TEHNIČKA ŠKOLA RUĐERA BOŠKOVIĆA
GETALDIĆEVA 4, ZAGREB

ZAVRŠNI STRUČNI RAD:
Deep Messenger

MENTOR: Marko Stojanović

UČENIK: Roko Dobovičnik
RAZRED: 4.G

Zagreb, travanj 2024.

Sadržaj

1. Uvod

Deep Messenger je decentralizirana chat aplikacija za anonimnu komunikaciju. Kako bi ostvarila anonimnost i *peer-to-peer* vezu među klijentima, aplikacija koristi TOR (The Onion Router) mrežu, odnosno skrivene servise (Hidden Services). Iz slanja poruka *peer-to-peer* proizlazi problem, naime klijenti mogu izmjenjivati poruke samo kada su oboje na mreži, kako bi riješili ovaj problem svaki od korisnika može konfigurirati svoj *mailbox* servis koji će primati poruke za njega dok nije na mreži. Naravno jedan *mailbox* servis može koristiti više klijenata. Aplikacija ima tekstualno korisničko sučelje TUI, dok ovakvo sučelje možda na prvu nije privlačno svim korisnicima, ono kao i GUI ima svoje prednosti poput mogućnosti korištenja aplikacije putem SSH. Aplikacija je napisana u C programskom jeziku, uz korištene više biblioteka navedenih u nastavku.

Ovu temu završnog rada odabrao sam jer sam kroz izradu i dizajn rada htio istražiti koncepte mrežnog programiranja u programskom jeziku C i način na koji Tor mreža funkcionira te kako ju koristiti van samog *Tor Browsera*. Način na koji Tor mreža radi mi je oduvijek bio interesantan i kroz ovaj rad sam ga htio dublje istražiti.

2. Teorijski dio

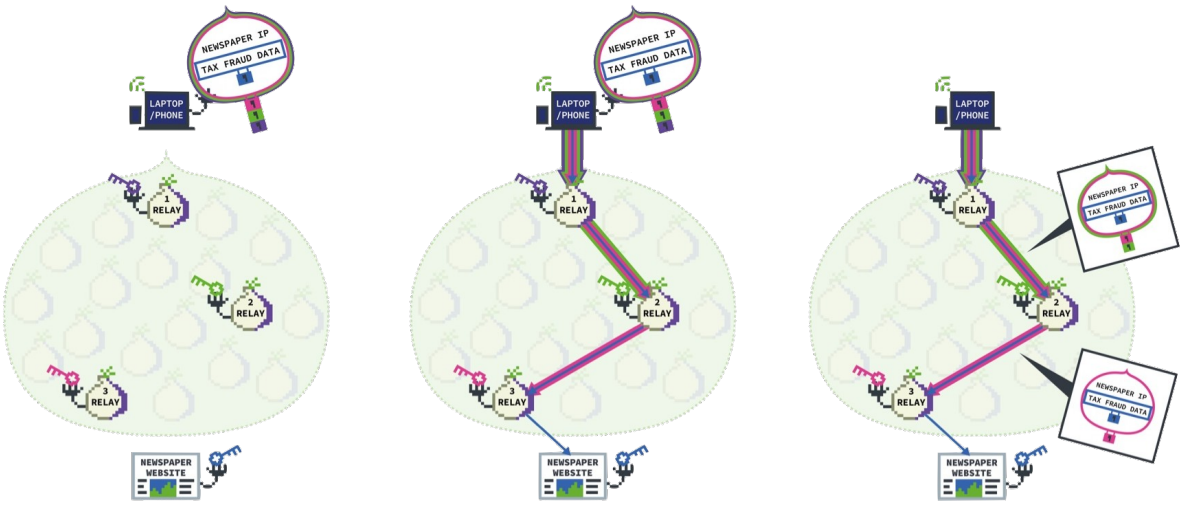
Ovo poglavlje navodi i opisuje biblioteke, alate i tehnologije korištene u izradi ovog završnog rada.

2.1. TOR (The Onion Router)

Tor je najpoznatija implementacija koncepta *onion* usmjeravanja, koje je osmislio i implementirao *Naval Research Lab* Sjedinjenih Američkih država devedesetih godina prošlog stoljeća. Kasnije kada se projektu pridružio Roger Dingledine projekt je dobio danas poznati naziv *The Onion Router* kako bi se razlikovao od ostalih sličnih implementacija. Danas je projekt pod vodstvom neprofitne organizacije *Tor Project* i sav softver koji razvijaju je otvorenog koda, tako da ga bilo tko može besplatno koristiti ili doprinijeti u njegovom razvoju.

Ono što nam Tor omogućuje je jest anonimni pristup internetu tako što naš promet preusmjerava kroz Tor mrežu. Tor mreža sastoji se od takozvanih Tor releja odnosno čvorišta koja se nalaze diljem svijeta, a postavljaju i održavaju ih volonteri koji žele doprinijeti funkcioniranju mreže. Što više aktivnih releja postoji, mreža postaje prohodnija i sigurnija. [1]

Ukratko, način na koji Tor funkcionira je sljedeći. Kako bi korisnik mogao pristupiti internetu kroz Tor mrežu, prvo mora dobaviti Tor klijenta, kojeg je moguće preuzeti sa službenih stranica Tor projekta ili u slučaju mnogih Linux distribucija, direktno iz repozitorija distribucije. Kada korisnik uspije dobaviti i pokrenuti klijenta, on će nasumično odabrati 3 dostupna releja i kroz njih uspostaviti vezu prema lokaciji kojoj korisnik želi pristupiti. Klijent će također 3 puta enkriptirati sav promet koji šalje, po jednom za svaki od releja. Kada enkriptirani podaci stignu do prvog releja on će ih dekriptirati svojim ključem, pa će tada podaci biti enkriptirani samo dva puta. Ovaj proces se nastavlja dok podaci ne dođu do trećeg releja kada napuštaju Tor mrežu. Proces možemo usporediti sa skidanjem pojedinih ljuski luka, od kuda i dolazi naziv *The Onion Router*. Dijagram ispod grafički prikazuje putovanje podataka kroz Tor mrežu.



Slika 2.1 – ilustracija tor mreže – https://gitlab.torproject.org/tpo/community/training/-/tree/master/2023

Svakom od releja poznata je IP adresa (odnosno identitet) točke prije i poslije njega te stoga, za slučaj sa slike u teoriji nije moguće odrediti tko je osoba koja pristupa stranici. Naravno Tor mreža nije savršena, a postoji i mnogo drugih načina kako je moguće ugroziti svoju anonimnost, ali to nije tema ovog završnog rada.

2.1.1. Skriveni servisi (Hidden services)

Osim sustava koji pruža anonimnost korisnicima dok pristupaju internet servisima, *Tor project* je razvio i metodu kojom omogućuje i samim servisima da ostanu anonimni, odnosno skriveni. Ova metoda vrlo je praktična iz više razloga i jedna je od ključnih funkcionalnosti koja je omogućila izradu Deep Messenger aplikacije. U normalnim okolnostima kako bi pristupili nekom internet servisu potrebno je znati njegov priključak (eng. port) i IP adresu. S druge strane u slučaju skrivenih servisa to nije potrebno, dokle god uređaj ima pristup internetu i može pristupiti Tor mreži, može i posluživati skriveni servis. Skriveni servisi koriste sličan uzorak kao i pri normalnom slanju prometa kroz Tor mrežu, odnosno koriste sklopove od po 3 releja, ali uz više dodatnih koraka koje moraju napraviti kako bi postali dostupni klijentima. Kako bi se klijent povezao na skriveni servis, kao i obično mora znati priključak na koji se želi povezati, ali uz to umjesto njegove IP adrese mora znati njegovu *onion* adresu. [2]

Danas su u uporabi *onion* adrese verzije 3. Svaka takva adresa sastoji se od 56 naizgled nasumičnih znakova i završetka „`.onion`”. Jedan primjer takve adrese je:

`i4mcwgorejxtforxrd7dsf73hsiiphhlgxxz3aeuef3hixdcv4vg3bid.onion`

Kada dekodiramo prvi dio *onion* adrese base32 algoritmom, iz adrese možemo iščitati ED25519 javni ključ skrivenog servisa, i još neke dodatne podatke. [3]

Ova karakteristika primjenjuje se kod zahtjeva za prijateljstvo u Deep Messenger protokolu opisanom ispod. Pomoću potpisa privatnim ključem koji odgovara javnom ključu iz adrese, klijent može dokazati da je stvarno vlasnik dane *onion* adrese, bez ikakvih dodatnih komunikacija sa Tor mrežom.

2.1.2. Tor klijent (program)

Već je spomenuto da kako bi se povezao sa Tor mrežom korisnik mora imati Tor klijenta, također poznatog i kao *little-t-tor*, *tor binary* ili *tor network daemon*. Tor klijent je jednostavan program koji se pokreće u komandnoj liniji, odnosno terminalu. On uspostavlja vezu sa Tor relejima i provodi sve potrebne operacije kako bi se mogli spojiti. Nakon što je završio sa inicijalizacijom, Tor klijent podiže *SOCKS proxy* server verzije 5, koji omogućuje ostalim aplikacijama na sustavu da šalju promet kroz Tor mrežu. Tor klijenta moguće je konfigurirati pomoću konfiguracijske datoteke *torrc* koju specificiramo pri pokretanju, moguće ga je konfigurirati i da pokrene već spomenuti skriveni servis i dopusti pristup nekom od lokalnih priključaka (eng. portova) kroz Tor mrežu koristeći *onion* adresu. Na taj način moguće je vrlo jednostavno postaviti skrivenu stranicu koristeći web server kao što je na primjer *Apache2*. Isto tako Deep messenger klijent omogućuje drugim klijentima da mu šalju poruke.

