ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

IMPLEMENTACIJA BEZBEDNE KLIJENT/SERVER KOMUNIKACIJE U PROGRAMSKOM JEZIKU JAVA

– Diplomski rad –

Kandidat:

Vanja Efremov 2008/196

Mentor:

doc. dr Zoran Čiča

Beograd, Jul 2014.

SADRŽAJ

1. UVOD

Implementacija sigurne server/klijent komunikacije korišćenjem AES šifrovanja pri čemu se komunikacija obavlja na transportnom sloju pomoću UDP protokola predstavlja cilj ovog rada.

Dva najpopularnija protokola transportnog sloja su TCP (*Transmission Control Protocol*) i UDP (*User Datagram Protocol*). TCP ima direktnu uspostavu veze, omogućava pouzdan prenos i kompleksan je za implementaciju. UDP radi bez uspostave veze, ne obezbeđuje pouzdan prenos i znatno je jednostavniji od TCP-a. Protokol UDP je opisan u RFC dokumentu 768. UDP prenosi segmente koji se sastoje iz 8-bajtnog zaglavlja i korisničkih podataka. Zaglavlje se sastoji iz:

- izvorišnog porta,
- odredišnog porta,
- dužine UDP paketa i
- kontrolnog zbira paketa

Izvorišni port predstavlja adresu porta strane koja je kreirala UDP paket, a odredišni port predstavlja adresu porta na odredištu dotičnog UDP paketa. Dužina UDP paketa predstavlja ukupnu dužinu zaglavlja i korisničkih podataka. Kontrolni zbir UDP nije obavezan, ali se preporučuje njegova upotreba. Kontrolni zbir je predstavljen nulom ako nije izračunat, a ako je izračunat i jeste nula onda se postavlja na vrednost sve jedinice. UDP protokol ne upravlja tokom, ne kontroliše greške i ne šalje pogrešno primljene pakete ponovo. Ovaj protokol je posebno koristan u kratkim serversko/klijentskim komunikacijama (pa je zato i odabran u ovoj tezi, a ne TCP), jer klijent šalje kratak zahtev i očekuje isto tako kratak odgovor. Ovakav način komunikacije je jednostavniji i zahteva kraću razmenu poruka, jer nema prethodne uspostave, kontrole i raskidanja veze kao u slučaju TCP protokola.

AES (*Advanced Encryption Standard*) je simetričan algoritam za šifrovanje koji je moguće i softverski i hardverski implementirati i podržava dužine ključa od 128, 192 i 256 bita. AES vrši šifrovanje blokova dužine 128 bita koji su predstavljeni pomoću matrice 4x4 koja se naziva AES blok. U slučaju AES-128 (128-bitni ključ) obavlja se 10 rundi, a svaka runda ima 4 operacije:

1) Supstituciju (*SubBytes*) – nelinearni deo algoritma koji konvertuje osmobitne podatke u druge (različite od početnih) osmobitske podatke;
2) Pomeranje redova (*ShiftRows*) – redovi AES bloka se rotiraju u desno za odgovarajući broj pozicija u zavisnosti od pozicije reda;
3) Linearno mešanje (*MixColumns*) – kolone AES bloka se posmatraju kao polinomi i množe se sa odgovarajućim polinomom po modulu x^4+1, ova operacija se ne obavlja u poslednjoj rundi;
4) Sabiranje ključem runde (*AddRoundKey*) – sabiranje po modulu dva AES bloka i ključa trenutne runde.

Prednosti AES algoritma su mali memorijski zahtevi, fleksibilnost, tajnost je zagarantovana samo tajnošću ključa, a mogu se postići i veoma velike brzine šifrovanja.

Ostatak rada je organizovan na sledeći način. U drugom poglavlju je opisan programski jezik Java, sa posebnim osvrtom na pakete i klase korišćene u okviru ove teze. U poglavlju 3 je detaljno objašnjena realizovana implementacija. Rezultati testiranja i primena su opisani u poglavlju 4. Poglavlje 5 predstavlja rezime teze. Na kraju je dat spisak literature i dva priloga, u prvom je celokupan kod programa, a u drugom spisak svih skraćenica korišćenih u radu.

2. PROGRAMSKI JEZIK JAVA

Java je objektno orijentisan programski jezik, koji je platformski nezavisan, jednostavan i siguran. Specifičan je po tome što ima svoju Java virtuelnu mašinu (JVM). To znači da se programi napisani u programskom jeziku Java ne prevode na mašinski jezik nekog stvarnog procesora, već u mašinski jezik Java virtuelne mašine, koji se naziva java bajtkod (*bytecodes*). Java bajtkod se ne može direktno izvršiti na stvarnom računaru, pa je neophodan interpretator Java bajtkoda, koji je dosta jednostavniji od kompleksne JVM i koji je različit za svaki tip računara, odnosno procesora.

Java platforma predstavlja Java programski jezik sa svim svojim softverskim komponentama (*Java Application Programming Interface*, skraćeno Java API). Java platforma je podeljena u 3 grupe:

- Java SE (*Standard Edition*) – za personalne računare, korišćen za izradu ovog diplomskog;
- Java ME (*Micro Edition*) – za uređaje sa manjim hardverskim mogućnostima, poput smartfonova, tableta;
- Java EE (*Enterprise Edition*) – za najzahtevnije aplikacije.

Klasa je osnovna jedinica u programiranju u jeziku Java. Klasa opisuje objekte, pomoću polja (definišu stanja objekta) i metoda (definišu ponašanje objekta) koje poseduju svi objekti te klase. Sami objekti se na Java platformi ne pišu, već se konstruišu iz napisanih klasa. Java platforma ima veliki broj unapred definisanih klasa koje su grupisane u pakete. Interfejsi deklarišu objekat koji je obrazovan samo od konstanti i apstraktnih metoda, razlika u odnosu na klasu je u tome što je interfejs stvar samo ugovora, a klasa kombinacija ugovora i implementacije.

U ovom poglavlju detaljnije će biti objašnjene samo klase i interfejsi koji su primenjeni u implementaciji koja je realizovana u okviru ove teze, kao i višenitno programiranje koje je ključno za rad realizovane implementacije.

