Sigurnosni poslužitelj za buduću sigurnost e-pošte

Mohorić, Antonio

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Mohorić

Sigurnosni poslužitelj za buduću sigurnost e-pošte

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Mohorić

JMBAG: 0016142069

Studij: Informacijski sustavi

Sigurnosni poslužitelj za buduću sigurnost e-pošte

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Nikola Ivković

Varaždin, rujan 2022.

Antonio Mohorić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj rad opisuje implementaciju i funkcioniranje sigurnosnog poslužitelja koji omogućava privatnost, sigurnost i buduću sigurnost e-pošte. Na sigurnosni poslužitelj se putem klijentske aplikacije šalju dijelovi ključeva za kriptiranje koji pripadaju određenom korisniku (Diffie-Hellmanova metoda) i njihovo vremensko trajanje. Klijentska aplikacija prilikom slanja e-pošte traži od sigurnosnog poslužitelja dijelove ključeva određenog primatelja, a poslužitelj ih vraća (ako postoje). Također, ključ se briše iz datoteke ako mu je isteklo vremensko trajanje. Ovakav sigurnosni poslužitelj i klijentska aplikacija koja kriptira sadržaj zajedno osiguravaju buduću sigurnost e-pošte.

Ključne riječi: sigurnosni poslužitelj, buduća sigurnost, e-pošta, Diffie-Hellman

Sadržaj

1. Uvod

Slanje e-pošte nije sigurno. Koristi li se SMTP (*simple mail transfer protocol*) za slanje e-pošte bez TLS-a (*transport layer security*), napadač može presresti poruku na bilo kojoj od poveznica. Ako se koristi TLS, veza na poveznicama je sigurna, ali naša poruka na poslužitelju nije, odnosno napadač može čitati, mijenjati i preusmjeravati naše e-mailove na poslužitelju. PGP (*pretty good privacy*) nudi rješenje za problem sigurnosti na poslužiteljima, ali nema buduće sigurnosti. [1]

Kako bismo osigurali privatnost i sigurnost možemo uvesti paralelan sustav za sigurnost e-pošte koji će raditi sa svim postojećim e-mail poslužiteljima. Takav sustav bi omogućio sigurno kriptiranje poruke, te bi se onda tako kriptirana poruka slala kroz nesigurni sustav za slanje e-maila, bez mogućnosti dekriptiranja. Dekriptirati poruku mogu samo pošiljatelj i primatelj poruke. [1]

U današnje vrijeme su sigurnost i privatnost sve bitniji aspekti korištenja ne samo e-pošte nego i cjelokupnog interneta. Implementacija sustava kojega čine klijentska aplikacija i sigurnosni poslužitelj bi omogućila sigurnost, privatnost i buduću sigurnost u slanju e-pošte, što je detaljnije objašnjeno u radu.

2. Teorijski principi rada sigurnosnog poslužitelja

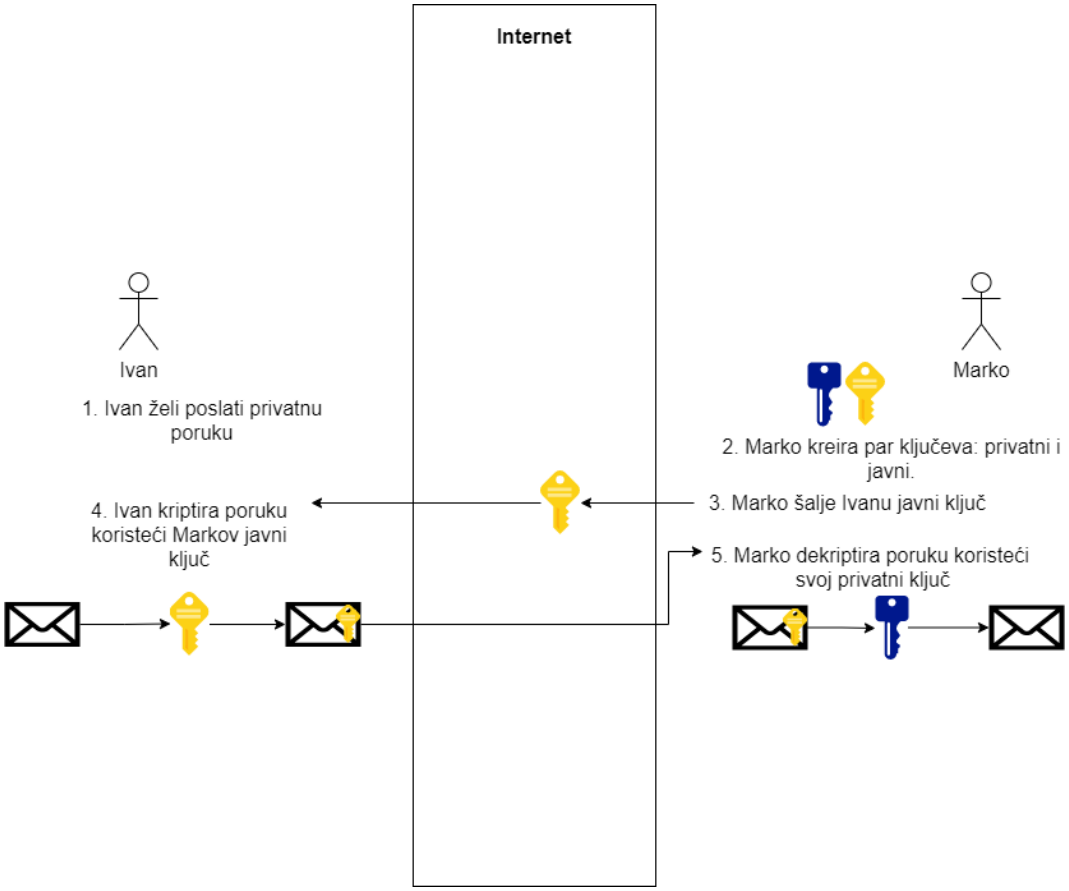
U ovom poglavlju detaljno je objašnjeno što i na koji način radi sigurnosni poslužitelj koji omogućava buduću sigurnost e-pošte.

2.1. SMTP i TLS

SMTP ili jednostavni protokol za slanje e-maila je standardni protokol koji se koristi u elektroničkoj komunikaciji. Počeo se globalno koristiti 80-ih godina, a 1995. godine se definira proširena verzija SMTP protokola ESMTP (*extended*) koja je nadopunila sve prethodno primijećene nedostatke. Običan SMTP ne koristi nikakvo kriptiranje pa naše poruke mogu biti lako snimljene i manipulirane od strane napadača. Kako bi se to izbjeglo uveden je TLS i tada SMTP postaje SMTPS (*secure*). Omogućavanjem TLS-a kriptira se SMTP protokol na transportnom sloju unutar TLS veze. Na taj način smo riješili problem kada napadač snima poruke na poveznici.[2]

2.2. PGP

PGP je način asimetrične enkripcije koji je uveden kako bi se kriptirale poruke i osigurala privatnost i sigurnost, a nastao je 1991. godine. Radi na način koji je prikazan na sljedećoj slici:



Slika 1: PGP (autorski rad)

Možemo primijetiti da napadač u ovom slučaju može dohvatiti javni ključ korisnika i poruku kriptiranu tim javnim ključem, ali je nikako ne može dekriptirati bez privatnog ključa korisnika [3]. Iz slike je jasno vidljivo na koji način PGP funkcionira i pruža privatnost poruka, no i kod ovakve komunikacije postoji problem koji je objašnjen u sljedećem poglavlju.

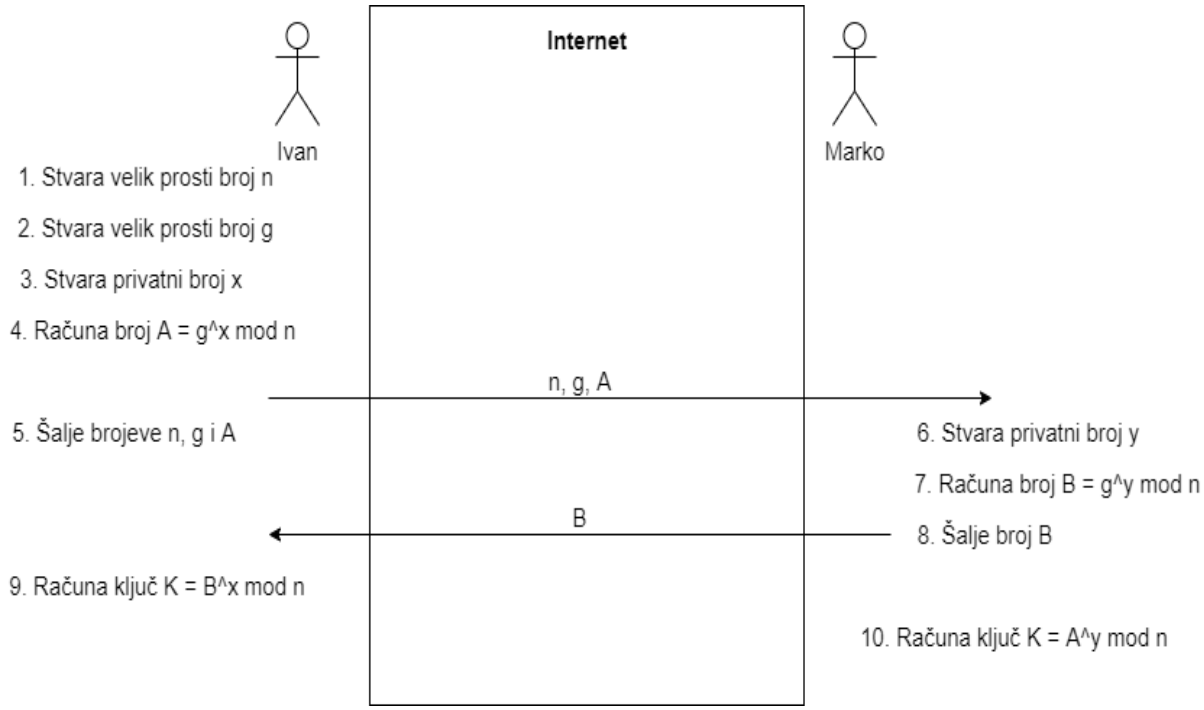
2.3. Buduća sigurnost

Dakle, zaključili smo da je PGP dobar način komunikacije na koji se može izbjeći napad na poveznici i na poslužitelju, ali se javlja novi sigurnosni zahtjev. Ako bi napadač snimao komunikaciju neko duže vrijeme i uspio nakon nekog vremena saznati privatni ključ za dekriptiranje, mogao bi pročitati sve što je do sada snimio. Ako je netko uporan i snima više godina promet, može čekati pad poslužitelja, napredak tehnologije ili neki propust korisnika i doći do tog privatnog ključa, a upravo to je problem buduće sigurnosti.[4]

Buduća sigurnost znači da napadač neće moći pročitati sve poruke koje je snimio ako na neki način uspije probiti privatni ključ.