2.2. OpenSSL

OpenSSL je vrlo poznati set softverskih alata za kriptografiju i sigurnu komunikaciju. Otvorenog je koda i licenciran je Apache2 licencom. Uključuje *openssl* program naredbenog retka koji se može koristiti za kreiranje ključeva, potpisivanje, računanje sažetaka (eng. hash), enkodiranje, enkriptiranje i slično, sve to koristeći razne vrste algoritama. Osim programa OpenSSL sadrži i biblioteku pod nazivom *libcrypto*, koja ima vrlo široku primjenu i danas se koristi u implementacijama mnogih programa i protokola. S obzirom da većina kriptografskih algoritama iste vrste imaju zajedničke značajke i funkcioniraju na sličan način, OpenSSL je dizajniran tako da se iste funkcije mogu koristiti za različite tipove algoritama. Implementacije algoritama zajedno su grupirane u *providere*, uz zadane *providere* koje OpenSSL pruža moguće je koristiti i druge *providere* trećih strana kako bi proširili funkcionalnost biblioteke.

U Implementaciji Deep Messenger protokola koristi se viši *libcrypto* API pod nazivom EVP (eng. envelope), i još nekoliko drugih navedenih u nastavku. Konkretno Deep Messenger klijenti koriste sljedeće grupe funkcija:

EVP_Seal* i EVP_Open*

Grupe funkcija za kreiranje takozvane omotnice. Funkcije primaju asimetrični ključ i generiraju simetrični ključ kojim će enkriptirati podatke, nakon što su podaci enkriptirani simetričnim ključem, simetrični ključ se enkriptira asimetričnim. Kod dekriptiranja podataka, simetrični ključ se prvo dekriptira asimetričnim kojim se zatim dekriptiraju podaci. Funkcije su tako dizajnirane jer proces enkriptiranja simetričnim ključem zahtjeva puno manje procesorske snage na većim količinama podataka, u usporedbi sa asimetričnim enkriptiranjem.

OSSL_ENCODER* i OSSL_DECODER*

Grupe funkcija koje se koriste za enkodiranje asimetričnih ključeva, konkretno u slučaju Deep Messenger aplikacije koriste se za enkodiranje i dekodiranje RSA 2048 bitnih ključeva u i iz DER formata.

EVP_DigestSign* i EVP_DigestVerify*

Koriste se za kreiranje i provjeru potpisa korištenjem asimetrične kriptografije. Funkcije primaju ključ koji se koristi za potpis ili provjeru ovisno o operaciji i zatim podatke koje je potrebno potpisati, ovisno o algoritmu podaci mogu biti predani u više navrata (eng. stream). U slučaju Deep Messengera koristi se ED25519 algoritam koji podržava samo potpisivanje svih podataka od jednom.

Uz navedene koriste se još neke funkcije za generiranje i manipulaciju ključevima i kreiranje sažetaka, više o tipovima ključeva i sažetaka u nastavku.

Svi podaci vezani uz OpenSSL iščitani su iz službene OpenSSL dokumentacije. [4]

2.3. Libevent

Libevent je biblioteka koja se koristi za pisanje brzih i portabilnih IO operacija. Koristi se u mnogim poznatim mrežnim aplikacijama kao što je *Chrome* internet preglednik i već spomenuti Tor klijent. Biblioteka je dizajnirana kako bi bila portabilna, odnosno kako bi programi za mrežnu komunikaciju implementirani pomoću ove biblioteke bili kompatibilni sa svim poznatim operacijskim sustavima. Ovisno u kakvom se okruženju nalazi, Libevent bira najpogodniju i najbržu IO implementaciju dostupnu u tom okruženju. Omogućuje programima da funkcioniraju uz desetke tisuća paralelno otvorenih veza.

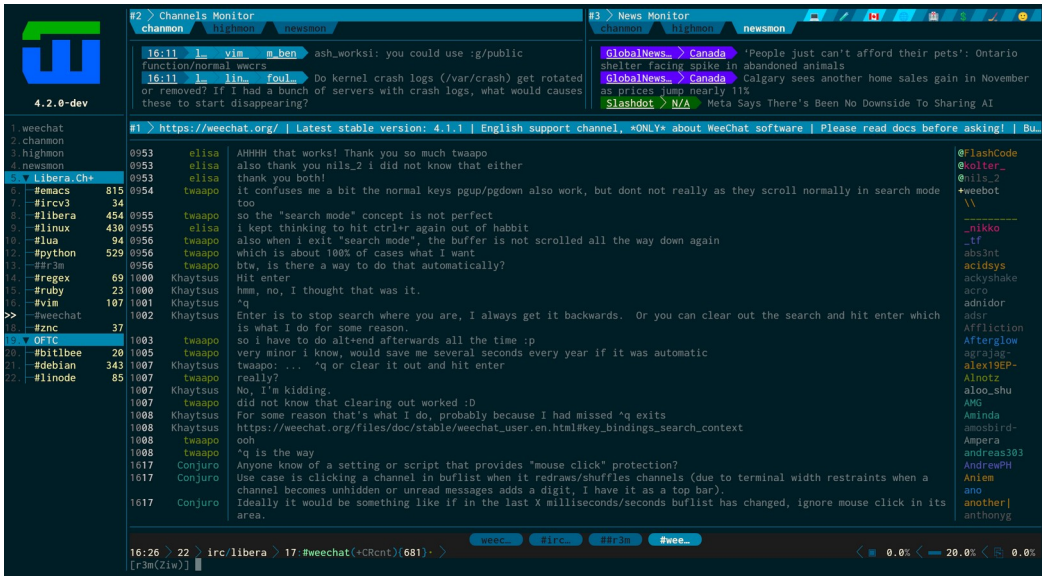
Slika 2.2 – Libevent logotip – https://libevent.org/images/libevent3.png

Libevent pruža nekoliko odvojenih skupina funkcija koje uvelike pojednostavljuju pisanje mrežnih aplikacija pružajući programeru razne apstrakcije nad učestalim operacijama koje bi u suprotnom bio prisiljen sam implementirati. [5]

U implementaciji Deep Messenger aplikacije sav kod povezan sa mrežom i mrežnim prometom koristi Libevent.

2.4. Ncurses

Ncurses je biblioteka koja se koristi za izradu takozvanog *TUI* sučelja, odnosno tekstualnog korisničkog sučelja. Naime još od samih početaka sučelja naredbenog retka (eng. terminal) uz ispisivanje normalnih znakova i slova podržavaju i mogućnost ispisa takozvanih kontrolnih znakova. Ti znakovi omogućuju nam da primijenimo način na koji se terminal ponaša. Tako možemo promijeniti boju slova, poziciju kursora na zaslonu i slično.



Slika 2.3 – TUI primjer – https://weechat.org/about/screenshots/weechat/weechat_2023-12-01_r3m.png/

Ncurses pruža apstrakciju nad samim kontrolnim znakovima i pomoću seta funkcija omogućuje kreiranje prozora, podprozora, virtualnih prozora (eng. pad), mijenjanje stilova znakova (koliko terminal dozvoljava), primanje znakova sa standardnog ulaza i drugo.

Deep Messenger koristi ncurses za implementaciju kompletnog korisničkog sučelja, s time da Deep messenger aplikacija sadrži dodatnu apstrakciju nad ncurses funkcijama.

2.5. SOCKS protokol verzije 5

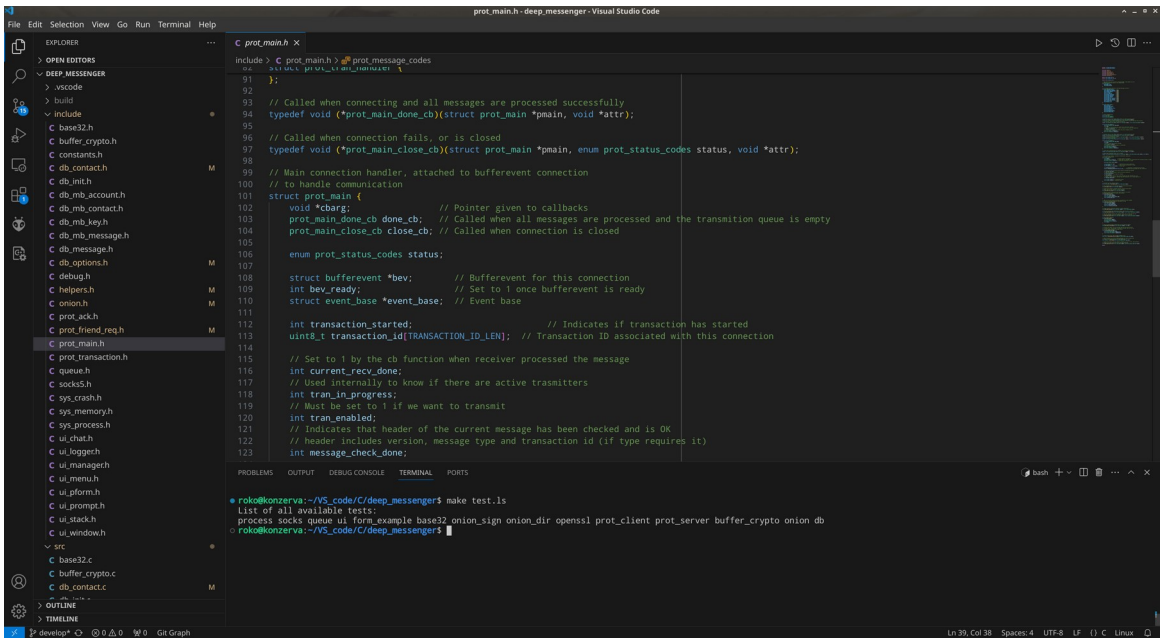
Deep Messenger koristi vlastitu implementaciju SOCKS 5 klijenta. Ovaj protokol je nužan za komunikaciju sa već spomenutim Tor klijentom.

