Izdelava programa za kriptografsko zaščiteno izmenjavo sporočil

Maturitetna seminarska naloga pri informatiki

Avtor: Nikola Brković
Mentor: Andrej Šuštaršič
Šola: Gimnazija Bežigrad
Kraj in datum: Ljubljana, 14. april 2022

1 Kazala

1.1 Kazalo vsebine

1.2 Kazalo slik

2 Povzetek in ključne besede

Ta maturitetna naloga razlaga najbolj aktualne vrste šifriranja in mehanizme njihovega delovanja ter opisuje proces izdelave odprtokodne aplikacije za šifrirano komunikacijo. Naloga obravnava programsko opremo in znanja na področju kriptografije, ki so bila potrebna za izdelavo, ter opisuje pomanjkljivosti in možne izboljšave aplikacije. Aplikacija je sestavljena iz treh delov: komponente za šifriranje, komponente za omrežno komunikacijo in grafičnega vmesnika. Napisana je v jeziku C++.

2.1 Ključne besede

šifriranje, kriptografija, kriptografska zaščita, hipno sporočanje, izmenjava ključev

3 Uvod

Kriptografija danes predstavlja eno od ključnih področij računalništva. Prve primitivne oblike šifriranja so uporabljali že stari Rimljani, a se je v zadnjih 50 letih zgodil bliskovit razvoj novih kriptografskih tehnologij, ki se v uporabi tudi danes. Napredne kriptografske tehnologije danes niso več v uporabi le na področjih bančništva, vojske in državne varnosti, temveč tudi v osebni komunikaciji in za zaščito osebnih podatkov. V zadnjem desetletju je nastalo več tehnologij, protokolov in spletnih platform, ki uporabnikom omogočajo kriptografsko zaščiteno hipno sporočanje oziroma pošiljanje sporočil, kot so WhatsApp, Tox, Signal, OMEMO, Protonmail, Matrix in Session.

Glavni problem teh aplikacij je, kako ustvariti uporabnikom prijazen program oziroma protokol, tudi tistim, ki principov delovanja kriptografije oziroma šifriranja ne razumejo, ki hkrati zagotavlja varnost, celovitost in preverljivost podatkov oziroma sporočil. V današnjem času so morebitne grožnje naši varnosti in zasebnosti zaradi vseprisotnosti tehnologije v vsakdanjem življenju vedno večje, kar terja razvoj vse bolj naprednih tehnologij za šifriranje sporočil.

Cilj izdelave mojega programa je bil izdelati aplikacijo, ki bo uporabljala iste sodobne kriptografske tehnologije in pristope kot bolj napredne aplikacije, kakršen je Signal, a hkrati uporabljala preproste in zanesljive protokole za komunikacijo, kot je TCP. Glavni vzor pri izdelavi moje aplikacije je bil protokol Tox, iz več razlogov.

1. Tox je zgolj splošen protokol, ki ga lahko uporabljajo različne aplikacije. Zaradi tega so Tox in vse aplikacije, ki uporabljajo ta protokol popolnoma odprtokodne, uporablja licenco *GNU General Public License 3*.

2. Tako kot sam protokol, tudi kriptografska knjižnica, ki jo Tox uporablja, *NaCl* (Networking and Cryptography Library) ali *libsodium*, je popolnoma prosta za uporabo pod odprtokodno licenco ISC.

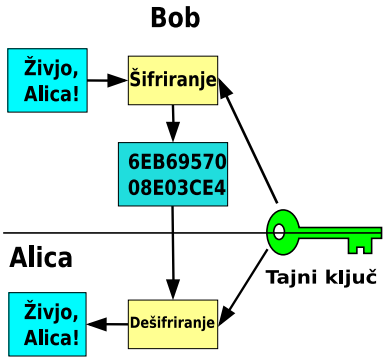
3. Knjižnica *libsodium* je preprosta, lahka, ima zelo jasno in podrobno dokumentacijo ter je napisana v jeziku C, zaradi česar sem se odločil, da bom to knjižnico uporabil tudi za mojo aplikacijo.

