SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-8717

VYUŽITIE MCELIECEOVHO KRYPTOSYSTÉMU V OS ANDROID

DIPLOMOVÁ PRÁCA

2016 **Bc. Andrej Boledovič**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-8717

VYUŽITIE MCELIECEOVHO KRYPTOSYSTÉMU V OS ANDROID

DIPLOMOVÁ PRÁCA

Študijný program:	Aplikovaná informatika
Číslo študijného odboru:	2511
Názov študijného odboru:	9.2.9 Aplikovaná informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	doc. Ing. Pavol Zajac, PhD.
Konzultant:	Ing. Juraj Varga

Bratislava 2016 **Bc. Andrej Boledovič**

Slovenská technická univerzita v Bratislave
Ústav informatiky a matematiky

Fakulta elektrotechniky a informatiky
Akademický rok: 2015/2016
Evidenčné číslo: FEI-5384-8717

ZADANIE DIPLOMOVEJ PRÁCE

Študent:	**Bc. Andrej Boledovič**
ID študenta:	8717
Študijný program:	Aplikovaná informatika
Študijný odbor:	9.2.9. aplikovaná informatika
Vedúci práce:	doc. Ing. Pavol Zajac, PhD.
Konzultant:	Ing. Juraj Varga
Miesto vypracovania:	Ústav informatiky a matematiky

Názov práce: **Využitie McElieceovho kryptosystému v prostredí OS Android**

Špecifikácia zadania:

McElieceov kryptosystém je jedným zo základných predstaviteľov postkvantovej kryptografie. Tento kryptosystém je súčasťou Android krypto knižnice SpongyCastle. Najprv bude potrebné zistiť, či je táto implementácia v poriadku a použiteľná pre naše účely. Následne bude tento systém implementovať v klient-server mobilnej aplikácii a otestovať použiteľnosť výslednej implementácie v reálnych podmienkach.

Úlohy:
1. Naštudujte problematiku adaptácie McElieceovho kryptosystému v klient-server scenári a krypto knižnicu SpongyCastle.
2. Navrhnite systém s mobilnými klientami a centrálnym serverom.
3. Implementujte navrhnutý systém pomocou McEliece implementácie v krypto knižnici SpongyCastle.
4. Otestujte implementované riešenie.

Zoznam odbornej literatúry:

1. F. Uhrecký. Implementácia kryptografickej knižnice s McEliece kryptosystémom. Diplomová práca. FEI STU, 2015.
2. Spongy Castle. http://rtyley.github.io/spongycastle/

Riešenie zadania práce od: 21. 09. 2015

Dátum odovzdania práce: 20. 05. 2016

L.S.

Bc. Andrej Boledovič
študent

prof. RNDr. Otokar Grošek, PhD.
vedúci pracoviska

prof. RNDr. Gabriel Juhás, PhD.
garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Bc. Andrej Boledovič
Diplomová práca:	Využitie McElieceovho kryptosystému v OS Android
Vedúci záverečnej práce:	doc. Ing. Pavol Zajac, PhD.
Konzultant:	Ing. Juraj Varga
Miesto a rok predloženia práce:	Bratislava 2016