SOCKS 5 je vrlo jednostavan *proxy* protokol definiran 1996. godine. Dizajniran je kako bi se mogao koristiti sa bilo kojim postojećim protokolom aplikacijskog sloja te kako bi uz dovoljno snažnu autentikaciju omogućio pristup servisima koji se nalaze iza vatrozida. Tor klijent koristi blago proširenu verziju ovog protokola kako bi omogućio aplikacijama na računalu da tuneliraju promet kroz Tor mrežu, jedna od takvih aplikacija je sam Tor Browser [6]

2.6. Opis korištenih alata

2.6.1. Visual Studio Code

Visual Studio Code je tekstualni editor otvorenog koda razvijan od strane Microsofta. VSCode je postao jedan od najpopularnijih tekstualnih editora današnjice, primarno zbog svoje mogućnost proširenja odnosno ekstenzija. Zbog svoje popularnosti danas za VSCode postoji proširenje gotovo za svaki postojeći programski jezik i mnoge dodatne funkcionalnosti.



Slika 1.4 – Sučelje Visual Studio Codea

Konkretno u izradi Deep Messengera korišteno je proširenje *C/C++ Extension Pack*.

2.6.2. ChatGPT 3.5

ChatGPT je AI model koji je razvila tvrtka OpenAI. Dostupan za besplatno korištenje putem web sučelja na adresi https://chat.openai.com. U okviru razvoja Deep Messengera korišten je

ispravljanje jednostavnijih grešaka u kodu i traženja odgovora na neka nespecifična pitanja. Odgovara vrlo neprecizno i netočno kod pitanja vezanih uz određenu specifičnu tematiku.

2.6.3. Git version control

Git je besplatan sustav otvorenog koda koji se koristi za kontrolu verzija. Posebno je poznat u *opensource* zajednici, a koriste ga i mnoge komercijalne tvrtke.

Slika 1.5 – Git logotip – https://git-scm.com/images/logo@2x.png

Git je jednostavan za korištenje i pruža razne funkcionalnosti koje nam između ostalog omogućuju da zasebno spremamo svaku promjenu koju napravimo u kodu. Na taj način ako dođe do nekih grešaka uvijek možemo pregledati i vratiti se na stare verzije koda. Git također omogućuje i grananje, pomoću kojeg možemo raditi na više verzija koda istovremeno, i mnoge druge funkcionalnosti. [7]

Za pohranu ovog projekta uz sami *git* komandni alat korištena je i web stranica github.com za pohranu koda u javnom repozitoriju.

2.6.4. GNU Make

Make je alat koji se koristi za automatizaciju procesa kompajliranja i povezivanja koda. Proces je opisan koristeći skriptu koja se naziva *Makefile*, pozivom naredbe *make*, skripta se interpretira i proces kompajliranja započinje. Na ovaj način čak i korisnici koji ne znaju kako ručno kompajlirati program svejedno ga mogu instalirati na svoj sustav.

Make također prati koje izvorne datoteke su promijenjene i kod ponovnog kompajliranja, kompajlirati će samo promijenjene datoteke, čime smanjuje vrijeme kompajliranja, pogotovo na sporijim sustavima. Uz to Make se koristi i za mnoge druge operacije poput pokretanja *unit* testova.

U okviru Deep Messengera Make se koristi u kombinaciji sa *gcc* kompajlerom za kompajliranje i povezivanje izvornog koda.

3. Tehničko tehnološki dio

Ovo poglavlje opisuje dizajn i implementaciju Deep Messenger protokola i klijentskog programa te detaljnije opisuje mjesto uporabe gore navedenih biblioteka i alata.

3.1. Deep Messenger protokol

Ovaj protokol opisuje način komunikacije između klijentskih programa i *mailbox* servisa Deep Messenger aplikacije. Cilj protokola je da omogući anonimnu komunikaciju između dvaju klijenata koristeći Tor mrežu. Kada je na mreži, svaki od klijenata podiže prethodno spomenuti skriveni servis kako bi ga drugi klijenti mogli kontaktirati. Za slučaj kada klijent nije na mreži, može odabrati jedan *mailbox* servis koji će umjesto njega primati poruke dok se ne vrati na mrežu.

Svaki klijent i *mailbox* servis ima svoju jedinstvenu *onion* adresu prema kojoj ga ostali prepoznaju. Vlasnik *onion* adrese posjeduje pripadni javni i privatni ključ pomoću kojih dokazuje da je on stvarno vlasnik te *onion* adrese.

Problem ideje *mailbox* servisa je što klijent mora podijeliti svoju listu kontakata sa servisom kako bi servis kada zaprimi poruku, mogao odrediti je li pošiljatelj poruke stvarno na listi kontakata. U suprotnom bi bilo tko kome je adresa servisa poznata mogao poslati veliku količinu poruka na servis, samo kako bi zapunio memoriju servisa. Naravno ovdje se nameće problem jer ako svi klijenti podjele svoje kontakte sa *mailbox* servisom vlasnik servisa može ugroziti anonimnost klijenata, jer će iz tih podataka moći vidjeti tko sa kime razgovara. Čak i ako je vlasnik servisa osoba od povjerenja, u slučaju provale napadač će moći napraviti isto.

Kako bi spriječili takvo razotkrivanje povezanosti među klijentima, svaki od klijenata prilikom uspostave prijateljstva generira dva para ključeva i prosljeđuje javne ključeve iz oba para onom drugom. U svim daljnjim izmjenama poruka, klijenti više ne koriste *onion* adresu za međusobno prepoznavanje, već ključeve koje su izmijenili prilikom uspostave prijateljstva. Više o izmjeni ključeva slijedi u nastavku.

3.1.1. Kriptografski algoritmi i ključevi

Protokol koristi niz kriptografskih algoritama u svrhu potpisivanja i enkriptiranja poruka pomoću kojih klijenti komuniciraju. Slijedi lista svih tipova ključeva, potpisa i enkripcije koji se koriste:

Par ključeva *onion* adrese tipa *ED25519*, generiraju se jednom kod prvog pokretanja klijenta, iz tih ključeva generira se i sama adresa. Koriste se za pokretanje skrivenog servisa i za potpisivanje prilikom uspostave prijateljstva.

Par Ključeva za potpisivanje poruka tipa *ED25519*, svaki klijent generira jedan ovakav par ključeva prilikom uspostave prijateljstva i prosljeđuje javni ključ drugomu. Kasnije klijent potpisuje poruku privatnim ključem, a drugi može provjeriti potpis javnim ključem koji je ranije zaprimio.

Par ključeva za enkriptiranje poruka tipa *RSA* duljine 2048 bita, ovo je drugi par ključeva koji se generira prilikom uspostave prijateljstva, klijent opet prosljeđuje javni ključ drugom klijentu, koji će onda u budućnosti koristiti taj ključ kako bi enkriptirao sadržaj poruka koje šalje kako bi ih samo primatelj mogao dekriptirati i pročitati.

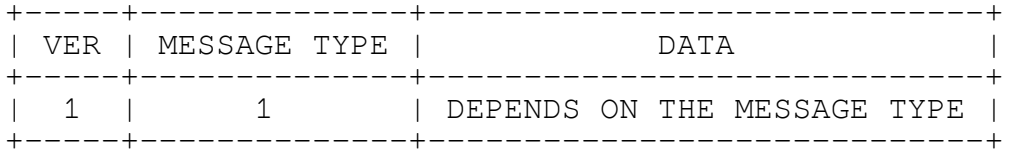
Proces potpisivanja poruka odvija se u dva koraka. U prvom koraku klijent računa *sha256* sažetak cijele poruke, uključujući i zaglavlje poruke koje sadrži verziju i tip poruke. Nakon toga klijent potpisuje izračunati sažetak *ED25519* ključem i dodaje potpis na kraj poruke.

Proces enkriptiranja poruke koristi OpenSSL funkcije za kreiranje omotnica. Kao algoritam za simetričnu enkripciju koristi se *AES 256 CBC*. Kako bi primatelj mogao dekriptirati sadržaj poruke, potrebno je uz enkriptirani simetrični ključ dostaviti i 16 bajtni IV broj generiran u procesu zatvaranja omotnice.