2.4. Diffie-Hellmanova metoda

Diffie-Hellmanova metoda razmjene ključeva koristi javne poveznice za razmjenu kriptografskih ključeva. Metoda je dobila ime po svojim osnivačima: Whitfield Diffie i Martin Hellman. Metoda radi na način da se simetrični ključ za kriptiranje ne šalje niti u jednom trenutku, već se šalju dijelovi ključa pomoću kojih se simetrični ključ računa. Na taj način obje strane dobiju jednak ključ, a niti u jednom trenutku taj ključ nije bio ugrožen od strane napadača [5]. Funkcioniranje ove metode razmjene ključa možemo vidjeti na sljedećoj slici:



Slika 2: Diffie-Hellmanova metoda (autorski rad)

Upravo prema ovoj metodi je implementiran sigurnosni poslužitelj e-pošte. Razmjena ključeva radi na način opisan na slici, a to je da prvi korisnik najprije kreira par velikih prostih brojeva i svoj privatni broj. Zatim računa broj A prema napisanoj formuli te šalje kreirane proste brojeve i izračunati broj A drugom korisniku. Drugi korisnik kreira svoj privatni broj i računa broj B s primljenim prostim brojevima. Nakon toga natrag šalje prvom korisniku izračunati broj B. Kada su korisnici razmijenili izračunate brojeve, mogu izračunati ključ za kriptiranje koristeći napisane formule. Primijetimo da su oba korisnika izračunala jednak ključ za kriptiranje bez da su razmijenili privatne ključeve [5], [6]. Koristeći ovu metodu i izračunavanjem novoga ključa prilikom svake nove razmjene poruka osiguravamo buduću sigurnost (perfect forward secrecy). Ako napadač snima duži period komunikaciju i na neki način uspije otkriti jedan

privatni ključ pomoću kojega može izračunati ključ za kriptiranje, moći će dekriptirati samo tu jednu poruku, što nam osigurava buduću sigurnost.

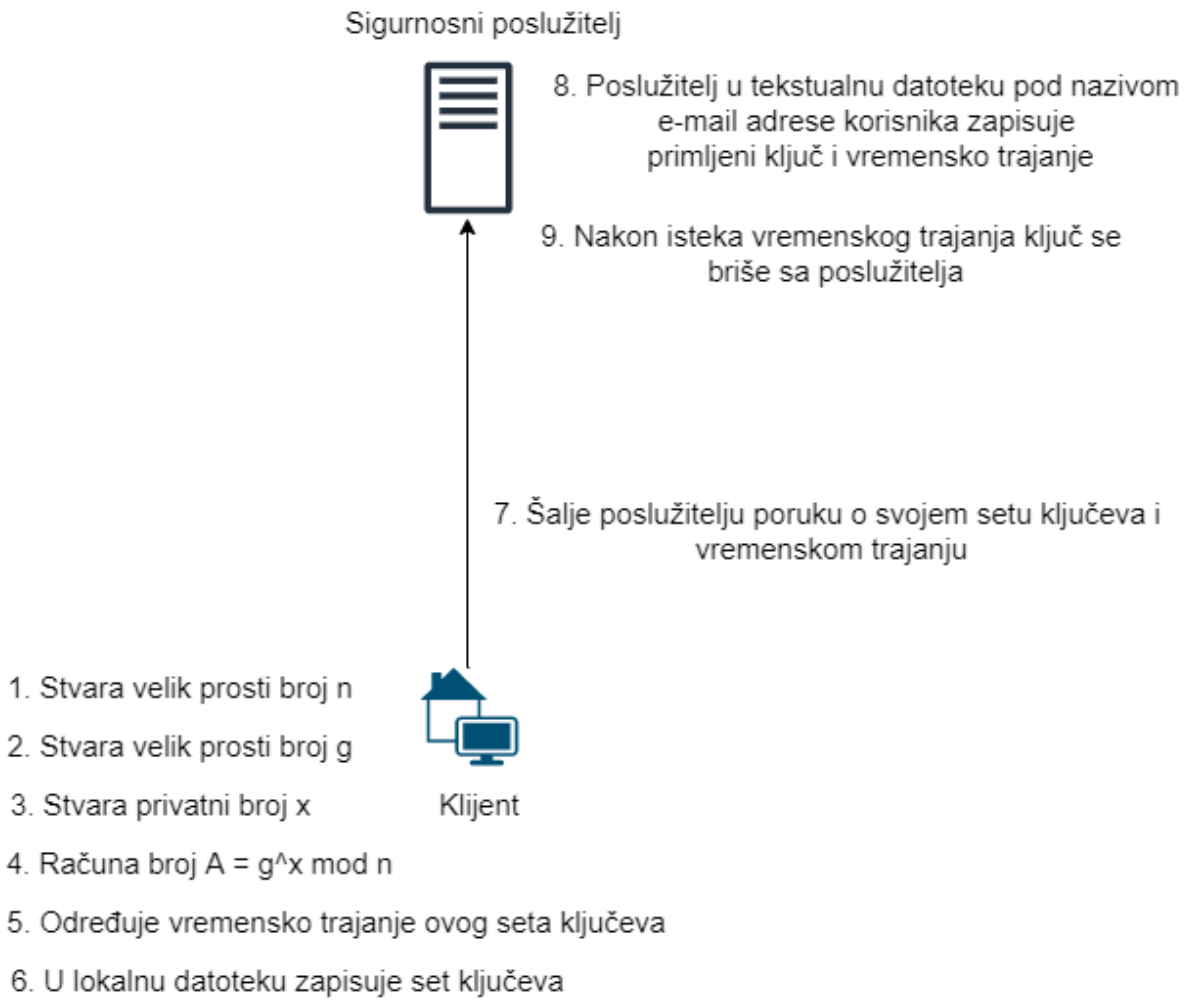
3. Implementacija sigurnosnog poslužitelja

U ovom poglavlju je najprije teorijski objašnjeno funkcioniranje sigurnosnog poslužitelja, a zatim i programski kod i sama konfiguracija sigurnosnog poslužitelja.

3.1. Teorijski opis rada sigurnosnog poslužitelja

3.1.1. Slanje ključeva na poslužitelj

Kako bi mogli pokazati korist sigurnosnog poslužitelja, potrebno je napraviti pomoćnu klijentsku aplikaciju koja bi komunicirala s poslužiteljem. Najprije korisnici moraju poslati na poslužitelj svoj set ključeva kako bi im drugi korisnici mogli slati e-poštu koristeći sigurnosni poslužitelj. To slanje ključeva je zamišljeno na način prikazan na slici:

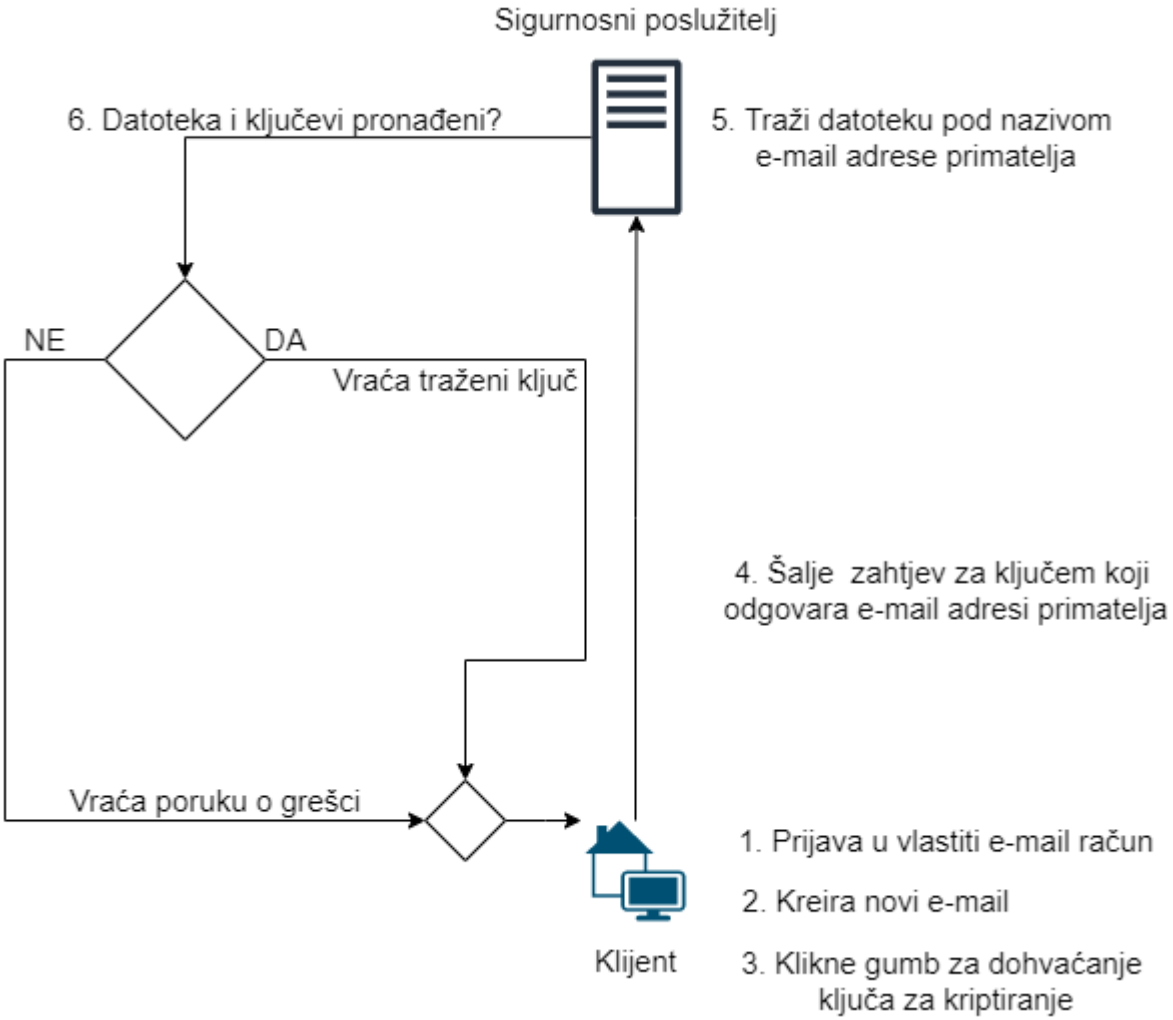


Slika 3: Slanje seta ključeva na sigurnosni poslužitelj (autorski rad)

Dakle, u klijentskoj aplikaciji korisnik kreira set ključeva kako je opisano u Diffie-Hellmanovoj metodi i određuje vremensko trajanje ključeva. Nakon toga šalje brojeve n, g, A i vremensko trajanje putem TCP socketa na određeni port na poslužitelj. Na poslužitelju je pokrenut program koji automatski obrađuje pristigle zahtjeve. Ovisno o tome kakav je zahtjev pristigao, program poduzima različite akcije. Ako je stigla poruka o slanju seta ključeva, onda poslužitelj u mapi gdje su datoteke u kojima su zapisani ključevi korisnika traži datoteku koja ima naziv jednak e-mail adresi korisnika od kojega je zahtjev stigao. Ako datoteka postoji, u nju se zapisuju dobiveni podatci, ako datoteka ne postoji, onda se kreira nova. Poslužitelj svake sekunde automatski provjerava je li vremensko trajanje isteklo, te ako jest onda briše set ključeva iz datoteke. Ako su izbrisani svi ključevi iz datoteke, to jest ako je datoteka prazna, onda se i ta datoteka briše.