Práca sa zaoberá navrhnutím a implementovaním klient-server aplikácie využívajúcej McElieceov kryptosystém v prostredí OS Android. Tento kryptosystém môžeme zaradiť do post-kvantovej kryptografie. Je teda odolný voči prípadnému útoku vykonanom na kvantovom počítači. Mobilné aplikácie sa dostávajú stále viac do popredia, preto je potrebné zabezpečiť bezpečnosť používania takýchto aplikácií. Využitie kryptografických prostriedkov v OS Android je do určitej miery obmedzené. Úlohou je vytvoriť prehľad dostupných knižníc zahrňujúcich McElieceov kryptosystém a otestovať riešenie na mobilnom zariadení s OS Android. V prvej časti práce je spracovaná teória k tejto problematike. Venuje sa taktiež protokolom, v ktorých sa dá použiť McElieceov kryptosystém vrámci klient-server scenára. V ďalšej časti sa nachádza návrh riešenia. Je v ňom popísaný spôsob realizácie zabezpečenia komunikácie, ako aj požiadavky na mobilnú aplikáciu a serverovú časť. Navrhnutá aplikácia slúži na výmenu správ, ide o tzv. 'messenger'. V ďalšej časti je popísaná implementácia a testovanie aplikácie. McElieceov kryptosystém má určité obmedzenia ako je napríklad veľkosť verejného kľúča, preto sa testuje jeho využitie v reálnych podmienkach a porovnáva sa s dostupnými riešeniami. Výsledná aplikácia je plne funkčná a spĺňuje podmienky bezpečnej komunikácie. Na rozdiel od iných podobných aplikácií ide o post-kvantové riešenie.

Kľúčové slová: Android, McEliece, klient-server, public-key protokol, zabezpečenie komunikácie

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Bc. Andrej Boledovič
Diploma Thesis:	Usage of McEliece cryptosystem in OS Android
Supervisor:	doc. Ing. Pavol Zajac, PhD.
Consultant:	Ing. Juraj Varga
Place and year of submission:	Bratislava 2016

The thesis deals with designing and implementing client-server application that use McEliece cryptosystem in OS Android. This cryptosystem can be included in post-quantum cryptography. It is resistant to any attack carried out on a quantum computer. Mobile applications are getting more and more into the foreground, so it is necessary to ensure the safe usage of such applications. Use of cryptography is limited in OS Android. The task is to make an overview of available libraries containing McEliece cryptosystem and test the solution on a mobile device with Android OS. The first part contains theoretical background on this issue. It also deals with protocols, in which McEliece cryptosystem can be used within a client-server scenario. The next section is proposed solution. The realization of secure communication and the requirements for a mobile app and backend is described here. The proposed application serves to exchange messages, it is also called 'messenger '. The next section describes the implementation and testing of application. McEliece cryptosystem has some limitations such as the size of a public key, therefore the solution is tested in real conditions and compared with other available solutions. The resulting application is fully functional and meets the conditions for secure communication. It is post-quantum secured unlike other similar applications.

Keywords: Android, McEliece, client-server, public-key protocol, secured communication

Vyhlásenie autora

Podpísaný Bc. Andrej Boledovič čestne vyhlasujem, že som diplomovú prácu Využitie McElieceovho kryptosystému v OS Android vypracoval na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.

Vedúcim mojej diplomovej práce bol doc. Ing. Pavol Zajac, PhD.

Bratislava, dňa 24.5.2016

.....

podpis autora

Poďakovanie

Touto cestou by som sa chcel poďakovať vedúcemu práce doc. Ing. Pavlovi Zajacovi, PhD. a konzultantovi Ing. Jurajovi Vargovi za odbornú pomoc, cenné rady a pripomienky, ktoré mi pomohli pri vypracovaní tejto práce.

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek a značiek

OS - Operačný Systém
AES - Advanced Encryption Standard
DES- Data Encryption Standard
RSA - Rivest Shamir Adleman
SHA - Secure Hash Algorithm
API - Application Programming Interface
CCA2 - Adaptive chosen-ciphertext attack
KDF - Key Derivation System
NP - Nondeterministic Polynomial time
MECS - McEliece Cryptosystem
PRNG - Pseudorandom Number Generator
FIPS - Federal Information Processing Standards
GCM - Google Cloud Messaging
CRL - Certificate Revocation List
PKCS - Public-Key Cryptography Standards
ASCII - American Standard Code for Information Interchange
HTTP - Hypertext Transfer Protocol
REST - Representational State Transfer
XML - eXtensible Markup Language
PDF - Portable Document Format
URI - Uniform Resource Identifier
DB - Database
UML - Unified Modeling Language
BLOB - Binary large object
JSON - JavaScript Object Notation
SQLite - Structured Query Language Lite
GUI - Graphical User Interface
RAM - Random Access Memory
CPU - Central Processing Unit
KDF - Key Derivation Function
hwt - Hamming Weight