Prilikom računanja *onion* adrese za javni ključ tipa *ED25519*, prema specifikaciji Tor protokola potrebna nam je i implementacija *sha3_256* algoritma. [8]

3.1.2. Zaglavlje poruke protokola

Sve poruke definirane u ovom protokolu započinju sa dva polja duljine 1B, prvo polje sadrži verziju protokola, odnosno u ovom slučaju broj 1, ukoliko klijent zaprimi poruku koja sadrži neispravnu verziju dužan je prekinuti vezu. Drugo polje sadrži tip poruke, ukoliko klijent zaprimi neispravan tip poruke dužan je prekinuti vezu. Nakon navedena dva polja slijede dodatna polja i podaci koji ovise o tipu poruke.



Dijagram 3.1 – prikazuje oblik zaglavlja poruke

Za sve tipove poruka definirane ispod, navedena su samo polja koja se nalaze u *DATA* dijelu dijagrama iznad, te se podrazumijeva da ispred njih dolaze polja *VER* i *MESSAGE TYPE*.

Svi tipovi poruka koji sadrže ID transakcije moraju ga dostaviti kao prvo polje radi lakšeg odbacivanja neispravnih poruka. *MESSAGE TYPE* polje svih poruka koje sadrže ID transakcije i

čija se izmjena vrši nakon uspostave transakcije mora imati bit najveće težine postavljen na 1 radi praktičnije implementacije.

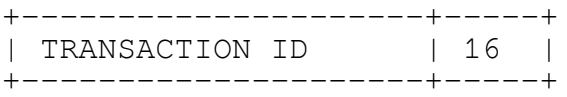
3.1.3. Tip poruke: Transaction request (0x01)

Prije nego što dva klijenta mogu početi komunicirati preko *socketa* potrebno je započeti transakciju. Klijent koji je započeo vezu šalje zahtjev za uspostavu, a drugi klijent tada odgovara sa porukom *TRANSACTION RESPONSE* koja sadrži ID ove transakcije. Taj ID transakcije vrijedi samo za tu vezu i čim se veza prekine oba klijenta će zaboraviti da je postojao. Na ovaj način čak i ako netko uspije uhvatiti jedan od potpisanih paketa ne mogu se predstaviti kao netko drugi jer transaction ID neće odgovarati.

S obzirom da transakcija još nije započela nema smisla da klijent išta potpisuje ili se predstavlja, stoga je ovo jedini tip poruke koji nema tijelo.

3.1.4. Tip poruke: Transaction response (0x02)

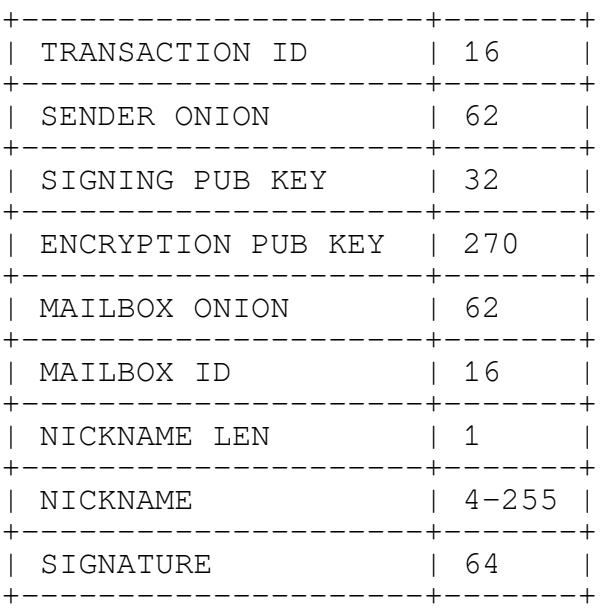
Ovim tipom poruke odgovara klijent koji je zaprimio *TRANSACTION REQUEST* i inicijalizirao transakciju za ovaj *socket*. Tijelo poruke sadrži samo 16 bajta koji predstavljaju ID ove transakcije. Ovom porukom je transakcija započela i sve poruke koje se nadalje izmjenjuju sadrže dani transaction ID, ako poruka ne sadrži odgovarajući transaction ID smatra se da je poruka komprimitirana i veza se prekida.



Dijagram 3.2 – prikazuje oblik tijela *Transaction response* poruke

3.1.5. Tip poruke: Friend request (0x81)

Kako bi dva klijenta mogla izmjenjivati poruke moraju jedan drugog smatrati prijateljima. Prilikom uspostave prijateljstva klijenti izmjenjuju podatke koji im omogućuju da izmjenjuju poruke direktno (peer-to-peer) ili putem mailbox servera. Svaki *friend request* sadrži slijedeće podatke.



Dijagram 3.3 – prikazuje oblik tijela poruke *Friend request*

TRANSACTION ID generirani ID za ovu transakciju (poruku).

SENDER ONION onion adresa pošiljatelja.

SIGNING PUB KEY javni ključ *ED25519* generiran samo za dani kontakt, odnosno klijenta kojeg tražimo da nam postane prijatelj. Primatelj pohranjuje ključ i koristi ga za buduće predstavljanje i potpisivanje poruka koje šalje, kao što je objašnjeno ranije.

MAILBOX ONION onion adresa *mailboxa* koji pošiljatelj koristi. U slučajevima kada pošiljatelj nije na mreži, ali primatelj mu želi poslati poruku može kontaktirati njegov *mailbox* servis na ovoj adresi. Ukoliko klijent ne koristi *mailbox* servis ovo polje mora biti ispunjeno nulama.

MAILBOX ID kako jedan *mailbox* server može koristiti više klijenata, ovo je broj pretinca pošiljatelja na koji je potrebno slati poruke. Ukoliko klijent ne koristi *mailbox* server ovo polje mora biti ispunjeno nulama.

NICKNAME LEN duljina nadimka (korisničkog imena) u bajtovima. Može sadržavati vrijednost između 4 i 255, jer je 4 najmanja dopuštena duljina nadimka. Ukoliko je duljina nadimka manja od 4, odbaciti zahtjev.

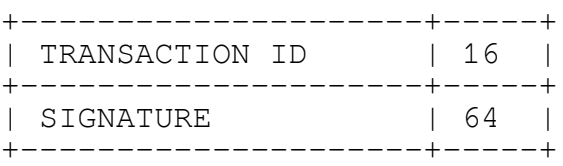
NICKNAME niz bajtova koji predstavljaju ASCII znakove korisničkog imena, duljina niza definirana je u *NICKNAME LEN* polju i ne može biti manja od 4.

SIGNATURE potpis gore navedenih podataka koristeći *ED25519* ključ *onion* adrese pošiljatelja. Ovim potpisom potvrđujemo da je pošiljatelj stvarno onaj tko se predstavlja. Ukoliko je potpis neispravan, zahtjev nije ispravan i odbacuje se.

Kada klijent zaprimi *FRIEND REQUEST*, odgovara sa *ACK ONION* porukom. Nakon što je korisnik potvrdio da želi biti prijatelj danom klijentu, odgovara na način da šalje svoj *FRIEND REQUEST* tom klijentu. Kada klijent odgovori sa *ACK ONION* prijateljstvo je uspješno uspostavljeno.

3.1.6. Tip poruke: Ack onion (0x82)

Jednostavna poruka koja potvrđuje da je dani zahtjev uspješno zaprimljen ili odrađen. Sadrži *TRANSACTION ID* trenutne transakcije kako bi bio nevažeći van te transakcije.



Dijagram 3.4 – prikazuje oblik tijela poruke *Ack onion*

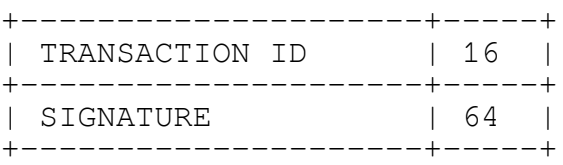
TRANSACTION ID generiran za trenutnu TCP vezu (stream).

SIGNATURE potpis podataka *ED25519* ključem *onion* domene.

Ovaj tip odgovora koristi se kod zaprimanja zahtjeva za prijateljstvo ili zaprimanja poruke na *mailbox* servisu.

3.1.7. Tip poruke: Ack signature (0x83)

Poput i *ACK ONION* odgovora ovaj odgovor se također koristi kao potvrda za odrađeni zadatak.



Dijagram 3.5 – prikazuje oblik tijela poruke *Ack signature*

TRANSACTION ID generiran za trenutnu TCP vezu (stream).

SIGNATURE potpis *ED25519* ključem koji smo izmijenili prilikom zahtjeva za prijateljstvo (*FRIEND REQUEST*).

Ovaj odgovor se koristi pri komunikaciji među klijentima, npr. kada klijent pošalje poruku drugom klijentu ovaj mu odgovara sa *ACK SIGNATURE* kako bi potvrdio primitak poruke.

3.1.8. Tip poruke: Message Container (0x84)

Ovo je najčešće korišteni tip poruke, ovaj tip poruke je spremnik koji koristimo kada šaljemo tekstualne poruke i još neke kontrolne poruke kao što je npr. obavijest o promjeni *mailboxa* ili korisničkog imena. Sve poruke koje kao klijent šaljemo na *mailbox* drugog klijenta su ovog formata kako bi sve izgledale identično.

