**Protokoly**

Pôvodná myšlienka bola spraviť TLS kanál a v ňom dodatočné šifrovanie. Zatiaľ sme stihli naimplementovať len šifrovanú komunikáciu cez TCP spojenie.

Každá správa sa skladá z dĺžky, tagu dĺžky, tagu správu a samotnej správy. Ako inicializačný vektor sa používa prvých 16 bitov hashu counteru.

**Zahájenie komunikácie klient – server:**

0. Pri spustení (server by mal bežať večne) servera si server vygeneruje RSA-4096 kľúč, ktorý overí CA,

ktorú zatiaľ neimplementujeme.

1. Klient sa napojí na server, ktorý mu zašle verejný kľúč.

2. Klient vygeneruje 256 bitový kľúč, zašifruje vereným kľúčom, pripojí k tomu číslo svojho portu na ktorom počúva spojenia od klientov a odošle serveru.

3. Server dešifruje a uloží do pamäte klúč.

-- -- -ďalej je všetko už šifrované symetricky-- -- -

**Registrácia klienta**

1. Klient odošle nejakú sekvenciu bitov signalizujúcu, že sa chce registrovať a svoj login a heslo. Heslo musí obsahovať minimálne 8 znakov a nemôže obsahovať znaky # a |

2. Server si spočíta hash (heslo + sol) a do databázy uloží login-hash-sol.

3. Server odošle správu s oznámením či to prebehlo úspešne(login mohol byť už zabraný). Ak nie

opakuje sa krok 1.

4. Po registrácií je klient prihlásený

**Login klienta**

1. Klient odošle svoj login a heslo.

2. Server si spočíta hash overí či prislúcha danému loginu.

3. Server odošle správu s oznámením či to prebehlo úspešne. Ak nie opakuje sa krok 1.

**Nadviazanie komunikácie s druhým klientom**

1. Každý klient má stále vytvorený server a zverejnené číslo portu (evidované u servera) na ktorom očakáva nové spojenie.

2. A pošle serveru požiadavku na komunikáciu s B. Server vytvorí správu **Eka(**ip Bčka || port Bčka ||64 náhodných bajtov **) || Ekb (**64 náhodných bajtov (rovnakých ako Ačku), login Áčka)**)**.

3. A sa za pomoci údajov zo serveru napojí na B, pošle časť zašifrovanú **Ekb** Bčku

4. Prebhne Diffie-Hellman zašifrovaný cez AES, kde klúč je prvých 32 bajtov z náhodného reťazca.

5. Ako dôkaz, že obaja vedia použiť ustanovený klúč, A pošle Bčku prvých 16 zo zvyšných 32 náhodných bajtov a Bčko druhých 16. Obidve strany si to porovnajú so svojimi dátami, ak to sedí, môžu začať chatovať.

**Logout klienta**

Server vždy pri prihlásení/odhlásení klienta aktualizuje zoznam on-line klientov. Keď chce klient zistiť kto je online môže si vyžiadať list online užívateľov. Pri disconnecte sa klient odhlási.

**Poznámky k implementácii**

Funkcie v súbore crypto.cpp:

**1.** void parseMessage(QTcpSocket\* socket, uint32\_t\* m\_inCounter, quint8\* message\_type, QString\* message, unsigned char\* m\_aesKey);

**2.** bool sendMessage(QTcpSocket\* socket, uint32\_t\* m\_outCounter, quint8 messageType, QString message, unsigned char\* m\_aesKey);

sú použité v komunikácií klient-server a funkcie:

**3.** bool parseMessage(QByteArray& input, uint32\_t\* m\_inCounter, quint8\* message\_type, QByteArray& message, unsigned char\* m\_aesKey);

**4.** bool sendMessage(QTcpSocket\* socket, uint32\_t\* m\_outCounter, quint8 messageType, QByteArray& message, unsigned char\* m\_aesKey);

sú použité v komunikácií klient-klient. Robia takmer to isté až na to, že druhé dve pracujú s QBiteArrayom, z ktorého sa potom dajú za pomoci QDataStreamu ťahať data a tie prvé dve spravia z celej správy jeden string, v ktorom musia byť jednodlivé časti rozdelené delimetrom. V našom program sme použili |#| preto sú tieto znaky pri registrácií zakázané. A ešte jeden rozdiel je v tom, že 4. funkcia nedešifruje dĺžku, ktorá sa musí dešifrovať zvlášť v messenger.cpp. Tento mierny nesúlad v použitých funkciách vznikol v dôsledku chabej komunikácie developer-developer.

Ostatné veci ako ustanovenie klúčov, kde sa nenačítavajú žiadne data do stringov od užívateľa pracujú čisto len z funkciami:

**5.** const unsigned char\* encryptMessage(quint8 messageType, uint32\_t\* counter, const unsigned char\* input, uint32\_t inputLength, size\_t& outputLength, unsigned char\* tag, const unsigned char\* key);

**6.** const unsigned char\* decryptMessage(quint8\* messageType, uint32\_t\* counter, const unsigned char\* input, uint32\_t inputLength, unsigned char\* tag, const unsigned char\* key);

**7.** bool decryptLength(uint32\_t& lenght, unsigned char\* input, unsigned char \*tag, uint32\_t\* counter, unsigned char\* m\_aesKey);

**8.** bool encryptLength(uint32\_t lenght, unsigned char\* output, unsigned char \*tag, uint32\_t\* counter, unsigned char\* m\_aesKey);

Tie prvé 4 funkcie vlastne tieto dalšie obalujú a kombinujú.

Po zapnutí servera treba chvílu čakať kým sa vygeneruje klúč. Niekedy to trvá dobrú pol minútu takže server nevypínajte pokiaľ netreba :D. Aplikácie sú v zložkách **src\applications\server\debug** a **src\applications\client\debug**