2.1. Korišćene Java klase

U programu ovog diplomskog korišćeni su sledeći već definisani Java paketi:

- java.net – klase DatagramSocket, DatagramPacket i InetAddress;
- java.sql – klasa DriveManager i interfejsi Connection, Statement i ResultSet;
- javax.crypto – klase Cipher i SecretKeySpec;
- java.util – klase Arrays, Formatter i Calendar;
- java.awt – klase Color, Font, EventQueue i Event;
- java.swing – klase JFrame, JPanel, border.EmptyBorder, JLabel, JTextField, JButton i JTextArea i interfejs SwingConstants.

2.1.1. DatagramPacket, DatagramSocket i InetAddress

DatagramPacket je klasa koja predstavlja datagram paket. Datagram paket je paket koji se transportuje bez direktne uspostave veze, odnosno predstavlja paket koji se prenosi UDP protokolom. Svaka poruka se rutira samo na osnovu informacija u datagram paketu. Paketi stižu neuređenim redosledom i nema garancije da će paket stići. Datagram paket se konstruiše naredbom: *DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)*, gde je *buf* podatak koji se šalje i dužine je *length*, sa ofsetom *offset*, na adresu *address* i port *port.* Metod klase DatagramPacket koji je korišćen je *getData()* koji vraća korisničke podatke iz datagram paketa i oni su dati u bajtima.

DatagramSocket je klasa koja predstavlja soket (priključak) koji prima i šalje datagram pakete. Svaki paket koji se šalje preko datagram soketa ima jedinstvenu adresu i port, a više paketa stiže na isto odredište pri čemu njihov redosled ne mora biti očuvan. Da bi soket primao i slao pakete on mora prvo biti povezan sa određenom adresom (koja može biti definisana na različite načine), odnosno kreiran. Korišćena je naredba *DatagramSocket(int port, InetAddress laddr)*, gde je napravljen soket koji ima lokalnu adresu *laddr* i port čiji je broj *port*. Naredbama *send* i *receive* se šalju i primaju datagram paketi, respektivno.

InetAddress je klasa koja predstavlja IP adresu. Metod koji je korišćen je *getLocalHost()* koji vraća InetAddress lokalnog hosta.

2.1.2. Paket java.sql

DriveManager je javna klasa za postavljanje JDBC (*Java Database Connectivity* – API koja omogućava programskom jeziku Java da pristupa bazama podataka). Metod *getConnection(String url, String user, String password)* pokušava da uspostavi vezu sa bazom podataka koja ima navedeni *url*, i sa datim *user* i *password* podacima.

Connection je javni interfejs koji predstavlja konekciju (sesiju) sa određenom bazom podataka. Bez konekcije je nemoguće pristupati bazi. Metod *createStatement()* kreira objekat Statement koji šalje SQL naredbe bazi podataka. Na kraju treba zatvoriti ovaj interfejs naredbom *close()*.

Statement je javni interfejs kojim se obavljaju statične SQL naredbe i vraćaju njihovi odgovori. Metodom *executeQuery(String sql)* se vraća objekat ResultSet, a string *sql* je naredba napisana na programskom jeziku SQL. Na kraju treba zatvoriti ovaj interfejs naredbom *close()*.

ResultSet je javni interfejs, i predstavlja rezultat naredbe date bazi podataka. ResultSet pokazuje na trenutnu vrstu u tabeli rezultata. Metodom *next()* se kursor pomera na sledeću vrstu. Kursor može da ide samo u jednom smeru, inicijalna pozicija je pre prve vrste, a kada nema više vrsta vraća rezultat *false*. Metodom *getString(String columnLabel)* se dobija string koji ima vrednost kolone čije je ime *columnLabel* u vrsti na koju pokazuje ResultSet. Na kraju treba zatvoriti ovaj interfejs naredbom *close()*.

2.1.3. Cipher i SecretKeySpec

Cipher je klasa koja se koristi u enkripciji i dekripciji. Ova klasa zahteva da se navede koje operacije se izvršavaju na ulaznim podacima da bi se dobili izlazni. To se deklariše u stringu koji se naziva *transformation* i mora da sadrži ime kriptografskog algoritma koji se primenjuje i opciono koji je mod i dopuna (*padding*). Odnosno ima oblik "*algoritam/mod/padding*" ili samo "*algoritam*". Java platforma podržava sledeće standarde:

- AES, DES i RSA algoritme
- CBC i ECB modove
- više tipova dopune i mogućnost da dopuna ne postoji
- različite veličine ključa za šifrovanje

Metod *getInstance(String transfmation)*, gde string *transformation* ima vrednost željenog algoritma za šifrovanje, stvara objekat cipher koji koristi željeni algoritam. Metodom *init(int opmode, Key key)* inicijalizuje se šifra sa ključom *key*, a *opmode* pokazuje da li se radi o enkripciji ili dekripciji. Na kraju se sa *doFinal(byte[] input)* završe sve operacije enkripcije ili dekripcije nad nizom bajtova *input*.

SecretKeySpec je klasa koja predstavlja tajni ključ koji je nezavisan. Ova klasa se samo može koristiti na ključevima koji se predstavljaju nizom bajtova i kada nema nekih dodatnih parametara ključa. Ovaj objekat se konstruiše sa *SecretKeySpec(byte[] key, String algorithm)* gde se od ključa *key* dobija tajni ključ korišćenjem algoritma *algorithm.*

2.1.4. Arrays, Formatter i Calendar

Arrays je klasa kojom su omogućene razne operacije na nizovima, poput sortiranja i pretrage. Takođe omogućava da se nizovi vide kao liste. Ako je niz null, klasa pokazuje NullPointerException. U programu je primenjen metod *copyOf(byte[] original, int newLength)* koji od niza *original* pravi novi koji je dužine *newLength* ili skraćivanjem na tu dužinu ili dopunjavanjem nulama do te dužine.

Formatter je klasa koja omogućava podešavanje formata stringa. U programu je korišćen za ispisivanje trenutnog datuma i vremena. Metod *format(String format, Object .. args)* ispisuje formatirani string, na način koji je deklarisan stringom *format* i koristeći objekte *args*.

