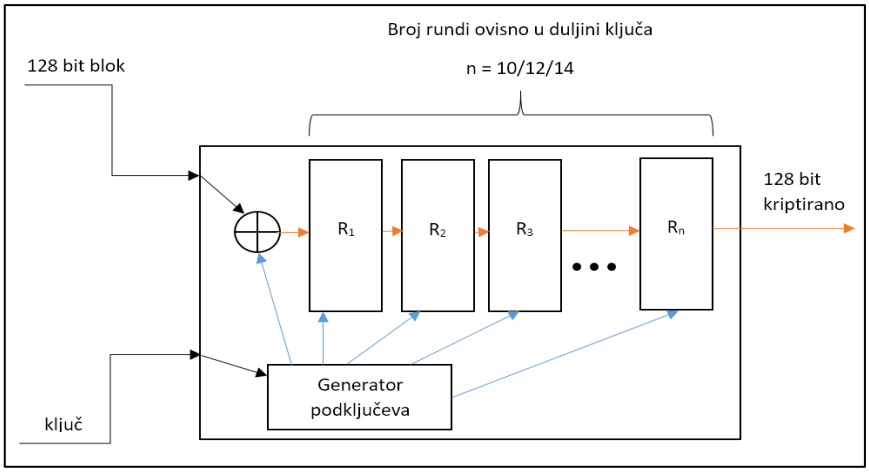
2.2. Specifikacija AES-a

AES algoritam može se specificirati preko dva osnovna parametra koja su ključna za njegov rad. To su duljina ključa i veličina blokova podataka nad kojima se izvršavaju operacije. Dozvoljene duljina ključeva su 128, 192 i 256 bita dok je veličina bloka uvijek 128 bita. Ovisno o duljini ključa provodi se određen broj ponavljanja tijekom izvršenja algoritma što je vidljivo na slici 2.1.

Duljina ključa	Broj ponavljanja
128	10
192	12
256	14

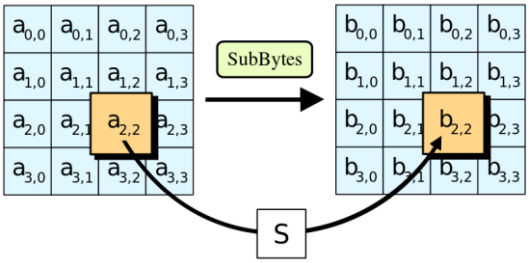
Sl. 2.1. Ovisnost broja ponavljanja o duljini ključa

Na slici 2.2. vidljiva je opća shema AES algoritma. Može se vidjeti kako su ulazni parametri podaci koji se trebaju kriptirati podijeljeni u blokove od 128 bita te ključ. Izlaz iz algoritma predstavljaju kriptirani blokovi od 128 bita.



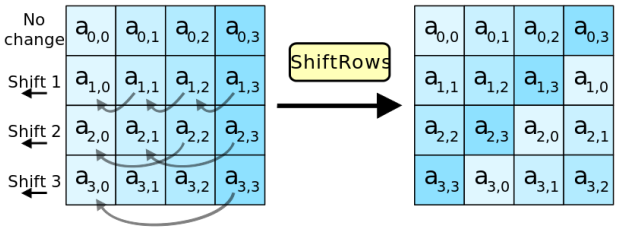
Sl. 2.2. Shema AES-a

Prije samog početka kriptiranja potrebno je stvoriti određen broj podključeva izvedenih iz osnovnog ključa koji će se koristiti u postupku kriptiranja. Broj tih podključeva određen je duljinom ključa odnosno brojem rundi koje će se izvoditi. Kao što je spomenuto u prethdonom dijelu rada broj rundi može biti 10, 12 ili 14. Ukupni broj podključeva mora biti jednak ukupnom broju rundi koje će se izvesti uz još jedan dodatni podključ. Prvi korak u algoritmu je operacija XOR koja se provodi nad ulaznim blokom podataka od 128 bita i prvim podključem. Takav niz prosljeđuje se prvoj rundi R_1. U ovoj rundi koristi se drugi generirani podključ. Nakon što se nad blokom odrade sve operacije unutar prve runde on se prosljeđuje drugoj rundi gdje se provode iste operacije kao i u prvoj rundi upotrebom sljedećeg podključa. Taj postupak ponavlja se sve do n-te odnosno zadnje runde koja se malo razlikuje od prethodnih. Nakon što su prošle sve runde kao izlaz se dobije 128 bitni kriptirani blok. Ovaj postupak ponavlja se za sve blokove koje je potrebno kriptirati dok se ne dođe do kraja cijelog niza podataka koje je potrebno kriptirati. U sljedećem dijelu rada razložit ćemo svaku rundu te objasniti koje se operacije događaju unutar svakog bloka R sa sheme. Svaki R blok sa slike 2.2. sastoji se od četiri podbloka odnosno operacije koje se u njemu izvršavaju nad podacima. Jedino se zadnji R blok razlikuje te se u njemu ne izvršava jedna od operacija. Prva operacija unutar svakog R bloka je supstitucija blokova bajtova. 128-bitni blok dijeli se na blokove od 16 podblokova. Nad tim podblokovima tada se izvršava supstitucija prema unaprijed generiranoj tablici. Tablica sadrži parove blokova odnosno za svaki podblok postoji drugi podblok s kojim će biti supstituiran. Ovime se postiže nelinearnost podataka. Primjer postupka vidljiv je na slici 2.3.



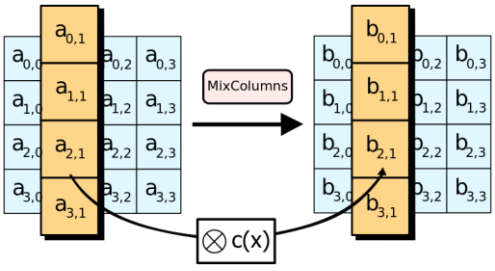
Sl. 2.3. Supstitucija

Nakon supstitucije blokova izvršava se transpozicija odnosno pomicanje blokova. Ono se odvija tako da se podblokovi ciklički pomiču ulijevo. Prvi red ostaje nepromijenjen dok se ostali redovi pomiču za jedno, dva odnosno tri mjesta ulijevo. Postupak je vidljiv na slici 2.4.



Sl. 2.4. Transpozicija

Nakon transpozicije na red ponovno dolazi supstitucija, ali u ovom slučaju ona se izvodi na četiri podbloka odjednom u odnosu na prvu supstituciju koja se izvodi na svakom podbloku odvojeno. Druga supstitucija prozivodi još viši stupanj nelinearsnoti budući da se izvršava na više blokova odjednom što daje veći mogući broj kombinacija. Na slici 2.5. može se vidjeti primjer ovog postupka.



Sl. 2.5. Druga supstitucija

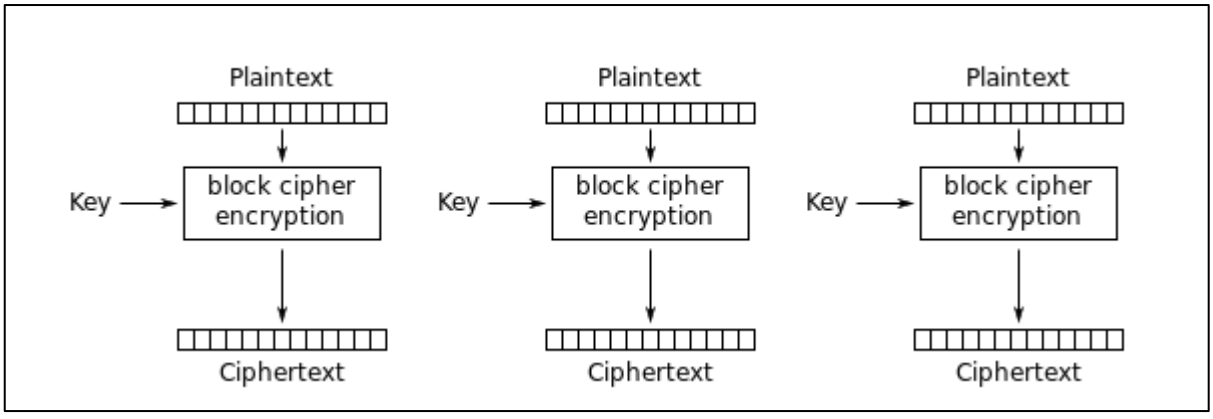
Svaki od blokova sastoji se od ovih operacija osim posljednjeg koji ne sadrži drugu supstituciju. Nakon prolaska svih podataka kroz algoritam kao izlaz se dobije šifrirani tekst jednake duljine kao i ulazni nešifrirani tekst.

2.3. Načini rada

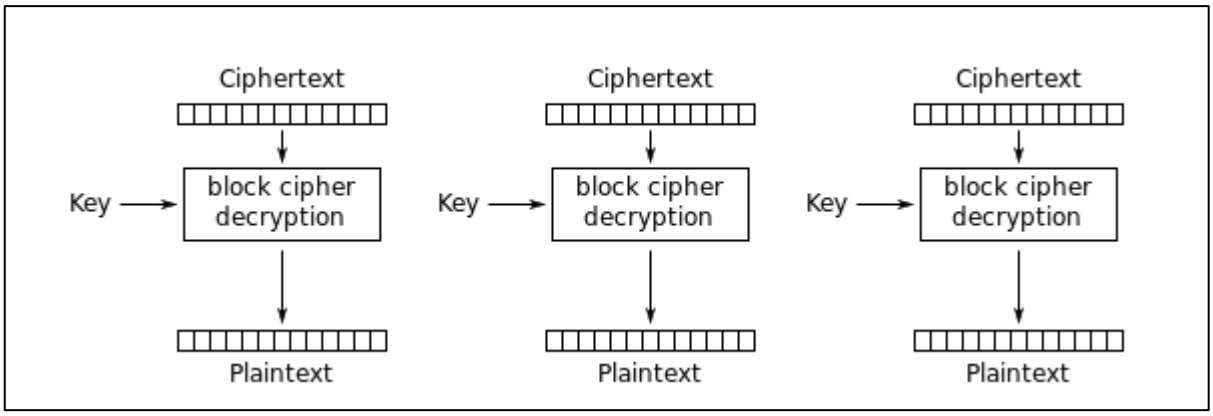
Budući da je duljina bloka nad kojim se izvršava algoritam konstanta pokazala se potreba za razvijanjem različitih načina rada samog algoritma. To se odnosi na način na koji se šifriraju i potom dešifiraju podaci čija je duljina duža od jednog bloka, u slučaju AES-a radi se o blokovima od 128 bita. Primjerice, ako je podatak koji se šifrira kraći od 128 bita, pri šifriranju će se uz korištenje istoga ključa dobiti isti rezultat. Isto tako u slučaju ponavljanja takvih blokova dolazi do ponavljanja šifrata što može dovesti do otkrivanja podataka. Zbog toga su osmišljeni različiti načini rada koji onemogućavaju ili barem svode šansu za pojavljivanje takvih slučajeva na minimum. To se postiže dodavanjem dodatnih operacija u algoritam te dodavanjem dodatnih parametara kao što je inicijalizacijski vektor u sam proces enkripcije i dekripcije. U nastavku će biti objašnjeni neki od najčešće korištenih načina rada enkripcijskih algoritama.

2.3.1. ECB

ECB (eng. *Electronic Codebook*) je najjednostavniji način rada. Sastoji se samo od ulaznih podataka, ključa te izlaznog šifrata. Ulazni podaci dijele se na jednake blokove i na svakom bloku vrši se enkripcija pomoću istog ključa. U ovom načinu rada može doći do ponavljanja istih blokova što će rezultirati istim izlazom. Ne preporučuje se koristiti ovaj način rada, osim u slučaju slanja podataka koji su kraći od duljine bloka na kojem se provodi enkripcija. Na slici 2.6. vidljiv je shematski prikaz enkripcije dok je na slici 2.7. prikazan postupak dekripcije podataka.



Sl. 2.6. ECB enkripcija



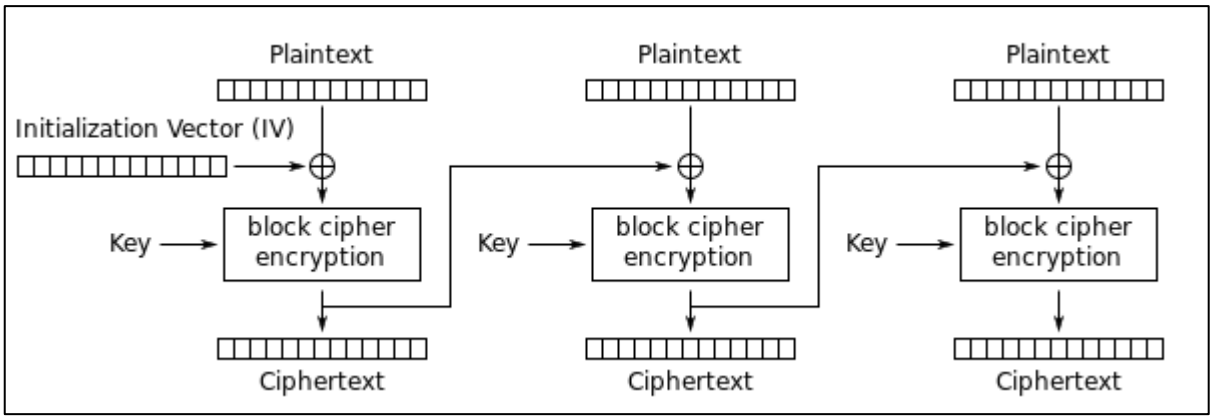
Sl. 2.7. ECB dekripcija

2.3.2. CBC

Drugi način rada je CBC (Cipher Block Chaining) i on je najčešće korišten način rada. Ovaj način uvodi znatno veću razinu sigurnosti u odnosu na ECB. To postiže uvođenjem dodatnog parametra – inicijalizacijskog vektora. Inicijalizacijski vektor je nasumično generiran niz koji pomoću kojega se postiže nasumičnost ulaznih podataka što dovodi do toga da ako se isti niz podataka enkriptira pomoću istog ključa, ali uz različit inicijalizacijski vektor rezultat kriptiranja neće biti isti. Upravo u tome CBC postiže veliki skok u sigurnosti u usporedbi s ECB načinom. U samom CBC načinu rada postupak se odvija na sljedeći način:

1) Ulazni podaci dijele se na jednake blokove
2) Provodi se operacija XOR (isključivo ili) nad prvim blokom podataka i inicijalizacijskim vektorom
3) XOR operacija se provodi nad sljedećim blokom ulaznih podataka i prethodnim kriptiranim blokom podataka

Ovim postupkom svaki kriptirani blok podataka ovisi o svim prethodnim blokovima što šansu za pojavljivanjem istog šifrata smanjuje na gotovo nepostojeću. Na slici 2.8. vidljiv je postupak enkripcije CBC načinom rada.

