**GIMNAZIJA VIČ**

SAFECHAT

**PROJEKTNO DELO PRI INFORMATIKI**

**SAMO BORZATTA**

**Mentor: KLEMEN BAJEC**

**LJUBLJANA, APRIL 2025**

## POVZETEK

Z mojim projektnim delom sem želel izvedeti več o delovanju interneta in aplikacij za pošiljanje sporočil. Prav tako me je zanimalo delovanje simetriče in asimetrične enkripcije, ki je uporabljeni v nalogi. Zato sem si zastavil cilj, da sam napišem program, ki bo omogočal enkriptirano pošiljanje sporočil med uporabniki. Program je napisan v programskem jeziku Python. Uporablja knjižnici *socket* in *cryptography*, ki prinašata funkcije potrebne za povezovanje preko omrežja in enkriptiranje podatkov. V prihodnosti bi programu lahko dodal še uporabniški vmesnik, ki pa ga trenutno ni, zato se ga v celoti izvaja s pomočjo terminala.

**Ključne besede:** *Python, omrežje, socket, ip naslov, povezovalni ključ*

*podatkovna baza*

**Key-words:** *Python, network, socket, ip address, connection key, database*

## KAZALO

[1 UVOD 3](#_Toc195444869)

[2 TEORIJA 4](#_Toc195444870)

[2.1 Omrežja 4](#_Toc195444871)

[2.2 Kriptografija 4](#_Toc195444872)

[2.2.1 Simetrična enkripcija: 5](#_Toc195444873)

[2.2.2 Asimetrična enkripcija: 5](#_Toc195444874)

[2.3 Python 6](#_Toc195444875)

[3 OPIS DELOVANJA PROGRAMA 7](#_Toc195444876)

[3.1 Potrebni dokumenti za delovanje 7](#_Toc195444877)

[3.1.1 Server 7](#_Toc195444878)

[3.1.2 Client 8](#_Toc195444879)

[3.2 Splošno delovanje 9](#_Toc195444880)

[3.2.1 Povezovanje 9](#_Toc195444881)

[3.2.2 Prijava in registracija 11](#_Toc195444882)

[3.2.3 Odziv strežnika po prijavi 12](#_Toc195444883)

[3.2.4 Pošiljanje sporočil 13](#_Toc195444884)

[3.2.5 Prejemanje sporočila 16](#_Toc195444885)

[3.2.6 Simetrična enkripcija 17](#_Toc195444886)

[3.2.7 Asimetrična enkripcija 18](#_Toc195444887)

[3.2.8 Delo z bazo 18](#_Toc195444888)

[4 ZAKLJUČEK 20](#_Toc195444889)

[5 VIRI 21](#_Toc195444890)

## UVOD

Moje projektno delo pri informatiki je program v programskem jeziku Python, ki omogoča pošiljanje enkriptiranih sporočil preko lokalnega omrežja. Program je preprosta verzija današnjih aplikacij za pošiljanje sporočil. Deluje s pomočjo Pythonovih knjižnic socket in cryptography, ki prinašata s seboj funkcije, ki omogočajo delovanje v omrežju in povezovanje naprav ter simetrično in asimetrično enkripcijo. Program potrebuje več datotek za delovanje, ki so smiselno razdeljene v svoje mape.

## TEORIJA

### Omrežja

Omrežja so danes prisotna povsod. Vendar pa je ta sistem za deljenje podatkov dokaj zapleten. Za sestavo programa je bil potrebno imeti znanje o pošiljanju sporočil preko omrežja.

Program omogoča pošiljanje sporočil v **LAN (Local Area Network)**. Ta postopek poteka preko vmesnega posrednika (strežnika), ki zagotavlja znan izvor sporočil. Da si lahko dva uporabnika izmenjata sporočila, morata biti oba povezana s strežnikom, ki edini hrani potrebne podatke, kot so javni ključi uporabnikov, tokeni in njihove povezave s strežnikom.

Povezava s strežnikom poteka s pomočjo prijave oziroma registracije uporabnika. Če se je ta v preteklosti že povezal, potem strežnik po vsej verjetnosti hrani povezovalni token, ki omogoča direktno povezavo in preskoči proces prijave.