Calendar je klasa koja ima ulogu kalendara i omogućava konverziju vremenskih veličina. Objekat se konstruiše sa *Calendar().*

2.1.5. Paket java.awt

Paket java.awt je podrška za programiranje grafičkog korisničkog interfejsa (*Graphical User Interface* – GUI). Klasa Color predstavlja boje, a klasa Font fontove koji se mogu dodeliti GUI-u. Paket java.awt.event je posvećen obradi događaja. Interfejs ActionListener osluškuje da li je došlo do događaja. Klasa ActionPerformed pokazuje da je došlo do događaja.

2.1.6. Paket java.swing

Paket java.swing je noviji paket za programiranje GUI-a od paketa java.awt. Klasa JFrame predstavlja glavni prozor i ima naziv i definisanu ivicu. Komponente koje se pojavljuju na ekranu su definisane klasama JLabel, JTextField, JButton i JTextArea i predstavljaju labelu (statički tekst), polje za tekst (dinamički tekst), dugme i prostor za višelinijski tekst, respektivno. Klasa JPanel služi za organizovanje komponenti u prozoru, tj. panel na koji se stavljaju komponente. Klasa EmptyBorder predstavlja transparentnu ivicu komponente. Interfejs SwingConstants je kolekcija konstanti koje se koriste za pozicioniranje i orijentisanje komponenti na ekranu.

2.2. Višenitno programiranje

Nit kontrole (*thread*) je sekvenca koraka koji se izvršavaju sekvencijalno. Uobičajno kod tradicionalnih programskih jezika postoji samo jedna nit kontrole. Java podržava višenitno programiranje. Višenitno programiranje (*multi-threading*) je simultano izvršavanje više niti koje mogu deliti neke zajedničke podatke. Višenitni program može da se izvršava:

- konkurentno (ne pretpostavlja više procesora)
- paralelno (pretpostavlja postojanje više procesora koji omogućavaju stvarni paralelizam)

Ako dve niti mogu da modifikuju i čitaju iste podatke dolazi do efekta "neizvesnosti trke" (*race hazard*). Rezultat neizvesnosti trke može biti neispravno stanje nekog objekta, ali problem neizvesnosti trke rešava se sinhronizacijom niti. Sinhronizacija niti treba da obezbedi samo jednoj niti ekskluzivan pristup podacima u određenom segmentu koda. Niti su definisane klasom *Thread* u standardnoj biblioteci. Niti se mogu kreirati na 2 načina:

- Prvi način je da se aktivan objekat (sadrži vlastitu nit kontrole) kreira na sledeći način: *Thread nit=new Thread();* Nakon kreiranja niti ona se može konfigurisati (postavljanjem prioriteta, imena,...) i pokrenuti (pozivom *start* metode). Metod *start* aktivira novu nit kontrole (nit postaje spremna), a zatim se metod završava. Okruženje (virtuelna mašina) pokreće *run* metod aktivne niti, koji treba da bude redefinisan u korisničkoj klasi koja proširuje klasu *Thread*. Kada se *run* metod niti završi, nit završava izvršenje. Metod *run* ne baca izuzetke;
- Drugi način kreiranja niti je implementacijom interfejsa *Runnable*. Interfejs *Runnable* je apstrakcija koncepta aktivnog objekta, koji izvršava neki kod konkurentno sa drugim takvim objektima. Ovaj interfejs deklariše samo jedan metod: run

Problem u prvom načinu kreiranja niti je da kada se izvodi iz klase *Thread* ne može se izvoditi i iz neke druge. Problem se rešava korišćenjem drugog pristupa u kreiranju niti, tako što se napiše klasa koja implementira interfejs *Runnable*, kreira se objekat te klase i na kraju se prosledi objekat konstruktoru klase *Thread.*

Nit se završava normalno kada se vrati iz metoda *run()*.

Za program je primenjen drugi način kreiranja niti, čime je omogućeno simultano primanje i slanje servera i klijenta.

3. OPIS IMPLEMENTACIJE

Osnovna ideja je da klijent pošalje zahtev preko korisničkog interfejsa, zahtev se šifruje AES algoritmom, zatim se šifrovan pošalje UDP protokolom do servera, koji primljenu poruku dešifruje i pristupa bazi podataka da bi našao odgovor na zahtev koji je poslao klijent. Server potom odgovor šifruje isto AES algoritmom i šalje isto UDP protokolom do klijenta, koji dešifruje odgovor i potom ispisuje konačan odgovor preko svog interfejsa.

Program se sastoji od klasa glavna i prozor. Klasa glavna ima klase Server i Klijent i metode DB, sifrovanje, desifrovanje i main. U klasi prozor je definisan GUI, metode su main, initGUI i actionPerformed.

3.1. Klasa glavna

Polja klase glavna su definisana tako da se njima može pristupati iz drugih klasa i metoda:

```
public class banka {
        static String odgovor = "";
        static String kljuc = "vanjin diplomski";
```

3.1.1. Klasa Server

Klasa Server se implementira sa interfejsom Runnable, odnosno predstavlja višenit, čime je omogućeno simultano opsluživanje više korisnika. Ima privatna polja soket i paket, tipa DatagramSocket i DatagramPacket, respektivno. Metode koje se koriste u ovoj klasi su DB, šifrovanje i dešifrovanje. Konstruktor (operacije za inicijalizaciju objekta) Servera koristi referencu this i ima jedan parametar *soket* tipa DatagramSocket. Kod koji ovo opisuje:

```
class Server implements Runnable {
        private DatagramPacket paket;
        private DatagramSocket soket;
        public Server(DatagramSocket soket) {
                this.soket = soket;
        }
```

U telu klase Server se zatim pokreće metod *public void run()*. Onda se u *while(true)* petlji prvo deklarišu statička polja niz od 1024 bajta nazvan *bafer*, stringovi *zahtev* i *odgovor* koji nemaju inicijalne vrednosti i datagram paket nazvan *paket*:

```
byte[] bafer = new byte[1024];
String zahtev = new String();
String odgovor = new String();
paket = new DatagramPacket(bafer, bafer.length);
```