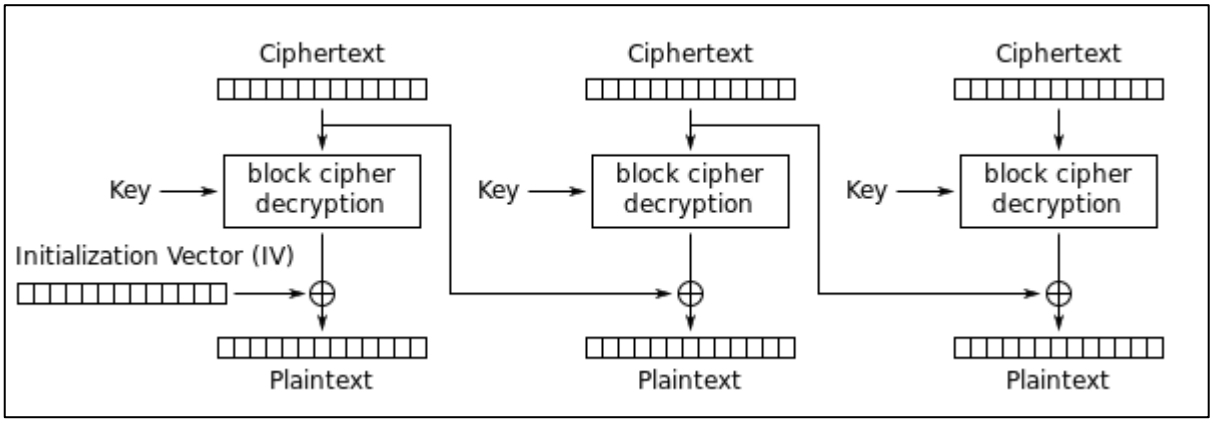


Sl. 2.8. CBC enkripcija

Na slici 2.9. vidljiv je postupak dekripcije podataka. U njemu se koristi isti inicijalizacijski vektor koji je korišten za enkripciju. U slučaju dekripcije provode se sljedeći koraci:

1) Prvi kriptirani blok podataka se dekriptira pomoću ključa
2) Tako dobiven niz se provlači kroz XOR s inicijalizaijskim vektorom te se dobiva prvi dekriptirani blok podataka
3) Svaki sljedeći kriptirani blok se provlači kroz XOR s prethodnim kriptiranim blokom

Može se primjetiti da će u slučaju krivog inicijalizacijskog vektora samo prvi blok biti krivo dekriptiran dok će ostali blokovi biti ispravno dekriptirani ukoliko se koristi ispravan ključ jer svi blokovi osim prvog ne ovise o inicijalizacijskom vektoru.

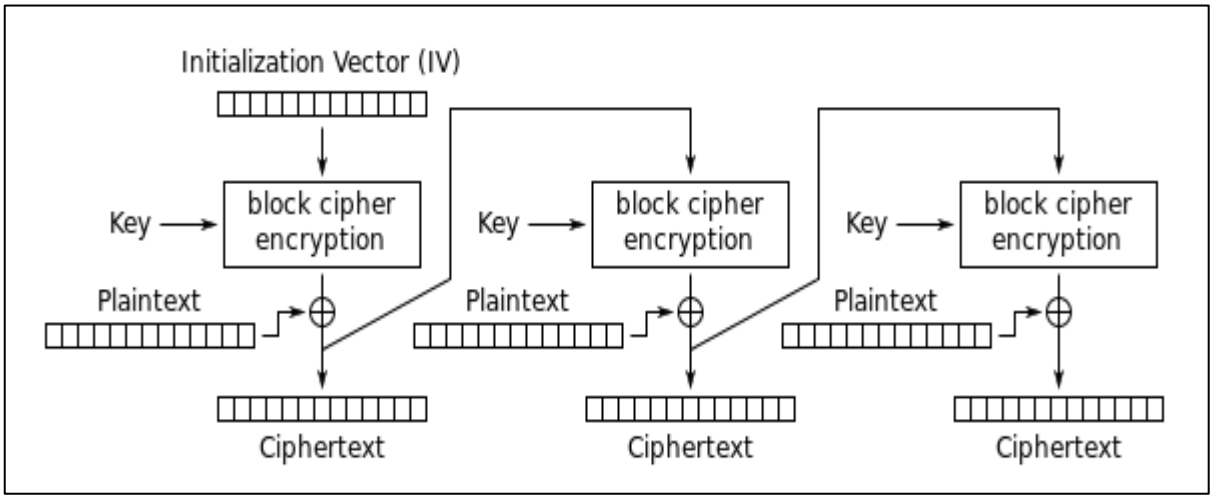


Sl. 2.9. CBC dekripcija

2.3.3. CFB

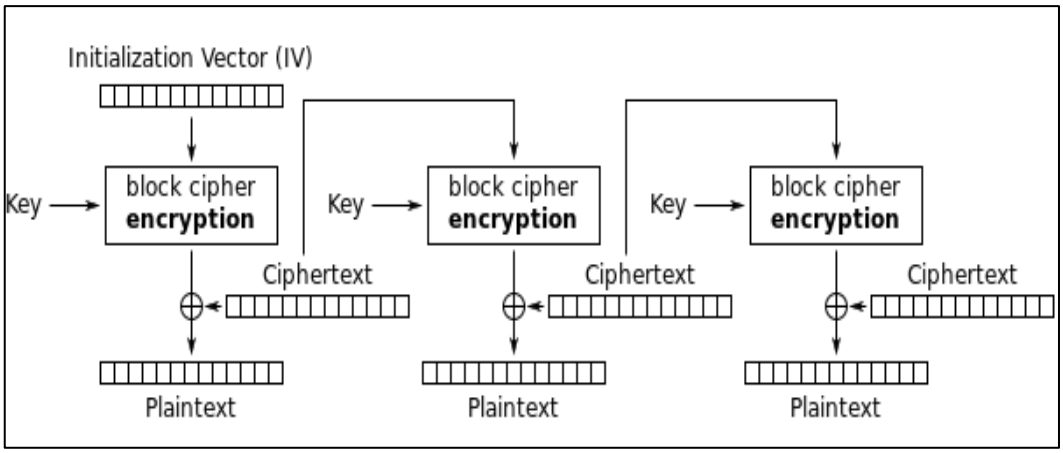
CFB (eng. *Cipher Feedback*) funkcionira na sličan način kao CBC. U ovom načinu također se koristi inicijalizacijski vektor i XOR operacije, ali na različitim mjestima. Postupak enkripcije vidljiv je na slici 2.10. i izvodi se na sljedeći način:

1) Inicijalizacijski vektor se enkriptira pomoću ključa
2) Nad dobivenim nizom i ulaznim blokom podataka se vrši XOR operacija te se dobiva prvi kriptirani blok
3) Dobiveni kriptirani blok se ponovno kriptira pomoću ključa te se rezultat provlači kroz XOR sa sljedećim ulaznim blokom te se ovaj korak ponavlja za svaki sljedeći blok



Sl. 2.10. CFB enkripcija

Postupak dekripcije u CFB načinu rada vidljiv je na slici 2.11. Velikim dijelom je identičan postupku enkripcije. Kao i kod enkripcije prvo se inicijalizacijski vektor kriptira pomoću ključa. Zatim se dobiveni niz preko XOR operacije uspoređuje te se dobiva prvi dekriptirani blok. Nakon toga se prvi kriptirani blok enkriptira pomoću ključa te se tako dobiveni niz preko XOR-a uspoređuje sa sljedećim kriptiranim blokom te se dobiva drugi dekriptirani blok. Ovaj korak ponavlja se sve dok se ne dođe do kraja niza podataka koje je potrebno dekriptirati.

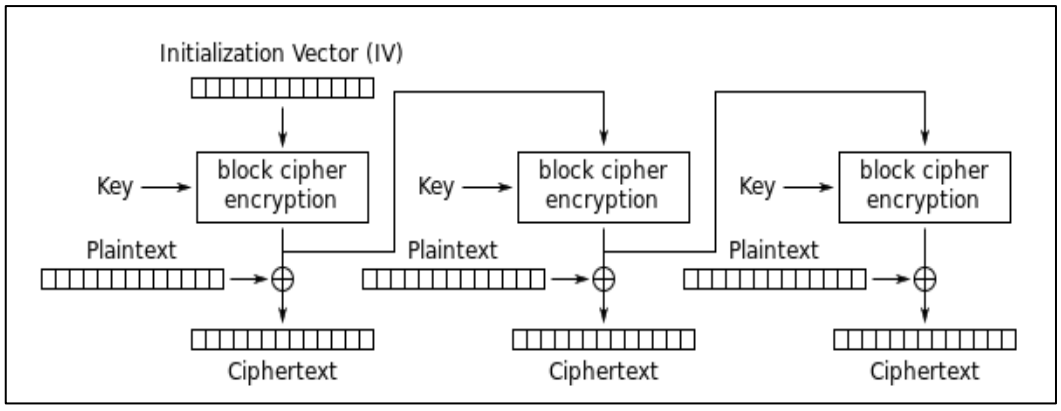


Sl. 2.11. CFB dekripcija

2.3.4. OFB

Sljedeći način rada koji će biti obrađen u ovom radu je OFB (eng. *Output Feeback*). Na slici 2.12. vidljiv je postupak enkripcije u OFB načinu:

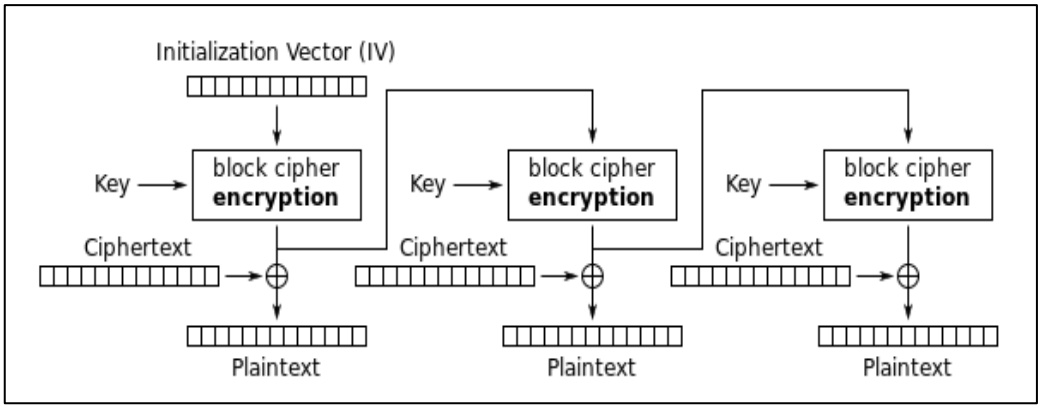
1) Pomoću inicijalizacijskog vektora i ključa stvara se niz podataka
2) Tako dobiveni niz se preko XOR-a uspoređuje s prvim blokom ulaznih podataka te se dobiva prvi kriptirani blok
3) Niz podataka dobiven prvim korako se pomoću ključa ponovno enkriptira te se preko XOR-a i sljedećeg ulaznog bloka dobiva drugi kriptirani blok
4) Niz dobiveni prethodnim korakom se koristi za enkripciju sljedećeg bloka i taj se postupak ponavlja do kraja niza ulaznih podataka



Sl. 2.12. OFB enkripcija

I kod dekripcije podataka u ovom načinu zapravo se koristi postupak enkripcije inicijalizacijskog vektora pomoću ključa. Daljnji postupak dekripcije analogan je postupku enkripcije. Pomoću niza dobivenog enkripcijom inicijalizacijskog vektora i prvog bloka kriptiranih podataka preko

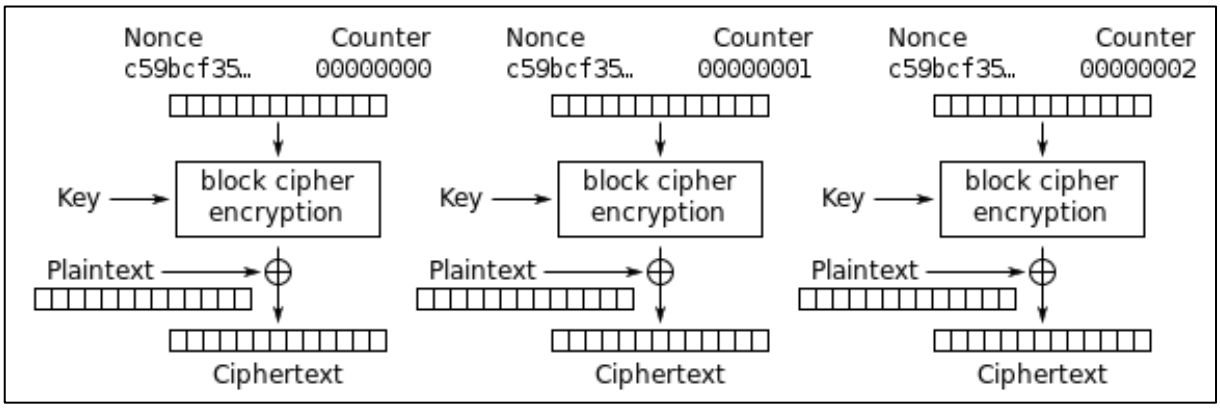
XOR-a se dobiva prvi blok dekriptiranih podataka. Niz dobiven enkripcijom inicijalizacijskog vektora se ponovno kriptira te primjenjuje XOR sa sljedećim blokom kirpitranih podataka te se taj postupak ponavlja do kraja niza kriptiranih podataka.