Pošiljanje sporočil po omrežju poteka s pomočjo paketkov, ki hranijo podatke o naslovniku ter pošiljatelju, o vrsti paketka in ostale podatke. Knjižnica socket omogoča dostop do **BSD** **(Berkeley Software Distribution)**, ki omogoča **FTP (File Transfer Protocol)**, s pomočjo katerega poteka pošiljanje sporočil **[12]**. Ko pošljemo sporočilo, najprej navedemo velikost sporočila, nato pa še sporočilo **[2]**. Sporočilo najprej potuje na strežnik, ki ga pošlje naprej naslovljenemu uporabniku. Danes strežniki pogosto hranijo sporočila pogovorov, zato ni potrebno, da sta oba uporabnika povezana z internetom. Tako se morata le na neki točki povezati z njim in bosta prejela sporočila.Vendar zaradi enostavnosti programa nisem sprogramiral shranjevanja sporočil na srežniku. Tako je pošiljanje sporočil mogoče samo, če sta oba uporabnika povezana.

### Kriptografija

Prenos informacij po internetu je prvotno bil mišljen samo za nekaj posameznikov, zato se na začetku niso preveč ukvarjali z varnostjo. Z večjo uporabo interneta, pa je varnost postala ključnega pomena. Zato se danes pogosto uporablja enkripcija. Poznamo več vrst enkripcij, sam pa sem uporabil *asimetrično* in *simetrično*. Zanimanje za enkripcijo me je privedlo, da sem tudi sam ustvaril program za enkriptiranje podatkov.

#### Simetrična enkripcija:

Delovanje simetrične enkripcije je bolje opisano v delu, kjer opišem delovanje datoteke ***Encryption.py*** in se uporablja predvsem za shranjevanje podatkov na računalniku. Pri mojem programu igra vlogo enkripcije zasebnih ključev uporabnikov in strežnika.

#### Asimetrična enkripcija:

Asimetrična enkripcija je kompleksen matematičen proces, ki ustvari javni in zasebni ključ. Javni ključ je uporabljen za enkripcijo podatkov. Ko so le ti enkriptirani, jih je mogoče dekriptirati samo z uporabo zasebnega ključa. Zasebni ključ je shranjen pri prejemniku. Če mu kdo hoče poslati podatke, mu prejemnik najprej deli svoj javni ključ, s pomočjo katerega pošiljatelj enkriptira sporočilo. Nato pošiljatelj sporočilo pošlje prejemniku, ki ga dekriptira s svojim zasebnim ključem.

Ta proces se danes pojavlja zelo pogosto in se imenuje *enkripcija od konca do konca* (**End-To-End encryption**) **[5]**.

Poznamo več načinov za ustvarjanje javnega in zasebnega ključa. V tem programu je bil uporabljen [**RSA (Rivest-Shamir-Adleman)**](https://www.cs.sjsu.edu/~stamp/CS265/SecurityEngineering/chapter5_SE/RSAmath.html), ki je eden izmed najstarejših algoritmov za asimetrično enkripcijo **[6]**. Kako ta algoritem ustvari ključa, je opisano na zgornji povezavi.

S tem algoritmom lahko enkriptiramo dolžino sporočil, ki so dolga toliko ali manj kot je dolg javni ključ **[6]**. V mojem programu je dolžina javnega ključa 256 znakov, kar pomeni da lahko enkriptira sporočila dolga do vključno 256 znakov.

Ključa sta shranjena v datotekah *.pem* (**Privacy Enhanced Mail**), ki sta namenjeni shranjevanju internetnih certifikatov, kot sta javni in zasebni ključ **[6]**.

### Python

Celoten program je napisan v programskem jeziku Python, ki je danes eden izmed najbolj pogosto uporabljenih programskih jezikov **[1]**. Za izdelavo programa sta bila potrebna dobro znanje enkodiranja datotek in uporaba pravih formatov za shranjevanje podatkov, pošiljanje, itd.