U try-catch sekvenci se prvo obavlja primanje paketa preko soketa. Iz paketa saznajemo adresu i port odakle je paket poslat. Potom se u baferu čuvaju korisnički podaci dobijeni iz novopristiglog paketa. Da bi podaci bili odgovarajuće dužine za dalju obradu oni se zatim skrate ili dopune nulama na dužinu od 16 bita:

```
soket.receive(paket);
InetAddress IPAddress = paket.getAddress();
int port = paket.getPort();
bafer = paket.getData();
bafer=Arrays.copyOf(bafer, 16);
```

Potom se redom primene metode dešifrovanje, DB i šifrovanje uz definisanje novog statičkog polja *bafer2* koji predstavlja niz bajtova:

```
zahtev = glavna.desifrovanje(bafer, glavna.kljuc);
odgovor = DB(zahtev);
byte[] bafer2 = glavna.sifrovanje(odgovor, glavna.kljuc);
```

Novi datagram paket se definiše, a kao korisnički podatak se koristi *bafer2* i on se šalje preko soketa na adresu i port koje smo sačuvali kada smo primili paket:

```
paket = new DatagramPacket(bafer2, bafer2.length, IPAddress, port);
soket.send(paket);
```

Na kraju se soket zatvara i naredbom *break* se izlazi iz while petlje.

```
soket.close();
break;
```

Posle *catch (Exception e)* završavaju se i while petlja i metod *run*, a i cela klasa Server.

3.1.2. Klasa Klijent

Klasa Klijent se implementira sa interfejsom Runnable, čime je omogućeno simultano postojanje više korisnika. Ima privatna polja soket i paket, tipa DatagramSocket i DatagramPacket, respektivno. Metode koje se koriste u ovoj klasi su šifrovanje i dešifrovanje. Konstruktor (operacije za inicijalizaciju objekta) Klijenta koristi referencu this i ima jedan parametar *soket* tipa DatagramSocket. Kod koji ovo opisuje:

```
class Klijent implements Runnable {
        private DatagramPacket paket;
        private DatagramSocket soket;
public Klijent(DatagramSocket soket) {
        this.soket = soket;
}
```

U telu klase Klijent se zatim pokreće metod *public void run()* i try-catch sekvenca. Zatim se pozove *while(true)* petlja. U petlji se deklariše niz bajtova nazvan *bafer2* koji dobija vrednost šifrovanog zahteva. Odnosno pozove se metod *sifrovanje* za string *prozor.zahtev* koji predstavlja podatke koje je klijent uneo preko GUI-a . Kreira se novi paket, čiji je korisnički podatak *bafer2*, a adresa i port na koju se šalje je poznata adresa i port servera, zatim se preko soketa taj paket i pošalje:

```
byte[] bafer2 = glavna.sifrovanje(prozor.zahtev, glavna.kljuc);
paket = new DatagramPacket(bafer2, bafer2.length, InetAddress.getLocalHost(),
18888);
  soket.send(paket);
```

Klijent treba i da primi odgovor koji mu server pošalje. To se radi tako što se soket podesi na primanje, u *bafer2* se smeste podaci iz pristiglog paketa. Važno je da *bafer2* ima dužinu od 16 bajtova da bi kada se na njega primeni metod dešifrovanje dobio tačan rezultat:

```
soket.receive(paket);
```

```
                bafer2 = paket.getData();
                bafer2=Arrays.copyOf(bafer2, 16);
            glavna.odgovor = glavna.desifrovanje(bafer2,glavna.kljuc);
```

Dešifrovan podatak je željeni odgovor dobijen od servera, pa je potrebno da se on prikaže klijentu preko GUI-a. Pored toga, hoćemo da prikažemo vreme i datum u kom smo pristupili podatku. Prvo kreiramo novi format *fmt*, pa *cal* tipa Calendar dobije vrednost trenutnog vremena, a *fmt* dobije željeni format vremena i datuma:

```
Formatter fmt = new Formatter();
Calendar cal = Calendar.getInstance();
fmt.format("%tc", cal);
prozor.textArea.insert(fmt+"\nStanje na računu "+prozor.zahtev+" je: \n"+glavna.odgovor+"
RSD\n", 0);
```

Petlja while se završava *break* naredbom, pa se posle *catch (Exception e)* završava metod *run*, a i cela klasa Klijent.

3.1.3. Metod DB

Ovaj metod se nalazi u okviru klase Server, jer ga samo ta klasa i poziva. Predstavlja komunikaciju servera sa bazom podataka. Treba da se na osnovu zahteva klijenta pronađe odgovor u bazi podataka. To se radi pretragom baze. Ako se odgovor pronađe on se prosledi serveru, a ako se ne nađe obaveštava se i server i klijent da za upućeni zahtev nema odgovora.

DB je javni statični metod koji ima za argument jedan string, a kao rezultat isto daje string. Prvo je neophodno pozvati JDBC Driver:

```
try {
        Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
}
```

Za rad sa bazama podataka u programskom jeziku Java neophodni su interfejsi: Connection, Statement i ResultSet koji se deklarišu sa imenima *con*, *stmt* i *rs*, respektivno i dobijaju null početnu vrednost:

```
Connection con = null;
Statement stmt = null;
ResultSet rs = null;
```

Onda se uspostavlja konekcija sa bazom podataka, koja se obavlja interfejsom Connection, gde je adresa "localhost:3306/banka", korisničko ime "root", a lozinka "" :

```
try {
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/banka",
"root", "");
} catch (SQLException e) {
}
```

Obavi se provera da li je baza konektovana sa serverom. Ako je con null to znači da konekcija nije uspostavljena i korisnik se o tome obaveštava:

```
if (con == null) {
        stanje="nepoznato";
        prozor.textArea.insert("Došlo je do greške prilikom povezivanja na bazu podataka,
molimo pokušajte ponovo. \n",0);
}
```

U try-catch-finally sekvenci se obave upiti baze podataka i dobije odgovor. Prvo se iz Connection interfejsa napravi Statement koji predstavlja SQL naredbu, a onda se u ResultSet upiše rezultat SQL upita. Upit traži u tabeli *racun* koja je vrednost :

```
stmt = con.createStatement();
rs = stmt.executeQuery("SELECT StanjeRacuna FROM racun WHERE BrojRacuna='" + broj + "'");
```