Za izdelavo moje aplikacije sem izbral programski jezik C++. Ta jezik sem izbral, ker je knjižnica *libsodium* napisana v jeziku C in ga tako lahko uporabim tudi v C++-ju. Nisem se pa odločil za C, ker jezik C++ ima večji izbor knjižnic za izdelavo grafičnega vmesnika in omrežne komunikacije preko TCP. Prednost jezika C++ v primerjavi s C-jem je tudi to, da je objektno orientiran. Za omrežno komunikacijo sem izbral TCP, saj je zelo zanesljiv

in preprost protokol za izmenjavo aplikacij in večina operacijskih sistemov omogoča izdelavo programov, ki uporabljajo TCP, brez dodatnih knjižnic, s pomočjo omrežnih vtičnic (ang. network socket). Knjižnica *libsodium* omogoča več različnih vrst šifriranja, tako simetričnega kot tudi asimetričnega, ampak sem za svojo nalogo izbral šifriranje s pomočjo standardizirane krivulje *Curve25519* in funkcije *X25519* za izmenjavo ključev po načinu Diffie-Hellman, ki predstavlja eden od najbolj uveljavljenih načinov šifriranja v sodobnem svetu ter bločno šifro *XSalsa20* za simetrično šifriranje sporočil.

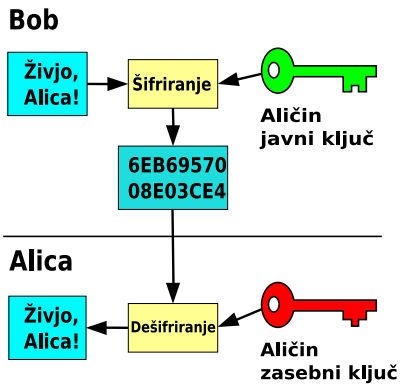
4 Šifriranje

Šifriranje delimo na simetrično in asimetrično (z javnim ključem) šifriranje. Glavna razlika je v tem, da pri simetričnem šifriranju uporabljamo isti ključ za šifriranje in dešifriranje sporočila. Pošiljatelj sporočilo šifrira s skupnim ključem in prejemnik ga dešifrira s ta istim skupnim ključem.



Slika 1: Shema simetričnega šifriranja

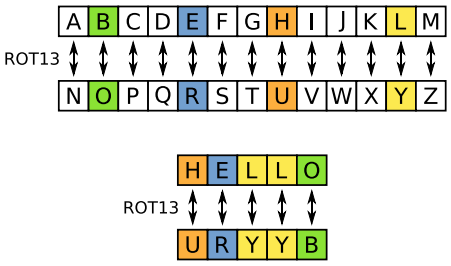
Pošiljatelj in prejemnik uporabljata skupen ključ. Po drugi strani, pri asimetričnem šifriranju uporabljamo par ključev - javnega in zasebnega. Zato to vrsto šifriranja imenujemo tudi šifriranje z javnim ključem (ang. *public-key encryption*). Za šifriranje uporabimo javni ključ, za dešifriranje pa zasebni. Pošiljatelj sporočilo šifrira z javnim ključem prejemnika in ga prejemnik dešifrira s svojim zasebnim ključem.



Slika 2: Shema asimetričnega šifriranja

4.1 Simetrično šifriranje

Simetrično šifriranje je preprostejša oblika šifriranja in je nastala pred asimetričnim. Do sedemdesetih let prejšnjega stoletja smo poznali le simetrično šifriranje. Za prvo obliko simetričnega šifriranja veljajo t. i. zamenjalna šifre. Najbolj znan primer zamenjalnih šifer je Cezarjeva šifra. Cilj zamenjalnih šifer je šifriranje besedila. Princip delovanja zamenjalnih šifer je zelo preprost - vsako črko v abecedi nadomestimo z neko drugo črko. Zamenjalne šifre so bile zgodovinsko dolgo v uporabi. Zaradi tega so s časom nastale tudi bolj napredne oblike zamenjalnih šifer, kot je Vigenèreva šifra. Vigenèreva šifra je ena od mnogoabecednih šifer, ki za zamenjavo črk uporabljajo več različnih abeced. Zamenjalne šifre danes veljajo za zelo nevarne, saj napadalec zelo lahko dešifrira sporočila s pomočjo analize pogostosti črk in dolžin besed (ang. *letter frequency analysis*). Zato so danes zgolj v uporabi za izobraževalne namene kot preprosta oblika kriptografije in njihova uporaba za šifriranje dejanskih podatkov je odsvetovana.