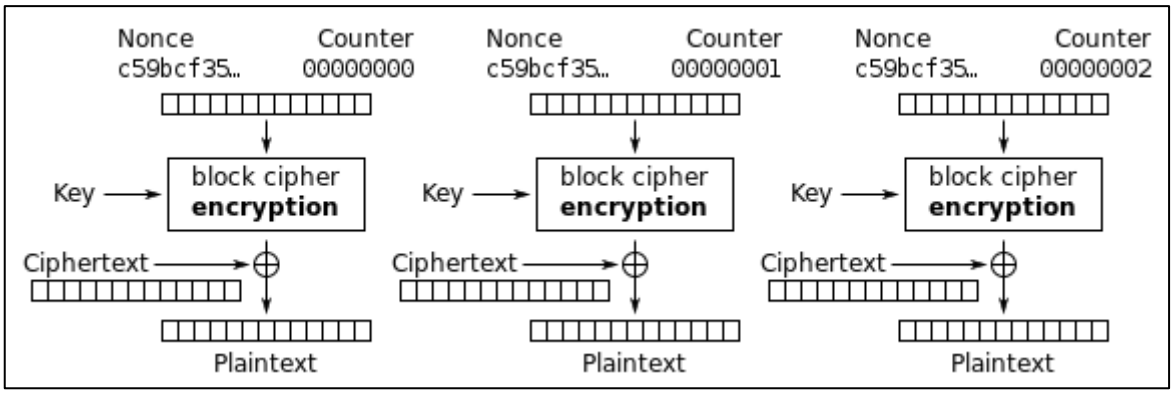
Sl. 2.13. OFB dekripcija

2.3.5. CTR

Posljednji način rada je CTR (eng. *Counter*). Ovaj način rada umjesto inicijalizacijskog vektora koristi brojač. Brojač može biti bilo koja funkcija koja jamči raznovrsnost odnosno jamči da se isti niz podataka neće pojaviti kroz veći broj ponavljanja. Najjednostavniji i najpopularniji brojač je onaj koji samo povećava vrijednost niza za jedan prilikom svakog novog bloka nad kojim se izvodi enkripcija. Osim brojača, prilikom enkripcije koristi se i nasumični niz podataka koji se za razliku od brojača ne mijenja kroz proces enkripcije. Kombinacijom ovih dvaju nizova postiže se jednostavnost, ali u isto vrijeme i određena razina sigurnosti budući da je potrebno poznavati oba parametra kako bi se moglo izvesti dešifriranje. Na slici 2.14. vidljiv je postupak enkripcije pomoću CTR načina rada, a na slici 2.15. je postupak dekripcije.



Sl. 2.14. CTR enkripcija



Sl. 2.15. CTR dekripcija

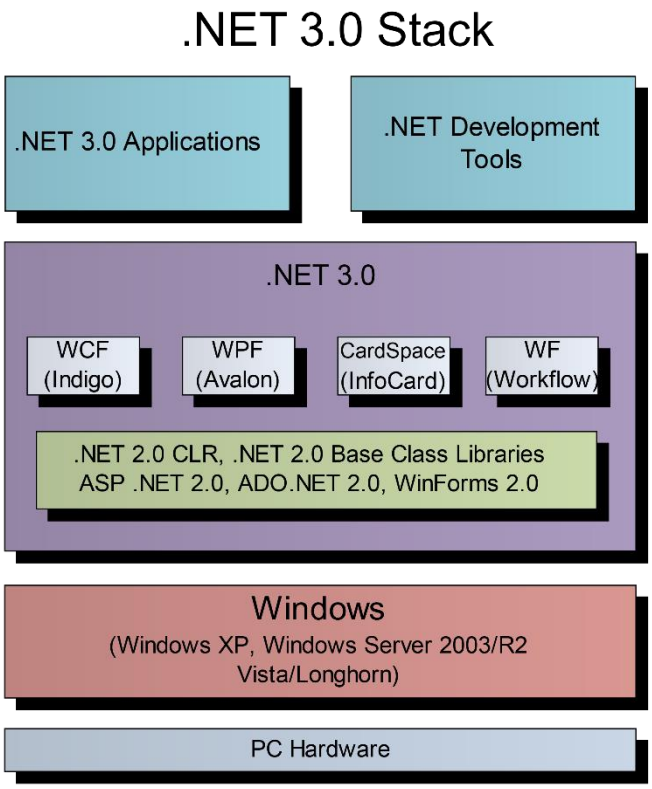
3. APLIKACIJA ZA SIGURNU RAZMJENU PORUKA

3.1. Pozadina aplikacije i korištene tehnologije

Prilikom planiranja izrade aplikacije u obzir su se morali uzeti zahtjevi koje aplikacija treba zadovoljavati te samim time i tehnologije i tehnike kojima se to može izvesti. Glavni zahtjevi na aplikaciju su sljedeći:

1) namijenjena računalima
2) slanje tekstualnih poruka
3) enkripcija poruka
4) slanje datoteka

Prvi uvjet koji se morao zadovoljiti je taj da je aplikacija namijenjena desktop računalima. Budući da je najrašireniji operacijski sustav na svijetu Windows - izbor je pao na njega. Postoji više mogućnosti odnosno jezika i tehnologija kojima se može napraviti aplikacija kompatibilna s Windows operacijskim sustavom, međutim najbolje rješenje se pokazalo u obliku C# programskog jezika s obzirom da on ima najbolju integraciju s Microsoftovim proizvodima pa tako i Windows operacijskim sustavima. Osim samog C# programskog jezika temelj ove aplikacije nalazi se u WCF (eng. *Windows Communication Foundation*) uslugama o kojima će biti riječi u sljedećem potpoglavlju. WCF je servisno orijentirani model koji omogućava programima komunikaciju preko mreže ili lokalno. Pojavio se u verziji 3 .NET Framework platforme. WCF je često korišten za implementaciju i razvijanje servisno orijentirane arhitekture koja je temeljena na servisima kao poveznicama među više klijenata. Na slici 3.1. vidljiva je struktura .NET 3 platforme i položaj WCF-a unutar nje. WCF se temelji na servisima i klijentima koji mogu koristiti usluge tih servisa. Klijentske aplikacije se na servis povezuju preko krajnjih točaka (eng. *endpoints*). Svaki servis ima definirane ugovore preko jedne ili više krajnjih točaka. Krajnje točke opisane su URL adresom, načinom povezivanja koji određuje protokol koji će se koristiti za prijenos podataka te ugovorom koji predstavlja sučelje (eng. *interface*) koje sadrži informacije o funkcijama koje su dostupne klijentskim aplikacijama. Ova tri parametra često se opisuju kao ABC (*adress, binding, contract*). Povezivanje određuje komunikacijski protokol koji će se koristiti za pristupanje servisu. Za najkorištenije protokole postoje predefinirana povezivanja unutar WCF-a. Neki od najpopularnijih protokola za pristupanje servisima su HTTP i TCP.



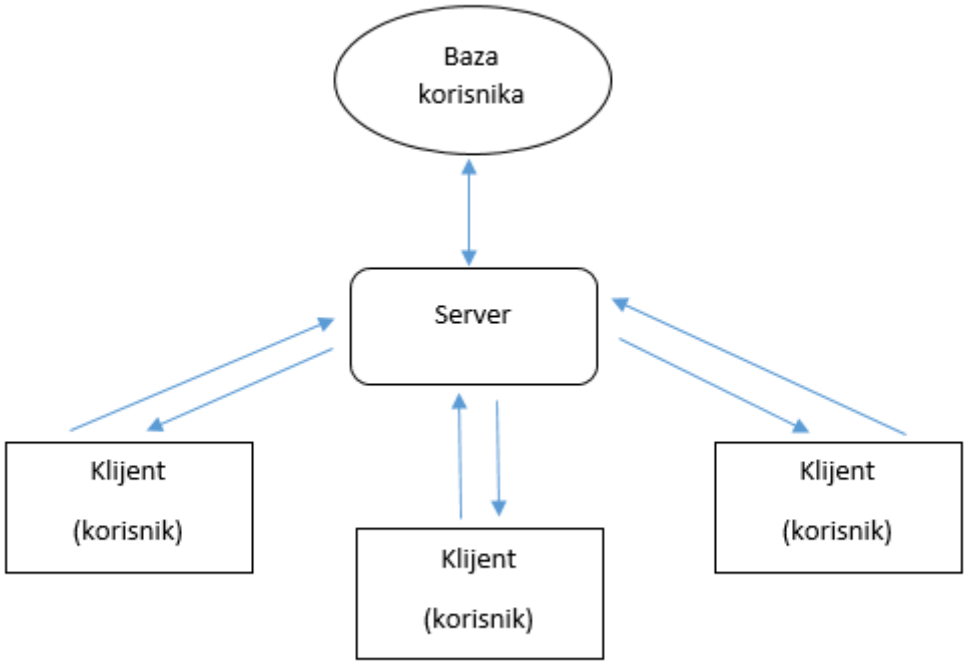
Sl. 3.1. .NET struktura

Sama komunikacija između krajnje točke i klijenta odvija se pomoću SOAP protokola odnosno SOAP omotnica (eng. *envelope*). SOAP omotnice su dokumenti formatirani u obliku XML-a koji time WCF servis čine neovisnim o platformi na kojoj se nalazi. Kako bi došlo do povezivanja klijenta i krajnje točke, klijent i server moraju imati kompatibilne krajnje točke.

Budući da je aplikacija zamišljena tako da ju koristi više korisnika te međusobno razmjenjuje tekstualne poruke i datoteke, neophodno je ostvariti i podsustav koji će se brinuti o samim korisnicima te upravljati njihovim aktivnostima. Kako bi se ostvario takav podsustav najbolje rješenje je baza podataka korisnika. Ta baza podataka sadržavat će osnovne podatke o korisnicima kao što su korisničko ime i lozinka te će služiti kao referenta točka za provjeru tih podataka prilikom korisnikova korištenja aplikacijom. Time će se ograničiti krug korisnika koji će moći koristiti aplikaciju te ona neće biti dostupna svima koji imaju pristup samoj aplikaciji već će za korištenje morati proći i određeni postupak provjere podataka. Baza podataka biti će SQL baza te će se svi upiti prema njoj odrađivati pomoću SQL jezika.

Osim baze podataka korisnika koja neće biti vidljiva samim korisnicima potrebno je osmisliti i korisnička sučelja preko kojih će korisnici pristupati funkcionalnostima aplikacije. To je

klijentski dio aplikacije i on je jedini vidljiv samim korisnicima. Također, potrebno je imati i posrednički dio sustava koji će koordinirati radom cjelokupnog sustava te služiti kao sučelje između klijenta i baze podataka koja se nalazi u pozadini te kao povezna točka između više različitih klijenata. Na slici 3.2. može se vidjeti shematski prikaz cijelog sustava koji uključuje sve komponente dosad spomenute u ovom poglavlju.



Sl. 3.2. Arhitektura aplikacije

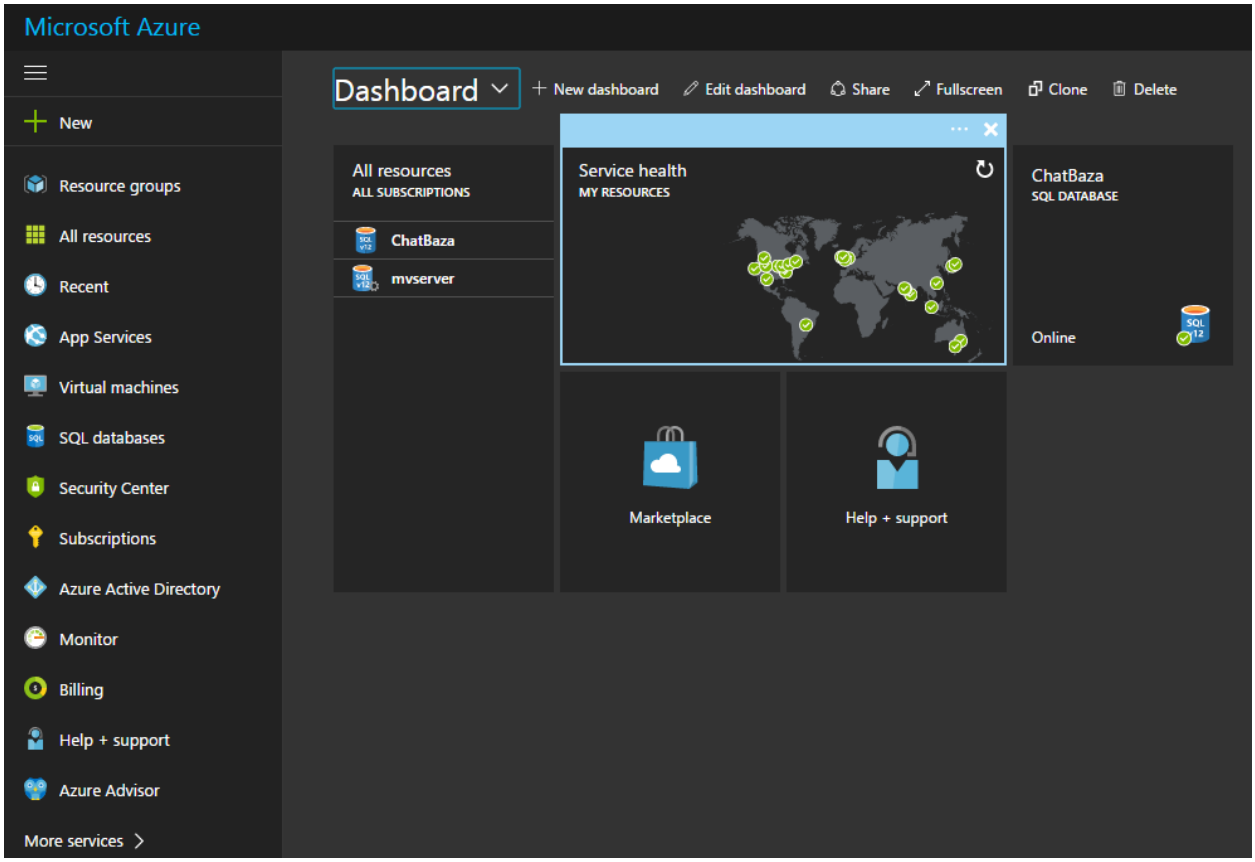
Kao što je vidljivo na slici 3.2. klijenti ne komuniciraju izravno međusobno nego se sva komunikacija odvija preko poslužiteljskog dijela aplikacije. Također, klijenti ne komuniciraju niti s bazom podataka izravno nego isto preko poslužitelja. U nastavku rada biti će detaljnije objašnjeni svi dijelovi sustava kao i veze među njima.