Zoznam algoritmov

Úvod

Veľa aplikácií využíva v dnešnej dobe kryptografiu s verejným kľúčom na zabezpečenie komunikácie. Ide o kryptosystémy využívajúce známe matimatické problémy. Tieto šifry zatiaľ neboli zlomené, no existuje určitá hrozba ich prelomenia pri vzniku kvantových počítačov. Ide o Schorov algoritmus, ktorý dokáže riešiť tieto matematické problémy na kvantovom počítači. Zatiaľ nie je dôkaz o existencii fuknčného kvantového počítača, avšak boli predložené rôzne prototypy a sú vynakladané veľké prostriedky na jeho výskum. Preto je treba zaoberať sa kryptosystémami, ktoré by boli voči takýmto útokom odolné. Touto tematikou sa zaoberá post-kvantová kryptografia. Azda najvhodnejším kandidátom pre post-kvantovú asymetrickú kryptografiu sa javí byť McEliecov kryptosystém.

Mobilné aplikácie sa čoraz viac dostávajú do popredia a sú stále viac využívané. Veľká časť týchto aplikácií komunikuje medzi sebou, prípadne so serverom, vzdialenou databázou atď. Sú aplikácie, ktoré komunikujú nezabezpečeným kanálom. Môže ísť napr. o hry alebo informačné aplikácie. Tieto aplikácie nevyžadujú šifrovanie, pretože prípadne zachytené dáta nie sú tajné alebo dôležité. Na druhú stranu, niektoré aplikácie vyžadujú zabezpečenie kryptografiou. Medzi takéto aplikácie môžme zaradiť internetové bankovanie, komunikačné služby alebo iné aplikácie, ktoré prenášajú citlivé dáta. Takéto prípady využívajú dnes známe šifry, ktoré nie sú post-kvantovo bezpečné. Preto je úlohou vytvoriť taký klient-server scenár, aby spĺňal podmienky bezpečnej komunikácie. Cieľom tejto práce je navrhnúť klient-server aplikáciu využívajúcu McEliecov kryptosystém pre mobilné zariadenia so systémom Android. McEliecov kryptosystém patrí medzi najstaršie kryptosystémy s verejným kľúčom. Je treba zanalyzovať použitie McEliecovho kryptosystému na mobilných zariadeniach a vyskúšať ho v reálnych podmienkach.

V tomto dokumente je na úvod spracovaná teória k tejto problematike a sú v nej spísané dostupné riešenia. V ďalšej časti sa nachádza návrh implementácie, ktorý popisuje požiadavky na navrhnutú aplikáciu a serverovú časť. Niektoré časti z návrhu sa môžu mierne líšiť od realizácie, keďže implementácia bola prispôsobovaná použitým technológiám. Samotná realizácia je popísaná v implementačnej časti dokumentu. Je v nej popísaná implemetácia klientskej aj serverovej časti, použité technológie a taktiež databázové modely. Taktiež je tu popísané použitie kryptografických knižníc. V neposlednom rade sa v dokumente nachádza testovanie aplikácie, ako aj vyhodnotenie dosiahnutých výsledkov. V prílohe dokumentu môžeme nájsť manuál k inštalácii a používaniu aplikácie.

1 Analýza problému

1.1 Android OS

Android je operačný systém bežiaci na mobilných platformách a tabletoch. Patrí do skupiny OS založených na Linuxovom jadre, preto je volne dostupný. Android je viacvrstvový OS počnúc Linuxovým jadrom až po samotné aplikácie. Z toho dôvodu vyžaduje robustné bezpečnostné opatrenia na ochranu dát, užívateľov, siete... V tejto sekcii sa budeme venovať vybraným bezpečnostným mechanizmom tohto OS zverejnených v [7].