3.1.2. Dohvaćanje ključeva s poslužitelja

Kako bi korisnik poslao e-mail drugome korisniku, najprije se prijavljuje u svoj korisnički račun. Nakon toga može čitati i slati poštu. Kada želi poslati poštu određenom korisniku, upisuje njegov e-mail u za to predviđeno mjesto, a zatim piše poruku i šalje. Ako korisnik želi, poruka se može kriptirati. Pritiskom na određeni gumb se šalje zahtjev za dohvaćanjem seta ključeva koji odgovaraju e-mail adresi primatelja na naš sigurnosni poslužitelj, što vidimo na sljedećoj slici:



Slika 4: Dohvaćanje seta ključeva sa sigurnosnog poslužitelja (autorski rad)

Dakle, dohvaćanje dijelova ključa, kako je opisano Diffie-Hellmanove metode, započinje kada korisnik kreira e-poštu i stisne gumb za dohvaćanje ključa. U tom trenutku aplikacija šalje zahtjev za dohvaćanjem ključa na sigurnosni poslužitelj. Program koji automatski obrađuje zahtjeve prepoznaje da je stigao zahtjev za dohvaćanjem ključa, i tada kreće tražiti datoteku koja ima naziv jednak e-mail adresi primatelja. Ako datoteka nije pronađena, vraća grešku korisniku od kojega je pošta pristigla. Ako je datoteka pronađena, poslužitelj ju čita i vraća poruku s ključem za traženog korisnika korisniku od kojega je pošta pristigla. Klijentska aplikacija čega odgovor na odgovor od poslužitelja nakon čega dalje nastavlja s radom.

3.2. Opis implementacije sigurnosnog poslužitelja

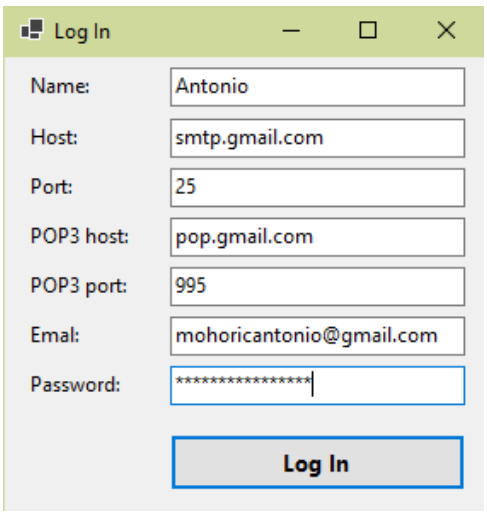
3.2.1. Implementacija pomoćne klijentske aplikacije

Za izradu klijentske aplikacije korišten je programski jezik C# u okruženju Visual Studio 2022. U programsko rješenje su dodana 2 NuGet paketa: MailKit za jednostavniji rad s e-mailom i Guna.UI2.WinForms za vizualno uljepšavanje korisničkih sučelja. Za pohranu i verzioniranje programskog koda korištena je aplikacija GitHub Desktop.

Kao što smo do sada već objasnili, kako bismo mogli objasniti funkcionalnosti sigurnosnog poslužitelja potrebno je bilo napraviti pomoćnu klijentsku aplikaciju. Klijentska aplikacija se sastoji od 5 dijelova:

1. Prijava korisnika svojim korisničkim podatcima
2. Čitanje pristigle e-pošte
3. Slanje nove e-pošte
4. Slanje ključa za kriptiranje na poslužitelj
5. Dohvaćanje ključa s poslužitelja

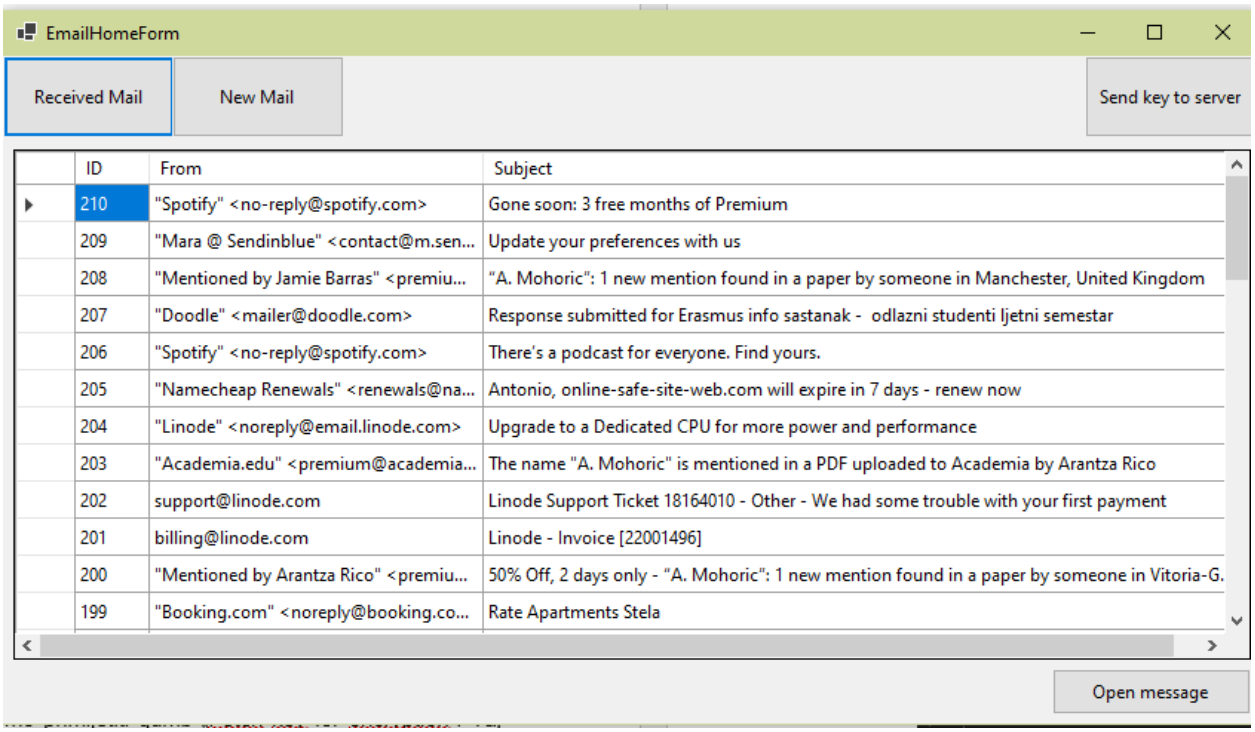
Otvaranjem aplikacije korisniku se prikazuje obrazac putem kojega unosi podatke za prijavu u svoj korisnički račun, što vidimo na sljedećoj slici:



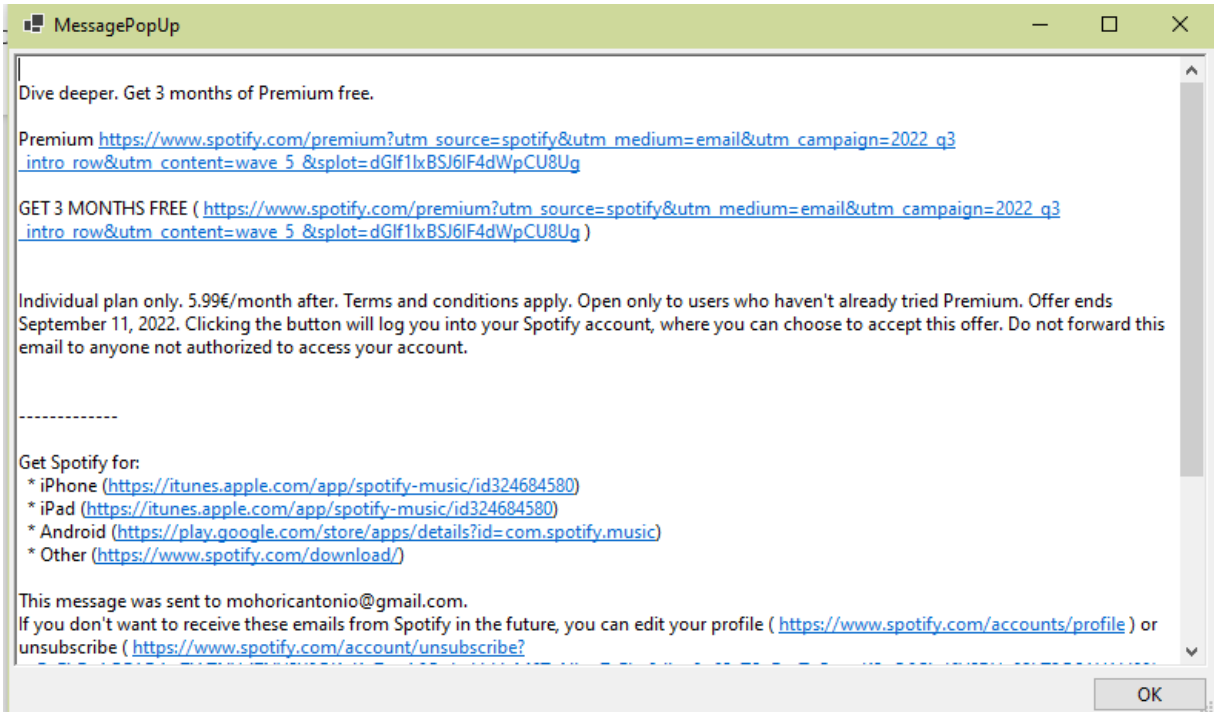
Slika 5: Log In obrazac (autorski rad)

Nakon što se korisnik prijavio može koristiti svoj e-mail račun za slanje i čitanje e-mailova. Po uspješnoj prijavi kreće učitavanje pristigle e-pošte i korisnik treba pričekati nekoliko sekundi dok se poruke učitaju, a zatim se otvara obrazac na kojemu imamo dva

izbornika: „Received Mail“ i „New Mail“. Automatski je odabran izbornik „Received Mail“ na kojemu su prikazani redni broj poruke, pošiljatelj i predmet e-pošte. Također vidimo gumb „Open message“ na kojega možemo kliknuti te nam se otvara tekst poruke. Izbornik „Received Mail“ vidimo na sljedeće dvije slike:

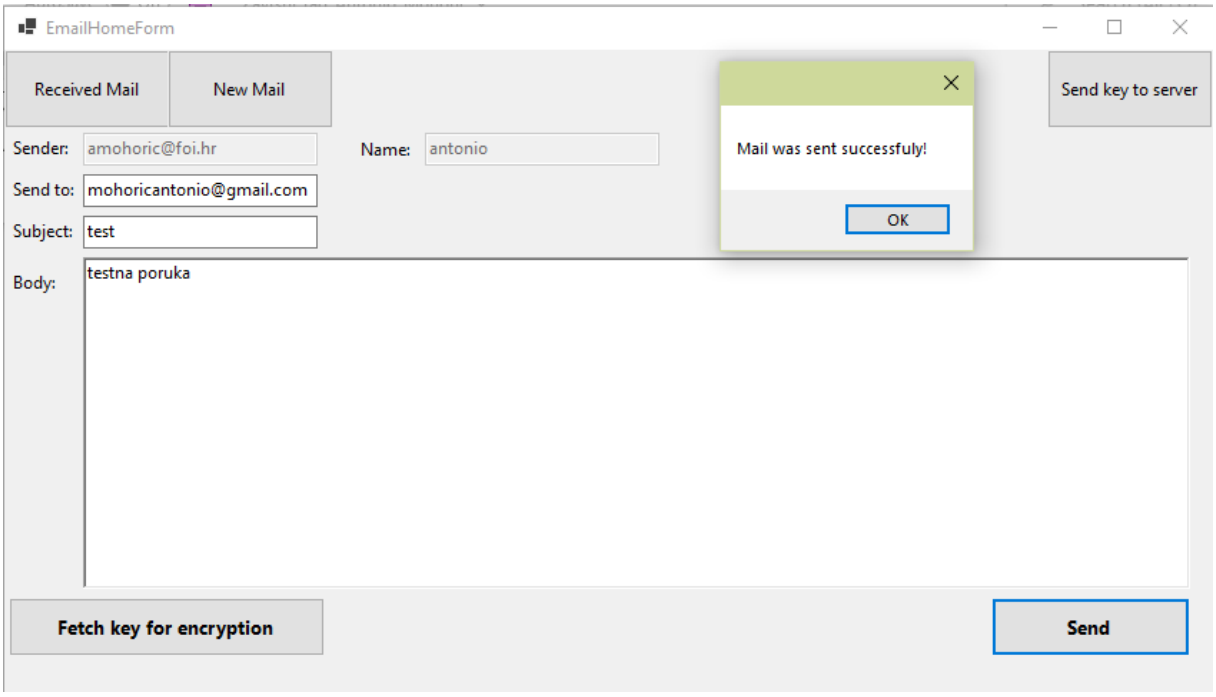


Slika 6: Obrazac pristigle e-pošte (autorski rad)