TRANSACTION ID	16
RECEIVER MAILBOX ID	16
SENDER SIGNING KEY	32
MESSAGE ID	16
DATA LEN	4
DATA	VAR
DATA KEY	512
DATA IV	16
SIGNATURE	64

Dijagram 3.6 – prikazuje oblik tijela *Message Container* poruke

RECEIVER MAILBOX ID nasumičnih 16 bajta koji predstavljaju pretinac na *mailbox* serveru u koji šaljemo poruku. Ukoliko šaljemo poruku klijentu ovo polje mora biti ispunjeno nulama.

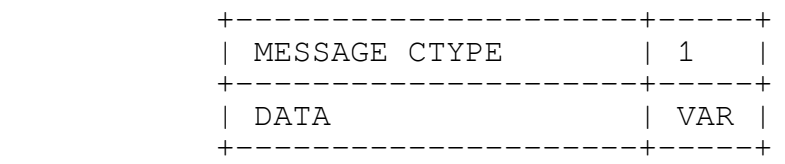
SENDER SIGNING KEY javni ključ tipa *ED25519* koji nam je dostavljen prilikom zahtjeva za prijateljstvo.

TRANSACTION ID generirani ID koji identificira transakciju.

MESSAGE ID nasumično generirani jedinstveni ID poruke.

DATA LEN broj bajta koji slijedi u *DATA* dijelu poruke, mora biti najmanje 1 u suprotnom se poruka odbacuje.

DATA sami podaci koji se prenose, enkriptirani simetričnim ključem pohranjenim u *DATA KEY* polju. Nakon dekriptiranja prvi bajt predstavlja tip poruke pohranjene u spremniku. Ovisno o tipu poruke različita je i struktura same poruke.



Dijagram 3.7 – prikazuje sadržaj *DATA* polja poruke

Dopuštene vrijednosti *MESSAGE CTYPE* polja (Message Content Type) su:

- *TEXT* (0x01)
  Obična tekstualna poruka, ostatak poruke se interpretira kao poruka koju je korisnik poslao.
- *NICK* (0x02)
  Izmjena nadimka (korisničkog imena) koje korisnik koristi, odnosno postavljanje novog koji je naveden u bajtovima koji slijede.
- *MBOX* (0x03)
  Postavljanje novog *mailbox* servisa za klijenta koji šalje poruku. Prvih 16 bajta je *mailbox* ID koji je dodijeljen klijentu na novom *mailboxu*, zatim slijede 62 bajta koji su *onion* adresa novog *mailbox* servisa.
- *RECV* (0x04)
  Dojava da je dana poruka uspješno poslana, poruka ovog sadržaja nije namijenjena da se šalje putem *mailbox* servisa već samo direktno među klijentima, kako bi oba klijenta bila sigurna da ju je drugi zaprimio. Nakon tipa sadržaja poruke slijedi 16 bajta koji predstavljaju ID poruke (*MESSAGE ID*) za koju se šalje potvrda.

DATA KEY simetrični ključ tipa *AES 256 CBC* korišten za enkriptiranje podataka, enkriptiran *RSA* javnim ključem dostavljenim u polju *ENCRYPTION PUB KEY* prilikom zahtjeva za prijateljstvo.

DATA IV je nepredvidljivi nasumično generirani broj koji se koristi kod *AES 256 CBC* simetrične enkripcije.

SIGNATURE potpis *ED25519* privatnim ključem koji odgovara ključu dostavljenom u polju *SENDER SIGNING KEY*.

Ukoliko bilo koja od provjera poput provjere potpisa ili duljine određenih dijelova poruke ne bude uspješna, *mailbox* ili sam klijent (ovisno kome je poruka upućena) odmah će zatvoriti vezu, bez ikakvog odgovora. Ukoliko su sve provjere uspješno prošle klijent će odgovoriti odgovorom *ACK SIGNATURE*, a *mailbox* sa *ACK ONION*.

3.1.9. Tip poruke: Mailbox register (0x85)

Ovaj tip poruke koristi se prilikom registracije na novi *mailbox*. Nakon što korisnik aplikacije unese *onion* adresu svog novog *mailboxa* i pristupni ključ, ovakva poruka se šalje kako bi se registrirao na *mailbox*.

TRANSACTION ID	16
ACCESS KEY	25
SIGNING PUB KEY	32

Dijagram 3.7 – prikazuje sadržaj tijela *Mailbox register* poruke

ACCESS KEY pristupni ključ generiran na *mailbox* serveru. Radi se o nizu nasumičnih znakova kojim klijent dokazuje da ima dopuštenje registrirati se na *mailbox* serveru.

TRANSACTION ID generirani ID koji označavaju transakciju.

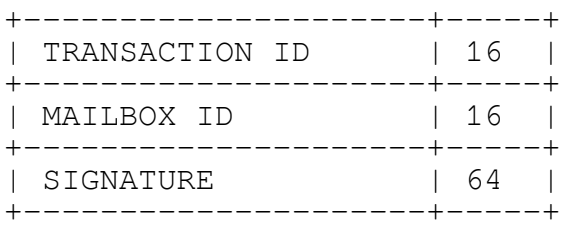
SIGNING PUB KEY javni ključ tipa *ED25519*, koristi se za autentikaciju klijenta u danjoj komunikaciji sa serverom. Klijent odgovarajućim privatnim ključem potpisuje sve poruke koje šalje *mailbox* serveru.

Na javnim mailbox serverima *ACCESS KEY* mora biti postavljen na 0, iako u trenutnoj verziji Deep Messengera funkcionalnost javnih mailbox servera nije implementirana

Ukoliko je *ACCESS KEY* ispravan ili se radi o javnom *mailbox* serveru, server će odgovoriti sa *MAILBOX GRANTED* odgovorom. U suprotnom će samo zatvoriti vezu.

3.1.10. Tip poruke: Mailbox granted (0x86)

Nakon što je klijent zatražio *mailbox* pretinac na danom *mailbox* serveru, ukoliko je priložio ispravan pristupni ključ, server će odgovoriti ovim tipom poruke.



Dijagram 3.8 – prikazuje sadržaj tijela *Mailbox granted* poruke

MAILBOX ID broj pretinca na *mailbox* serveru koji je dodijeljen klijentu koji je zatražio registraciju.

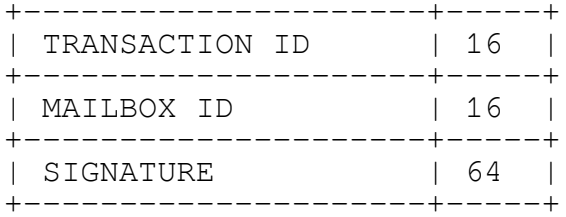
TRANSACTION ID generirani ID kopiran iz *MAILBOX REGISTER* poruke.

SIGNATURE potpis podataka *ED25519* ključem *onion* adrese *mailboxa*.

Nakon što se je klijent uspješno registrirao na *mailbox* server, može proslijediti svoj novi *MAILBOX ID* i *MAILBOX ADDRESS* svim svojim kontaktima.

3.1.11. Tip poruke: Mailbox fetch (0x87)

Pribavi sve poruke sa *mailbox* servera za svoj račun. *Mailbox* odgovara sa listom poruka koje se nalaze u korisničkom pretincu



Dijagram 3.9 – prikazuje sadržaj tijela *Mailbox fetch* poruke

MAILBOX ID jedinstveni broj pretinca klijenta koji traži listu poruka.

TRANSACTION ID generirani ID transakcije.

SIGNATURE potpis *ED25519* privatnim ključem koji odgovara javnom ključu dostavljenom prilikom registracije na *mailbox* server.

Server odgovara sa tipom poruke *MESSAGE LIST*.

3.1.12. Tip poruke: Mailbox set contacts (0x88)

Postavi kontakte za svoj pretinac na mailbox serveru. Mailbox server će odbaciti sve kontakte koji su bili prije postavljeni i od primitka na dalje će koristiti novu listu kontakata.

TRANSACTION ID	16
MAILBOX ID	16
CONTACTS LEN	2
CONTACTS	VAR
SIGNATURE	64

Dijagram 3.10 – prikazuje sadržaj tijela *Mailbox set contacts* poruke

MAILBOX ID jedinstveni broj pretinca klijenta koji dodaje kontakt.

CONTACT LEN broj kontakata koje želimo postaviti za danog klijenta. *Mailbox* će u ime klijenta prihvatiti poruke samo od kontakata koji su na listi.

CONTACT LEN broj kontakata koje želimo postaviti za danog klijenta. *Mailbox* će u ime klijenta prihvatiti poruke samo od kontakata koji su na listi.

TRANSACTION ID generirani ID transakcije.

SIGNATURE potpis *ED25519* privatnim ključem koji odgovara javnom ključu dostavljenom prilikom registracije na *mailbox* server.

Ukoliko je operacija uspješna server odgovara sa *ACK ONION* porukom.

3.1.13. Tip poruke: Mailbox del account (0x89)

Briše registrirani pretinac sa *mailbox* servera, uključujući i sve podatke unutar pretinca, odnosno sve kontakte i poruke koje su poslali.

TRANSACTION ID	16
MAILBOX ID	16
SIGNATURE	64

Dijagram 3.11 – prikazuje sadržaj tijela *Mailbox del account* poruke

MAILBOX ID jedinstveni broj pretinca klijenta koji se odjavljuje sa servera.

TRANSACTION ID generirani ID transakcije.