Linuxové zabezpečenie

Ako bolo spomenuté, Android je založený na Linuxovom jadre. Linuxové distribúcie sú rozšírené po celom svete a využívané v rôznych bezpečnostných systémoch. Ako volne dostupný OS sa rozvíjal po mnohé roky a vývojári si ho prispôsobovali podľa rôznych bezpečnostných potrieb. Preto môžme povedať, že bezpečnosť tohto systému je v dnešnej dobe na vysokej úrovni.

Povolenia systémových súborov

Jednotlivé súbory v telefóne podliehajú UNIX-ovým povoleniam. Každý súbor ma pridelené povolenia na čítanie, zapisovanie, či upravovanie údajov. Súbory vytvorené jednou aplikáciou nemôžu byť upravované inými aplikáciami, pokiaľ im toto povolenie nie je pridelené. Takéto opatrenia majú aj systémové aplikácie, s ktorými systém neumožňuje manipulovať. Toto opatrenie je možné obísť pridelením 'rootovských' práv aplikáciám. S ním sa avšak zvyšuje bezpečnostné hrozba útoku škodlivých aplikácií.

Kryptografia

Android poskytuje implementáciu viacerých kryptografických mechanizmov. Tieto API dovoľujú využívať známe šifry ako AES, DES, RSA... Novšie verzie Androidu (4.0 a vyššie) dovoľujú aplikáciam využívanie systémového úložiska pre ukladanie certifikátov a verejných kľúčov. Na zabezpečenie komunikácie je k dispozícii SSL protokol.

Šifrovanie súborového systému

Android zariadenia podporujú šifrovanie užívateľských súborov. Tieto súbory sú zabezpečené 128-bitovým AES kľúčom. Tento kľúč je chránený 128-bitovým AES kľúčom odvodeným z užívateľského hesla. Pre docielenie zabezpečenia voči hádaniu hesla je kľúč skombinovaný s náhodným 'saltom', ktorý je ďalej hašovaný SHA1 algoritmom.

Implementácie McEliece v systéme Android

McElieceov kryptosystém je súčasťou viacerých kryptografických knižníc ako sú:

- BouncyCastle
  Podľa [11] ide o volne dostupnú knižnicu podporovanú austrálskou charitatívnou organizáciou Legion of the Bouncy Castle Inc. Jej API podporujú programovacie jazyky Java a taktiež C Sharp.

- BitPunch
  Jedná sa o samostatnú implementáciu McElieceovho kryptosystému vytvorenú pre programovací jazyk C [16].

Tieto knižnice však nie sú vhodné pre operačný systém Android. Implementácia McElieceovho kryptosystému na mobilných zariadeniach so systémom Android je možná vďaka knižnici SpongyCastle [15]. Jedná sa o prekompilovanú verziu knižnice BouncyCastle pre účely využitia v systéme Android. Kvôli zabezpečeniu je McEliece zabezpečený určitou CCA2-bezpečnou konverziou. Táto knižnica obsahuje jej viaceré verzie, ktoré sú popísané v ďalšej časti. Sú to:

- Pointcheval,
- Kobara Imai,
- Fujisaki.

1.2 Kryptografické pozadie

Dnešné kryptosystémy sa rozdeľujú na symetrické a asymetrické. Na rozdiel od symetrickej kryptografie, ktorá využíva jeden a ten istý kľúč na šifrovanie a dešifrovanie správ, asymetrická kryptografia využíva dva kľúče. Verejný kľúč e sa používa na šifrovanie a súkromný kľúč d sa využíva na dešifrovanie. Je kryptograficky nemožné odvodiť jeden kľúč od druhého. Je pravidlom, že súkromný kľúč nie je verejne dostupný a má ho u seba uchovaný majiteľ. Na druhú stranu, verejný kľúč je verejne dostupný a môže ho vidieť hocikto. Preto sa verejný kľúč používa na šifrovanie.