3.2. Izrada aplikacije

Nakon izrade plana aplikacije te okvirnih funkcija pojedinih elemenata sustava moglo se početi sa samom izradom aplikacije. Budući da je baza podataka koja će sadržavati korisnike zapravo najjednostavniji dio sustava prvo će biti objašnjen tijek njene izrade i funkcija.

3.2.1. Baza podataka korisnika

Kao što je spomenuto u prethodnom dijelu rada, baza podataka koja će sadržavati podatke o korisnicima biti će SQL baza. Kao najbolje rješenje za rješavanje ovog dijela sustava pokazao se Microsoft Azure program koji nudi besplatnu izradu i održavanje SQL baza podataka. Mogućnosti su ograničene, ali za potrebe ovoga rada i više nego dovoljne za ispunjavanje zahtjeva aplikacije. Za početak rada potrebno se prijaviti na Microsoft Azure Portal koji je vidljiv na slici 3.3.



Slika 3.3. Microsoft Azure Portal

Nakon prijave na Microsoft Azure Portal imamo mogućnost biranja servera na kojem će se nalaziti baza podataka te stvaranja same baze podataka. Prilikom postupka stvaranja baze podataka potrebno je postaviti parametre baze podataka koji će se poslije koristiti za pristup istoj bazi. Neki od tih parametara su ime baze podataka, lozinka, ime servera na kojem se baza nalazi. Na temelju zadanih parametara stvara se tzv. *Connection String* koji će se poslije koristiti u programskom kodu za spajanje na tu bazu, primjer se može vidjeti na slici 3.4.

```
Server=tcp:mvserver.database.windows.net,1433;Initial Catalog=ChatBaza;
Persist Security Info=False;User ID={your_username};Password={your_password};
MultipleActiveResultSets=False;Encrypt=True;
TrustServerCertificate=False;Connection Timeout=30;
```

Sl. 3.4. Connection String

Nakon stvaranja baze podataka moguće se spojiti na istu te upravljati podacima koji se nalaze u njoj. Budući da je baza tek stvorena ona je prazna te je potrebno prvo stvoriti strukturu baze te nakon toga popuniti tu strukturu stvarnim podacima. Postoji više načina za spajanje na bazu podataka u ovom slučaju. Jedan od njih je i preko razvojnog okruženja Microsoft Visual Studio. U samom okružju potrebno je stvoriti vezu prema bazi podataka koristeći prije spomenuti Connection String s ispravnim podacima za spajanje. Nakon što je veza s bazom uspostavljena moguće je uređivati bazu prema potrebama.

Za potrebe aplikacije obrađene u ovom radu baza podataka je vrlo jednostavna. Ona sadrži samo jednu tablicu s dva stupca, a to su korisničko ime i lozinka svakog korisnika. Upit za stvaranje takve tablice u bazi prikazan je na slici 3.5.

```
1  CREATE TABLE users (
2      username varchar(255),
3      password varchar(255)
4  );
```

Sl. 3.5. Stvaranje tablice u bazi podataka

Nakon kreiranja tablice moguće je u nju unjeti podatke koji će se koristiti u aplikaciji. Unos podataka moguće je doraditi preko SQL upita kao i kreiranje tablica, a moguće je i ručno unijeti podatke preko grafičkog sučelja koristeći Microsoft Visual Studio razvojno okruženje. U ovom slučaju podaci su ručno unešeni te se primjer tablice može vidjeti na slici 3.6. kao što je vidljivo na slici tablica se sastoji od četiri stupca, a to su identifikacijski broj, korisničko ime i lozinka svakog korisnika te status korisnika ovisno o tome je li prijavljen u aplikaciju ili ne. Lozinka nije upisana u obliku običnog teksta nego u kriptiranom obliku kako bi se spriječilo čitanje iste u slučaju neovlaštenog pristupa bazi. O tom mehanizmu više će riječi biti u sljedećim poglavljima rada.

	ID	UserName	Password	IsOnline
▶	1	matko	c70b5dd9ebfb6...	False
	2	ivan	7b6ad79b346fb...	False
	5	josip	7b6ad79b346fb...	False
*	NULL	NULL	NULL	NULL

Sl. 3.6. Tablica korisnika

Unosom podataka u bazu završen je dio aplikacije koji se tiče same baze podataka te se već nakon ovih par koraka ona može koristit na način da se prema njoj šalju upiti. Ovisno o vrsti upita baza će vratiti određenu vrstu odgovora prema kojem se može dalje postupati u samoj aplikaciji. Bazu podataka uvijek je moguće na isti način izmijeniti, dodati ili obrisati nove korisnike. Naravno, kako bi se pristupilo bazi potrebno je znati podatke za pristup istoj. Ovime je objašnjen postupak stvaranja baze podataka i unosa podataka u istu te se u nastavku rada govori o izvedbi ostatka aplikacije odnosno poslužitelja i klijenta.

3.2.2. Konfiguracija poslužitelja i klijenta

Poslužitelj ili server može se definirati kao aplikacija odnosno servis koji je zadužen za koordinaciju cjelokupnim sustavom te za međusobnu komunikaciju između svih ostalih elemanata u sustavu. Poslužitelj se uvijek pokreće prije klijenata kako bi oni mogli uspostaviti vezu s njim te preko njega s bazom podataka u kojoj se nalaze njihovi podaci. U slučaju aplikacije za razmjenu poruka poslužitelj izvršava dvije glavne uloge:

1) provjera točnosti podataka korisnika prilikom prijave u aplikaciju
2) razmjena poruka i datoteka između klijenata

Kako bi se postigle ove funkcionalnosti poslužitelja bilo je potrebno definirati sučelja koristeći WCF načela. Server je prvo potrebno opisati adresom te protokolom kojim će se izvršavati razmjena podataka. Na slici 3.7. vidljiva je konfiguracija servisa.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <configuration>
3      <startup>
4          <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
5      </startup>
6
7    <system.serviceModel>
8      <services>
9        <service name="SafeChatServer.Server">
10         <endpoint address="net.tcp://localhost:9000/ChattingService"
11           binding="netTcpBinding" bindingConfiguration="chatServerBinding"
             name="ChattingServiceEndPoint"
12           contract="Interfaces.IService" />
13       </service>
14     </services>
15     <bindings>
16       <netTcpBinding>
17         <binding name="chatServerBinding" maxBufferSize="2147483647"
           maxReceivedMessageSize="2147483647" closeTimeout="01:50:00"
           openTimeout="01:50:00" sendTimeout="01:50:00" receiveTimeout=
           "01:50:00" >
18           <readerQuotas maxDepth="128" maxStringContentLength="8388608"
             maxArrayLength="10000000" maxBytesPerRead="4096"
             maxNameTableCharCount="16384" />
19         </binding>
20       </netTcpBinding>
21     </bindings>
22   </system.serviceModel>
23 </configuration>
```

Sl. 3.7. Konfiguracija servisa

Sa slike 3.1. možemo iščitati tri parametra koja određuju WCF servis, a to su adresa, način povezivanja te ugovor. Ovi parametri vidljivi su u 10. liniji koda. Adresa je postavljena na vrijednost “net.tcp://localhost:9000/ChattingService“. To znači da se servisu pristupa preko porta 9000 na lokalnom računalu na kojem je servis pokrenut. Način povezivanja određen je parametrom “netTcpBinding“ te se može iz njega iščitati da se za izmjenu podataka koristi TCP protokol. Posljednji parametar koji određuje ovaj servis je ugovor čija je vrijednost putanja do datoteke unutar projekta u kojoj su definirane funkcije kojima će klijenti moći pristupati preko poslužitelja. Programski kod kojim je opisano sučelje poslužitelja vidljiv je na slici 3.8. Sučelje predstavlja samo deklaraciju funkcija bez njihove implementacije. Implementacija tih funkcija nalazi se u zasebnom dijelu programskog koda koji nastaje nasljeđivanjem ovog sučelja te sadrži sve funkcije koje su ovdje deklarirane te im naknadno dodaje i samu funkcionalnost odnosno opisuje njihov rad.

```
namespace Interfaces
{
    [ServiceContract(CallbackContract = typeof(IClient))]

    public interface IService
    {
        [OperationContract]
        LoginResponse Login(string korisnickoIme, string key = "", int
        keyLenght = 0, int mode =-1);

        [OperationContract]
        void PosaljiPoruku(string poruka, string korisnickoIme);

        [OperationContract]
        void PosaljiDatoteku(byte[] datoteka,string filename, string
        korisnickoIme);

        [OperationContract]
        int Logout();

        [OperationContract]
        bool SetKeyDetails(string key, int mode, int lenght, string username);
    }
}
```

Sl. 3.8. Sučelje servisa

Klijenti su aplikacije koje će biti vidljive korisnicima i preko kojih će korisnici komunicirati sa poslužiteljem te onda dalje preko njega i sa bazom podataka te s ostalim klijentima. Osnovna uloga klijentskih aplikacija je omogućiti pristup poslužitelju, a kako bi se to postiglo potrebno je i klijentske aplikacije konfigurirati isto kao što je bilo potrebno konfigurirati i poslužiteljsku aplikaciju. Kako bi se klijenti mogli spojiti na servis konfiguracije poslužitelja i klijenta moraju biti usklađene. Na slici 3.9. vidljiva je konfiguracija klijenta.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
    </startup>

  <appSettings>
    <add key="ConnectionString" value="Data
    Source=mvserver.database.windows.net;Initial Catalog=ChatBaza;Integrated
    Security=False;User ID=mvujnovac;Password=bEPl6IbKOfuj;Connect
    Timeout=60;Encrypt=False;TrustServerCertificate=True;ApplicationIntent=Read
    Write;MultiSubnetFailover=False" />
  </appSettings>

  <system.serviceModel>
    <client>
      <endpoint address="net.tcp://localhost:9000/ChattingService"
          binding="netTcpBinding" bindingConfiguration="chatServerBinding"
          contract="Interfaces.IService" name="ChattingServiceEndPoint" kind=
          "" endpointConfiguration="" />
    </client>
    <bindings>
      <netTcpBinding>
        <binding name="chatServerBinding" maxBufferSize="2147483647"
        maxReceivedMessageSize="2147483647" closeTimeout="01:50:00"
        openTimeout="01:50:00" sendTimeout="01:50:00" receiveTimeout=
        "01:50:00" >
          <readerQuotas maxDepth="128" maxStringContentLength="8388608"
          maxArrayLength="10000000" maxBytesPerRead="4096"
          maxNameTableCharCount="16384" />
        </binding>
      </netTcpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

Sl. 3.9. Konfiguracija klijenta

Kao što je vidljivo na prethodnoj slici, adresa i način povezivanja u klijentovoj konfiguraciji su identični onima u konfiguraciji poslužitelja. To omogućuje klijentima da znaju točno određenu adresu na koju moraju slati svoje zahtijeve. U ovom slučaju to je adresa lokalnog računala na kojemu je poslužitelj pokrenut, točnije na portu 9000 tog računala. Parametar koji se razlikuje od onoga u konfiguraciji klijenta je naziv ugovora i u ovom slučaju on je predstavljen putanjom do datoteke u kojoj su opisane funkcije koje će se obavljati na klijentu. Popis tih funkcija vidljiv je na slici 3.10. te se i u ovom slučaju radi samo u deklaraciji funkcija bez njihove implementacije koja se nalazi u drugom dijelu programskog koda koji koristi sučelje kao izvor iz kojeg nasljeđuje određena svojstva, u ovom slučaju funkcije zajedno s njihovim parametrima.