SIGNATURE potpis *ED25519* privatnim ključem koji odgovara javnom ključu dostavljenom prilikom registracije na *mailbox* server.

Mailbox odgovara sa *ACK ONION* porukom.

3.1.14. Tip poruke: Mailbox del messages (0x8A)

Nakon što je klijent uspješno obradio i pohranio poruke koje je pribavio sa *mailbox* servera, ovim tipom poruke javlja serveru da obriše poruke prema ID-u svake od poruka.

TRANSACTION ID	16
MAILBOX ID	16
IDS LEN	2
MESSAGE IDS	VAR
SIGNATURE	64

Dijagram 3.12 – prikazuje sadržaj tijela Mailbox del messages poruke

MAILBOX ID jedinstveni broj pretinca klijenta.

TRANSACTION ID generirani ID trenutne transakcije.

IDS LEN broj poruka koje brišemo, ukoliko je postavljen na 0, server ne briše ništa, ali svejedno vraća uspješan odgovor.

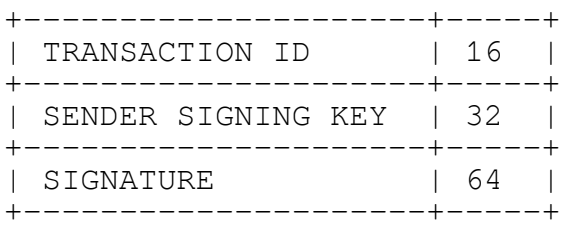
MESSAGE IDS lista blokova od 16 bajta, gdje je svaki blok *message id* poruke koju treba obrisati, duljina liste jednaka je vrijednosti u *IDS LEN* polju.

SIGNATURE potpis *ED25519* privatnim ključem koji odgovara javnom ključu dostavljenom prilikom registracije na *mailbox* server.

Mailbox odgovara sa *ACK ONION* porukom.

3.1.15. Tip poruke: Client fetch (0x8B)

Pošalji upit danom klijentu za listu poruka koje ima za tebe, ukoliko je upit ispravan klijent odgovara sa listom svih poruka za koje nije dobio obavijest da su zaprimljene (*RECV*).



Dijagram 3.13 – prikazuje sadržaj tijela *Client fetch* poruke

SENDER SIGNING KEY javni ključ tipa *ED25519*, generiran za vrijeme zahtjeva za prijateljstvo. To je ključ koji je klijent poslao primatelju kako bi mogao provjeriti poslane poruke.

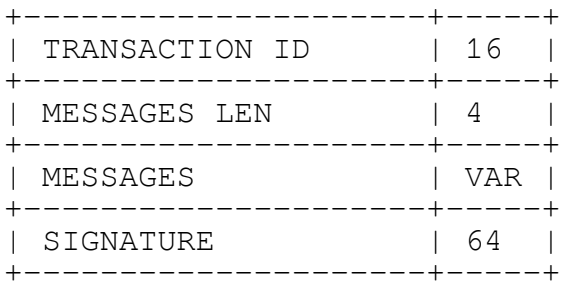
TRANSACTION ID predstavlja *UUID* trenutne transakcije.

SIGNATURE potpis, potpisan privatnim ključem koji odgovara ključu u polju *SENDER SIGNING KEY*.

Kao odgovor na ovu poruku klijent šalje *MESSAGE LIST* poruku. Ukoliko ne postoji niti jedna ne dostavljena poruka, klijent odgovara sa praznom listom.

3.1.16. Tip poruke: Message list (0x8C)

Šalje se kao odgovor na *FETCH* poruke. Koristi se kako bi dostavili listu *MESSAGE CONTAINER* poruka.



Dijagram 3.14 – prikazuje sadržaj tijela *Message list* poruke

MESSAGES LEN duljina *MESSAGES* polja u bajtovima.

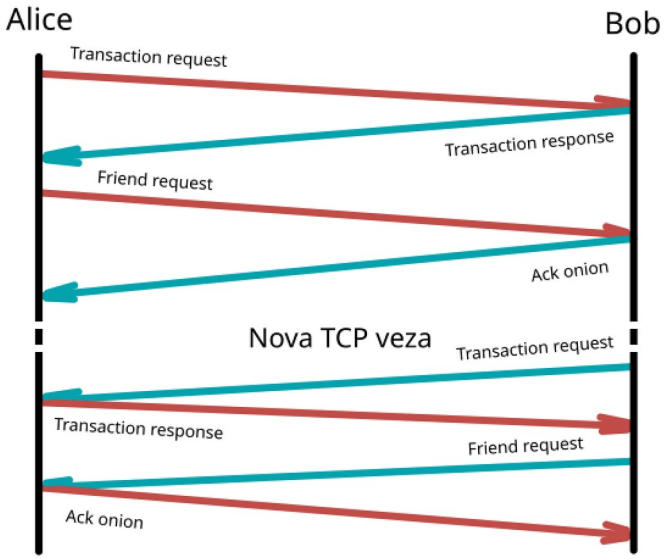
MESSAGES niz *MESSAGE CONTAINER* poruka. S time da se šalju cijele poruke uključujući *VER* i *MESSAGE TYPE* polja. Klijent koji zaprimi poruke, dužan je napraviti validaciju za svaku od poruka i odbaciti sve neispravne poruke, te poslati serveru zahtjev za brisanje.

SIGNATURE potpis privatnim ključem *ED25519*, ukoliko ovaj odgovor šalje *mailbox* server, ovo je potpis ključem *onion* domene, u suprotnom radi se o ključu čiji je javni dio poslan prilikom zahtjeva za prijateljstvo.

3.2. Primjeri komunikacije

U nastavku slijedi nekoliko primjera komunikacije korištenjem Deep Messenger protokola, nisu navedeni primjeri za sve tipove poruka s obzirom da većina poruka slijedi vrlo sličan uzorak.

3.2.1. Zahtjev za prijateljstvo



Dijagram 3.15 – proces uspostavljanja prijateljstva

Kao što je već spomenuto, kako bi dva klijenta mogla komunicirati odnosno izmjenjivati poruke prvo moraju uspješno uspostaviti prijateljstvo jer u tom procesu izmjenjuju sve ključeve koji su im potrebni za danju komunikaciju. Iz dijagrama iznad vidljivo je da se taj proces izvodi u 4 koraka.

Recimo da Alice želi Bobu poslati zahtjev za prijateljstvo. Alice će koristeći Tor klijenta otvoriti TCP vezu prema Bobovom skrivenom servisu i zatim mu poslati *Transaction request* poruku kako bi započela transakciju. Kada Bob zaprimi poruku generirati će nasumični ID transakcije i poslati ga Alice u *Transaction response* poruci. Od trenutka kada Alice zaprimi Bobovu poruku generirani ID transakcije postaje važeći za tu vezu i mora biti naveden u svim porukama koje slijede ako zahtijevaju ID transakcije.

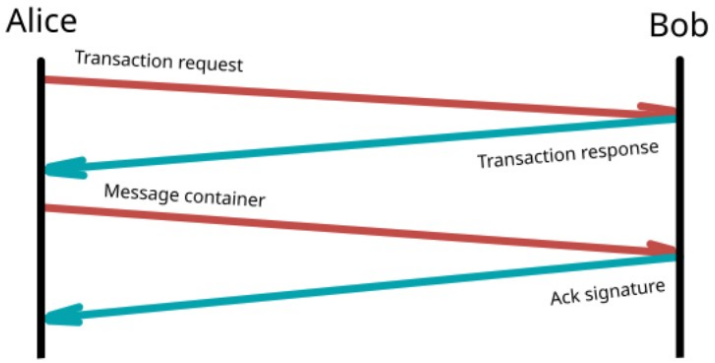
Alice sada može generirati potrebne parove ključeva i poslati njihove javne dijelove i ostale informacije o sebi u *Friend request* poruci. Kada Bob zaprimi *Friend request*, on će pohraniti sve podatke o Alice i njezine javne ključeve te će odgovoriti *Ack onion* porukom. Veza će tada biti zatvorena.

Kada Bob odluči prihvatiti zahtjev za prijateljstvo koji mu je Alice poslala, otvorit će vezu prema Aliceinom skrivenom servisu i kao što je i Alice ranije započeti transakciju.

Bob će zatim generirati potrebne parove ključeva i poslati njih i svoje podatke Alice pomoću *Friend request* poruke. Kada Alice zaprimi poruku, ona će pohraniti Bobove podatke koji joj nedostaju i njegove ključeve te će odgovoriti Bobu *Ack onion* porukom.

Ovim korakom prijateljstvo je uspješno uspostavljeno i klijenti sada mogu izmjenjivati tekstualne poruke.