Potom se proverava rezultat upita. U slučaju da postoji odgovor on se upisuje u string *stanje*, u suprotnom se u *stanje* upisuje string „nepoznato“:

```
if (rs.next()) {
   stanje = rs.getString("StanjeRacuna");
}else {
if (stanje==""){//ako zahteva nema u bazi podataka
        stanje="nepoznato";
        prozor.textArea.insert("Ovo nije račun UDP-AES banke \n",0);
        }
}
```

U catch delu koda se hvata izuzetak (SQLExceotion), a u okviru finally dela se obavi zatvaranje rs, stmt i con interfejsa:

```
rs.close();
stmt.close();
con.close();
```

Na kraju se kao rezultat metoda vraća vrednost stringa *stanje*:

```
return (stanje);
```

3.1.4. Metod sifrovanje

Ovaj metod obavlja AES šifrovanje, to je javan statičan metod koji ima dva argumenta stringove *original* i *kljuc*, a rezultat je niz bajtova. Metod se poziva i u klasi Server i u klasi Klijent.

Cipher je klasa koja reprezentuje šifru u kriptografiji. Deklarišemo cipher klase Cipher i naznačimo da je ovo šifra za AES algoritam:

```
Cipher cipher = Cipher.getInstance("AES");
```

Novi SecretKeySpec nazvan *key* se konstruiše od stringa *kljuc* tako da bude tajni ključ za AES šifrovanje. String *kljuc* se prvo mora prebaciti u bajtove koristeći *charset* UTF-8 i to predstavlja ključ šifrovanja *key*:

```
SecretKeySpec key = new SecretKeySpec(kljuc.getBytes("UTF-8"), "AES");
```

Inicijalizuje se šifra cipher za šifrovanje koristeći tajni ključ *key*:

```
cipher.init(Cipher.ENCRYPT_MODE, key);
```

Metodom *cipher.doFinal* se obavlja šifrovanje stringa *original* koji se prvo prebaci u bajtove, a rezultat ovog metoda je i rezultat metoda *sifrovanje*:

```
return cipher.doFinal(zaSif.getBytes("UTF-8"));
```

3.1.5. Metod desifrovanje

Ovaj metod obavlja AES dešifrovanje, to je javan statičan metod koji ima dva argumenta niz bajtova *sifrovano* i string *kljuc*, a rezultat je niz bajtova. Metod se poziva i u klasi Server i u klasi Klijent.

Analogno kao kod metoda sifrovanje koristimo klasu Cipher jer reprezentuje šifru u kriptografiji. Deklarišemo cipher klase Cipher i naznačimo da je ovo šifra za AES algoritam:

```
Cipher cipher = Cipher.getInstance("AES");
```

Isto se konstruiše novi SecretKeySpec nazvan *key* od stringa *kljuc* tako da bude tajni kluč za AES dešifrovanje. String *kljuc* se prvo mora prebaciti u bajtove koristeći *charset* UTF-8 i to predstavlja ključ šifrovanja :

```
SecretKeySpec key = new SecretKeySpec(kljuc.getBytes("UTF-8"), "AES");
```

Inicijalizuje se šifra cipher za dešifrovanje koristeći tajni ključ *key*:

```
cipher.init(Cipher.DECRYPT_MODE, key);
```

Metodom *cipher.doFinal* se obavi dešifrovanje niza bajtova *sifrovano*, a rezultat ovog metoda je i rezultat metoda *desifrovanje*:

```
return new String(cipher.doFinal(zaDesif),"UTF-8");
```

3.2. Klasa prozor

Klasa prozor proširuje klasu JFrame i implementira interfejs ActionListener. Polja klase prozor su definisana na sledeći način:

```
static String zahtev = "";
static JTextArea textArea = new JTextArea();
private JPanel contentPane;
private final JLabel lblUdpAes = new JLabel("UDP - AES banka");
private final JLabel lblBrojRauna = new JLabel("Broj računa:");
private final JButton btnPoalji = new JButton("Pošalji");
private final JTextField textField = new JtextField();
```

Konstruiše se objekat prozor:

```
public prozor() {
        initGUI();
}
```

3.2.1. Metod main

Metod *main* klase prozor prvo kreira novu nit specifičnu za javax.swing paket. U okviru *run()* metoda ove niti i try-catch sekvence deklariše se prozor nazvan *frame* i podešava se da on bude vidljiv:

```
EventQueue.invokeLater(new Runnable() {
        public void run() {
                try {
                        prozor frame = new prozor();
                        frame.setVisible(true);
                } catch (Exception e) {
                        e.printStackTrace();
                        }
        }
```

```
});
```

3.2.2. *Metod initGUI*

Privatni metod *initGUI* ne vraća nikakvu vrednost, već inicijalizuje sve komponente GUI prozora.

Prvo definiše podrazumevano zatvaranje i veličinu prozora:

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setBounds(100, 100, 450, 300);
```

Potom se konstruišu i deklarišu osobine panela:

```
contentPane = new JPanel();
contentPane.setBackground(new Color(224, 255, 255));
contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
setContentPane(contentPane);
contentPane.setLayout(null);
```

Naslovna labela se opisuje i konstruiše na sledeći način:

```
lblUdpAes.setForeground(new Color(0, 0, 0));
lblUdpAes.setHorizontalAlignment(SwingConstants.CENTER);
lblUdpAes.setFont(new Font("Dialog", Font.BOLD, 30));
lblUdpAes.setBounds(12, 12, 424, 36);
contentPane.add(lblUdpAes);
```

Labela pored polja za tekst:

```
lblBrojRauna.setBounds(12, 87, 100, 15);
contentPane.add(lblBrojRauna);
```

Polje za tekst pored opisa i konstrukcije, sadrži i interfejs za osluškivanje događaja:

```
textField.setToolTipText("");
textField.setBounds(145, 85, 153, 19);
textField.setColumns(10);
textField.addActionListener(this);
contentPane.add(textField);
```

Dugme takođe sadrži intefejs za osluškivanje događaja sa osobinama:

```
btnPoalji.setBackground(new Color(224, 255, 255));
btnPoalji.setBounds(310, 82, 117, 25);
btnPoalji.addActionListener(this);
contentPane.add(btnPoalji);
```

Višelinijsko polje za tekst se konstruiše tako da tekst koji se ispiše u njemu ne može da se edituje:

```
textArea.setEditable(false);
textArea.setBackground(new Color(245, 255, 250));
textArea.setBounds(12, 145, 415, 110);
contentPane.add(textArea);
```

3.2.3. *Metod actionPerformed*

Metod *actionPerformed* ima argument *ActionEvent*, što znači da se metod pokreće ako se desio neki događaj.

Prvo se u string *zahtev* sačuva šta je korisnik uneo u polje za tekst. Višelinijsko polje za tekst se isprazni, tako da bude prazno za novi unos:

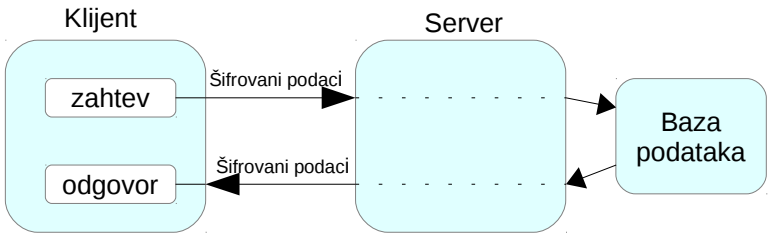
```
zahtev = textField.getText();
textArea.setText("");
```

U try-catch sekvenci se deklarišu i startuju nove niti klasa Server i Klijent. Klasi Server je argument datagram soket sa portom broj 18888 (za potrebe rada je izabran ovaj broj, ali može da se postavi i druga vrednost) i na tom portu će server osluškivati da li su pristigli zahtevi klijenata. Argument klase Klijent nema specificiran broj porta, već se bira neki slobodan:

```
new Thread(new Server(new DatagramSocket(18888))).start();
new Thread(new Klijent(new DatagramSocket())).start();
```

4. TESTIRANJE

Klijent šalje zahtev preko korisničkog interfejsa, server ga prima i pristupa bazi podataka da bi našao odgovor na zahtev koji je poslao klijent. Server šalje odgovor klijentu, kome se odgovor ispiše preko interfejsa. Podaci se šalju šifrovani, klijent i server obavljaju šifrovanje i dešifrovanje.



Slika 4.1. Klijent/server komunikacija

Testiranje je obavljeno tako što su server i klijent bili na istoj mašini. Za adresu je korišćen *localhost*, odnosno adresa servera i klijenta je ista 127.0.0.1, ali portovi su različiti. Za server je odlučeno da port bude 18888 i on je sve vreme isti i poznat je klijentu. Klijent šalje sa slobodnih portova, koji nisu unapred definisani. Server na osnovu primljenog paketa saznaje port sa kog je poslat paket i na taj port šalje odgovor. Testiranje je moguće i ako server i klijent nisu na istoj adresi, treba samo promeniti njihove IP adrese u kodu.

Slika 4.2. Server i klijent na localhost-u

Za pokretanje baze podataka korišćena je aplikacija *MySQL server*. Baza podataka *banka* koja je korišćena ima jednu tabelu *racun* sa kolonama *idRacuna*, *BrojRacuna*, *ImeIPrezime* i *StanjeRacuna*. Kolona *idRacuna* je tipa *int(13)* i predstavlja redni broj u tabeli. Kolona *BrojRacuna* je tipa *varchar(13)* i predstavlja broj računa u banci. Kolona *ImeIPrezime* je tipa *varchar(45)* i predstavlja ime i prezime osobe koja ima račun u banci. Kolona *StanjeRacuna* je tipa varchar(20) i predstavlja stanje računa u banci.

Tabela 4.1. Tabela *racun* baze podataka *banka*

idRacuna	***BrojRacuna***	***ImeIPrezime***	***StanjeRacuna***
1	1234567890123	Petar Petrović	12.345,67
2	1234567890234	Marko Marković	5.678,90
3	1234567890345	Marija Marić	6.789.012,34
4	1234567890456	Ivana Ivanović	7,89

Korisnik treba da unese broj računa čije stanje želi da sazna u polje za tekst. Kada se pritisne dugme Pošalji u prostoru za višelinijski tekst se ispisuje vreme i datum, stanje računa i eventualno obaveštenje zašto ne može da se ispiše stanje računa.

Tokom testiranja prvo je poslat upit dok baza podataka banka još nije bila povezana na server i tada se dobija obaveštenje da nije došlo do povezivanja sa bazom podataka.

Slika 4.3. GUI kada server nije povezan sa bazom

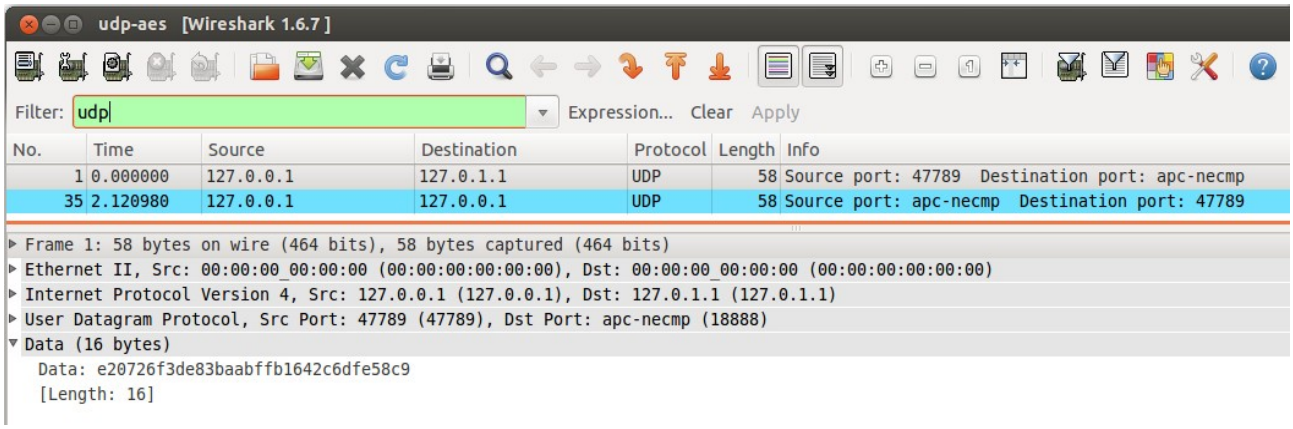
Zatim je poslato više zahteva koji nisu brojevi računa i dobijen je odgovor da ne postoje takvi računi u bazi podataka.

Slika 4.4. GUI kada nema odgovora na zahtev

Potom je u polje za tekst unet string „1234567890123“, koji jeste broj računa iz baze podataka i pritisnuto je dugme „Pošalji“ i dobijeno je ispravno stanje računa.

Slika 4.5. GUI sa uspešnim ispisom

Koristeći program *Wireshark* praćen je tok UDP paketa tokom testiranja:



Slika 4.6. Provera UDP protokola u programu *Wireshark*

Ovime je potvrđeno da aplikacija uspešno radi.

Aplikacija je testirana uz manju izmenu i za slučaj kada postoje dva klijenta koje server treba da opsluži. Testirano je ako oba klijenta pošalju zahteve na koje nema odgovora u bazi podataka, ako jedan klijent pošalje ispravan zahtev, a jedan ne i ako oba klijenta pošalju ispravne zahteve. U sva tri slučaja aplikacija je radila ispravno.

5. ZAKLJUČAK

Zadatak da se napravi sigurna komunikacija između klijenta i servera je urađen uspešno. Programski jezik Java se pokazao odličnim izborom za implementaciju. Lako su nađena rešenja za kriptografiju, komunikaciju UDP protokolom i povezivanje na bazu podataka u postojećim paketima jezika Java i time je pisanje koda i razvoj implementacije značajno skraćeno. Višenitno programiranje je omogućilo da se sve obavlja u realnom vremenu.

Program je moguće dalje usavršavati i unapređivati. Na primer, može se napraviti složenija komunikacija između klijenta i servera, tako što bi se slalo više podataka. Takođe, moguće je dodati još klijenata koje će server opsluživati i to je podržano višenitnim programiranjem. Manjim izmenama ovog programa moguće je i napraviti zaštićeno dopisivanje (*chat*) između dva ili više korisnika. Jednostavno se menja algoritam šifrovanja, na primer umesto AES može da se koristi DES ili RSA. Protokol UDP može biti zamenjen TCP protokolom samo korišćenjem drugih klasa java.net paketa. Namene i primene programa su razne, svugde gde je potreban brz pristup udaljenoj bazi podataka uz kriptografsku zaštitu prenosa, a bez zahtevanja kontrole greške.

LITERATURA

[1] Endru S. Tanenbaum, Računarske mreže, Mikro knjiga, 2005.

[2] Oracle documentacion, http://docs.oracle.com/javase/

[3] IR2OO2 predavanja, http://rti.etf.bg.ac.rs/rti/ir2oo2/predavanja/

A. KOD PROGRAMA

A.1. Fajl prozor.java