```
 8   namespace Interfaces
 9   {
10       public interface IClient
11       {
12           [OperationContract]
13           void DohvatiPoruku(string poruka, string korisnickoIme);
14
15           [OperationContract]
16           void DohvatiDatoteku(byte[] datoteka, string naziv, string
             korisnickoIme);
17
18           [OperationContract]
19           void SkipConfigPage(KeyDetails keyDetails);
20
21       }
22   }
```

Sl. 3.10. Sučelje klijenta

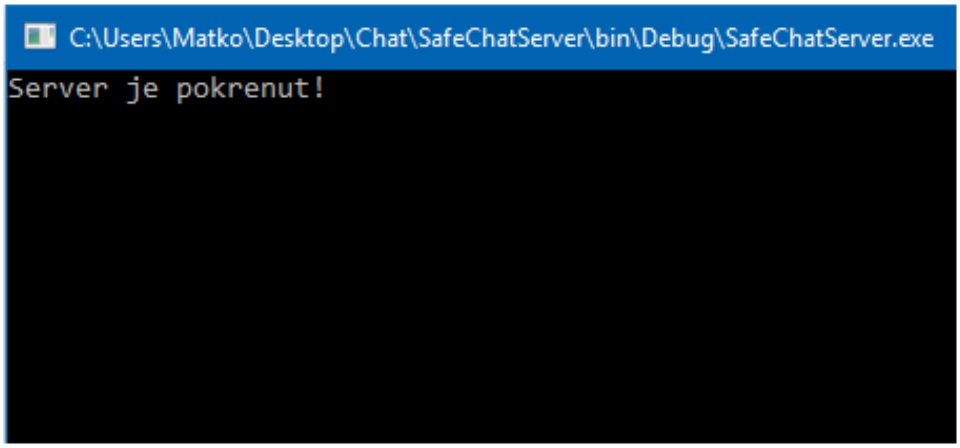
3.2.3. Implementacija poslužitelja

Nakon konfiguracije poslužitelja bilo je potrebno ostvariti samu aplikaciju koja će biti pokretana svaki puta kada se klijenti budu željeli spojiti na servis. Poslužitelj će u našem slučaju biti konzolna aplikacija koja će biti pokretana na lokalnom računalu te će se svi klijenti spajati na nju preko tog računala. Uvjet da bi se klijenti mogli spojiti na poslužitelja je da se nalaze u istoj lokalnoj mreži te da su ispravno konfigurirani odnosno da znaju koje računalo u mreži trebaju tražiti i na kojem portu će biti izvršavani njihovi zahtjevi. Na slici 3.11. vidljiv je programski kod kojim je ostvarena poslužiteljska aplikacija odnosno server. Ovaj dio koda ne sadržava puno linija te je vrlo jednostavan odnosno obavlja vrlo jednostavnu funkciju, a to je stvaranje serverske aplikacije u trenutku pokretanja konzolne aplikacije. U liniji 16 vidljiva je inicijalizacija novog objekta koji je tipa Server(). Nakon toga je potrebno tom serveru dati domaćina na kojem će biti dostupan ostatku aplikacije odnosno klijentima. To je učinjeno u liniji 17 pomoću ServiceHost klase unutar C# jezika koja služi upravo tome. Ovime se postiže da domaćin servisu postaje lokalno računalo na kojem je poslužiteljska aplikacija pokrenuta. Nakon stvaranja samog servisa potrebno je otvoriti vezu prema njemu te je to napravljeno u liniji 19 pomoću funkcije Open() koja je implementirana unutar ServiceHost klase. Preostaje samo dati do znanja da je cijeli posuptak stvaranja i otvaranja servisa bio uspješan te se to čini jednostavnim ispisivanjem linije unutar konzolnog prozora koja govori da je server uspješno pokrenut.

```
 8   namespace SafeChatServer
 9   {
10       class Program
11       {
12           public static Server _server;
13
14           static void Main(string[] args)
15           {
16               _server = new Server();
17               using (ServiceHost host = new ServiceHost(_server))
18               {
19                   host.Open();
20                   Console.WriteLine("Server je pokrenut!");
21                   Console.ReadLine();
22               }
23           }
24       }
25   }
```

Sl. 3.11. Server programski kod

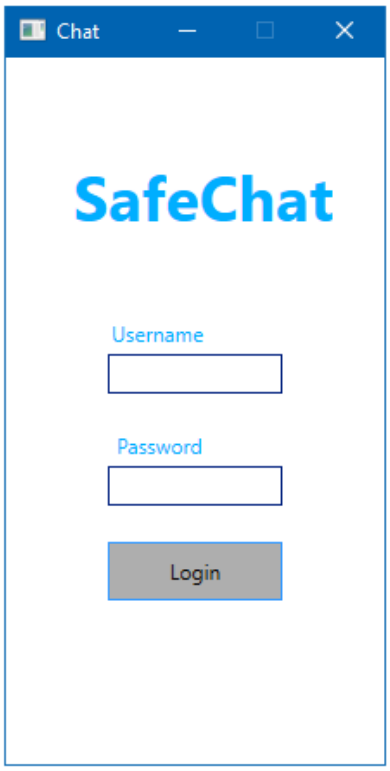
Nakon pokretanja aplikacije na ekranu možemo vidjeti prozor sa slike 3.12. Ovime je server uspješno stvoren i pokrenut te je sad moguće pokrenuti klijente koji će se spajati na njega te njegovim posredništvom izvršavati određene radnje s bazom podataka te ostalim klijentima spojenima na isti server.



Sl. 3.12. Server aplikacija

3.2.4. Implementacija klijenta

Nakon pokretanja poslužitelja ostavreni su uvjeti za pokretanje klijenta i njihovo povezivanje na isti. U slučaju da poslužitelj nije pokrenut, a klijent se pokuša prijaviti u sustav dolazi do iznimke te se aplikacija prisilno zaustavlja. Zbog toga je uvijek prvo potrebno pokrenuti poslužiteljsku aplikaciju. Prilikom pokretanja klijenta na ekranu se pojavljuje početni prozor koji je vidljiv na slici 3.13.



Sl. 3.13. Klijentska aplikacija

Kao što je vidljivo na slici iznad, početni prozor klijentske aplikacije sadrži tri funkcionalna elementa. To su polje za unos korisničkog imena, polje za unos lozinke te gumb za prijavu. Prilikom pritiska gumba za prijavu pokreće se programski kod sa slike 3.14.

```
67  private void LoginButton_Click(object sender, RoutedEventArgs e)
68  {
69      if (DBConnection.DBLogin(UserNameTextBox.Text.ToString(),
        PasswordBox.Password.ToString()) == 1)
70      {
71          korisnik = UserNameTextBox.Text.ToString();
72
73          var result = Server.Login(korisnik);
74
75          if (result.Success)
76          {
77              PasswordBox.Visibility = Visibility.Hidden;
78              LoginButton.Visibility = Visibility.Hidden;
79              label.Visibility = Visibility.Hidden;
80              label1.Visibility = Visibility.Hidden;
81              UserNameTextBox.Visibility = Visibility.Hidden;
82              label2.Visibility = Visibility.Hidden;
83              keyLabel.Visibility = Visibility.Visible;
84              keyTextBox.Visibility = Visibility.Visible;
85              ModeComboBox.Visibility = Visibility.Visible;
86              ModeLabel.Visibility = Visibility.Visible;
87              keySizeComboBox.Visibility = Visibility.Visible;
88              keySizeLabel.Visibility = Visibility.Visible;
89              ConfirmButton.Visibility = Visibility.Visible;
90
91              LoginButton.IsDefault = false;
92              SendButton.IsDefault = true;
93
94              if (result.KeyDetails != null)
95              {
96                  this.KeyDetails = result.KeyDetails;
97                  HideConfirmDetails();
98              }
99          }
100         else
101         {
102             MessageBox.Show("Već ste prijavljeni!");
103         }
104
105     }
106 }
```

Sl. 3.14. Programski kod klijenta

Kao što je vidljivo na slici, nakon pritiskanja gumba za prijavu pokreće se funkcija DBLogin koja se nalazi u klasi DBConnection. Ta funkcija kao parametre prima korisničko ime i lozinku iz polja za unos tih podataka s početnog prozora klijentske aplikacije. Rezultat izvršavanja te funkcije odmah se provjerava pomoću "if" petlje te se ovisno o njemu izvršava programski kod. U slučaju uspješne prijave izvršava se dio programskog koda od linije 71 do linije 99 dok se u

slučaju da je korisnik već prijavljen izvršava 102. linija koda. Kako bi znali o čemu ovisi rezultat funkcije DBLogin potrebno je objasniti programski kod te funkcije koji je vidljiv na slici 3.15.

```
12  class DBConnection
13  {
14  public static int DBLogin(string korisnickoIme, string lozinka)
15  {
16      try
17      {
18
19
20          using (var conn = new SqlConnection(Helper.GetValueFromConfig(
            "ConnectionString")))
21          {
22              using (var cmd = new SqlCommand("pr_User_Login", conn))
23              {
24                  cmd.CommandType = CommandType.StoredProcedure;
25                  cmd.Parameters.AddWithValue("@Username", korisnickoIme);
26                  cmd.Parameters.AddWithValue("@Password", Helper.
                    SHA512Hash(lozinka));
27                  conn.Open();
28
29                  int UserID = Int32.Parse(cmd.ExecuteScalar().ToString());
30                  if (UserID > 0)
31                  {
32                      return 1;
33                  }
34
35                  conn.Close();
36              }
37          }
38      }
39      catch
40      {
41          MessageBox.Show("Unijeli ste krive podatke!");
42      }
43      return 0;
```

Sl. 3.15. Spajanje na bazu podataka

Prilikom pokušaja prijave korisnika najprije je potrebno ostvariti vezu s bazom podataka. Upravo to se izvršava u liniji 20 programskog koda sa slike i to pomoću C# klase SqlConnection koja kao parametar prima tzv. “Connection string“ o kojem je riječi bilo u prethodnom dijelu rada koji je govorio o bazi podataka. Taj niz znakova sadrži sve bitne informacije koje su potrebne za pristup bazi podataka. Kako se ti podaci ne bi vidjeli unutar samo programskog koda te kako bi aplikacija bila konfigurabilna (u slučaju promjene podataka za pristup bazi) taj podatak zapisan je u konfiguracijskoj datoteci te se odande dohvaća pomoću funkcije GetValueFromConfig čiji je kod vidljiv na slici 3.16. Ova funkcija jednostavno prolazi kroz konfiguracijsku datoteku te dohvaća vrijednost koja se nalazi pod imenom koje primi kao parametar, u ovom slučaju to je “Connection string“.

```
30      public static string GetValueFromConfig(string key)
31      {
32          return ConfigurationManager.AppSettings[key].ToString();
33      }
```

Sl. 3.16. Dohvaćanje Connection Stringa

Kada se uspostavi veza s bazom podataka moguće je izvršavati procedure koje se nalaze unutar te baze podataka. Te procedure zapisane su u SQL jeziku, ali se njima može manipulirati pomoću C# ugrađenih klasa i funkcija. U liniji koda 22 na slici vidljivo je da se stvara novi objekt tipa SqlCommand pomoću kojega će se izvršiti procedura u bazi podataka koja ima naziv "*pr_User_Login*". Ova procedura vidljiva je na slici 3.17.

```
1  CREATE PROCEDURE pr_User_Login
2          @UserName nvarchar(50),
3          @Password nvarchar(max)
4      AS
5      BEGIN
6          SET NOCOUNT ON;
7          SELECT ID FROM Users WHERE UserName = @Username AND Password = @Password
8      END
```

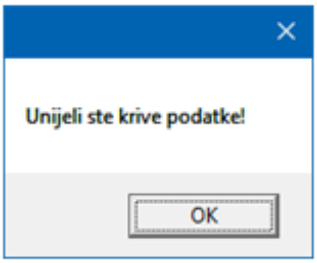
Sl. 3.17. Procedura za provjeru korisnika

U sljedećim linijama definira se vrsta procedure te joj se prosljeđuju parametri, u ovom slučaju korisničko ime i lozinka. Korisničko ime predaje se u obliku običnog niza znakova dok se za lozinku poziva funkcija SHA512Hash. Ova funkcija lozinku u teksutalnom obliku pretvara u tzv. „hash" oblik odnosno pomoću algoritma SHA512 nizom operacija prebacuje običan tekst u niz znakova koje je nemoguće dekriptirati natrag u prvotni oblik. Lozinka se u tom kriptiranom obliku uspoređuje s lozinkom u bazi podataka korisnika gdje je također pohranjena u kriptiranom obliku. Ovime se povećava razina sigurnosti aplikacije budući da lozinka u nijednom trenutku ne biva zapisana u memoriju u izvornom obliku niti se bilo koja operacija obavlja nad lozinkom u izvornom obliku. Funckija koja odrađuje „hashiranje" vidljiva je na slici 3.18.

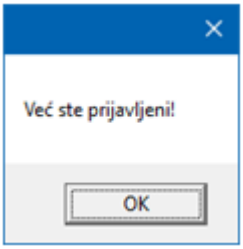
```
15      public static string SHA512Hash(string inputString)
16      {
17          SHA512 sha512 = SHA512Managed.Create();
18          byte[] bytes = Encoding.UTF8.GetBytes(inputString);
19          byte[] hash = sha512.ComputeHash(bytes);
20          StringBuilder sb = new StringBuilder();
21
22          for (int i = 0; i < hash.Length; i++)
23          {
24              sb.Append(hash[i].ToString("x2"));
25          }
26
27          return sb.ToString();
28      }
```

Sl. 3.18. Hash funkcija

Nakon pripreme procedure za izvođenje potrebno je otvoriti vezu prema bazi podataka te nakon toga izvršiti proceduru. Izvršenje procedure vidljivo je u liniji 22 na slici 3.8. Ova linija koda sprema broj rezultata koje je procedura pronašla u bazi podataka. Taj broj može biti 0 ukoliko korisnik ne postoji u bazi ili ukoliko su uneseni pogrešni podaci. U tom slučaju na ekranu će se izvršiti kod iz linije 41 sa slike 3.8. te će se na ekranu pojaviti prozor sa slike 3.19. Oba polja za unos su obavezna te ne smiju biti ostavljena prazna inače će pritiskanje na gumb za prijavu izbaciti prozorčić s greškom. U slučaju da je korisnik već prijavljen u bazu podataka varijabla „result“ neće zadovoljavati uvjet te će se ne ekranu pojaviti prozor s porukom koji je vidljiv na slici 3.20.

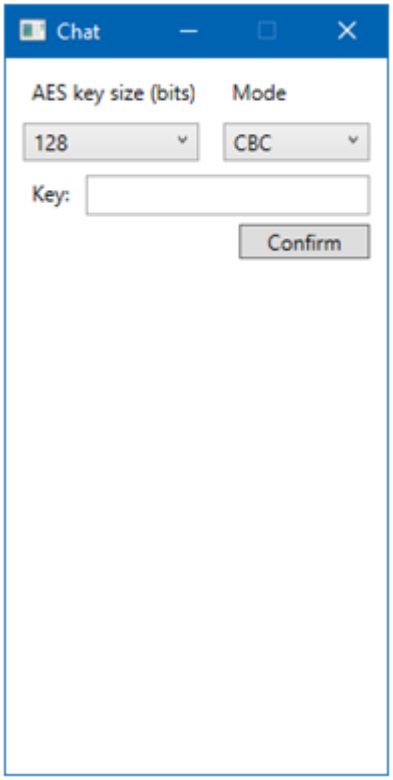


Sl. 3.19. Krivi podaci



Sl. 3.20. Korisnik je već prijavljen

U slučaju kada je korisnik pronađen i ima ispravno upisane podatke procedura će vratiti vrijednost veću od 0, odnosno vrijednost 1 budući da u bazi ne postoji više identičnih korisnika. U tom slučaju će i funkcija *DBLogina* vratiti vrijednost 1 funkciji koja ju je pozvala, a to je funkcija koja se izvršava pritiskom na tipku za prijavu, ta funkcija je vidljiva na slici 3.14. Time će se izvršiti petlja koja počinje u 75. liniji koda sa slike. Prvo će se promijeniti grafičko sučelje aplikacije, odnosno elementi koji služe za prijavu korisnika će prestati biti vidljivi te će se pojaviti elementi za odabir parametara enkripcije. Ovaj prozor vidljiv je na slici 3.21. Kao što je vidljivo moguće je odabrati duljinu ključa AES algoritma, način rada te sami ključ koji će biti korišten pri enkripciji poruka.



Sl. 3.21. Odabir parametara enkripcije

Prvi parametar koji je moguće izabrati je duljina ključa o kojoj ovisi složenost samog algoritma. Što je ključ dulji to će se više operacija izvršiti nad podacima nad kojima će se izvršavati enkripcija te će se time povećati i razina sigurnosti. Moguće je odabrati duljine ključeva od 128, 192 i 256 bita. Drugi parametar koji je moguće odabrati je način rada algoritma. Ova aplikacija podržava dva načina rada, a to su ECB i CBC. Na kraju je potrebno upisati i željeni ključ pomoću kojega će se izvršavati sve daljnje operacije u sklopu enkripcije. Pritiskom na tipku za potvrdu odabira izvršava se kod sa slike 3.22.