Pri izdelavi enkripcije, ki je zelo preprosta in večinoma temelji na seštevanju binarnih vrednosti, sem moral uporabiti različne vrste spremenljivk, ki so med drugim vsebovale tudi binarne zapise. Ta program ima strukturo razreda, za lažjo implementacijo v drugih programih.

## OPIS DELOVANJA PROGRAMA

### Potrebni dokumenti za delovanje

Projekt SafeChat je na Githubu na voljo za testiranje. Program je narejen iz dveh delov. Prvi del predstavlja strežnik (mapa *server*), ki skrbi za prijavo uporabnikov in pošiljanje sporočil med njimi. Drugi del pa predstavlja uporabniški del (mapa *client*), v katerem so datoteke potrebne za povezovanje na strežnik.

Slika 1: Pregled datotek potrebnih za delovanje

A screen shot of a computer program

AI-generated content may be incorrect.

Vir: lasten 2025

#### Server

Strežnik je sestavljen iz treh podmap in glavnega programa **server.py**, ki vsebuje vse glavne funkcije za povezovanje ter prepošiljanje sporočil. Skrbi tudi za prijavo in registracijo uporabnikov.

Podmapa *Database* (podatkovna baza) ima naslednje datoteke:

* **DatabaseRequests.py**: Vsebuje funkciji za branje in spreminjanje podatkovne baze.
* **baza.py**: Vsebuje funkcije za ustvarjanje podatkovne baze. Ker je pri projektu bil uporabljen sqlite, ima ta mapa še bazno datoteko.
* **accounts.db**: Je datoteka tipa baze podatkov, ki hrani tabelo za shranjevanje uporabnikov.

Podmapa *Encryption*, ki je popolnoma enaka tudi za uporabnika, vsebuje dve datoteki:

* **AsymmetricEncryption.py**: Vsebuje funkcije za asimetrično enkripcijo in dekripcijo.
* **Encryption.py**: Vsebuje funkcije za preprosto simetrično enkripcijo, ki sem je naredil sam kot del drugega projekta.

Nazadnje imamo še podmapo *keys*, ki je na strežniku sestavljena iz treh, pri uporabniku pa iz štirih datotek. Pri strežniku je sestavljena iz naslednjih datotek:

* **generate\_keys.py**: Ta datoteka je enaka kot pri uporabniku in ima funkcije za ustvarjanje strežnikovega javnega in zasebnega ključa, ki sta potrebna za varno komunikacijo.
* **privat\_key.pem** in **public\_key.pem** datoteki sta tako kot pri uporabniku namenjeni hranjenju ključev.

#### Client

Del projekta, ki ga mora imeti uporabnik se nahaja v mapi client, ki je prav tako sestavljena iz treh podmap in glavnega programa **client.py**, ki vsebuje vse glavne funkcije za povezovanje in komunikacijo s strežnikom ter funkcije za pošiljanje sporočil.

Podmapa *Encryption* je popolnoma enaka tisti na strežniku.

Podmapa *keys* vsebuje naslednje datoteke:

* **generate\_keys.py**, **privat\_key.pem** in **public\_key.pem** vsebujejo enake funkcije in shranjujejo ključe enako kot na strežniku.
* **server\_public\_key.pem** je prisotna samo pri uporabniku in vanjo se zapiše javni ključ strežnika.

Podmapa *token* vsebuje eno datoteko:

1. **token.txt**: v to tekstovno datoteko se zapiše enkriptirani ključ, ki omogoča hitrejšo prijavo na strežniku oziroma preskok koraka prijave.

### Splošno delovanje

#### Povezovanje

Ko zaženemo program server.py, prične program na izbranih vratih poslušati za morebitnimi povezavami (*funkcija start()*).

Slika 2: Funkcija start()

A computer screen shot of a program code

AI-generated content may be incorrect.

Vir: lasten 2025

Nato zaženemo client.py, ki se poskusi povezati na ip naslov strežnika preko izbranih vrat. Ko strežnik zazna povezavo ustvari stranski proces, s katerim nadaljnje obravnava uporabnika (funkcija *handle\_client.py*). Pri tem mu pošlje svoj javni ključ in prejme javni ključ uporabnika ter preveri njegovo delovanje.