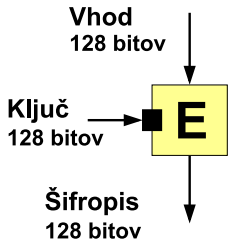
Slika 3: Prikaz zamenjalne šifre ROT13, variacije na Cezarjevo šifro

Bolj aktualni vrsti simetričnega šifriranja sta tekoča in bločna šifra. Tekoča šifra šifrira vsak posamezen bit. To je doseženo tako, da na vhodni bit (oz. bit iz podatkov, ki jih želimo šifrirati) dodamo bit iz toka ključa (ang. *key stream*). Govorimo o sinhronih in asinhronih tekočih šifrah. Razlika je v tem, da pri asinhronih tekočih šifrah šifropis vpliva tudi na tok ključa. Ena od najbolj znanih in najpreprostejših oblik tekočih šifer je šifra izključno ALI (ang. *XOR cipher*). Pri tej šifri podatke šifriramo tako, da opravimo bitno operacijo izključno ALI med vhodnimi podatki in tokom ključa. Rezultat te operacije predstavlja naš šifropis. Če šifropis želimo dešifrirati, operacijo izključno ALI opravimo na šifropisu in dobimo nazaj prvotne podatke. Ključnega pomena pri varnosti šifre izključno ALI je naš tok ključa. Za tok ključa lahko uporabimo kriptografsko varen generator psevdonaključnih števil. Šifropis, ki nastane z uporabo takšnega generatorja imenujemo tudi enkratna prevleka (ang. *one-time pad*) in je v teoriji nezlomljiv, če je ključ daljši kot vhodni podatki. V resnici pa je tak pristop nepraktičen in ne prav

zelo pogost, saj bi pošiljatelj in prejemnik morala na varen način deliti tajni ključ, kar ni vedno možno.

Bolj praktična tekoča šifra, ki je prišla tudi v splošno uporabo je šifra Salsa oziroma ChaCha. ChaCha je veliko bolj zapletena metoda šifriranja kot šifra izključno ALI, ampak uporablja pa zgolj 3 različne operacije: seštevanje, rotacijo v levo in izključno ALI, zaradi česar je računsko poceni. Zato šifriranje s ChaCha tudi vedno traja enako procesorskih ciklov, kar jo dela odporno proti časovnim napadom (ang. *timing attack*). Šifra ChaCha za šifriranje uporablja 128-bitno konstanto, 256-biten ključ, 32-bitni števec in 96-bitno enkratno število (ang. *nonce*), kar skupno sestavlja en 512-biten blok. Najbolj aktualna shema za šifriranje s šifro ChaCha je *ChaCha20-Poly1305*, ki opravi 20 krogov šifriranja s šifro ChaCha in zapiše z overitveno oznako Poly1305. Sorodno šifro *XSalsa20-Poly1305* uporabljam tudi v moji nalogi za simetrično šifriranje.

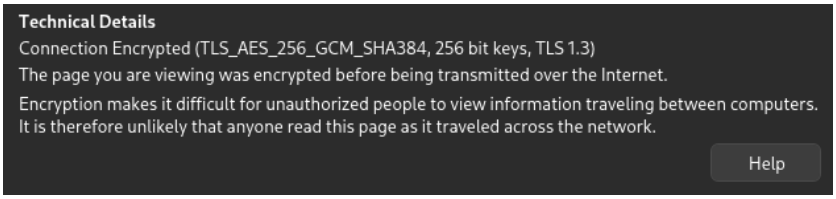
Za razliko od tekočih šifer, pri bločnih šifrah šifriramo en celoten blok bitov hkrati. V teoriji je lahko dolžina bloka poljubna, ampak se je v praksi uveljavila dolžina od 128, 192 ali 256 bitov.