```
import java.awt.EventQueue;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.*;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JTextArea;

import java.net.DatagramSocket;

//prozor prosiruje klasu jframe i implementira slusanje dogadjaja----------
public class prozor extends JFrame implements ActionListener {
	static String zahtev = "";			//koristi se u klasi glavna
	static JTextArea textArea = new JTextArea();  //koristi se u klasi glavna

	private JPanel contentPane;
	private final JLabel lblUdpAes = new JLabel("UDP - AES banka");
	private final JLabel lblBrojRauna = new JLabel("Broj računa:");
	private final JButton btnPoalji = new JButton("Pošalji");
	private final JTextField textField = new JTextField();

	//pokretanje programa-----------------------------------------------------
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					prozor frame = new prozor();
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}
	//kreiranje objekta prozor------------------------------------------------
	public prozor() {
		initGUI();
	}
	//definisanje izgleda prozora---------------------------------------------
	private void initGUI() {
```

```
        //zatvaranje i velicina prozora
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setBounds(100, 100, 450, 300);
                //panel, boja pozadine, granica
                contentPane = new JPanel();
                contentPane.setBackground(new Color(224, 255, 255));
                contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
                setContentPane(contentPane);
                contentPane.setLayout(null);
                //labela, boja, pozicija, font, velicina
                lblUdpAes.setForeground(new Color(0, 0, 0));
                lblUdpAes.setHorizontalAlignment(SwingConstants.CENTER);
                lblUdpAes.setFont(new Font("Dialog", Font.BOLD, 30));
                lblUdpAes.setBounds(12, 12, 424, 36);
                contentPane.add(lblUdpAes);
                //labela, velicina i pozicija
                lblBrojRauna.setBounds(12, 87, 100, 15);
                contentPane.add(lblBrojRauna);
                //polje za tekst, prazno,velicina i pozicija, broj kolona
                textField.setToolTipText("");
                textField.setBounds(145, 85, 153, 19);
                textField.setColumns(10);
                textField.addActionListener(this);//ako pritisnemo enter dogadjaj!
                contentPane.add(textField);
                //dugme, pozadina, velicina i pozicija
                btnPoalji.setBackground(new Color(224, 255, 255));
                btnPoalji.setBounds(310, 82, 117, 25);
                btnPoalji.addActionListener(this);//osluskuje da li je pritisnuto
                contentPane.add(btnPoalji);
                //viselinijsko polje, pozadina, velicina i pozicija
                textArea.setEditable(false);//tekst se ne moze menjati
                textArea.setBackground(new Color(245, 255, 250));
                textArea.setBounds(12, 145, 415, 110);
                contentPane.add(textArea);

        }

        //dogadjaj je ili pritisak dugmeta posalji
        //ili enter na tastaturi u polju za tekst
        //sta se desava ako imamo dogadjaj
        public void actionPerformed(ActionEvent e){
                zahtev = textField.getText();//zahtev cuva sta je korisnik uneo
                textArea.setText("");//brisanje sta je ranije pisalo u viselinijskom polju
                try {
                        new Thread(new Server(new DatagramSocket(18888))).start();//nova nit
Server
                        new Thread(new Klijent(new DatagramSocket())).start();//nova nit Klijent
                        } catch (Exception e1) {
                        e1.printStackTrace();
                        }