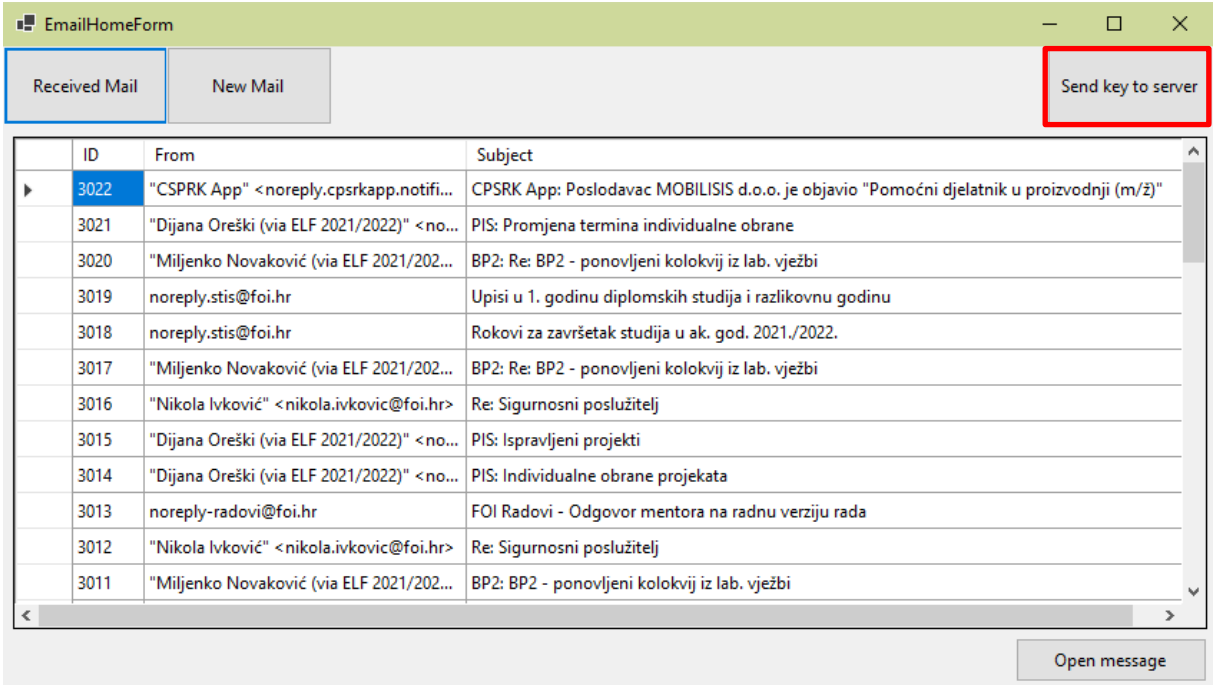
Slika 7: Obrazac prikaza teksta odabranog e-maila (autorski rad)

Drugi izbornik koji možemo odabrati je izbornik „New mail“. Klikom na taj gumb se otvara obrazac za kreiranje nove e-pošte. Klikom na gumb „Send“ poruka se šalje primatelju te aplikacija javlja korisniku uspješnost slanja te e-pošte:



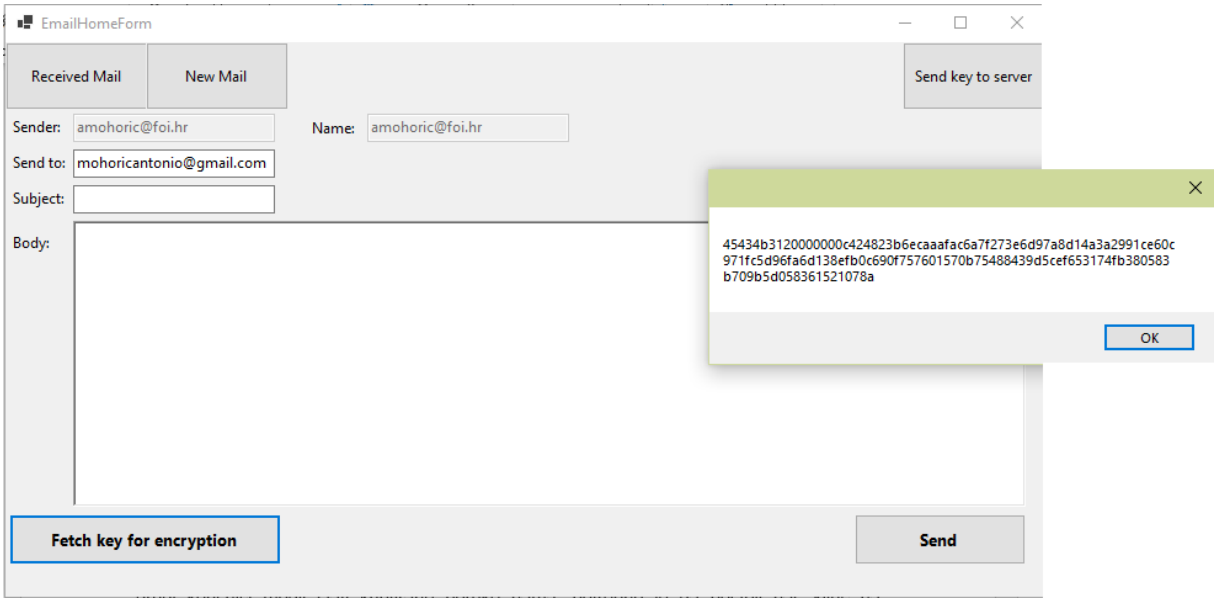
Slika 8: Obrazac za slanje e-pošte (autorski rad)

Desno od izbornika „New Mail“ možemo vidjeti gumb „Send key to server“. Kako bi drugi korisnici mogli slati kriptirane poruke nama, potrebno je da postoji naš ključ na poslužitelju kako bi ga drugi korisnici mogli dohvatiti. Pritiskom na taj gumb kreira se novi par ključeva za kriptiranje pomoću Diffie-Hellmanove metode, a zatim se određuje vremensko trajanje ključa od jedan dan. Nakon toga se šalje poruka na poslužitelj u kojoj je zapisano da se radi o zahtjevu za spremanjem ključa, javni ključ i vremensko trajanje ključa.



Slika 9: Gumb "Send key to server" (autorski rad)

Kod kreiranja nove e-pošte možemo primijetiti gumb „Fetch key for encryption“. Taj gumb nije moguće kliknuti dok god ne upišemo primatelja e-pošte. Nakon što smo upisali primatelja, možemo kliknuti na taj gumb, nakon čega se šalje zahtjev za dohvaćanjem ključa korisnika čiji smo e-mail napisali kao primatelja. Aplikacija čeka na odgovor od poslužitelja, ako ključ postoji poslužitelj ga vraća, ako ne postoji onda poslužitelj vraća poruku greške.



Slika 10: Gumb "Fetch key for encryption" (autorski rad)

3.2.1.1. Objašnjenje programskog koda

Aplikacija je napravljena u programskom jeziku C# koristeći Visual Studio 2022. Obrasci su dizajnirani u dizajn pogledu u VIsual Studiu pomoću kojega na jednostavan način možemo urediti elemente obrasca. Za olakšavanje rada s e-poštom, u programsko rješenje je dodan NuGet paket MailKit, a za korištenje korisničkih kontrola na formama je dodan paket Guna.UI2.WinForms.

Kod prijave korisnika u aplikaciju najprije se provjerava popunjenost svih potrebnih polja. Nakon što su sva polja popunjena, autentificira se korisnik pomoću vrlo jednostavne funkcije koja vraća *true* ili *false* ovisno o uspješnosti autentifikacije. Pomoću dodanog paketa MailKit uz funckije „Connect“ i „Authenticate“ jednostavno autentificiramo korisnika, a nakon toga zatvaramo vezu funkcijom „Disconnect“.:

```
private bool ValidateCredentials(string host, int port, string email, string password)
{
    using (var client = new MailKit.Net.Smtp.SmtpClient())
    {
        try
        {
            client.Connect(host, port, false);
            client.Authenticate(email, password);
            client.Disconnect(true);

            return true;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return false;
        }

    }
}
```

Slika 11: Autentifikacija korisničkog računa (autorski rad)

Ako prilikom pokušaja autentifikacije ona ne uspije, program hvata iznimku i korisniku se prikazuje poruka iznimke. S druge strane, ako je autentifikacija uspjela, program kreira novu instancu klase tipa „Account“ u koju sprema sve podatke o prijavljenom korisniku kako bi kasnije mogao čitati i slati e-poštu. Klasa se sastoji od 7 svojstava i konstruktora, a njen programski kod izgleda ovako:

```
public class Account
{
    public string Name { get; set; }
    public string Password { get; set; }
    public string Email { get; set; }
    public string Host { get; set; }
    public int Port { get; set; }
    public int POP3port { get; set; }
    public string POP3host { get; set; }
    public Account(string name, string password, string email, string host, int port, int pop3port, string pop3host)
    {
        Name = name;
        Password = password;
        Email = email;
        Host = host;
        Port = port;
        POP3port = pop3port;
        POP3host = pop3host;
    }
}
```

Slika 12: Klasa "Account" (autorski rad)

Nakon što se novi objekt klase „Account“ instancirao, otvara se nova forma „EmailHomeForm“ kojoj se prosljeđuje taj objekt. Na formi se automatski otvara novo korisničko sučelje „ReceivedMailUC“ kojemu se također prosljeđuje isti objekt klase „Account“.