```
150     private void ConfirmButton_Click(object sender, RoutedEventArgs e)
151     {
152
153
154         HideConfirmDetails();
155
156         key = keyTextBox.Text;
157         ComboBoxItem typeItem = (ComboBoxItem)keySizeComboBox.SelectedItem;
158         string value = typeItem.Content.ToString();
159         keyLenght = int.Parse(value);
160         ComboBoxItem modei = (ComboBoxItem)ModeComboBox.SelectedItem;
161         string v = modei.Content.ToString();
162         switch (v)
163         {
164             case "CBC":
165                 mode = 1;
166                 break;
167             case "EBC":
168                 mode = 2;
169                 break;
170         }
171
172         Server.SetKeyDetails(key, mode, keyLenght, this.korisnik);
173         this.KeyDetails = new KeyDetails { key = key, mode = mode, keyLenght = keyLenght };
174     }
```

Sl. 3.22. Postavljanje parametara enkripcije

Prvo se izvršava funkcija koja sakriva elemente za odabir parametara te se prikazuju elementi koji će služiti za slanje poruka i fileova. Nakon toga se odabrani parametri spremaju u objekt koji se stvara iz klase vidljive na slici 3.23. Taj objekt će od trenutka stvaranja ostati nepromjenjiv te će se prilikom prijave drugih korisnika u aplikaciju provjeravati koje vrijednosti sadrži. Ukoliko je korisnik koji se prijavljuje prvi, parametri koje on odabere će se koristiti za svu daljnju enkripciju te ostali korisnici neće imati mogućnost odabira parametara nego će nakon prijave odmah biti preusmjereni na dio aplikacija koji služi za slanje poruka i datoteka. Tek prilikom gašenja servera objekt se briše iz memorije te se nakon ponovnog pokretanja servera pruža mogućnost korisnicima da odabiru parametre enkripcije.

```
20      public class KeyDetails
21      {
22          public string key { get; set; }
23
24          public int mode { get; set; }
25          public int keyLenght { get; set; }
26      }
27  }
```

Sl. 3.23. Parametri enkripcije

Nakon pohranjivanja parametara enkripcije aplikacija je spremna za rad odnosno za slanje poruka i datoteka. Na slici 3.24. vidljiv je prozor aplikacije u kojemu je moguće izvršiti slanje poruka i datoteka. On se sastoji od okna za prikaz poruka, polja za unos poruka koje se šalju, gumba za slanje poruka te gumba za slanje datoteka.

Sl. 3.24. Prozor za slanje poruka i datoteka

Prilikom pritiska na gumb za slanje poruke izvršava se kod sa slike 3.25. Prva provjera koja se obavlja je sadržaj polja za unos poruke. Ono ne smije biti prazno odnosno nemoguće je poslati praznu poruku. Ukoliko polje sadrži neke znakove izvršava se daljnji dio koda. Znakovi iz polja se spremaju u varijablu u obliku stringa. Nakon toga poziva se funkcija za enkripciju teksta koja će biti opisana u kasnijem dijelu rada.

```
133     private void SendButton_Click(object sender, RoutedEventArgs e)
134     {
135
136         if (!string.IsNullOrWhiteSpace(MessageTextBox.Text.ToString()))
137         {
138             var msg = MessageTextBox.Text.ToString();
139             var e_msg = Helper.EncryptText(msg, KeyDetails);
140
141             Server.PosaljiPoruku(e_msg, korisnik);
142             PreuzmiPoruku(MessageTextBox.Text, "You");
143             MessageTextBox.Text = "";
144         }
145     }
```

Sl. 3.25. Slanje poruke

Nakon što je poruka enkriptirana poziva se funkcija servera koja služi za slanje poruke svim trenutno prijavljenim korisnicima. Kao što je vidljivo na slici 3.25. ova funkcija kao parametre prima samu poruku i korisničko ime pošiljatelja. Funkcija prolazi kroz sve prijaveljene korisnike te na svakom klijentu poziva funkciju *DohvatiPoruku* te njoj predaje parametre koje je primila što omogućuje ispis korisničkog imena korisnika koji je poslao poruku na svim ostalim klijentima.

```
50      public void PosaljiPoruku(string poruka, string korisnickoIme)
51      {
52          foreach (var klijent in online_klijenti)
53          {
54              if (klijent.Key.ToLower() != korisnickoIme.ToLower())
55              {
56                  klijent.Value.veza.DohvatiPoruku(poruka, korisnickoIme);
57              }
58      }
```

Sl. 3.26. Funkcija PosaljiPoruku

Funkcija *DohvatiPoruku* vidljiva je na slici 3.27. i ona se izvršava na svakom od klijenata.

```
15      public void DohvatiPoruku(string poruka, string korisnickoIme)
16      {
17          ((MainWindow)Application.Current.MainWindow).PreuzmiPoruku(poruka, korisnickoIme);
18      }
```

Sl. 3.27. Funkcija DohvatiPoruku

Nakon toga program se vraća u funkciju koja se pokrenula prilikom pritiska gumba za slanje poruke te se tamo poziva funkcija *PreuzmiPoruku* koja kao parametar korisničkog imena prima string *„You“*. Ovime se postiže da se na klijentu koji je poslao poruku umjesto vlastitog korisničkog imena prikaže riječ *„You“* kao pošiljatelj. Osim tekstualnih poruka moguće je slati i

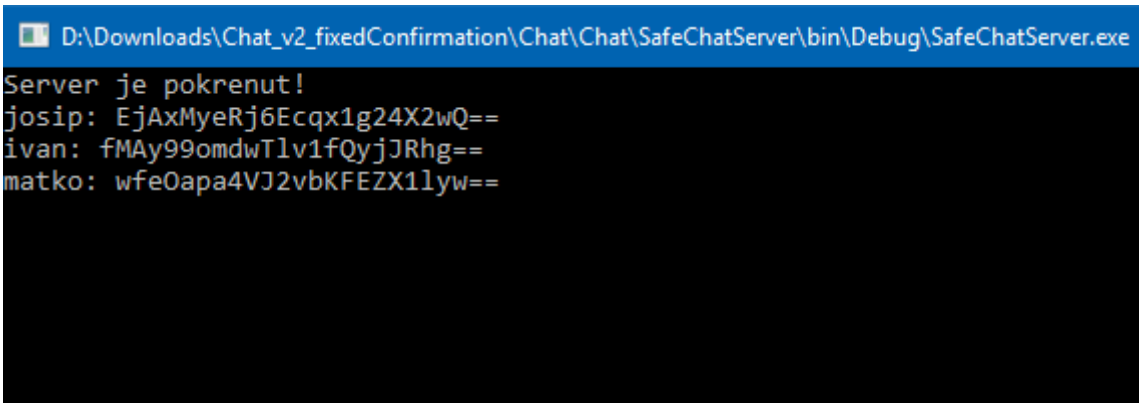
datoteke. Datoteku je moguće poslati pritiskom na gumb *„Send file“* na klijentu. Time se pokreće kod sa slike 3.28.

```
187     private void SendFileButton_Click(object sender, RoutedEventArgs e)
188     {
189         OpenFileDialog dlg = new OpenFileDialog();
190         if (dlg.ShowDialog() == true)
191         {
192             var bytes = File.ReadAllBytes(dlg.FileName);
193             var filename = System.IO.Path.GetFileName(dlg.FileName);
194             Server.PosaljiDatoteku(bytes, filename, korisnik);
195 
196             PreuzmiDatoteku(bytes, filename, "You");
197         }
198     }
```

Sl. 3.28. Slanje datoteke

Prvo što se izvrši je otvaranje Windows prozora u kojem je moguće odabrati datoteku koju korisnik želi poslati. Nakon odabira datoteke ona se učitava u niz bajtova koji se prosljeđuje funkciji *PosaljiDatoteku.* Ova funkcija kao parametre prima niz bajtova koji predstavljaju samu datoteku, ime datoteke te ime korisnika koji ju šalje. Nakon slanja datoteke na isti način na koji se šalju i tekstualne poruke svim prijavljenim korisnicima se prikazuje gumb za preuzimanje datoteke što je vidljivo na slici 3.29. Nakon pritiska na taj gumb ponovno se otvara Windows prozor u kojem je moguće odabrati željenu lokaciju na računalu na koju će se spremiti datoteka nakon preuzimanja.

Na slici 3.29. možemo vidjeti primjer kako izgleda poslužitelj prilikom slanja poruka između korisnika. Za potrebe ovog rada i testiranja aplikacije u poslužitelj je dodana funkcija da prilikom slanja poruke ispiše ime korisnika koji je poslao poruku te poruku u enkriptiranom obliku.



Sl. 3.29. Ispis poruke na poslužitelju

4. IMPLEMENTACIJA AES ALGORITMA U C# PROGRAMSKOM JEZIKU

Obzirom na to da je C# jedan od najkorištenijih programskih jezika on ima ima unaprijed implementirane mnoge funkcije i algoritme odnosno klase koje se mogu iskoristiti bez da se od nule implementira željeno rješenje. Tako C# sadrži i kolekciju klasa i funkcija vezanih uz kriptografiju u kojoj se nalazi i AES klasa. Kako bi se u aplikaciji koristile te klase i funkcije potrebno je u projekt uključiti namespace System.Security.Cryptography pomoću ključne riječi „using“. Nakon toga imamo pristup svim klasam i njihovim metodama koje se nalaze unutar te kolekcije. Za potrebe izrade aplikacije u ovom radu najvažnija je klasa AES koja sadrži funkcije potrebne za točno određivanje parametara koji će se koristiti prilikom enkripcije.

4.1. Enkripcija podataka

Na slici 4.1. vidljiv je primjer korištenja AES algoritma u C# programskom jeziku. Slika 4.1. prikazuje funkciju koja će služiti za enkripciju podataka. U prvoj liniji koda nalazi se deklaracija funkcije. Funkcija ima povratni tip byte[] što znači da kao rezultat vraća niz bajtova. Kao parametre prima niz bajtova koji trebaju biti enkriptirani te niz bajtova pomoću kojih će se generirati ključ. Cijeli postupak izvodit će se pomoći MemoryStreama koji je kreiran u liniji 7. Nakon toga stvara se nova instanca Aes klase kako bi imali pristup funkcijama koje ona sadrži. Zatim je potrebno postaviti osnovne parametre AES algoritma, a to su duljina ključa, veličina bloka, generirati sami ključ te odrediti način rada algoritma. Veličina ključa može biti 128, 192 ili 256 bita, veličina bloka kod AES-a je uvijek 128 bita. Ključ se generira pomoću funkcije u liniji 14 koda na slici 4.1. Ta funkcija je Rfc2898DeriveBytes i ona kao parametre prima niz bajtova koji su proslijeđeni glavnoj funkciji, niz bajtova saltBytes koji predstavljaju dodatak prvom nizu te promjenom ovog niza možemo dobiti različiti ključ čak i ako je prvi parametar uvijek isti. Zadnji parametar predstavlja broj iteracija koje će proći funkcija pri kreiranju ključa, u ovom slučaju postavljen je na 1000 iteracija. Nakon toga stvara se ključ ovisno u zadanoj duljini te inicijalizacijski vektor duljine 16 bita. Preostali parametar za postaviti je način rada, u ovom slučaju je postavljen na ECB. Osim njega moguće je postaviti način rada u CBC, CFB, ECB te OFB.