Slika 4: Diagram delovanja bločne šifre

Daleč najbolj uveljavljena bločna šifra je šifra AES (*Advanced Encryption Standard* ali tudi *Rijndael)*. AES je postal zelo priljubljen, saj je del TLS (*Transport Layer Security*) protokola, katerega HTTPS, IEEE 802.11i standard za šifriranje Wi-Fi omrežij in protokol SSH (*Secure Shell*) za oddaljen pristop uporabljajo za šifriranje. Torej, skoraj vedno ko se povežemo na spletno stran, ki podpira HTTPS, za šifriranje podatkov uporabljamo prav AES.

AES je bolj zapleten in računsko zahteven način šifriranja. Za razliko od starejših bločnih šifer, AES šifrira celoten blok v eni iteraciji. En krog šifriranja z AES poteka v treh slojih: seštevanje ključa, zamenjava bitov (S-Box) in difuzija. Seštevanje ključa deluje podobno kot šifriranje s šifro izključno ALI. Trenutno ne obstajajo nobeni znani učinkoviti napadi proti AES-u. Če želimo šifrirati več kot 256 bitov, seveda moramo naše podatke



Slika 5: Prikaz uporabe šifre AES v brskalniku Firefox

na nek način razdeliti v manjše bloke. Za to obstaja več načinov, a najbolj priljubljeni so ECB (*Electonic Code Book*), CFB (*Cipher Feedback*), ki se obnaša kot tekoča šifra in GCM (*Galois Counter Mode*). Način GCM ima nekaj prednosti pred prvima dvema. Pri načinih ECB in CFB mora dolžina vhodnih podatkov biti večkratnik dolžina bloka, v primeru AES večkratnik 16 bajtov. Poleg tega, GCM vsebuje tudi preverjeno šifriranje oziroma izračuna overitveno oznako.

4.2 Asimetrično šifriranje

Asimetrično šifriranje je bilo izumljeno šele leta 1976. Whitfield Diffie, Martin Hellman in Ralph Merkle so tega leta izumili prvi sistem šifriranja z javnim ključem. Njihov sistem je temeljil na preprosti ugotovitvi: Ni potrebno, da je ključ osebe, ki sporočilo šifrira (v našem primeru Alica), tajen. Ključni del je ta, da prejemnik (Bob) lahko sporočilo dešifrira s tajnim/zasebnim ključem. Za uresničitev takega sistema Bob mora posedovati javni ključ, ki je javno dostopen vsem in ujemajoč se tajni ključ. Takšen sistem omogoča tudi komunikacijo s simetrično šifro. Recimo, da Alica ima simetrični ključ. Alica lahko ta ključ šifrira, z uporabo algoritma z javnim ključem in ga pošlje Bobu. Vsi sistemi šifriranja z javnim ključem matematično temeljijo na skupnem načelu: enosmernih funkcijah. Enosmerna funkcija je taka funkcija, za katero je $y = f(x)$ računsko poceni, inverzna funkcija $x = f^{-1}(y)$ pa izjemno računsko draga. Glede na to, katero vrsto enosmerne funkcije uporabljajo, algoritme šifriranja z javnim ključem delimo na tri vrste:

1. Prafaktorizacijske sheme: RSA
2. Logaritemske sheme: Izmenjava ključev po načinu Diffie-Hellman, DSA (Digital Signature Algorithm)
3. Sheme z eliptično krivuljo (EC): Elliptic Curve Diffie-Hellman (ECDH) in Elliptic Curve Digital Signature Algorithm (ECDSA)

Algoritmi za šifriranje z javnim ključem tudi omogočajo druge funkcije, kot je izmenjava ključev preko nezaščitenega kanala, predvsem z izmenjavo ključev po načinu Diffie-Hellman, in identifikacijo oziroma preverljivost sporočil z algoritmi za digitalne podpise, kot je DSA oziroma ECDSA. V primerjavi z simetričnim šifriranjem, so algoritmi za asimetrično šifriranje veliko počasnejši in računsko zahtevnejši in tudi zahtevajo daljše ključe. RSA in DSA imajo 1024 do 15360 bitov varnosti, ECDH pa ECDSA pa od 160 do 512 bitov.