Slika 3: Funkcija handle\_client()

A screen shot of a computer program

AI-generated content may be incorrect.

Vir: lasten 2025

Ko je povezava sprejeta, client.py pošlje ključ na strežnik. Ta nato preveri, če je ključ v bazi z uporabniškimi podatki *accounts.db*. Če je ključ napačen je potrebna prijava ali registriracija*,* v nasprotnem primeru pa nadaljuješ v funkciji *send\_msg().*

Slika 4: Del za preverjanje ključa

A screen shot of a computer code

AI-generated content may be incorrect.

Vir: lasten 2025

#### Prijava in registracija

Ko uporabnik zažene *client.py*, se ta poveže na strežnik in pošlje ključ. Če ključa ni oziroma če se le ta ne ujema s tistim v *accounts.db*, strežnik vrne odgovor »!FALSE\_TOKEN«, ki povzroči aktivacijo funkcije *login()*, kjer izbereš ali se želiš prijaviti ali registrirati.

Slika 5: Funkcija login()

A computer screen shot of text

AI-generated content may be incorrect.

Vir: lasten 2025

Na strežniku se tudi aktivira zadnji del funkcije, ki preverja poslane podatke s tistimi v bazi in ustvarja nove s pomočjo funkcij *login()* in *register()*.

Slika 6: Del funkcije handle\_client(), ki skrbi za prijavni postopek

A screen shot of a computer program

AI-generated content may be incorrect.

Vir: lasten 2025

#### Odziv strežnika po prijavi

Po prijavi strežnik shrani uporabnikov novi ključ za povezovanje, ki ga ustvari funkcija *create\_token()* in je dolg 64 mest. Ta ključ nadomesti potrebo po vnašanju uporabniških podatkov vsakič, ko se znova povežemo na strežnik.

Slika 7: Funkcija create\_token()

A screenshot of a computer program

AI-generated content may be incorrect.

Vir: lasten 2025

Ključ se pri uporabniku zapiše v datoteko **token.txt** s pomočjo enkripcijskega algoritma in gesla, ki ga določi uporabnik.

#### Pošiljanje sporočil

Po prijavnem procesu uporabnik nadaljuje v funkciji *send\_msg()*. Najprej je potrebno vnesti uporabniško ime naslovnika, nato pa še sporočilo. Za pregled prejetih sporočil se uporabi tipka Enter.

Slika 8: Funkcija send\_msg() pri uporabniku

A computer screen with text on it

AI-generated content may be incorrect.

Vir: lasten 2025

Funkcija *send()* poskrbi za enkriptiranje besedila in pošiljanje na strežnik. Ta funkcija se uporablja ob vsakem pošiljanju na strežnik, med drugim tudi ob pošiljanju ključa. Funkcija preveri tudi dolžino sporočil, saj sporočila daljša od 256 mest, ne morejo biti enkriptirana, ker RSA omogoča samo enkripcijo besedil, dolgih toliko kot ključ ali manj.

Slika 9: Funkcija send()

A screen shot of a computer program

AI-generated content may be incorrect.

Vir: lasten 2025

Strežnik sprejme sporočila prijavljenih uporabnikov s pomočjo funkcije *send\_msg()*. Ta funkcija v osnovi prejme sporočilo, nato s pomočjo funkcije *get\_target\_info()* pridobi podatke naslovnika iz baze in preveri, če je povezan ter mu pošlje sporočilo. Vmes gre sporočilo čez več stopenj, ki preverjajo obliko sporočila, da se izognemo napakam, ki bi lahko ustavile strežnik.

Slika 10: Lovljenje napak in pošiljanje sporočil

A screen shot of a computer program

AI-generated content may be incorrect.

Vir: lasten 2025

Strežnik vsebuje tudi funkcijo, ki je namenjena pošiljanju sporočil uporabnikom.

Slika 11: Funkcija send() na strežniku

A screen shot of a computer code

AI-generated content may be incorrect.

Vir: lasten 2025

#### Prejemanje sporočila

Ob zagonu programa se pri uporabniku zažene funkcija *rec()* kot stranski proces, ki omogoča prejemanje sporočil, ki jih nato shrani v tabelo.