Na formi „EmailHomeForm“ imamo dva gumba za odabir izbornika, prvi otvara korisničko sučelje „ReceivedMailUC“ koje se i automatski otvara po instanciranju forme, i drugi gumb koji otvara korisničku kontrolu „NewMailUC“. Važna je funkcija „AddUserControl“ koja prima korisničko sučelje i prikazuje ga u za to predviđeni panel:

```
public partial class EmailHomeForm : Form
{
    private Account LoggedUser;
    public EmailHomeForm(Account loggedUser)
    {
        InitializeComponent();
        LoggedUser = loggedUser;
    }

    private void btnMailRcv_Click(object sender, EventArgs e)
    {
        ReceivedMailUC uc = new ReceivedMailUC(LoggedUser);
        AddUserControl(uc);

    }
    private void AddUserControl(UserControl userControl)
    {
        userControl.Dock = DockStyle.Fill;
        panelContainer.Controls.Clear();
        panelContainer.Controls.Add(userControl);
        userControl.BringToFront();
    }

    private void btnNewMail_Click(object sender, EventArgs e)
    {
        NewMailUC uc = new NewMailUC(LoggedUser);
        AddUserControl(uc);
    }

    private void EmailHomeForm_Load(object sender, EventArgs e)
    {
        ReceivedMailUC uc = new ReceivedMailUC(LoggedUser);
        AddUserControl(uc);
    }
}
```

Slika 13: "EmailHome Form" (autorski rad)

Prilikom učitavanja korisničke kontrole „ReceivedMailUC“ učitava se primljena e-pošta prijavljenog korisnika i prikazuje na ekran. Prilikom instanciranja „ReceivedMailUC“ korisničke kontrole se stvara novi objekt tipa „ReadEmail“ koji također prima objekt tipa „Account“ kako bi znao podatke o prijavljenom korisniku. Klasa „ReadEmail“ ima 2 svojstva, konstruktor i 3 funkcije. Prvo je svojstvo tipa „Account“, a drugo je cijeli broj koji nam pokazuje koliko je e-mailova dohvaćeno. Zbog jednostavnosti i brzine izvođenja programa dohvaća se zadnjih 50 e-mailova ako ih toliko ima. Funkcija „FetchAllMessages“ se putem POP3 protokola spaja na korisnički e-mail račun te dodaje u listu zadnjih 50 poruka, nakon čega tu listu vraća.

Funkcija „NumberOfMessages“ se ponovno spaja putem POP3 protokola i jednostavnom funkcijom „GetMessageCount“ klase „MailKit“ dohvaća broj poruka. Ova funkcija je važna za kasnije dohvaćanje poruke prema indeksu.

Zadnja funkcija je ona koja nam omogućava čitanje poruke, pa se zato i zove „FetchMessage“. Ta funkcija prima argument tipa cijelog broja, spaja se na korisnički račun koristeći POP3 protokol, a zatim koristi još jednu funkciju iz klase „MailKit“, „GetMessage“ koja prima cijeli broj. Taj cijeli broj predstavlja indeks tražene poruke. Cijelu klasu vidimo na sljedeće dvije slike:

```
public class ReadEmail
{
    public Account LoggedUser { get; set; }
    public int Fetched { get; set; }
    public ReadEmail(Account loggedUser)
    {
        LoggedUser = loggedUser;
        Fetched = 0;
    }

    public List<MimeKit.MimeMessage> FetchAllMessages()
    {
        List<MimeKit.MimeMessage> returnMe = new List<MimeKit.MimeMessage>();

        using (Pop3Client client = new Pop3Client())
        {
            try
            {
                client.Connect(LoggedUser.POP3host, LoggedUser.POP3port, true);
                client.Authenticate(LoggedUser.Email, LoggedUser.Password);
                for(int i = client.GetMessageCount()-1; i>=0 && i>client.GetMessageCount() - 51; i--)
                {
                    returnMe.Add(client.GetMessage(i));
                    Fetched++;
                }
            }catch(Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

        }

            return returnMe;
    }
    public int NumberOfMessages()
    {
        int returnMe = 0;
        using (Pop3Client client = new Pop3Client())
        {
            try
            {
                client.Connect(LoggedUser.POP3host, LoggedUser.POP3port, true);
                client.Authenticate(LoggedUser.Email, LoggedUser.Password);
                returnMe = client.GetMessageCount();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
        return returnMe;
    }
```

Slika 14: Klasa "ReadEmail" 1/2 (autorski rad)

```
public MimeKit.MimeMessage FetchMessage(int index)
{
    MimeKit.MimeMessage returnMe = null;
    using (Pop3Client client = new Pop3Client())
    {
        try
        {
            client.Connect(LoggedUser.POP3host, LoggedUser.POP3port, true);
            client.Authenticate(LoggedUser.Email, LoggedUser.Password);
            returnMe = client.GetMessage(index);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    return returnMe;
}
```

Slika 15: Klasa "ReadEmail" 2/2 (autorski rad)

Nakon što korisnička kontrola „ReceivedMailUC“ kreira objekt tipa „ReadEmail“ dohvaćaju se indeksi, pošiljatelji i predmeti poruka u liste. Nakon što se liste popune, podatci iz lista se prikazuju u „datagridview“. Kako bi prikazali poruku određenog e-maila, klikom na gumb „Open message“ dohvaća se indeks odabranog retka i poziva spomenuta funkcija „FetchMessage“. Kada se poruka dohvati, otvara se nova forma i poruka se ispisuje:

```
public partial class ReceivedMailUC : UserControl
{
    private Account LoggedUser;
    private ReadEmail read;
    public ReceivedMailUC(Account loggedUser)
    {
        InitializeComponent();
        this.LoggedUser = loggedUser;
        read = new ReadEmail(LoggedUser);
    }

    private void ReceivedMailUC_Load(object sender, EventArgs e)
    {
        FillRcvMail(read);
    }
    private void FillRcvMail(ReadEmail read)
    {
        List<string> from = new List<string>();
        List<string> subject = new List<string>();
        List<string> id = new List<string>();
        int i = read.NumberOfMessages() - 1;
        foreach(MimeMessage m in read.FetchAllMessages())
        {
            id.Add(i.ToString());
            from.Add(m.From.ToString());
            subject.Add(m.Subject.ToString());
            i--;
        }
        for(int j = 0; j < read.Fetched; j++)
        {
            dgvRcvMail.Rows.Add(id[j], from[j], subject[j]);
        }
    }

    private void btnOpenMessage_Click(object sender, EventArgs e)
    {
        int index = Convert.ToInt32(dgvRcvMail.CurrentRow.Cells[0].Value);
        MessagePopUp form = new MessagePopUp(read.FetchMessage(index).TextBody.ToString());
        form.ShowDialog();
    }
}
```

Slika 16: "ReceivedMailUC" (autorski rad)

Klikom na gumb „New Mail“ se instancira korisnička kontrola „NewMailUC“ za kreiranje nove e-pošte. Objektu se ponovno prosljeđuje isti objekt klase „Account“. Kada se popune podatci o primatelju, predmetu i tekstu e-pošte, e-pošta se može poslati pritiskom na gumb „Send“. Klikom na taj gumb se poziva metoda „SendMail()“. U toj se metodi najprije kreira „MailMessage“ objekt kojem se prosljeđuju pošiljatelj, primatelj, predmet i tekst e-maila. Zatim se kreira objekt tipa „SmtpClient“ koji prima argumente o poslužitelju i broju porta, a nakon toga se prosljeđuju korisničko ime i lozinka korisnika koji šalje e-poštu. Navedeni atributi su zapisani u proslijeđenom objektu tipa „Account“. Nakon toga se poziva metoda „Send“ koja prima objekt tipa „MailMessage“ i poruka se šalje:

```
private void SendMail()
{

    MailMessage mail = new MailMessage(LoggedUser.Email , txtSendTo.Text, txtSubject.Text, richTxtBody.Text);
    SmtpClient client = new SmtpClient(LoggedUser.Host, LoggedUser.Port);
    client.UseDefaultCredentials = false;
    client.Credentials = new NetworkCredential(LoggedUser.Email, LoggedUser.Password);
    client.EnableSsl = true;
    try
    {
        client.Send(mail);
        MessageBox.Show("Mail was sent successfuly!");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

}
```

Slika 17: Metoda "SendMail" (autorski rad)

Dalje je bitno pojasniti kako se šalju i dohvaćaju ključevi za kriptiranje. Za to je napravljena posebna klasa pod nazivom „ServerConnection“. Ta se klasa sastoji od 3 svojstva i 4 metode i konstruktora. Prva metoda je „ConnectToServer()“. Ona služi za povezivanje na IP adresu servera na port 54000, jer je na serveru na tom portu namješten program koji čeka na zahtjev. Metoda najprije kreira novi objekt tipa „TcpClient“ kojemu se prosljeđuju IP adresa i broj porta. Zatim se u objekt tipa „NetworkStream“ sprema *stream* dohvaćen iz objekta „TcpClient“ metodom „GetStream()“. Za kraj se postavlja zastavica „Connected“ u istinu.

Druga metoda je „SendMessageToServer (string message)“ koja prima poruku koja se šalje na server. Najprije se u cjelobrojnu varijablu dohvaća broj bajtova primljene poruke. Zatim se poruka pretvara u niz bajtova, pa se taj niz šalje putem kreiranog „NetworkStream“ objekta metodom „Write“ koja prima niz bajtova, pomak i duljinu bajtova.

Sljedeća metoda služi za čitanje poruka koje stižu s poslužitelja pa se zato zove „ReadMessageFromServer()“. Kreira se objekt tipa „StreamReader“ koji prima naš postojeći „NetworkStream“. Nakon toga se poruka čita metodom „ReadLine()“ i vraća.

Zadnja metoda služi za zatvaranje veze s poslužiteljem, a zove se „CloseConnectionToServer()“. Poziva se metoda „Close()“ nad objektima „NetworkStream“ i „TcpClient“ kako bi se veza zatvorila, a zatim se postavlja zastavica „Connected“ u laž. Cijelu klasu možemo vidjeti na sljedeće dvije slike:

```
private TcpClient? Client { get; set; }
private NetworkStream? Stream;
bool Connected;
public ServerConnection()
{
    Client = null;
    Stream = null;
    Connected = false;
}
public int ConnectToServer()
{
    try
    {
        Client = new TcpClient("192.46.236.42", 54000);
        Stream = Client.GetStream();
        Connected = true;
        return 0;
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message);
        return -1;

    }
}
public void SendMessageToServer(string message)
{
    if (Connected)
    {
        int byteCount = Encoding.ASCII.GetByteCount(message + 1);
        byte[] sendData = Encoding.ASCII.GetBytes(message);
        Stream.Write(sendData, 0, sendData.Length);
        MessageBox.Show("Request sent!");
    }
```

Slika 18: klasa "ServerConnection" 1/2 (autorski rad)

```
}
public string ReadMessageFromServer()
{
    string returnMe = "";
    if (Connected)
    {
        StreamReader sr = new StreamReader(Stream);
        returnMe = sr.ReadLine();
    }


    return returnMe;
}
public void CloseConnectionToServer()
{
    if (Connected)
    {
        Stream.Close();
        Client.Close();
        Connected = false;
    }
}
```

Slika 19: klasa "ServerConnection" 2/2 (autorski rad)

Dakle, ovu klasu koristimo pri komunikaciji s našim sigurnosnim poslužiteljem, odnosno pri slanju svojeg ključa za kriptiranje na poslužitelj i pri dohvaćanju ključa primatelja s poslužitelja.