```
1   public static byte[] AES_Encrypt(byte[] bytesToBeEncrypted, byte[] passwordBytes)
2   {
3       byte[] encryptedBytes = null;
4
5       byte[] saltBytes = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
6
7       using (MemoryStream ms = new MemoryStream())
8       {
9           using (Aes AES = Aes.Create())
10          {
11              AES.KeySize = 128;
12              AES.BlockSize = 128;
13
14              var key = new Rfc2898DeriveBytes(passwordBytes, saltBytes, 1000);
15              AES.Key = key.GetBytes(AES.KeySize / 8);
16              AES.IV = key.GetBytes(AES.BlockSize / 8);
17
18              AES.Mode = CipherMode.ECB;
19
20              using (var cs = new CryptoStream(ms, AES.CreateEncryptor(), CryptoStreamMode.Write))
21              {
22                  cs.Write(bytesToBeEncrypted, 0, bytesToBeEncrypted.Length);
23                  cs.Close();
24              }
25              encryptedBytes = ms.ToArray();
26          }
27      }
28
29      return encryptedBytes;
30  }
```

Sl. 4.1. AES enkripcija u C#

Kako bi mogli koristit prethodnu funkciju AES_Encrypt potrebno ju je negdje u programu pozvati te joj proslijediti točno određene parametre koje one prima. To su dva niza bajtova: niz bajtova koji se treba kriptirati te niz bajtova pomoću kojih će se kreirati ključ. Na slici 4.2. vidi se primjer funkcije koja to omogućuje. Funkcija EncryptText kao parametre prima dva stringa koji su ekvivalenti dvama nizovima bajtova koje prima AES_Encrypt. Odnosno, unutar funkcije EncryptText ta se dva stringa prebacuju u oblik niza bajtova kao što je vidljivo u linijama 3 i 4 na slici 3.2. Nakon toga se bajtovi koji služe za kreiranje ključa „hashiraju“ kako se nigdje u računalu ne bi zapisivala stvarna vrijednost tog niza i kako bi se otklonila bilo kakva mogućnost otkrivanja tog podataka budući da je to jednosmjerni postupak. Sada je sve spremno za pozivanje funkcije AES_Encrypt. Ona se poziva liniji 8 koda sa slike 4.2. te se njen rezultat automatski sprema u niz bajtova budući da je to njen povratni tip. Nakon toga samo preostaje rezultat koji je u obliku niza bajtova pretvoriti u string te vratiti taj podatak pozivatelju ove funkcije.

```
1   public static string EncryptText(string input, string password)
2           {
3               byte[] bytesToBeEncrypted = Encoding.UTF8.GetBytes(input);
4               byte[] passwordBytes = Encoding.UTF8.GetBytes(password);
5
6               passwordBytes = SHA256.Create().ComputeHash(passwordBytes);
7
8               byte[] bytesEncrypted = AES_Encrypt(bytesToBeEncrypted, passwordBytes);
9
10              string result = Convert.ToBase64String(bytesEncrypted);
11
12              return result;
13          }
```

Sl. 4.2. Enkripcija teksta

4.2. Dekripcija podataka

Funkcija za dekripciju podataka ekvivalentna je funkciji za enkripciju. Ona je vidljiva na slici 4.3. Isto kao i funkcija za enkripciju i ova funckija mora primati iste tipove parametara od kojih je prvi niz bajtova zapravo kriptirani tekst, a drugi niz bajtova je niz pomoću kojeg se stvara ključ te taj niz mora biti identičan onome iz funkcije za enkripciju. To je potrebno zbog toga što je AES simetrična šifra te moramo imati isti ključ pomoću kojega smo radili enkripciju kako bi mogli iste te podatke dekriptirati. Jedina razlika između ovih dviju funkcija je u liniji 21 koda na slici 4.3. gdje se pri stvaranju CryptoStreama umjesto funkcije CreateEncryptor pozivanje funkcija CreateDecryptor.

```
1   public static byte[] AES_Decrypt(byte[] bytesToBeDecrypted, byte[] passwordBytes)
2   {
3       byte[] decryptedBytes = null;
4
5       byte[] saltBytes = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
6
7       using (MemoryStream ms = new MemoryStream())
8       {
9           using (Aes AES = Aes.Create())
10          {
11              AES.KeySize = 128;
12              AES.BlockSize = 128;
13
14              var key = new Rfc2898DeriveBytes(passwordBytes, saltBytes, 1000);
15              AES.Key = key.GetBytes(AES.KeySize / 8);
16              AES.IV = key.GetBytes(AES.BlockSize / 8);
17
18              AES.Mode = CipherMode.ECB;
19
20              using (var cs = new CryptoStream(ms, AES.CreateDecryptor(), CryptoStreamMode.Write))
21              {
22                  cs.Write(bytesToBeDecrypted, 0, bytesToBeDecrypted.Length);
23                  cs.Close();
24              }
25              decryptedBytes = ms.ToArray();
26          }
27      }
```

Sl. 4.3. AES dekripcija u C#

```
public static string DecryptText(string input, string password)
        {

            byte[] bytesToBeDecrypted = Convert.FromBase64String(input);
            byte[] passwordBytes = Encoding.UTF8.GetBytes(password);

            passwordBytes = SHA256.Create().ComputeHash(passwordBytes);

            byte[] bytesDecrypted = AES_Decrypt(bytesToBeDecrypted, passwordBytes);

            string result = Encoding.UTF8.GetString(bytesDecrypted);

            return result;
        }
```

Sl. 4.4. Dekripcija teksta

Analogno funkciji za enkripciju teksta može se napraviti funkcija za dekripciju kao što je prikazano na slici 4.4. Ulazi ove funkcije su string koji treba dekriptirati te string koji predstavlja izraz pomoću kojega se kreira ključ korišten u AES_Decrypt funkciji. Ta dva stringa se prebacuju i nizove bajtova kako bi se mogli proslijediti spomenutoj funkciji koja kao rezultat vraća dekriptirani niz bajtova. Taj niz na kraju treba prebaciti u oblik stringa budući da je povratni tip funkcije DecryptText upravo string.

5. ZAKLJUČAK

Razvojem računala i računalnih sustava, a time i količnom komunikacije koja se odvija preko njih pojavila se potreba razvoja algoritama koji pružaju zadovoljavajuću zaštitu prilikom prijenosa podataka. Današnji standard u svijetu enkripcije je AES algoritam. U ovom radu obrađen je AES algoritam za enkripciju, opisan je način na koji on radi te su objašnjeni njegovi načini rada. Osim teoretskog opisa algoritma dodatno pojašnjenje može se pronaći u slikama i dijagramima koji olakšavaju shvaćanje tematike. Drugi dio rada bavi se izradom aplikacije za sigurno slanje poruka i datoteka. Aplikacije ja izrađena za Windows operacijski sustav. Tehnologije koje su korištene su: SQL pomoću kojeg je izgrađena baza podataka s korisnicima, C# programski jezik kao temelj na kojemu je aplikacija izgrađena, WCF servis kao dio .NET platforme koji omogućava komunikaciju između svih elemenata aplikacije. Kroz čitav rad objašnjeni su najvažniji dijelovi programskog koda te su slikama prikazani osnovni dijelovi aplikacije odnosno korisničko sučelje aplikacije s kojim se korisnik susreće. Aplikacija je testirana u stvarnom okruženju te je rezultat testiranja objašnjen odmah prilikom objašanjavanja pojedinih dijelova aplikacije kako bi se postigla bolja preglednost rezultata. Nakon istraživanja područja računalne sigurnosti i izrade aplikacije za sigurnu razmjenu poruka može se zaključiti da je u današnje vrijeme obavezno koristiti neku vrstu zaštite prilikom bilo kakve razmjene podataka, a naročito onih povjerljivih. Aplikacija koja je napravljena za potrebe ovog rada pruža određenu zaštitu, međutim u stvarnom svijetu potrebna razina zaštite je neuspredivo veća. Potrebno je kontinuirano razvijati mjere zaštite podataka kako bi se smanjila vjerojatnost zloupotrebe računalne moći koja svakim danom sve više raste i pruža sve veće mogućnosti. Zbog toga je bitno usmjeriti razvoj na zaštitu podataka umjesto na njihovo otuđivanje bez obzira i kojoj vrsti podataka se radi.

6. LITERATURA

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

[2] J. Daemen, V. Rijmen, The Design of Rijndael, Springer-Verlag, Berlin, 2002.

[3] https://msdn.microsoft.com/en-us/library/ms731082(v=vs.110).aspx

7. SAŽETAK

RAČUNALNA APLIKACIJA ZA SIGURNU RAZMJENU TEKSTUALNIH PORUKA U STVARNOM VREMENU

Ovaj rad opisuje AES enkripcijski algoritam te njegove načine rada. Opisan je postupak izrade aplikacije za razmjenu poruku i datoteka u stvarnom vremenu koja koristi AES algoritam kao metodu enkripcije. Spomenute su i ukratko opisane tehnologije i alati koji su se koristili prilikom izrade aplikacije. Objašnjeni su najbitniji dijelovi programskog koda te su priložene slike koje prikazuju korisničko sučelje aplikacije. Obavljeno je testiranje u stvarnom mrežnom okruženju kako bi se ispitala funkcionalnost i ispravnost aplikacije.

ABSTRACT

REAL-TIME ENCRYPTED INSTANT MESSAGING DESKTOP APPLICATION

This paper describes AES encryption algorithm and its modes of operation. It also describes development process of chat application which uses AES algorithm as encryption method. The most important technologies and tools used in the process are mentioned and described as well as crucial parts of source code. The paper contains images and screenshots of applications user interface. Testing in real network environment has been conducted to examine functionality.

8. ŽIVOTOPIS

Matko Vujnovac rođen je 14. studenoga 1992. godine u Osijeku. Živi u Samatovcima, na adresi Osječka 21. Osnovnu školu upisuje 1999. godine te prva četiri razreda završava u područnoj školi u Samatovcima. Osnovnoškolsko obrazovanje završava u OŠ Bratoljuba Klaića u Bizovcu. 2007. godine upisuje se u I. Gimanziju Osijek te s odličnim uspjehom završava sva četiri razreda srednje škole što mu donosi izravan upis na Elektrotehnički fakultet u Osijeku. Iste godine uspješno polaže državnu maturu te se upisuje na Elektrotehnički fakultet u Osijeku, smjer Računarstvo. Preddiplomski studij računarstva završava 2014. godine te iste godine upisuje diplomski studij, smjer Procesno računarstvo na istom fakultetu, gdje i u vrijeme pisanja ovog životopisa studira.

9. PRILOZI

9.1. Interfaces

```
namespace Interfaces
{
    public class ConnectedClient
    {
        public IClient veza;
        public string korisnickoIme { get; set; }

        public string key { get; set; }

        public int mode { get; set; }
        public int keyLenght { get; set; }
    }

    public class KeyDetails
    {
        public string key { get; set; }

        public int mode { get; set; }
        public int keyLenght { get; set; }


    }
}

namespace Interfaces
{
    public interface IClient
    {
        [OperationContract]
        void DohvatiPoruku(string poruka, string korisnickoIme);

        [OperationContract]
        void DohvatiDatoteku(byte[] datoteka, string naziv, string korisnickoIme);

        [OperationContract]
        void SkipConfigPage(KeyDetails keyDetails);

    }
}

namespace Interfaces
{
    [ServiceContract(CallbackContract = typeof(IClient))]

    public interface IService
    {
        [OperationContract]
        LoginResponse Login(string korisnickoIme, string key = "", int keyLenght = 0, int
mode =-1);
        [OperationContract]
        void PosaljiPoruku(string poruka, string korisnickoIme);

        [OperationContract]
        void PosaljiDatoteku(byte[] datoteka,string filename, string korisnickoIme);

        [OperationContract]
        int Logout();
```

```
        [OperationContract]
        void IspisiPoruku(string korisnik, string msg);

        [OperationContract]
        bool SetKeyDetails(string key, int mode, int lenght, string username);
    }
}

namespace Interfaces
{
    public class LoginResponse
    {
        public bool Success { get; set; }
        public KeyDetails KeyDetails { get; set; }
    }
}
```

9.2. Client

```
namespace SafeChatClient
{
    public class ChatItem
    {
        public ChatItem(string message, string sender, string fileName = null, byte[]
file = null)
        {
            Message = message;
            Sender = sender;
            FileName = fileName;
            File = file;
        }

        public string Message { get; set; }
        public string Sender { get; set; }
        public string FileName { get; set; }
        public byte[] File  { get; set; }
        public Visibility DownloadButtonVisible { get { return (File != null) ?
Visibility.Visible : Visibility.Collapsed; } }
    }
}

namespace SafeChatClient
{
    [CallbackBehavior(ConcurrencyMode = ConcurrencyMode.Multiple)]
    public class ClientCallback : IClient
    {
        public void DohvatiPoruku(string poruka, string korisnickoIme)
        {
            ((MainWindow)Application.Current.MainWindow).PreuzmiPoruku(poruka,
korisnickoIme);
        }

        public void DohvatiDatoteku(byte[] datoteka, string naziv, string korisnickoIme)
        {
            ((MainWindow)Application.Current.MainWindow).PreuzmiDatoteku(datoteka,naziv,
korisnickoIme);
        }

        public void SkipConfigPage(KeyDetails keyDetails)
        {
            ((MainWindow)Application.Current.MainWindow).HideConfirmDetails();
```

```
            ((MainWindow)Application.Current.MainWindow).KeyDetails = keyDetails;
        }
    }
}

namespace SafeChatClient
{
    class DBConnection
    {
        public static int DBLogin(string korisnickoIme, string lozinka)
        {
            try
            {
                //return 1;

                using (var conn = new
SqlConnection(Helper.GetValueFromConfig("ConnectionString")))
                {
                    using (var cmd = new SqlCommand("pr_User_Login", conn))
                    {
                        cmd.CommandType = CommandType.StoredProcedure;
                        cmd.Parameters.AddWithValue("@Username", korisnickoIme);
                        cmd.Parameters.AddWithValue("@Password",
Helper.SHA512Hash(lozinka));
                        conn.Open();

                        int UserID = Int32.Parse(cmd.ExecuteScalar().ToString());
                        if (UserID > 0)
                        {
                            return 1;
                        }

                        conn.Close();
                    }
                }
            }
            catch
            {
                MessageBox.Show("Unijeli ste krive podatke!");
            }
            return 0;
        }
    }
}

namespace SafeChatClient
{
    class Helper
    {
        public static string SHA512Hash(string inputString)
        {
            SHA512 sha512 = SHA512Managed.Create();
            byte[] bytes = Encoding.UTF8.GetBytes(inputString);
            byte[] hash = sha512.ComputeHash(bytes);
            StringBuilder sb = new StringBuilder();

            for (int i = 0; i < hash.Length; i++)
            {
                sb.Append(hash[i].ToString("x2"));
            }

            return sb.ToString();
```

```
        }

        public static string GetValueFromConfig(string key)
        {
            return ConfigurationManager.AppSettings[key].ToString();
        }

        public static byte[] AES_Encrypt(byte[] bytesToBeEncrypted, byte[] passwordBytes,
int keySize, int mode)
        {
            byte[] encryptedBytes = null;