Slika 12: Funkcija rec() pri uporabniku

A screen shot of a computer program

AI-generated content may be incorrect.

Vir: lasten 2025

Za prikaz novih sporočil mora uporabnik klikniti tipko Enter, ki nato izpiše tabelo z sporočili.

#### Simetrična enkripcija

SafeChat uporablja dve vrsti enkripcije, simetrično in asimetrično.

Simetrično enkripcijo sem razvil sam in je namenjena enkripciji zasebnega ter povezovalnega ključa. Na začetku mora uporabnik vnesti geslo, ki se uporabi za enkripcijo. Geslo mora biti dolgo med 6 in 30 znakov, da se prepreči vsote, ki bi potrebovale ali manj ali več kot 3 števke za zapis v šestnajstiškem sistemu. Program ne preverja kompleksnosti gesla ampak samo število znakov.

***Ustvarjanje objektov in nastavljanje parametrov:***

Podatke se enkriptira po naslednjem postopku. Najprej se ustvari objekt za simetrično enkripcijo, ki mu kot parameter dovedemo geslo. Nato pa kličemo funkcijo *encrypt(text)* in kot *text* parameter navedemo besedilo poljubne dolžine.

Geslo (ključ): 'ab' 🡪 ['0110 0001', '0110 0010']

Besedilo: 'ba' 🡪 ['0110 0010', '0110 0001']

***Enkripcija:***Za začetek se ustvari spremenljivka *value*, ki je seštevek vseh binarnih vrednosti simbolov iz ključa. S tem se naredi dodatna masko, ki dodatno zakrije začetno vrednost. Nato pa gre algoritem čez vsako vrednost v tabeli z binarnimi vrednostimi besedila in hkrati tudi čez ključ. Vsaki vrednosti besedila se prišteje vrednost iz ključa, ki je na enakem indeksu. Če je besedilo daljše od ključa, ko uporabimo vse vrednosti iz ključa, začnemo ponovno z začetno vrednostjo in tako nadaljujemo dokler ne enkripritiramo celotnega besedila.

(Binarne vrednosti se seštevajo s pomočjo pretvorbe v desetiški sistem).

Value: *97(10) + 98(10) = 195*

*besedilo[0]+ ključ[0] + 195 = 97 + 98 + 195 = 39010 = 18616*

*besedilo[1]+ ključ[1] + 195 = 98 + 97 + 195 = 39010 = 18616*

Rezultat: *'186'* + *'186' = '186186'* 🡪 Ta vrednost se uporabi kot enkriptirano besedilo.

**Dekripcija:**

Postopek za dekripcijo je obraten enkripciji. Najprej enkriptirano besedilo razdeli na dele po 3 in nato vsakemu kosu odšteje *value* in vrednost iz ključa uporabljenega za dekripcijo in s tem nadaljuje do konca enkriptiranega besedila.

Enkriptirano besedilo: *'186186'* 🡪 [186, 186]

*39010 – value – ključ[0] = 390 – 195 – 97 = 9810* 🡪 'b'

*39010 – value – ključ[1] = 390 – 195 – 98 = 9810* 🡪 'a'

Besedilo: 'b' + 'a' = 'ba'

#### Asimetrična enkripcija

Za asimetrično enkripcijo je bil uporabljen postopek, ki je že opisan pod naslovom teorija. Uporabljen je za pošiljanje sporočil med strežnikom in uporabnikom. Ko uporabnik kontaktira strežnik, mu ta najprej pošlje svoj javni ključ, nato pa še uporabnik pošlje strežniku svojega. To omogoča zaščiteno izmenjavo sporočil.

#### Delo z bazo

V datoteki DatabaseRequests.py se nahajata funkciji, ki skrbita za poizvedbe v sql bazi. Prva funkcija select() poišče eno ali več podatkov iz baze, funkcija update() pa posodablja podatke v bazi ali ustavlja nove podatke. Za delovanje je bila uporabljena knjižnica sqlite3.

Slika 13: Funkciji za delo z bazo

A screen shot of a computer screen

AI-generated content may be incorrect.

Vir: lasten 2025

Datoteka baza.py pa ustvari to bazo, ki je namenjena shranjevanju uporabnikov.