Klikom na gumb za slanje ključa na poslužitelj kreira se novi objekt tipa „ServerConnection“ nakon čega se poziva metoda „ConnectToServer“. Zatim se poziva metoda „SendMessageToServer“ kojoj se prosljeđuje znakovni niz koji sadrži tekst „save!/&“, e-mail adresu prijavljenog korisnika ponovno popraćeno znakovnim nizom „!/&“, novokreirani javni ključ za kriptiranje, pa „!/&“, i na kraju datum i vrijeme trajanja toga ključa, odnosno točno jedan dan od kreiranja ključa. Jedan primjer takvog znakovnog niza je ovo: save!/&amohoric@foi.hr!/&45434b3120000000236c110e4596903ee17cb07580be1c7be2680 20015363907cd842be75470dd7c8f4d10b816a4572b3f4c0ee2633585c4c0ba9083764c9391 93c0a4ba6866380a!/&03.09.2022. 19:51:31“. Ovakav format znakovnog niza je kreiran kako bi poslužitelj znao o kojem zahtjevu se radi. Nakon slanja poruke se prikazuje odgovor od poslužitelja nakon čega se zatvara veza prema poslužitelju.

Za kreiranje seta ključeva se koristi klasa „ECDiffieHellmanCng“ pomoću čijih metoda se kreiraju privatni i javni ključ. Programski kod za kreiranje i slanje ključeva vidimo na sljedećoj slici [7]:

```
private void btnSendKey_Click(object sender, EventArgs e)
{
    ServerConnection server = new ServerConnection();
    if (server.ConnectToServer() == 0)
    {
        server.SendMessageToServer(CreateKeyPair());
        MessageBox.Show(server.ReadMessageFromServer());
        server.CloseConnectionToServer();
    }
}
private string CreateKeyPair()
{
    string returnMe = "";

    byte[] publicKey;
    byte[] privateKey;

    using (ECDiffieHellmanCng edc = new ECDiffieHellmanCng(CngKey.Create(CngAlgorithm.ECDiffieHellmanP256, null,
        new CngKeyCreationParameters { ExportPolicy = CngExportPolicies.AllowPlaintextExport })))
    {
        edc.KeyDerivationFunction = ECDiffieHellmanKeyDerivationFunction.Hash;
        edc.HashAlgorithm = CngAlgorithm.Sha256;
        publicKey = edc.Key.Export(CngKeyBlobFormat.EccPublicBlob);
        privateKey = edc.Key.Export(CngKeyBlobFormat.EccPrivateBlob);
    }

    StringBuilder stringBuilder = new StringBuilder(publicKey.Length *2);
    foreach (byte b in publicKey)
    {
        stringBuilder.AppendFormat("{0:x2}", b);
    }

    DateTime now = DateTime.Now.AddDays(1);

    returnMe = "save!/&" + LoggedUser.Email + "!/&" + stringBuilder.ToString() + "!/&" + now.ToString("dd.MM.yyyy. HH:mm:ss");

    return returnMe;
}
```

Slika 20: Slanje ključa na poslužitelj (autorski rad)

S druge strane, klikom na gumb za dohvaćanje ključa primatelja prilikom kreiranje nove pošte, na poslužitelj se šalje drugačija poruka. Ta poruka je sljedećeg formata: „fetch!/&“ i tekst koji je zapisan pod kućicom primatelja. Primjer te poruke je ovo: „fetch!/&mohoricantonio@gmail.com“. Nakon toga se ponovno prikazuje odgovor od poslužitelja, ako je korisnik pronađen vraća se ključ, ako ga nema vraća se poruka greške, te se za kraj veza zatvara.

```
private void btnFetch_Click(object sender, EventArgs e)
{
    ServerConnection server = new ServerConnection();
    if (server.ConnectToServer() == 0)
    {
        server.SendMessageToServer("fetch!/&" + txtSendTo.Text);
        MessageBox.Show(server.ReadMessageFromServer());
        server.CloseConnectionToServer();
    }
}
```

Slika 21: Dohvaćanje ključa s poslužitelja (autorski rad)

3.2.2. Implementacija sigurnosnog poslužitelja

3.2.2.1. Objašnjenje konfiguracije poslužitelja

Kako bih imao pristup nekom udaljenom poslužitelju, preko online aplikacije *Linode* [8] sam zakupio poslužitelj s 1GB RAM memorije i 25GB memorije za pohranu. Smatram da je takav poslužitelj dovoljan za demonstraciju funkcioniranja sigurnosnog poslužitelja. Na taj sam poslužitelj instalirao Linux Ubuntu 20.04. IP adresa poslužitelja je 192.46.236.42, a port na kojem program osluškuje na zahtjev klijentske aplikacije je 54000. Na poslužitelj sam se spajao preko naredbenog retka naredbom „ssh“ kao „root“ korisnik.

Implementacija programa sigurnosnog poslužitelja jednako će raditi na bilo kojem drugom poslužitelju koji radi na operacijskom sustavu Linux, a ne samo na ovome.

Ovaj sigurnosni poslužitelj se sastoji od jednog programa koji radi u dvije dretve: prva dretva obrađuje pristigle zahtjeve od strane klijentske aplikacije, a druga dretva provjerava vremensko trajanje ključeva te ih po potrebi briše. Program radi u pozadini u beskonačnoj petlji kako bi sustav mogao funkcionirati konstantno. Program je napisan u Linuxovom uređivaču teksta nano u programskom jeziku C++. Zbog korištenja „thread“ biblioteke, prilikom kompiliranja je bilo potrebno dodati zastavicu „-lpthread“, tako da naredba za kompiliranje izgleda ovako: „g++ Server.cpp --lpthread“. Ovakvim kompiliranjem dobivamo izlaznu datoteku

„a.out“ koju možemo preimenovati. Kako bi program radio nakon prekida veze sa serverom, stavljen je da radi u pozadini naredbom „nohup“. Cijela naredbe za pokretanje programa izgleda ovako: „nohup ./Server &“.

3.2.2.2. Objašnjenje programskog koda – obrada pristiglih zahtjeva

Prije početka izvođenja programa, potrebno je globalno deklarirati određene varijable i funkcije kako bi im obje dretve mogle pristupati, a to su:

- Funkcija koja pomaže pri provjeri vremenskog trajanja ključa: „IsValid“
- Funkcija za brisanje linije teksta iz datoteke: „delete_line“
- Glavna funkcija za provjeru vremenskog trajanja ključa u koju ulazi novokreirana dretva: „CheckIfValid“
- Funkcija u koju se ulazi kada stigne „SIGINT“ signal: „end“
- Varijabla za kreiranje nove dretve: „Check_t“
- Varijabla za kreiranje socketa za slušanje: „listeningS“
- Varijabla za zaključavanje dijela programa kako ne bi obje dretve istovremeno čitale ili pisale u datoteke: „mutex“

```
bool IsValid(string file, string now);
void delete_line(const char *file_name, int n);
void *CheckIfValid(void *x);
void end (int sig);
pthread_t *Check_t = new pthread_t;
int listeningS;
pthread_mutex_t mutex;
```

Slika 22: Globalno deklarirane varijable i funkcije (autorski rad)

Program za obradu pristiglih zahtjeva se sastoji od 5 dijelova:

1. Kreiranje socketa za osluškivanje i dodjeljivanje porta
2. Čekanje na povezivanje s klijentom
3. Primanje poruke
4. Obrada zahtjeva
5. Slanje odgovora

Socket se kreira naredbom „socket“ koja prima argumente kojima specificiramo vrstu socketa i tip adrese. Nakon toga se kreira objekt „sockaddr_in“ koji služi za vezanje socketa na određeni port. Ponovno specificiramo vrstu socketa, definiramo broj porta i onda definiramo adresu pomoću funkcije „inet_pton“. Ta funkcija prima argument znakovnog niza koji

predstavlja IP adresu, i sprema ju u oblik internetske adrese. Kada je to uspješno odrađeno, mogu se povezati kreirani socket i pomoćni „sockaddr_in“ objekt metodom „bind“.

Ako su svi od prethodno navedenih koraka uspješno odrađeni, socket je spreman za osluškivanje na definiranom portu. Odabrao sam port 54000, ali mogao sam odabrati bilo koji slobodan port (osim prvih 1023 koji su rezervirani za operacijski sustav i ostalih zauzetih protova). Pozovemo funkciju „listen“ kojom označavamo da će se proslijeđeni socket koristiti za prihvat pristiglih zahtjeva za povezivanje. Programski kod navedenog dijela izgleda ovako [9]:

```
int listeningS = socket(AF_INET, SOCK_STREAM, 0);
if(listeningS == -1){
        cout<< "Can't create a socket!"<<endl;
        return -1;
}

sockaddr_in hint;
hint.sin_family = AF_INET;
hint.sin_port = htons(54000);
inet_pton(AF_INET, "0.0.0.0", &hint.sin_addr);

if(bind(listeningS, (sockaddr *)&hint, sizeof(hint)) == -1){
        cout<< "Can't bind socket!"<<endl;
        return -2;
}

if(listen(listeningS, SOMAXCONN) == -1){
        cout<< "Can't listen!"<<endl;
        return -3;
}
```

Slika 23: Kreiranje socketa za osluškivanje (autorski rad)

Nakon ovoga dijela program čeka na povezivanje s klijentom. Taj dio koda se izvodi u beskonačnoj petlji kako bi stalno bio na raspolaganju klijentima, ali prije ulaska u beskonačnu petlju se kreira nova dretva. Ta dretva ulazi u funkciju „CheckIfValid“. Također se kreira „mutex“ objekt koji služi za zaključavanje dijela koda kako obje dretve ne bi istovremeno čitale i pisale u datoteke. Zadnja stvar prije ulaska u beskonačnu petlju je postavljanje funkcije „sigset“ koja hvata signal „SIGINT“ i poziva funkciju „end“. Funkcija „end“ služi kako bi se ispravno dealocirala memorija koju je program koristio prilikom njegova gašenja. Dretva se gasi funkcijom „pthread_kill“, a „mutex“ objekt se briše funkcijom „pthread_mutex_destroy“. Funkcija „end“ još zatvara socket koji osluškuje na zahtjev klijenta za povezivanje.

```
void end(int sig){

    pthread_kill(*Check_t, SIGKILL);
    pthread_mutex_destroy(&mutex);

    close(listeningS);

    exit(0);
}
```

Slika 24: Funkcija "end" (autorski rad)

Pri povezivanju klijenta na server stvara se novi socket koji se koristi za razmjenu poruka. Funkcijom „accept“ se prihvaća zahtjev za vezu nad socketom koji osluškuje i kreira se novi socket preko kojega onda možemo čitati pristiglu poruku. Poruka se čita funkcijom „recv“ kojoj se prosljeđuje specifikator socketa, spremnik, veličina spremnika i zastavice, a poruka ostaje spremljena u spremniku [9]:

```
    sigset(SIGINT, end);
    pthread_mutex_init(&mutex, NULL);
    pthread_create (Check_t, NULL, CheckIfValid, NULL);


while(true){

    int clientSocket = accept(listeningS, (sockaddr *)&client, &clientSize);

    if(clientSocket == -1){
        cout<< "Problem with client connecting"<< endl;
        return -4;
    }

    char buf[4096];

    memset (buf, 0, 4096);
    int bytesRecv = recv(clientSocket, buf, 4096, 0);

    if(bytesRecv == -1){
        cout<< "There was a connection issue!" <<endl;
    }
```

Slika 25: Prihvaćanje veze (autorski rad)

Nakon što je poruka dohvaćena, program određuje o kojoj se vrsti zahtjeva radi. Da bi to napravio, mora razdijeliti primljenu poruku s delimiterom „!/&“. To radimo pomoću funkcije „strtok“. Kada je primljenu poruku razdijeljena, program može razlikovati radi li se o zahtjevu za spremanjem ili dohvaćanjem ključa. Ako se radi o zahtjevu za spremanjem, onda će na prvom mjestu nakon razdjeljivanja poruke pisati „save“, ako se radi o zahtjevu za dohvaćanjem ključa će pisati „fetch“. Vrsta zahtjeva se provjerava funkcijom „strcmp“ koja prima dva

znakovna niza te vraća 0 ako su oni jednaki. Kada je određena vrsta zahtjeva, pripadajuća zastavica se postavlja u istinu. Po razdijeljenoj poruci iteriramo kroz „while“ petlju dok god spremnik u koji se zapisuje poruka nakon razdjeljivanja nije „NULL“. Prije ulaska u tu „while“ petlju zaključava se „mutex“ kako druga dretva ne bi mijenjala datoteke. Za određivanje na kojem smo dijelu poruke koristimo brojač koji povećavamo svakim korakom petlje.