Prva simetrična shema šifriranja je RSA. Osnovna enosmerna funkcija RSA je problem razcepa velikih števil: Množenje dveh velikih števil je računsko poceni, ampak razcep produkta teh dveh števil je pa izjemno težek. Zaradi tega v kriptografiji imajo praštevila izjemen pomen. Za število n, ki se pri RSA uporablja kot modul in predstavlja del javnega ključa je produkt dveh praštevil. Tukaj govorimo o zelo velikih številih, kajti število n mora biti dolgo vsaj 1024 bitov, kar pomeni da je dolžina teh dveh praštevil 512 bitov. RSA je bil v široki uporabi zelo dolgo časa, glede na to da je nastal leta 1977, ampak se je v zadnjih letih izkazalo, da je RSA lahko ranljiv na določene napade. Poleg tega, v prihodnosti bi RSA lahko kvantni računalniki z lahkoto v prihodnosti dešifrirali.

4.3 Šifriranje z eliptično krivuljo

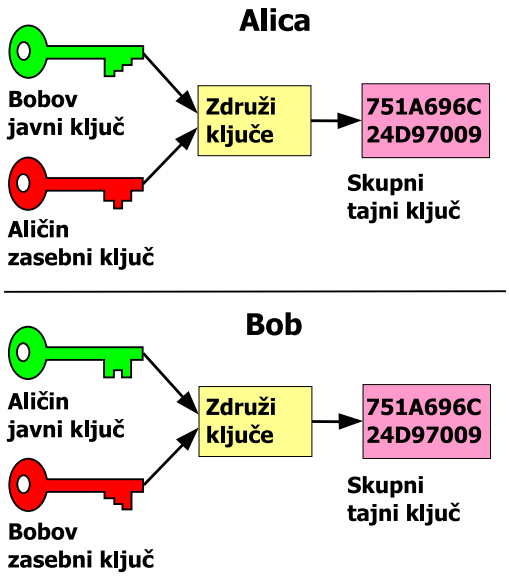
Šifriranje z eliptično krivuljo predstavlja najnovejšo obliko asimetričnega šifriranja. Šifriranje z eliptično krivuljo (v nadaljevanju ECC - Elliptic Curve Cryptography) ima primerljiv nivo ravnosti kot RSA ali logaritmske sheme (npr. DSA), ampak operira z veliko manjšimi številkami (160-256 bitov v primerjavi z 1024-3072 bitov). ECC temelji na logaritskem problemu na eliptični krivulji (ang. *Elliptic Curved Discrete Logarithm Problem - ECDLP*). Pri ECC za šifriranje podatkov uporabljamo skalarno množenje točke na krivulji, to pomeni, da neko točko na krivulji večkrat seštevamo samo s sabo. Na primer, skalarni produkt točke P in števila n je $nP = P + P + P + ... + P$. Logaritemski problem na eliptični krivulji temelji na tem, da je za neko točko Q na eliptični krivulji, ko $Q = nP$, zelo težko izračunati n, če sta Q in P zelo veliki vrednosti. Eliptična krivulja je pa krivulja oblike $y^2 = x^3 + ax + b$, ki je definirana na nekem omejenem praštevilskem polju.

Ena od najpogostejših eliptičnih krivulj je *Curve25519*, ki jo uporabljam tudi pri moji aplikaciji za izmenjavo ključev. Ta krivulja je definirana na praštevilskem polju definiranem po praštevilu $2^{55} - 19$, od koder prihaja ime. Ta krivulja je ena od najhitrejših algoritmov, ki uporablja eliptično krivuljo in je bila narejena prav za izmenjavo ključev po načinu Diffie-Hellman. Funkcija, ki uporablja *Curve25519* za izmenjavo ključev se imenuje *X25519*.

Curve25519 ponuja 128 bitov varnosti in uporablja ključe od 256 bitov.

4.4 Izmenjava ključev po načinu Diffie-Hellman z eliptično krivuljo

Izmenjava ključev je eden od najpomembnejših mehanizmov v kriptografiji, saj omogoča pošiljatelju in prejemniku, da ustvarita skupen tajni ključ prek nezaščitenega kanala. Pred izmenjavo ključev je za šifrirano komunikacijo s tajnim ključem med dvema osebama bilo potrebno ta ključ deliti prek »zanesljivega kurirja«. Izmenjava ključev po načinu Diffie-Hellman omogoča dvem osebam to, da ustvarita skupen tajni ključ brez predhodnega poznavanja ali komunikacije. Ta tajni ključ lahko pošiljatelj in prejemnik uporabljata s katero koli simetrično šifro, kot so AES in ChaCha.