Ak chce entita A poslať správu entite B, tak správu zašifruje pomocou verejného kľúča entity B. Entita B vie potom správu dešifrovať pomocou svojho súkromného kľúča. Je zachovaná dôvernosť správy, keďže správu dokáže dešifrovať iba entita B. Avšak nie je zaručená autenticita odosielateľa. Asymetrické systémy sú z tohoto hľadiska bezpečnejšie a jednoduchšie ako symetrické, pretože nie je treba prenášať symetrický kľúč cez zabezpečený kanál. Na druhú stranu sú pomalšie ako symetrické šifry. Preto sú asy-

metrické kryptosystémy zväčša používané na vytvorenie bezpečného kanálu pre prenos symetrického kľúča, ktorý je ďalej využívaný na dešifrovanie [2].

1.2.1 Notácie

$Px(m)$ - zašifrovanie správy m verejným kľúčom entity X,
$Gen(P_E, S_E)$ - vygenerovanie kľúčového páru asymetrického kryptosystému,
$KDF(a_1, a_2)$ - vytvorenie symetrického kľúča z hodnôt a_1, a_2,
$hwt(v)$ - Hammingova váha vektora v.

1.2.2 Protokoly

Na prenos symetrického kľúča existujú rôzne protokoly, pričom niektoré využívajú kryptografiu s verejným kľúčom.

Needham-Schroeder public-key protocol
Tento protokol zabezpečuje výmenu symetrického kľúča a taktiež zabezpečuje vzájomnú autentifikáciu účastníkov komunikácie.

Algoritmus 1 Needham-Schroeder public-key protokol

1: Entita A pošle $P_B(k_1, A)$ entite B
2: Entita B pošle $P_A(k_1, k_2)$ entite A
3: Entita A pošle $P_B(k_2)$ entite B

Alg. 1 popisuje proces výmeny správ medzi entitami A a B. V kroku (1) pošle entita A prvú časť kľúča k_1 a svoju identitu entite B. Celú správu vie dešifrovať iba legitímny príjemca s prislúchajúcim súkromným kľúčom, čiže entita B. B ďalej (2) pošle entite A kľúče k_1, k_2 zašifrované verejným kľúčom entity A. A dešifruje správu svojím súkromným kľúčom. B sa autentifikovalo entite A, keďže správa obsahuje kľúč k_1. V poslednom kroku (3) pošle A entite B zašifrovanú správu obsahujúcu obdržaný kľúč k_2. Tým sa autentifikuje entite B, keďže nikto iný tento kľúč poznať nemôže. Ďalej bude komunikácia medzi A a B prebiehať šifrovaním pomocou symetrického kľúča k, ktorý bude výstupom nejakej funkcie f. Táto funkcia je predom dohodnutá a známa pre všetky strany. Jej vstupom sú dve časti kľúča k_1 a k_2 [2].

Modifikovaný Needham-Schroeder public-key protocol
Tento protokol je podobný pôvodnému Needham-Schroeder protokolu. Vznikol pre zrýchlenie celého procesu. Eliminuje sa v ňom posledné šifrovanie z kroku (3) v Alg. 1. V tomto protokole každá strana generuje navyše dve náhodné čísla r_1 a r_2, ktoré vôbec nesúvisia

s výsledným symetrickým kľúčom. Proces výmeny správ je potom znázornený v Alg. 2.

Algoritmus 2 Modifikovaný Needham-Schroeder public-key protokol

1: Entita A pošle $P_B(k_1, A, r_1)$ entite B
2: Entita B pošle $P_A(k_2, r_1, r_2)$ entite A
3: Entita A pošle r_2 entite B

Vidno, že namiesto nadobudnutej časti kľúča k_1 a k_2 sa späť posielajú už iba nadobudnuté čísla r_1 a r_2. Keďže si ľubovoľná entita vie dešifrovať správu svojím súkromným kľúčom a nadobudnúť zaslané náhodné číslo, je jasné že má taktiež potrebnú časť kľúča. Preto sa ďalej na overenie a autentifikáciu posiela už iba číslo. Tento algoritmus je kryptograficky rovnako bezpečný ako pôvodný Needham-Schroeder protokol, ale je vhodnejší pre implementáciu [1].