Kod spremanja ključeva postoji pomoćna datoteka koja služi za iteriranje po datotekama koja se zove „listofusers.txt“. Ta datoteka je bitna kako bi program za provjeru vremenskog trajanja ključa mogao posebno otvoriti svaku datoteku i provjeriti ključeve. Dakle, kada dođe zahtjev za spremanjem ključa, u tekstualnoj datoteci „listofusers.txt“ se provjerava je li korisnik već upisan, te ako nije se upisuje njegova e-mail adresa. Kroz ovu datoteku se ponovno iterira „while“ petljom dok god nismo prošli sve zapise u datoteci, te ako je pronađeno korisničko ime postavlja se zastavica „add“ u laž.

Kada je to odrađeno, na drugom mjestu razdijeljene poruke je zapisan ključ. Otvara se tekstualna datoteka pod nazivom „e-mail_adresa.txt“ i u tu datoteku se zapisuje ključ popraćen znakovnim nizom „!/&“. Na trećem mjestu se nalaze datum i vrijeme trajanja toga određenoga ključa te se to zapisuje u datoteku. Za rad s datotekama korišteni su „fstream“ objekti.

```
char *bufS = strtok(buf, "!/&");
bool save = false;
bool fetch = false;
int counter = 0;
fstream saveKey;
fstream addToList;
fstream fetchKey;
bool add = true;
char sendMessage[4096];
memset (sendMessage, 0, 4096);
bool found = true;

pthread_mutex_lock(&mutex);

while(bufS != NULL){
        if(strcmp(bufS, "save") == 0){
                save = true;
                addToList.open("listofusers.txt", ios::in);
        }
        if(save == true && counter ==1){
                saveKey.open( string(bufS) + ".txt", ios::out | ios::app);
                string line;
                while(getline(addToList, line))
                        if(line == string(bufS))add = false;


                if(add == true){
                        addToList.close();
                        addToList.open("listofusers.txt", ios::out | ios::app);
                        addToList<<string(bufS) + "\n";
                }
                addToList.close();

        }
        if(save == true && counter ==2){
                saveKey<<string(bufS) +"!/&";
        }
        if(save == true && counter == 3){
                saveKey<<string(bufS) + "\n";
                saveKey.close();
        }
```

Slika 26: Spremanje ključa u tekstualnu datoteku (autorski rad)

Ukoliko se radi o zahtjevu za dohvaćanjem, program pokušava otvoriti datoteku pod nazivom traženog korisničkog imena. Ako je datoteka uspješno otvorena, prebrojavaju se svi zapisi u njoj. Nakon toga se slučajnim odabirom izabire ključ koji će se vratiti korisniku. Taj ključ se dohvaća tako da se otvori datoteka i iterira kroz nju slučajni broj puta. Tim načinom smo dobili taj jedan cijeli zapis od kojega moramo odrezati vremensko trajanje kako bismo dobili samo ključ, što ponovno radimo „strtok“ funkcijom. Za kraj sam ključ stavljamo u spremnik koji šaljemo natrag korisniku funkcijom „strcpy“.

```
        if(strcmp(bufS, "fetch")==0){
                fetch = true;
        }
        if(fetch == true && counter ==1){
                fetchKey.open("Keys/"+ string(bufS) + ".txt", ios::in);
                if(fetchKey){

                        string line;
                        int count = 0;;
                        while(getline(fetchKey, line))count++;
                        srand(time(NULL));
                        int random = rand() % count +1;
                        line = "";

                        fetchKey.close();

                        fetchKey.open("Keys/"+ string(bufS) + ".txt", ios::in);
                        for(int i = 0; i< random; i++) getline(fetchKey,line);
                        fetchKey.close();

                        int l = line.length();
                        char fetchedKey[l+1];
                        memset(fetchedKey, 0, l+1);
                        strcpy(fetchedKey, line.c_str());

                        char *f = strtok(fetchedKey, "!/&");
                        strcpy(sendMessage, f);
                }
        }
        counter++;
        bufS= strtok(NULL, "!/&");
}
```

Slika 27: Dohvaćanje ključa (autorski rad)

Na pokazani način program sigurnosnog poslužitelja obrađuje pristigle zahtjeve. Nakon što je zahtjev obrađen klijentu se vraća povratni odgovor. Ako je ključ spremljen šalje se poruka „Key saved“, ako je trebalo dohvatiti ključ i on je uspješno dohvaćen onda se on i šalje, ako nije dohvaćen se šalje poruka „Requested user was not found“. Ako nekim slučajem na program dođe zahtjev nekog trećeg formata, vraća se poruka „Bad request“. Poruka se šalje funkcijom „send“ kojoj se prosljeđuje socket, poruka, veličina poruke i zastavice.

Dok smo još unutar beskonačne petlje u kojoj program radi bitno je zatvoriti socket za komunikaciju naredbom „close“ kako bi se mogao kreirati novi. Nakon toga otključavamo „mutex“ kako bi druga dretva mogla nastaviti s radom s datotekama. Kada program želimo ugasiti, izvan beskonačne petlje zatvaramo socket koji osluškuje.

```
        if(save == true) strcpy (sendMessage, "Key saved!");
        else if(fetch == true && strcmp (sendMessage, "") == 0) strcpy (sendMessage, "Requested user was not found");
        else if(fetch != true) strcpy (sendMessage, "Bad request");

        send(clientSocket, sendMessage, sizeof(sendMessage) + 1, 0);
        close(clientSocket);
        pthread_mutex_unlock(&mutex);

}
        close(listeningS);
        return 0;
}
```

Slika 28: Slanje povratne poruke i kraj programa (autorski rad)

3.2.2.3. Objašnjenje programskog koda – provjera vremenskog trajanja ključeva

Program za provjeru vremenskog trajanja ključeva sastoji se od sljedećih dijelova:

1. Otvaranje tekstualne datoteke „listofusers.txt“
2. Iteriranje po otvorenoj datoteci i iteriranje po datotekama čiji su nazivi zapisani u datoteci „listofusers.txt“
3. Provjera je li isteklo vremensko trajanje ključa za svaki zapis u otvorenoj datoteci
4. Brisanje zapisa ako mu je rok istekao
5. Brisanje zapisa iz datoteke „listofusers.txt“ ako je datoteka prazna

Za lakši rad sam kreirao 3 funkcije koje se koriste u programskom kodu:

1. Funkcija „IsValid“ koja prima dva znakovna niza od kojih prvi predstavlja vremensko trajanje određenog ključa, a drugi trenutno vrijeme. Funkcija vraća „true“ ili „false“ ovisno o tome je li isteklo vremensko trajanje ključa
2. Funkcija „IsPastTime“ koja prima 6 cjelobrojnih argumenta koji predstavljaju sate, minute i sekunde u određenom zapisu i trenutno. Ako je vrijeme prošlo funkcija vraća „true“, ako nije onda „false“. Ova funkcija se poziva kod funkcije „IsValid“ kako bi se jednostavnije usporedilo vrijeme.
3. Funkcija „delete_line“ koja prima naziv datoteke i broj linije koju je potrebno izbrisati iz nje. Ta nam je funkcija važna i za brisanje zapisa u pojedinim datotekama i za brisanje zapisa u datoteci „listofusers.txt“

Funkcija „IsValid“ iz primljenih znakovnih nizova uzima znakove na određenim pozicijama i pretvara ih u cjelobrojne vrijednosti. Za uzimanje točno određenih dijelova znakovnog niza se koristi funkcija „substr“, a za pretvorbu u cjelobrojnu vrijednost funkcija

„stoi“. Nakon što je program u posebne varijable zapisao godinu, mjesec, dane, sate, minute i sekunde, može ih početi uspoređivati. Godine, mjeseci i dani se uspoređuju unutar ove funkcije, dok se sati, minute i sekunde uspoređuju unutar funkcije „IsPastTime“. Programski kod ove dvije funkcije možemo vidjeti na sljedećoj slici:

```
bool IsPastTime(int hourF, int minF, int secF, int hourN, int minN, int secN){
        if(hourF < hourN) return true;
        else if(hourF == hourN){
                if(minF < minN) return true;
                else if(minF == minN){
                        if(secF < secN) return true;
                        }
        }
        return false;
}

bool IsValid(string file, string now){

        int dayF = stoi(file.substr(0,2));
        int monthF = stoi(file.substr(3,2));
        int yearF = stoi(file.substr(6,4));
        int hourF = stoi(file.substr(12,2));
        int minF = stoi(file.substr(15,2));
        int secF = stoi(file.substr(18,2));

        int dayN = stoi(now.substr(0,2));
        int monthN = stoi(now.substr(3,2));
        int yearN = stoi(now.substr(6,4));
        int hourN = stoi(now.substr(12,2));
        if(hourN == 22) hourN = 0;
        else if(hourN == 23) hourN = 1;
        else hourN = hourN +2;

        int minN = stoi(now.substr(15,2));
        int secN = stoi(now.substr(18,2));

        if(yearF < yearN) return false;
        else if(yearF == yearN){
                if(monthF < monthN) return false;
                else if(monthF == monthN){
                        if(dayF < dayN) return false;
                        else if(dayF == dayN){
                                if(IsPastTime(hourF, minF, secF, hourN, minN, secN)) return false;
                        }
                }
        }

        return true;
}
```

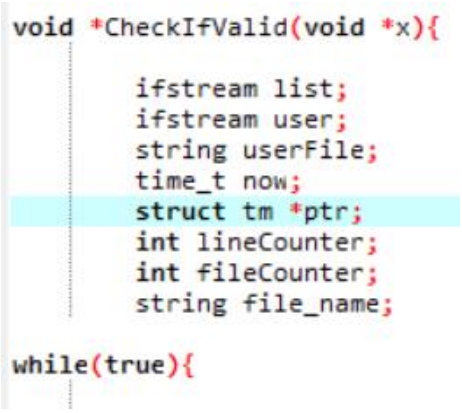
Slika 29: Funkcije "IsValid" i "IsPastTime" (autorski rad)