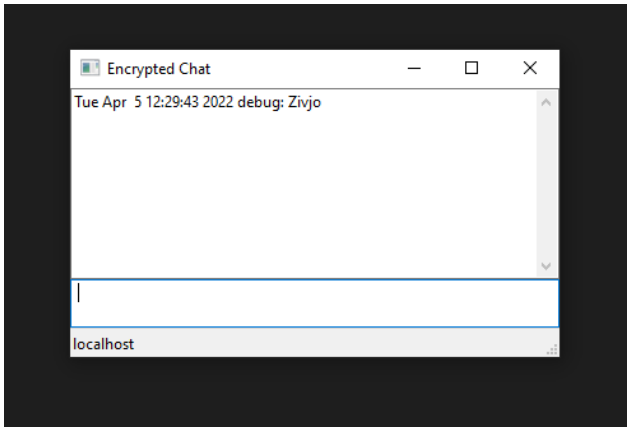
Slika 6: Prikaz delovanja izmenjave ključev

Pri izmenjavi ključev z eliptično krivuljo je pa ključno, da pošiljatelj in prejemnik uporabljata enako krivuljo.

5 Grafični vmesnik

Ključni del moje aplikacije je bil tudi grafični vmesnik. Danes obstaja veliko različnih knjižnic za izdelavo grafičnega vmesnika, ki podpirajo različne operacijske sisteme (Windows, MacOS X in Linux), vendar sem se jaz odločil za

knjižnico WXWidgets. Za to knjižnico sem se odločil, ker je zelo ustaljena, dobro dokumentirana in odprtokodna.



Slika 7: Zajem zaslona moje aplikacije

Knjižnica WXWidgets je narejena predvsem za jezik C++, podpira pa tudi druge jezike, kot Python. Zraven grafičnega vmesnika WXWidgets ponuja tudi veliko drugih možnosti, vključno s komponento za omrežno komunikacijo preko omrežnih vtičnic. WXWidgets ma pa to pomanjkljivost, da v primerjavi z nekimi drugimi knjižnicami ne omogoča izdelavo razporeda elementov grafičnega vmesnika v neki navadni tekstovni datoteki, kot je npr. XML, ampak mora vsak element biti posebno deklariran v kodi.

6 Omrežna komunikacija

Za omrežno komunikacijo sem uporabljal TCP preko omrežnih vtičnic. Za mojo aplikacijo sem izbral vrata 28015. Aplikacija je narejena le za komunikacijo med dvema uporabnikoma. Ob povezavi z novim uporabnikom, aplikacija prvo pošlje ime in javni ključ. To sporočilo je sestavljeno iz 50 bajtov: prvi bajt pove vrsto sporočila, drugi bajt dolžino sporočila. Naslednjih 32 bajtov vsebuje javni ključ, zadnjih 16 bajtov pa ime uporabnika, kar pomeni da so uporabniška imena omejena na 16 ASCII črk.

Ko oba uporabnika dobita javni ključ drugega uporabnika, ustvarita skupni tajni ključ. Potem izmenjata le šifrirana sporočila, ki sta dolga 240 bajtov. Prvih 24 bajtov predstavlja enkratno število sporočila (*nonce*). Ostalih 216 bajtov vsebuje šifropis, oziroma šifrirano vsebino sporočila.

7 Izdelava aplikacije

Mojo aplikacijo sem izdeloval v programu Visual Studio 2019 in v jeziku C++. Za to okolje za razvijanje programa sem se odločil zato, da bi na maturitetni nalogi lahko delal tudi v šoli. Pri izdelavi sem prvo začel s šifriranjem, potem pa se začel ukvarjati z grafičnim vmesnikom in na koncu izdelal omrežno komunikacijo. Pri izdelavi sem se srečal s številnimi izzivi.