Slika 14: Del programa, ki ustvari bazo

A computer screen with text and numbers

AI-generated content may be incorrect.

Vir: lasten 2025

## ZAKLJUČEK

Programiranje SafeChat-a mi je poglobilo znanje o delovanju omrežij in pokazalo kompleksnost, ki je potrebna za varno uporabo interneta.

V aplikaciji bi lahko izboljšal še veliko stvari, med drugim bi lahko naredil prijaznejši uporabniški vmesnik v *tkinterju*. Prav tako ima moj program za enkripcijo še prostor za izboljšavo z uvedbo novejših mask.

V aplikaciji manjka večja možnost kontrole uporabnikov nad njihovimi računi, saj trenutno še ne morejo spremeniti svojih podatkov po registraciji in računa prav tako ne morejo izbrisati.

Končno, trenutno uporabniki ne morejo enkriptirano poslatii sporočil daljših od 256 mest. To bi lahko popravil z razdeljevanjem sporočila na več manjših kosov. Prav tako bi lahko razvil boljše preverjanje kompleksnosti gesla, za večjo varnost.

***Povezava do programa:***

<https://github.com/BorzaTheKing123/SafeChat>

## VIRI

1. GeeksforGeeks. Python Tutorial: Learn Python Programming (Datum zadnjega popravljanja: 22.5.2024). Dostop na povezavi:

<https://www.geeksforgeeks.org/python-programming-language-tutorial/>

1. YouTube. Tech With Tim. Python Socket Programming Tutorial (Datum objave: 5.4.2020). Dostop na povezavi:

<https://www.youtube.com/watch?v=3QiPPX-KeSc>

1. YouTube. Corey Schafer. Python SQLite Tutorial: Complete Overview - Creating a Database, Table, and Running Queries (Datum objave: 18.4.2017). Dostop na povezavi: <https://www.youtube.com/watch?v=pd-0G0MigUA>
2. AsciiTable. ASCII Table. Dostop na povezavi: <https://www.asciitable.com/>
3. Medium. Ashiq KS. Asymmetric Cryptography with Python (Datum objave: 24.1.2019). Dostop na povezavi:

<https://medium.com/@ashiqgiga07/asymmetric-cryptography-with-python-5eed86772731>

1. GitHub. Ezequiel Leonardo Castaño. Python Security (Datum objave: 2021). Dostop na povezavi:

<https://elc.github.io/python-security/chapters/07_Asymmetric_Encryption.html#rsa-encryption>

1. GeeksforGeeks. Python Tutorial: Convert Hex String to Bytes in Python (Datum objave: 5.8.2024). Dostop na povezavi:

<https://www.geeksforgeeks.org/convert-hex-string-to-bytes-in-python/>

1. San José State University. The Mathematics behind RSA. Dostop na povezavi:

<https://www.cs.sjsu.edu/~stamp/CS265/SecurityEngineering/chapter5_SE/RSAmath.html>

1. BIOTEHNIŠKI IZOBRAŽEVALNI CENTER LJUBLJANA. mag. Dejan Cvitkovič, mag. Marina Vodopivec, Katarina Smole, Jana Marinček in mag. Irena Štrumbelj Drusany, Mojca Jevnikar. NAVODILA ZA IZDELAVO DIPLOMSKEGA DELA (Datum objave: december 2023). Dostop na povezavi:

<https://www.bic-lj.si/sites/default/files/inline-files/Navodila%20za%20izdelavo%20DDv2.pdf>

1. Python. Socket — low-level networking interface (Datum zadnjega popravljanja: 12.4.2025). Dostopno na povezavi:

<https://docs.python.org/3/library/socket.html>

1. GitHub. Samo Borzatta (BorzaTheKing123). SafeChat (Datum zadnjega popravljanja: 25.2.2025). Dostopno na povezavi:

<https://github.com/BorzaTheKing123/SafeChat>

1. Wikipedija. Berkeley Software Distribution (Datum zadnjega popravljanja: 17.8.2022). Dostopno na povezavi:

<https://sl.wikipedia.org/wiki/Berkeley_Software_Distribution>