3.2.2. Slanje poruka direktno



Dijagram 3.16 – slanje poruke

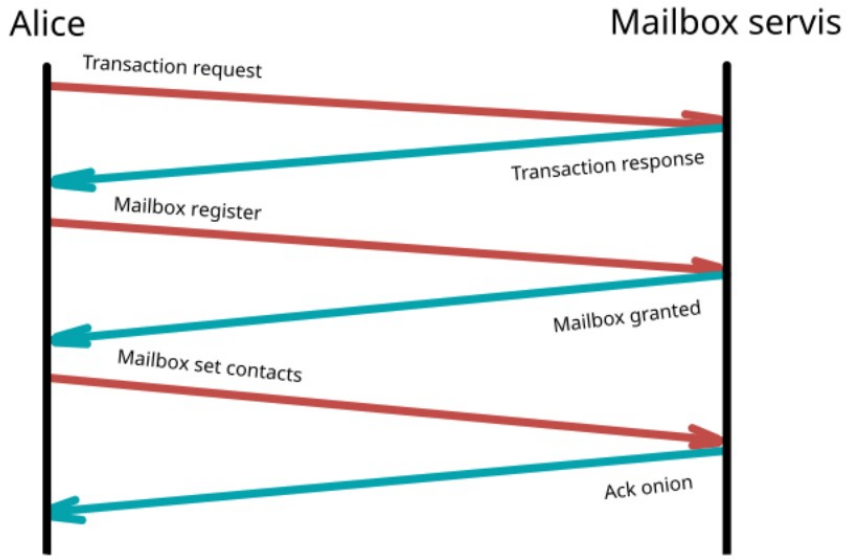
Slanje poruka je vrlo jednostavan proces, kako bi klijent mogao poslati poruku prvo mora započeti transakciju i zatim poslati *Message container* poruku. U primjeru iznad Alice šalje poruku Bobu, prvo započinje transakciju na isti način kao i u prethodnom primjeru. Zatim šalje Bobu *Message container* poruku koju je enkriptirala javnim *RSA* ključem koji je primila od Boba i potpisala privatnim *ED25519* ključem koji odgovara javnom ključu koji je Bob zaprimio tijekom uspostave prijateljstva.

Kada Bob zaprimi poruku on odgovara Ack signature porukom. Kasnije Bob će Alice poslati novu poruku *Message container* sadržaja tipa *RECV*. Kako bi Alice znala da je on sigurno zaprimio poruku.

Na isti način funkcionira i slanje poruka na *mailbox* servis, samo što *mailbox* servis umjesto *Ack signature* poruke šalje *Ack onion.*

3.2.3. Registracija na mailbox servis

Kako bi klijent mogao primati poruke i kada nije na mreži mora se registrirati na jedan *mailbox* servis.



Dijagram 3.17 – registracija i konfiguracija mailbox servisa

Dijagram iznad prikazuje kako se Alice registrira na svoj *mailbox* servis i konfigurira od kojih kontakata želi primati poruke.

Alice prvo kao i u prethodnim primjerima uspostavlja transakciju te nakon što je to učinila šalje *Mailbox register* poruku koja sadrži pristupni ključ koji je dobila od vlasnika *mailbox* servisa i javni ključ koji će Alice u budućnosti koristiti za autentikaciju sa servisom.

Kada servis zaprimi njezin zahtjev provjerit će ispravnost pristupnog ključa, ukoliko je ključ ispravan odgovorit će Alice sa *Mailbox granted* porukom. U toj poruci Alice će dobiti ID svojeg pretinca koji ju razlikuje od drugih korisnika na servisu.

Kako bi servis mogao primati poruke od njenih kontakata Alice mu mora dostaviti listu svojih kontakata, to čini slanjem *Mailbox set contacts* poruke na što servis odgovara sa *Ack onion*. Sve ostale konfiguracijske poruke za *mailbox* servis funkcioniraju na jednaki način.

3.3. Vlastite implementacije pomoćnih algoritama

Kako bi funkcionirao Deep Messenger zahtjeva set relativno specifičnih pomoćnih funkcija koje se ne nalaze u standardnim bibliotekama, u nastavku slijedi lista algoritama i protokola koji su implementirani u sklopu Deep Messenger projekta.

3.3.1. SOCKS 5 klijent

Vrlo jednostavna implementacija SOCKS 5 protokola prema dokumentu RFC 1928 [6]. Ne sadrži većinu naprednijih funkcionalnosti protokola i posebno je prilagođena za povezivanje na skrivene servise pomoću Tor klijenta pokrenutog lokalno na računalu.

3.3.2. Base32 enkoder i dekoder

Koristi se za enkodiranje i dekodiranje *onion* adresa implementiran prema dokumentu RFC 4648 [10]. Inspiriran je otvorenom implementacijom dostupnom u sljedećem *Github* repozitoriju:

https://github.com/mjg59/tpmtotp/blob/master/base32.c
https://github.com/mjg59/tpmtotp/blob/master/base32.h

3.3.3. Implementacija generičkog reda (queue)

Ova implementacija koristi se u glavnom mehanizmu za obradu poruka koje pristižu i odlaze u Deep Messenger aplikaciji. Dizajnirana je kako bi mogla funkcionirati sa bilo kojim tipovima podataka dostupnim u C programskom jeziku. Inspirirana je otvorenom implementacijom dostupnom u sljedećem *Github* repozitoriju:

https://github.com/kostakis/Generic-Queue

3.3.4. Pomoćne funkcije za upravljanje onion adresama

Deep Messenger sadrži set funkcija koje se koriste za provjeru, dekodiranje i enkodiranje *onion* adresa te pretvorbu *ED25519* ključeva u oblik pogodan za pokretanje skrivenih servisa. Podaci potrebni za implementaciju iščitani su iz Tor specifikacije i nekih dodatnih izvora. [9]

3.3.5. Ostale pomoćne funkcije

U aplikaciji postoje još neke pomoćne funkcije, poput *debug()* naredbe koja se posebice koristila tijekom razvoja u svrhu otklanjanja pogrešaka. Ona nam omogućuje da slično kao sa *printf* funkcijom ispisujemo podatke na standardni izlaz ili u datoteku dnevničkog zapisa. Aplikacija sadrži i set funkcija za potpisivanje i enkriptiranje sadržaja *libevent evbuffer* struktura s obzirom da su to operacije koje su jednake za većinu tipova poruka definiranih u protokolu.

3.4. Glavna logika Deep Messengera

Sav kod Deep Messenger aplikacije možemo podijeliti na tri dijela. Već spomenute pomoćne funkcije, korisničko sučelje napisano pomoću *ncurses* biblioteke i stražnjeg dijela koji obrađuje sav mrežni promet (Protocol handler). Oni su međusobno povezani pomoću sustava *callback* funkcija, a

za pohranu podataka koriste *SQLite* bazu podataka. Za svaku od poruka definiran je poseban objekt sa setom *callback* funkcija koji obrađuje pristigle podatke ili postavlja poruku u izlazni *buffer*.

Sav bitan kod je komentiran i dokumentiran, te se čitatelj na njega može referencirati putem poveznice u prilogu. U nastavku slijedi opis pojedinih važnih dijelova aplikacije.

3.4.1. Glavni upravitelj poruka (Main protocol handler)

Upravitelj poruka glavni je dio Deep Messenger aplikacije, za svaku otvorenu vezu alociran je po jedan upravitelj poruka koji ovisno o tipu poruke i operaciji (slanje ili primanje) procesuira mrežni promet. Upravitelj se sastoji od dva reda (eng. queue), jedan za slanje i jedan za primanje poruka. Uz to pruža mogućnost postavljanja *callback* funkcija koje će biti pozvane u slučaju prekida veze i drugih promjena.

Za svaki gore navedeni tip poruke implementirana je klasa (struktura sa setom funkcija) koja sadrži funkcije potrebne za slanje i primanje tog tipa poruke. Stoga je moguće kreirati objekt za svaki od tipova poruka i samo ga ubaciti u red glavnog upravitelja poruka. Način izvedbe je sljedeći:

Definirane su dvije strukture *prot_recv_handler*, koja se koristi za primanje i *prot_tran_handler*, koja se koristi za slanje poruka. Definicija struktura je sljedeća:

```
struct prot_recv_handler {
    enum prot_message_codes msg_code;
    void *msg;
    int require_transaction;
    prot_recv_handle_cb handle_cb;
    prot_recv_cleanup_cb cleanup_cb;
};

struct prot_tran_handler {
    enum prot_message_codes msg_code;
    void *msg;
    struct evbuffer *buffer;
    prot_tran_done_cb done_cb;
    prot_tran_setup_cb setup_cb;
    prot_tran_cleanup_cb cleanup_cb;
};
```

Obje strukture sadrže kod tipa poruke i pokazivač na objekt tipa poruke. Osim toga obje strukture sadrže i pokazivač na *cleanup* funkciju koja se poziva nakon što je operacija slanja/primanja završena kako bi se oslobodila memorija koju zauzima objekt tipa poruke. Struktura za primanje također sadrži i člana *require_transaction* iz kojeg glavni upravitelj poruka može iščitati mora li transakcija biti započeta kako bi bilo u redu primiti ovaj tip poruke. Dakle pomoću podataka iz ovih struktura upravitelj poruka može provesti određene provjere i ukoliko poruka nije ispravna odbaciti ju i bez pozivanja funkcija specifičnih za tip poruke.

Kod primanja poruka nakon što je provjera poruke bila uspješna, upravitelj će provjeriti red sa tipovima koji pristižu, ukoliko red nije prazan upravitelj će provjeriti je li pristigla poruka odgovarajućeg tipa i ako nije zatvorit će vezu. S druge strane ako je red prazan upravitelj će pokušati samostalno alocirati objekt za pristigli tip poruke pomoću *autogen* funkcije.