Funkcija „delete_line“ prima naziv datoteke koju otvara u objektu tipa „ifstream“ iz koje se samo čita. Kreira se i „ofstream“ objekt u koji se piše. To je pomoćni objekt u koji se zapisuje sve iz otvorene datoteke koju je funkcija primila osim određene linije koja se briše. Kroz otvorenu datoteku se iterira pomoću „while“ funkcije i provjerava se je li vrijednost brojača jednaka broju linije koju treba izbrisati. Ako jest, brojač se povećava i preskače se korak petlje naredbom „continue“, ako nije onda se pročitana linija zapisuje u pomoćnu datoteku i brojač se povećava. Nakon što završi „while“ petlja, zatvaraju se obje datoteke, originalna datoteka se briše, a pomoćna se preimenuje u naziv originalne datoteke. Na taj način smo izbrisali točno jednu liniju iz datoteke.

```
void delete_line(const char* file_name, int n){
        ifstream is(file_name);

        ofstream ofs;
        ofs.open("temp.txt");

        string line;
        int line_no = 1;


        while (getline(is, line))
        {
                if(n == line_no){
                        line_no++;
                        continue;
                }
                else{
                        ofs << line <<endl;
                        line_no++;
                }
        }

        ofs.close();
        is.close();

        remove(file_name);

        rename("temp.txt", file_name);
}
```

Slika 30: Funckija "delete_line" (autorski rad)

Glavni dio programa radi u beskonačnoj petlji, kako bi konstantno provjeravao je li isteklo vremensko trajanje određenog ključa. Prije ulaska u petlju, instanciraju se objekti s kojima program radi unutar petlje, a to su:



Slika 31: Objekti instancirani prije beskonačne petlje (autorski rad)

Datoteka „list“ je datoteka „listofusers.txt“, a u objekt „user“ se otvaraju sve datoteke redom. U znakovni niz „userFile“ se sprema korisničko ime koje se pročita iz datoteke „listofusers.txt“, a u „file_name“ se sprema „userFile“ + „.txt“ kako bi se datoteka mogla otvoriti. Brojač „lineCounter“ nam označava broj linije koju promatramo unutar svake pojedine datoteke, a brojač „fileCounter“ predstavlja broj linije koju promatramo unutar datoteke „listofusers.txt“. Varijable „now“ i „ptr“ nam služe za dobivanje trenutnog datuma i vremena.

Pri ulasku u petlju, dretva „spava“ jednu sekundu kako bi glavna dretva mogla obraditi pristigli zahtjev (ukoliko zahtjev postoji), a zatim se zaključava „mutex“ objekt. Potrebno je otvoriti datoteku „listofusers.txt“ i postaviti brojač na 0. Nakon toga se unutar nove „while“ petlje, koja traje dok nisu pročitani svi zapisi iz datoteke „listofusers.txt“, otvaraju datoteke redom. Brojač „lineCounter“ se postavlja na nulu te se ulazi u novu „while“ petlju koja ponovno traje dok nisu pročitani svi zapisi iz te datoteke.

U toj petlji se za svaki zapis provjerava ispravnost vremenskog trajanja, te se taj zapis briše ukoliko je potrebno. Da bi se to moglo provjeriti, najprije se iz zapisa dohvaća samo datum i vrijeme funkcijom „strtok“. Nakon toga se kreira znakovni niz trenutnog datuma i vremena u istom formatu kao što je zapisano u datoteci. Za dohvaćanje trenutnog datuma se koristi funkcija „localtime“, a za formatiranje datumskog zapisa funkcija „strftime“ kojoj se prosljeđuje spremnik, veličina spremnika, format i pokazivač na trenutni datum i vrijeme.

Nakon kreiranja znakovnog niza i dohvaćanja datuma iz zapisa, pomoću funkcije „IsValid“ se provjerava ispravnost vremenskog trajanja. Ako je ključ istekao, poziva se funkcija „delete_line“ kojoj se prosljeđuje naziv datoteke i broj linije koja se trenutno promatra, ondosno „lineCounter“. Nakon brisanja je potrebno „lineCounter“ smanjiti za 1, jer je u datoteci sada jedna linija manje.

Nakon što se iteriralo kroz cijelu datoteku, ona se zatvara. Program dalje provjerava je li datoteka prazna; ako je broj linija jednak nuli, datoteka je prazna. Ako je ona prazna, njen naziv se mora izbrisati iz datoteke „listofusers.txt“, za što ponovno koristimo funkciju „delete_line“. Kao i kod prethodnog brisanja, potrebno je smanjiti brojač za 1. Nakon što smo izbrisali zapis iz datoteke, potrebno je tu praznu datoteku izbrisati. Nakon što je završena provjera svih ključeva, „mutex“ objekt se otključava. Cijeli programski kod glavnog djela programa možemo vidjeti na sljedećoj slici:

```
while(true){

        sleep(1);
        pthread_mutex_lock(&mutex);
        list.open("listofusers.txt");

        fileCounter = 0;

        while(getline(list, userFile)){
                fileCounter++;
                file_name = "";
                user.open(userFile + ".txt");
                string line;

                lineCounter = 0;

                while(getline(user, line)){

                        lineCounter++;
                        int l = line.length();
                        char date[l+1];
                        memset(date, 0, l+1);
                        strcpy(date, line.c_str());

                        char *d = strtok(date, "!/&");
                        d = strtok(NULL, "!/&");

                        now = time(0);
                        char now_str[256];
                        ptr = localtime(&now);

                        strftime(now_str, sizeof(now_str), "%d.%m.%Y. %H:%M:%S", ptr);

                        if(!IsValid(d, now_str)){

                                string file_name = userFile + ".txt";
                                delete_line(file_name.c_str(), lineCounter);
                                lineCounter--;
                        }

                }

                user.close();
                string file_name = userFile + ".txt";

                if(lineCounter == 0){

                        delete_line("listofusers.txt", fileCounter);
                        fileCounter--;
                        remove(file_name.c_str());
                }

        }
        list.close();

        pthread_mutex_unlock(&mutex);
        }
}
```

Slika 32: Glavni dio programa "CheckIfValid.cpp" (autorski rad)

4. Mogućnosti nadogradnje sustava

Nakon implementacije ovakvog sigurnosnog poslužitelja, kako bi cijeli sustav za kriptiranje i slanje poruka mogao ispravno funkcionirati, postoje neke nadogradnje koje bi se mogle provesti nad klijentskom aplikacijom.

Kod kreiranja para ključeva, javni ključ se šalje na poslužitelj, dok se s privatnim ništa ne radi. Kako bi se poruka mogla kriptirati i dekriptirati, potrebno je spremiti oba ključa na klijentsko računalo. Dakako, još jedna nadogradnja bi onda bila i samo kriptiranje poruke i mogućnost čitanja tako kriptiranih poruka, odnosno dekriptiranje kriptirane poruke.

Kod slanja ključa na poslužitelj, unaprijed je definirano njegovo vremensko trajanje od točno jednog dana od trenutka kreiranja ključa. Moguća nadogradnja je da korisnik sam određuje njegovo vremensko trajanje.

Sve navedene nadogradnje su moguće bez ikakvog mijenjanja ovakve implementacije sigurnosnog poslužitelja. Kada bi se navedene nadogradnje uvele, cijeli sustav bi osigurao potpunu buduću sigurnost prilikom slanja e-pošte.

5. Zaključak

Buduća sigurnost je pojam koji označava sigurnost u budućnosti u slučaju probijanja ključeva za kriptiranje. Ako se razmjena poruka snima duži period i probije se ključ za kriptiranje, napadač će moći pročitati samo one poruke kriptirane tim ključem, a ne sve poruke. Koristeći sigurnosni poslužitelj, probijanje ključeva se dovodi na minimalnu razinu s obzirom da se javni dio ključa briše s poslužitelja nakon isteka njegova roka trajanja, te se taj ključ onda više ne može koristiti za kriptiranje. To znači da će se moći koristiti samo za manji broj poruka, što omogućuje buduću sigurnost.

Program sigurnosnog poslužitelja se sastoji od dvije dretve, prva obrađuje pristigle zahtjeve, dok druga provjerava je li isteklo vremensko trajanje ključa te briše zapise iz datoteka i same datoteke. Pristigli zahtjevi mogu biti dva tipa: zahtjev za spremanjem ključa i zahtjev za dohvaćanjem ključa. Program ih razlikuje te poduzima odgovarajuće akcije ovisno o zahtjevu. Komunikacija klijentske aplikacije sa sigurnosnim poslužiteljem je uspješno implementirana, sigurnosni poslužitelj poduzima ispravne akcije ovisno o zahtjevu. Provjera ispravnosti trajanja ključeva je također uspješno implementirana, zapisi o ključevima u datotekama se pravovremeno brišu.

Ovaj sigurnosni poslužitelj za buduću sigurnost e-pošte radi paralelno sa svim e-mail poslužiteljima. Ovako implementirani sigurnosni poslužitelj, klijentska aplikacija sa spomenutim nadogradnjama i e-mail poslužitelji zajedno mogu omogućiti potpunu buduću sigurnost e-pošte.

Popis literature

[1] N. Ivković, „Sigurnost e-pošte“ [Video datoteka], *Sveučilište u Zagrebu, Fakultet organizacije i informatike,* n.d., Dostupno: https://u.pcloud.link/publink/show?code=XZHv83XZj8eYTiesifFTa8nr2M2SbmugO08y, Pristupano: 11.1.2022.

[2] John Wilson, *SMTPS: Securing SMTP and the Differences Between SSL, TLS, and the Ports They Use,* 27.5.2022., Dostupno: https://www.agari.com/email-security-blog/smtps-how-to-secure-smtp-with-ssl-tls-which-port-to-use/#:~:text=SMTP%20can%20be%20secured%20through,frequently%20used%20for%20SMTPS%20traffic., Prisupano: 5.8.2022.

[3] Dark Web Academy, *How PGP Works*,[Video datoteka], 27.3.2016., Dostupno: https://www.youtube.com/watch?v=CHi2RclGvIM, Pristupano: 5.8.2022.

[4] F5 DevCentral, *Perfect Forward Secrecy,* [Video datoteka], 25.4.2017., Dostupno: https://www.youtube.com/watch?v=IkM3R-KDu44, Pristupano: 5.8.2022.

[5] D. A. Carts , „A Review of the Diffie-Hellman Algorithm and its Use in Secure Internet Protocols“, *SANS Institute 2001*., 5.11.2001.

[6] CBTVid, „*7 - Cryptography Basics - Diffie-Hellman Key Exchange“,* [Video datoteka], 19.1.2021., Dostupno: https://www.youtube.com/watch?v=d1KXDGgwIpA, Pristupano: 4.8.2022.

[7] D. Tomšić, „Sigurnosna nadogradnja sustava elektroničke pošte“, *Sveučilište u Zagrebu, Fakultet organizacije i informatike*, rujan 2020.

[8] Linode LLC, *linode,* verzija v1.73.0 [Na internetu], Dostupno: https://cloud.linode.com/linodes, Pristupano 4.8.2022.

[9] S. Kelly, *„Creating a TCP Server in C++ [Linux / Code Blocks]“*, [video datoteka], 4.10.2018, Dostupno: https://www.youtube.com/watch?v=cNdlrbZSkyQ&t=1237s, Pristupano: 22.8.2022.

[10] Hung-Min Sun, Bin-Tsan Hsieh, Hsin-Jia Hwang, „Secure E-mail Protocols Providing Perfect Forward Secrecy*“, IEEE COMMUNICATIONS LETTERS, VOL. 9, NO. 1*, siječanj 2005

Popis slika

Prilozi

- GitHub repozitorij programskoga koda: https://github.com/mohoricantonio/Forward_Secrecy_Server/tree/main/ForwardSecrecy