            // Set your salt here, change it to meet your flavor:
            // The salt bytes must be at least 8 bytes.
            byte[] saltBytes = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };

            using (MemoryStream ms = new MemoryStream())
            {
                using (Aes AES = Aes.Create())
                {
                    AES.KeySize = keySize;
                    AES.BlockSize = 128;

                    var key = new Rfc2898DeriveBytes(passwordBytes, saltBytes, 1000);
                    AES.Key = key.GetBytes(AES.KeySize / 8);
                    AES.IV = key.GetBytes(AES.BlockSize / 8);

                    switch (mode)
                    {
                        case 0:
                            AES.Mode = CipherMode.CBC;
                            break;
                        case 1:
                            AES.Mode = CipherMode.ECB;
                            break;
                        case 2:
                            AES.Mode = CipherMode.CFB;
                            break;
                    }
                    //AES.Mode = CipherMode.CBC;

                    using (var cs = new CryptoStream(ms, AES.CreateEncryptor(),
CryptoStreamMode.Write))
                    {
                        cs.Write(bytesToBeEncrypted, 0, bytesToBeEncrypted.Length);
                        cs.Close();
                    }
                    encryptedBytes = ms.ToArray();
                }
            }

            return encryptedBytes;
        }

        public static byte[] AES_Decrypt(byte[] bytesToBeDecrypted, byte[] passwordBytes,
int keySize, int mode)
        {
            byte[] decryptedBytes = null;

            // Set your salt here, change it to meet your flavor:
            // The salt bytes must be at least 8 bytes.
            byte[] saltBytes = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
```

```
            using (MemoryStream ms = new MemoryStream())
            {
                using (Aes AES = Aes.Create())
                {
                    AES.KeySize = keySize;
                    AES.BlockSize = 128;

                    var key = new Rfc2898DeriveBytes(passwordBytes, saltBytes, 1000);
                    AES.Key = key.GetBytes(AES.KeySize / 8);
                    AES.IV = key.GetBytes(AES.BlockSize / 8);

                    switch (mode)
                    {
                        case 0:
                            AES.Mode = CipherMode.CBC;
                            break;
                        case 1:
                            AES.Mode = CipherMode.ECB;
                            break;
                        case 2:
                            AES.Mode = CipherMode.CFB;
                            break;
                    }
                    //AES.Mode = CipherMode.CBC;

                    using (var cs = new CryptoStream(ms, AES.CreateDecryptor(),
CryptoStreamMode.Write))
                    {
                        cs.Write(bytesToBeDecrypted, 0, bytesToBeDecrypted.Length);
                        cs.Close();
                    }
                    decryptedBytes = ms.ToArray();
                }
            }

            return decryptedBytes;
        }

        public static string EncryptText(string input, KeyDetails details)
        {
            // Get the bytes of the string
            byte[] bytesToBeEncrypted = Encoding.UTF8.GetBytes(input);
            byte[] passwordBytes = Encoding.UTF8.GetBytes(details.key);

            // Hash the password with SHA256
            passwordBytes = SHA256.Create().ComputeHash(passwordBytes);

            byte[] bytesEncrypted = AES_Encrypt(bytesToBeEncrypted, passwordBytes,
details.keyLenght, 1);

            string result = Convert.ToBase64String(bytesEncrypted);

            return result;
        }

        public static string DecryptText(string input, KeyDetails details)
        {
            // Get the bytes of the string
            byte[] bytesToBeDecrypted = Convert.FromBase64String(input);
            byte[] passwordBytes = Encoding.UTF8.GetBytes(details.key);
            passwordBytes = SHA256.Create().ComputeHash(passwordBytes);
```

```
            byte[] bytesDecrypted = AES_Decrypt(bytesToBeDecrypted, passwordBytes,
details.keyLenght, 1);

            string result = Encoding.UTF8.GetString(bytesDecrypted);

            return result;
        }

    }
}

namespace SafeChatClient
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        string korisnik;
        public static IService Server;
        private static DuplexChannelFactory<IService> _channelFactory;

        public ObservableCollection<ChatItem> ChatItems { get; set; }

        public KeyDetails KeyDetails {get; set;}

        string key = "";
        int keyLenght = 0;
        int mode = -1;

        public MainWindow()
        {
            InitializeComponent();

            ChatItems = new ObservableCollection<ChatItem>();
            lbChatBox.ItemsSource = ChatItems;

            MessageTextBox.Visibility = Visibility.Hidden;
            lbChatBox.Visibility = Visibility.Hidden;
            SendButton.Visibility = Visibility.Hidden;
            SendFileButton.Visibility = Visibility.Hidden;
            ConfirmButton.Visibility = Visibility.Hidden;
            keyLabel.Visibility = Visibility.Hidden;
            keyTextBox.Visibility = Visibility.Hidden;
            ModeComboBox.Visibility = Visibility.Hidden;
            ModeLabel.Visibility = Visibility.Hidden;
            keySizeComboBox.Visibility = Visibility.Hidden;
            keySizeLabel.Visibility = Visibility.Hidden;

            _channelFactory = new DuplexChannelFactory<IService>(new ClientCallback(),
"ChattingServiceEndPoint");
            Server = _channelFactory.CreateChannel();

        }

        private void LoginButton_Click(object sender, RoutedEventArgs e)
        {
            if (DBConnection.DBLogin(UserNameTextBox.Text.ToString(),
PasswordBox.Password.ToString()) == 1)
            {
                korisnik = UserNameTextBox.Text.ToString();
```

```
                var result = Server.Login(korisnik);

                if (result.Success)
                {
                    PasswordBox.Visibility = Visibility.Hidden;
                    LoginButton.Visibility = Visibility.Hidden;
                    label.Visibility = Visibility.Hidden;
                    label1.Visibility = Visibility.Hidden;
                    UserNameTextBox.Visibility = Visibility.Hidden;
                    label2.Visibility = Visibility.Hidden;
                    keyLabel.Visibility = Visibility.Visible;
                    keyTextBox.Visibility = Visibility.Visible;
                    ModeComboBox.Visibility = Visibility.Visible;
                    ModeLabel.Visibility = Visibility.Visible;
                    keySizeComboBox.Visibility = Visibility.Visible;
                    keySizeLabel.Visibility = Visibility.Visible;
                    ConfirmButton.Visibility = Visibility.Visible;

                    LoginButton.IsDefault = false;
                    SendButton.IsDefault = true;

                    if (result.KeyDetails != null)
                    {
                        this.KeyDetails = result.KeyDetails;
                        HideConfirmDetails();
                    }
                }
                else
                {
                    MessageBox.Show("Već ste prijavljeni!");
                }

            }
        }

        private void window_Closing(object sender, System.ComponentModel.CancelEventArgs
e)
        {
            if ((Server.Logout()) == 1) MessageBox.Show("Uspješna odjava!");
        }

        public void PreuzmiPoruku(string poruka, string korisnickoIme)
        {
            if (korisnickoIme == "You")
            {
                ChatItems.Add(new ChatItem(poruka,korisnickoIme));
            }
            else
            {
               var msg = Helper.DecryptText(poruka, KeyDetails);
                ChatItems.Add(new ChatItem(msg, korisnickoIme));
            }

        }

        public void PreuzmiDatoteku(byte[] datoteka,string nazivDatoteke, string
korisnickoIme)
        {
            ChatItems.Add(new ChatItem(nazivDatoteke, korisnickoIme, nazivDatoteke,
datoteka));
        }
```

```
        private void SendButton_Click(object sender, RoutedEventArgs e)
        {


            if (!string.IsNullOrWhiteSpace(MessageTextBox.Text.ToString()))
            {
                var msg = MessageTextBox.Text.ToString();
                var e_msg = Helper.EncryptText(msg, KeyDetails);



                Server.PosaljiPoruku(e_msg, korisnik);
                Server.IspisiPoruku(korisnik, e_msg);
                PreuzmiPoruku(MessageTextBox.Text, "You");
                MessageTextBox.Text = "";
            }
        }

        private void ConfirmButton_Click(object sender, RoutedEventArgs e)
        {


            HideConfirmDetails();

            key = keyTextBox.Text;
            ComboBoxItem typeItem = (ComboBoxItem)keySizeComboBox.SelectedItem;
            string value = typeItem.Content.ToString();
            keyLenght = int.Parse(value);
            ComboBoxItem modei = (ComboBoxItem)ModeComboBox.SelectedItem;
            string v = modei.Content.ToString();
            switch (v)
            {
                case "CBC":
                    mode = 1;
                    break;
                case "EBC":
                    mode = 2;
                    break;
            }

            Server.SetKeyDetails(key, mode, keyLenght, this.korisnik);
            this.KeyDetails = new KeyDetails { key = key, mode = mode, keyLenght =
keyLenght };
        }

        internal void HideConfirmDetails()
        {
            MessageTextBox.Visibility = Visibility.Visible;
            lbChatBox.Visibility = Visibility.Visible;
            SendButton.Visibility = Visibility.Visible;
            SendFileButton.Visibility = Visibility.Visible;
            ConfirmButton.Visibility = Visibility.Hidden;
            keyLabel.Visibility = Visibility.Hidden;
            keyTextBox.Visibility = Visibility.Hidden;
            ModeComboBox.Visibility = Visibility.Hidden;
            ModeLabel.Visibility = Visibility.Hidden;
            keySizeComboBox.Visibility = Visibility.Hidden;
            keySizeLabel.Visibility = Visibility.Hidden;
        }
```

```
        private void SendFileButton_Click(object sender, RoutedEventArgs e)
        {
            OpenFileDialog dlg = new OpenFileDialog();
            if (dlg.ShowDialog() == true)
            {
                var bytes = File.ReadAllBytes(dlg.FileName);
                var filename = System.IO.Path.GetFileName(dlg.FileName);
                Server.PosaljiDatoteku(bytes, filename, korisnik);

                PreuzmiDatoteku(bytes, filename, "You");
            }
        }


        private void btnDownload_Click(object sender, RoutedEventArgs e)
        {
            var item = ((Button)sender).DataContext as ChatItem;
            if (item == null)
            {
                MessageBox.Show("No file to download");
                return;
            }

            var saveDialog = new SaveFileDialog();
            saveDialog.FileName = item.FileName;
            saveDialog.DefaultExt = System.IO.Path.GetExtension(item.FileName);
            saveDialog.Filter = string.Format("File type | *.{0}",
System.IO.Path.GetExtension(item.FileName));
            if (saveDialog.ShowDialog() == true)
            {
                File.WriteAllBytes(saveDialog.FileName, item.File);
            }
        }
    }
}
```

9.3. Server

```
namespace SafeChatServer
{
    class Program
    {
        public static Server _server;

        static void Main(string[] args)
        {
            _server = new Server();
            using (ServiceHost host = new ServiceHost(_server))
            {
                host.Open();
                Console.WriteLine("Server je pokrenut!");
                Console.ReadLine();
            }
        }
    }
}

namespace SafeChatServer
{
    [ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple, InstanceContextMode =
InstanceContextMode.Single)]
    public class Server : IService
```

```
    {
        public ConcurrentDictionary<string, ConnectedClient> online_klijenti = new
ConcurrentDictionary<string, ConnectedClient>();

        public KeyDetails KeyDetails { get; set; }

        public LoginResponse Login(string KorisnickoIme, string key = "", int keyLenght =
0, int mode = -1)
        {
            try
            {
                foreach (var klijent in online_klijenti)
                {
                    if (klijent.Key.ToLower() == KorisnickoIme.ToLower())
                    {
                        return new LoginResponse { Success = false };//allready logged in
                    }
                }
                var uspostavljenaVeza =
OperationContext.Current.GetCallbackChannel<IClient>();

                ConnectedClient noviKlijent = new ConnectedClient();
                noviKlijent.veza = uspostavljenaVeza;
                noviKlijent.korisnickoIme = KorisnickoIme;

                online_klijenti.TryAdd(KorisnickoIme, noviKlijent);

                return new LoginResponse { Success = true, KeyDetails = this.KeyDetails
};
            }
            catch (Exception)
            {
                return new LoginResponse { Success =false, KeyDetails = this.KeyDetails
};
            }


        }

        public void PosaljiPoruku(string poruka, string korisnickoIme)
        {

                foreach (var klijent in online_klijenti)
                {
                    if (klijent.Key.ToLower() != korisnickoIme.ToLower())
                    {
                        klijent.Value.veza.DohvatiPoruku(poruka, korisnickoIme);
                    }
            }
        }

        public ConnectedClient DohvatiKlijenta()
        {
            var uspostavljenaVeza =
OperationContext.Current.GetCallbackChannel<IClient>();

            foreach (var klijent in online_klijenti)
            {
                if(klijent.Value.veza == uspostavljenaVeza)
                {
                    return klijent.Value;
                }
```

```
            }
            return null;
        }

        public int Logout()
        {
            ConnectedClient klijent = DohvatiKlijenta();
            if(klijent != null)
            {
                ConnectedClient obrisaniKlijent;
                online_klijenti.TryRemove(klijent.korisnickoIme, out obrisaniKlijent);

                return 1;
            }

            if (online_klijenti.Count == 0)
                KeyDetails = null;

            return 0;
        }

        public void PosaljiDatoteku(byte[] datoteka,string filename, string
korisnickoIme)
        {
            foreach (var klijent in online_klijenti)
            {
                if (klijent.Key.ToLower() != korisnickoIme.ToLower())
                {
                    klijent.Value.veza.DohvatiDatoteku(datoteka,filename, korisnickoIme);
                }
            }
        }

        public bool SetKeyDetails(string key, int mode, int lenght, string korisnickoIme)
        {
            this.KeyDetails = new KeyDetails { key = key,mode =mode, keyLenght = lenght};
            if (online_klijenti.Any())
            {
                foreach (var klijent in online_klijenti)
                {
                    if (klijent.Key.ToLower() != korisnickoIme.ToLower())
                    {
                        klijent.Value.veza.SkipConfigPage(KeyDetails);
                    }
                }
            }
            return true;
        }

        public void IspisiPoruku(string korisnik, string msg)
        {
            Console.WriteLine(korisnik + ": " + msg);
        }
    }
}
```