Nakon što je uspješno dobavio objekt poruke iz reda ili ga generirao, pozvat će *handle* funkciju koja procesuira podatke iz ulaznog *buffera* za taj tip poruke. Funkcija *handle* može biti pozvana više puta ukoliko svi podaci nisu pristigli prvi puta. Nakon što je funkcija handle iz buffera izvadila sve što je trebala, promijenit će jednu varijablu upravitelja i tako mu dati do znanja da je završila. Upravitelj će tada pozvati *cleanup* i nastaviti sa sljedećim porukama.

Slanje poruka je nešto jednostavnije, nakon što ubacimo poruku za slanje u red, upravitelj će ako je slanje omogućeno pozvati *setup* funkciju iz strukture, ta će funkcija preoblikovati poruku u oblik za slanje kroz mrežu i postaviti ju u privremeni *buffer*. Upravitelj će zatim iz privremenog *buffera* poruku kopirati u izlazni *buffer*.

Nakon što je poruka uspješno poslana (izlazni buffer postane prazan), upravitelj će pozvati *done* funkciju kako bi obavijestio objekt poruke da je uspješno poslan i zatim *cleanup* funkciju kako bi oslobodio memoriju.

3.4.2. Modeli baze podataka

Kao što je već spomenuto Deep Messenger koristi SQLite biblioteku za pohranu podataka u bazi podataka. Kako bi pojednostavili proces spremanja i čitanja podataka, za svaku od tablica u bazi podataka definiran je model. Model se sastoji od strukture i seta funkcija za vršenje operacija nad strukturom. Struktura sadrži sve atribute koji se nalaze i u odgovarajućoj tablici. Za svaku strukturu definirane su funkcije za osnovne operacije nad podacima kao što su: *save* za pohranu podataka, *refresh* za dobavljanje promjena iz baze, *delete* za brisanje zapisa iz baze, funkcije new i free za alociranje i oslobađanje strukture te serija *get* funkcija koje dobavljaju zapis prema različitim kriterijima.

Na ovaj način interakcija sa bazom podataka uvelike je pojednostavljena iz ostalih dijelova aplikacije.

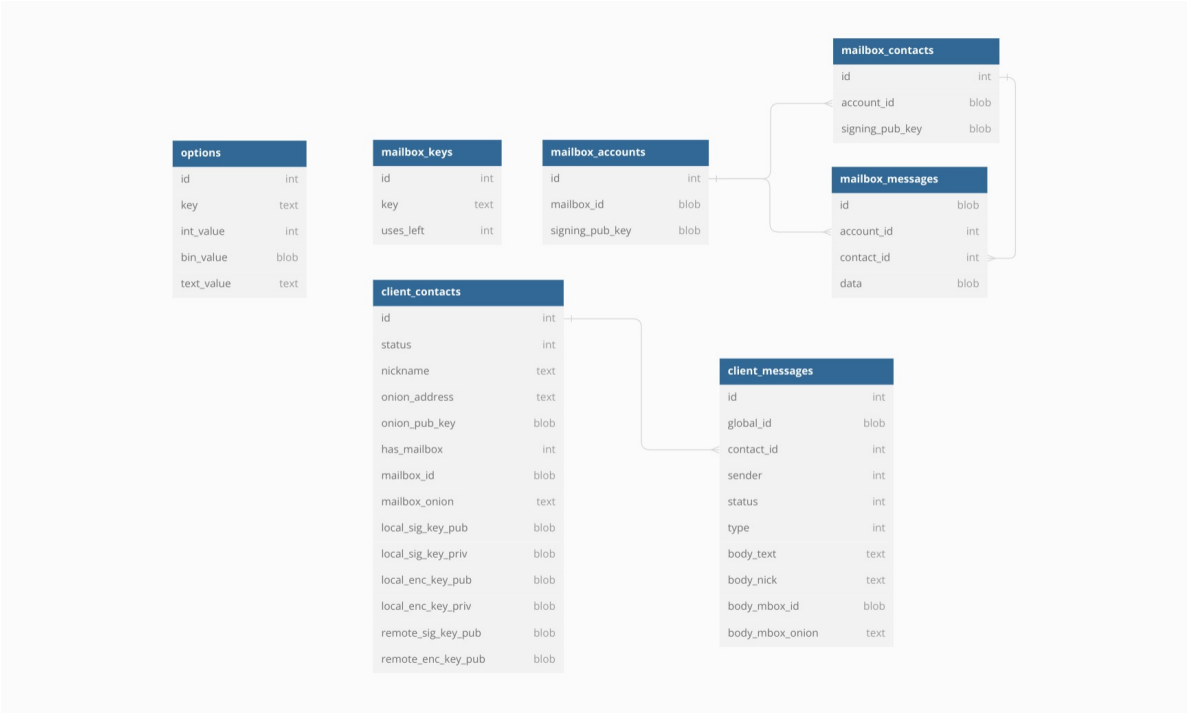
3.4.3. Korisničko sučelje

Korisničko sučelje dizajnirano je kao apstrakcija na biblioteku *NCurses*. Osnovni element korisničkog sučelja je prozor, odnosno struktura *ui_window* sa svojim setom funkcija. Sve ostale komponente sučelja se povezuju na instance strukture *ui_window*. Pomoću njenih funkcija svakoj

strukturi prozora moguće je nacrtati obrub, postaviti naslov pozicioniran lijevo, desno ili u sredini, pozicionirati ju na ekranu i odrediti njezine dimenzije. Kako bi prozor prikazivao nešto osim naslova potrebno je na njega zakačiti neku od komponenata, koja će onda unutar njega nešto ispisati koristeći *NCurses* funkcije. Neke od komponenata koje postoje u Deep Messenger aplikaciji su, *ui_manager* koji se koristi za selektiranje različitih prozora na zaslonu, *ui_logger* koja se koristi za ispis teksta u prozor uz mogućnost *scrollanja*, *ui_prompt* koja se koristi kao tekstualno polje za unos poruke i neke druge.

3.5. Dizajn baze podataka

Deep Messenger aplikacija sve svoje podatke pohranjuje u SQLite bazi podataka, raspoređene u nekoliko povezanih tablica. ER dijagram ispod prokazuje dizajn baze podataka kakav aplikacija koristi.



Dijagram 3.18 – ER dijagram baze podataka

Zbog jednostavnije izvedbe istu bazu podataka moguće je koristiti i za Deep Messenger klijenta i za *mailbox* servis. Ovisno u kojem je modu aplikacija pokrenuta ignorirati će one tablice koje joj nisu potrebne.

4. Zaključak

Rad na projektu Deep Messenger bio je vrlo zanimljiv i poučan, istražio sam više područja mrežnog programiranja i rada u C programskom jeziku sa kojima nikada ranije nisam radio. Koristio sam ukupno 4 poznate C biblioteke, O*penSSL* za kriptografske operacije, *libevent* za mrežnu komunikaciju, *SQLite* za pohranu podataka i *NCurses* za izradu korisničkog sučelja. Osmislio sam vlastiti binarni mrežni protokol aplikacijskog sloja, koji se koristi za komunikaciju među klijentima i sa *mailbox* servisom. Posebno izazovno bilo je korištenje kriptografskih funkcija *OpenSSL* biblioteke zbog vrlo velike količine dostupnih funkcija. Nadam se da ću u budućnosti imati priliku raditi i na drugim projektima iz ovog područja.

5. Literatura

1. Tor Project History
   URL: https://www.torproject.org/about/history/, (pristupljeno: 30.04.2024.)
2. How do Onion Services work
   URL: https://community.torproject.org/onion-services/overview/, (pristupljeno: 30.04.2024.)
3. Encoding onion addresses [ONIONADDRESS]
   URL: https://spec.torproject.org/rend-spec/encoding-onion-addresses.html,
   (pristupljeno: 30.04.2024.)
4. Službena OpenSSL dokumentacija
   URL: https://www.openssl.org/docs/, (pristupljeno: 30.04.2024.)
5. Nick Mathewson, Fast portable non-blocking network programming with Libevent,
   URL: https://libevent.org/libevent-book/, (pristupljeno: 30.04.2024.)
6. RFC1928 – SOCKS Protocol Verison 5
   URL: https://datatracker.ietf.org/doc/html/rfc1928, (pristupljeno: 30.04.2024.)
7. git --distributed-is-the-new-centralized
   URL: https://git-scm.com/, (pristupljeno: 30.04.2024.)
8. Hidden services: overview and preliminaries
   URL: https://spec.torproject.org/rend-spec/overview.html, (pristupljeno: 30.04.2024.)
9. Korisnik NULL, How to get the hostname from hs_ed25519_secret_key
   URL: https://tor.stackexchange.com/questions/23965/how-to-get-the-hostname-from-hs-ed25519-secret-key, (pristupljeno: 30.04.2024.)
10. RFC4648 – Base16, Base32 and Base64 Data Encodings, Sekcija 6 Base32 Encoding
    URL: https://datatracker.ietf.org/doc/html/rfc4648#section-6 (pristupljeno: 30.04.2024.)
11. SQLite Documentation, Programming Interfasce
    URL: https://www.sqlite.org/docs.html (pristupljeno: 30.04.2023.)
12. Dan Gookin, Programmer's Guide to NCurses, Indianapolis: Wiley Publishing Inc. 2007.

6. Prilozi

Sav kod ovog projekta uključujući i ovu dokumentaciju licenciran je MIT licencom otvorenog koda i dostupan je u *Github* repozitoriju putem poveznice https://github.com/rdobovic/deep-messenger.