        }
}
```

A.2. Fajl glavna.java

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

import java.sql.*;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

import java.util.Arrays;


import java.util.Formatter;
import java.util.Calendar;

//klasa opisuje rad klijenta----------------------------------------------------
class Klijent implements Runnable {

   private DatagramPacket paket;
   private DatagramSocket soket;

   public Klijent(DatagramSocket soket) {
      this.soket = soket;
   }

   public void run() {

      try {
         while (true) {
            byte[] bafer2 = glavna.sifrovanje(prozor.zahtev, glavna.kljuc);//sifrovanje unetog
zahteva
            paket = new DatagramPacket(bafer2, bafer2.length, InetAddress.getLocalHost(),
18888);
            soket.send(paket);//slanje paketa

            soket.receive(paket);//primanje paketa
            bafer2 = paket.getData();//korisnickicki podaci
            bafer2=Arrays.copyOf(bafer2, 16);//mora da bude duzina 16 zbog desifrovanja
            glavna.odgovor = glavna.desifrovanje(bafer2,glavna.kljuc);//desifrovanje

            Formatter fmt = new Formatter();//novi format
            Calendar cal = Calendar.getInstance();//novo vreme i datum
            fmt.format("%tc", cal);//kreiranje vremenskog formata
            prozor.textArea.insert(fmt+"\nStanje na računu "+prozor.zahtev+" je:
\n"+glavna.odgovor+" RSD\n", 0);//ispis primljenog podatka

            break;//prekid da ne bi se beskonacno izvrsavala petlja
         }
      } catch (Exception e) {
         System.err.println("Exception");
         e.printStackTrace();
      }
   }
}
```

```
//klasa koja opisuje rad servera----------------------------------------------
class Server implements Runnable {

    private DatagramPacket paket;
    private DatagramSocket soket;


    public Server(DatagramSocket soket) {
        this.soket = soket;
    }
    //metod za komunikaciju sa bazom podataka----------------------
    public static String DB(String broj) {
        try {
            Class.forName("com.mysql.jdbc.Driver");//povezivanje sa JDBC driver-om
        } catch (ClassNotFoundException e) {
        }
        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;
        String stanje = "";
        try {//na koju adresu, sa kojim  korisnickim imenom i sifrom se povezati
            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/banka", "root",
"gari");
        } catch (SQLException e) {
        }
        if (con == null) {//ako se nije povezao na bazu
           stanje="nepoznato";
           prozor.textArea.insert("Došlo je do greške prilikom povezivanja na bazu podataka,
molimo pokušajte ponovo. \n",0);
        }

        try {
            stmt = con.createStatement();
            rs = stmt.executeQuery("SELECT StanjeRacuna FROM racun WHERE BrojRacuna='" +
broj + "'");//upit baze podataka
            if (rs.next()) {
                stanje = rs.getString("StanjeRacuna");
            }else {
                    if (stanje==""){//ako zahteva nema u bazi podataka
                            stanje="nepoznato";
                            prozor.textArea.insert("Ovo nije račun UDP-AES banke \n",0);
                            }
                    }
        } catch (Exception e) {
        } finally {
            try {
                rs.close();
                stmt.close();
                con.close();
            } catch (SQLException ex) {
                System.err.println("SQLException");
            }
        }
        return (stanje);


    }
```

```
    //-------------------------------------------------------
    public void run() {

       while (true) {
          byte[] bafer = new byte[1024];
          String zahtev = new String();
          String odgovor = new String();
          paket = new DatagramPacket(bafer, bafer.length);
          try {
             soket.receive(paket);//primanje paketa
             InetAddress IPAddress = paket.getAddress();//cuvanje adrese sa koje je paket poslat
             int port = paket.getPort(); //cuvanje broja porta sa koje je paket poslat
             bafer = paket.getData();
             bafer=Arrays.copyOf(bafer, 16);//duzina podatka mora biti 16 zbog desifrovanja
             zahtev = glavna.desifrovanje(bafer,glavna.kljuc);

             odgovor = DB(zahtev);//za zahtev nadji odgovor u bazi podataka

             byte[] bafer2 = glavna.sifrovanje(odgovor,glavna.kljuc);
             paket = new DatagramPacket(bafer2, bafer2.length, IPAddress, port);
             soket.send(paket);//odgovor se salje na adresu sa koje je poslat zahtev

             soket.close();//zatvaranje soketa da bi bilo moguce ponovo poslati zahtev
             break;//prekid da ne bi se beskonacno izvrsavala petlja
          } catch (Exception e) {
             System.err.println("Exception");
             e.printStackTrace();
          }
       }
    }
}
//-------------------------------------------------------------------------

public class banka {

static String odgovor = "";
static String kljuc = "vanjin diplomski";

    public static void main(String[] args) throws Exception {

    }
//-------------------------------------------------------------------------

    public static byte[] sifrovanje(String zaSif, String kljuc) throws Exception {
          Cipher cipher = Cipher.getInstance("AES");//sifra za AES algoritam
          SecretKeySpec key = new SecretKeySpec(kljuc.getBytes("UTF-8"), "AES");//tajni kljuc
          cipher.init(Cipher.ENCRYPT_MODE, key);//inicijalizacija sifre za enkripciju
          return cipher.doFinal(zaSif.getBytes("UTF-8"));//sifrovanje podatka
    }
 //-------------------------------------------------------------------------

    public static String desifrovanje(byte[] zaDesif, String kljuc) throws Exception{
          Cipher cipher = Cipher.getInstance("AES");//sifra za AES algoritam
          SecretKeySpec key = new SecretKeySpec(kljuc.getBytes("UTF-8"), "AES");//tajni kljuc
          cipher.init(Cipher.DECRYPT_MODE, key);//inicijalizacija sifre za dekripciju
          return new String(cipher.doFinal(zaDesif),"UTF-8");//desifrovanje podatka
    }
```

B. SKRAĆENICE

AES - *Advanced Encryption Standard*

API - *Application Programming Interface*

CBC - *Cipher-block chaining*

DES - *Data Encryption Standard*

ECB – *Electronic CodeBook*

GUI - *Graphical User Interface*

JDBC - *Java Database Connectivity*

JVM - *Java Virtual Machine*

SQL - *Structured Query Language*

TCP - *Transmission Control Protocol*

UDP - *User Datagram Protocol*