7.1 Izzivi pri izdelavi

Prvi izziv je bil sploh izbor programske opreme, ki jo bom uporabljal za izdelavo moje aplikacije. Za šifriranje sem že od začetka vedel, da bom uporabljal knjižnico *libsodium*, a vendar nisem bil prepričan, katere knjižnice bom uporabljal za grafični vmesnik. Razklan sem bil med knjižnicama WXWidgets in Qt, saj obe dobro poznam in obe podpirata operacijski sistem Windows, a vendar zaradi tehničnih težav mi ni uspelo knjižnice Qt naložiti na šolski računalnik in sem se odločil za WXWidgets.

Tudi pri izbiri knjižnice za omrežno komunikacijo sem imel podobne težave. Sprva sem želel uporabiti knjižnico *boost*, ki omogoča tako asinhrono kot sinhrono omrežno komunikacijo, ampak se je ta knjižnica izkazala za preveč zahtevno za moje namene. Po tem sem izvedel, da WXWidgets že ima vgrajene omrežne vtičnice in sem jih uporabil v moji aplikaciji.

Pri testiranju aplikacije sem tudi naletel na podobne težave, ker so na šolskem lokalnem omrežju določena vrata zaprta in sem moral za mojo aplikacijo izbrati vrata, ki niso zaprta. Vrata od 1 do 2014 načeloma lahko uporabljajo le tiste aplikacije, ki imajo administratorske pravice.

7.2 Morebitne izboljšave

Pri moji aplikaciji je še nekaj stvari, ki bi jih želel izboljšati. Kot prvo, moja aplikacija je narejena zgolj za operacijski sistem Windows, a glede na to, da so vse knjižnice, ki jih uporabljam, dostopne tudi za operacijskem sistemu Linux, ne bi potreboval preveč časa, da bi mojo aplikacijo naredil tudi za ta operacijski sistem. Moja aplikacija ima še nekaj pomanjkljivosti in omejitev.

Sporočila in uporabniška imena lahko vsebujejo le ASCII črke. To je zaradi tega, ker je kodiranje s shemo UTF-8 veliko bolj zahtevno in moja prioriteta je bila delovanje šifriranja v moji kodi. Vsako sporočilo se pretvori v niz bajtov, ki so potem šifrirani in potem dešifrirani nazaj v nek niz bajtov. Pri ASCII šifriranju to ni velik problem, saj vsaka črka zaseda en bajt. Pri UTF-8 je pa lahko neka črka oziroma simbol več bajtov dolga. Vsako sporočilo je tudi omejeno na 200 bajtov oziroma 200 ASCII črk. Tudi iz tega

razloga, bi Unicode črke porabile več prostora. Uporabniška imena pa so omejena na 16 ASCII črk. Povrh tega, tudi nisem uspel dodati možnosti za spremembo uporabniškega imena. Vse to bi bilo možno izboljšati, če bi imel več časa za izdelavo aplikacije.

8 Viri

Authenticated encryption. [internet]. [citirano 27. 01. 2022]. Dostopno na naslovu: https://doc.libsodium.org/secret-key_cryptography/secretbox

Bernstein, D. *Curve25519: new Diffie-Hellman speed records.* [internet]. [citirano 13. 04. 2022]. Dostopno na naslovu: https://cr.yp.to/ecdh/curve25519-20060209.pdf

Bernstein, D. *The Salsa20 family of stream ciphers.* [internet]. [citirano 13. 04. 2022]. Dostopno na naslovu: http://cr.yp.to/snuffle/salsafamily-20071225.pdf

Elliptic Curve Cryptography (ECC). [internet]. [citirano 27. 01. 2022]. Dostopno na naslovu: https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc

Key exchange. [internet]. [citirano 27. 01. 2022]. Dostopno na naslovu: https://doc.libsodium.org/key_exchange

Paar, C. in Pelzl, J. 2009. *Understanding Cryptography: A Textbook for Practitioners and Students.* Heidelberg: Springer.

Public key cryptography. [internet]. [citirano 27. 01. 2022]. Dostopno na naslovu: https://web.stanford.edu/class/cs54n/handouts/22-PublicKeyCryptography.pdf

9 Priloge

Izvorna koda aplikacije je dostopna na Githubu: https://github.com/nikolabr/encrypted_chat

10 Zahvala

Zahvaljujem se svojemu mentorju za pomoč pri izdelavi naloge.