Perfect Forward secrecy

Tieto protokoly avšak nespĺňajú tzv. "forward secrecy" alebo aj inak nazývané "perfect forward secrecy". Ide o útoky na protokoly, pri ktorých sú kompromitované kryptografické kľúče.

Definícia: [2] "Protokol spĺňa "perfect forward secrecy" ak kompromitovanie dlhodobých kľúčov nekompromituje dočasné kľúče."

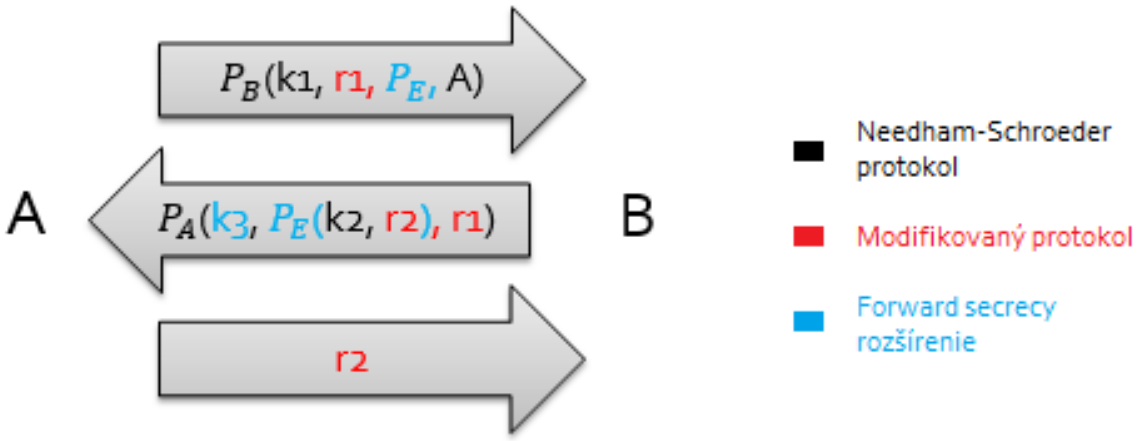
Forward secrecy chráni históriu komunikácie v prípade, že dlhodobé kľúče boli zlomené alebo odcudzené. Predpokladajme, že si útočník uchovával zašifrované texty. V takom prípade útočník nevie získať dočasné kľúče, ktorými bola zabezpečená komunikácia v minulosti, pretože nie sú nikde uchovávané [5].
V Alg. 3 je podrobne navrhnutý protokol, ktorý splňuje túto podmienku.

Algoritmus 3 Forward secrecy protokol

1: A: $Gen(P_E, S_E)$
2: A: generuj k_1, r_1
3: Entita A pošle $P_B(k_1, r_1, P_E, A)$ entite B
4: B: generuj k_2, k_3, r_2
5: Entita B pošle $P_A(k_3, P_E(k_2, r_2), r_1)$ entite A
6: Entita A pošle $r2$ entite B

V kroku (1) vygeneruje A nový kľúčový pár, ktorý je dočasný a nebude uchovaný. V

kroku (2) vygeneruje A časť kľúča k_1 a náhodné číslo r_1. V ďalšom kroku (3) je k_1, r_1 a vygenerovaný verejný kľúč P_E poslaný entite B. B získa tieto údaje dešifrovaním správy svojím súkromným kľúčom. Ďalej (4) entita B vygeneruje ďalšie časti kľúča a náhodné číslo r_2. Potom B zašifruje k_2 a r_2 verejným kľúčom P_E. Spolu s r_1 a k_3 to pošle entite A (5). A najskôr celú správu dešifruje svojim súkromným kľúčom S_A. V tomto momente A získalo k_3, r_1 a zašifrovanú časť. Vie teda, že entita B nadobudla k_1. Zašifrovanú časť dešifruje A privátnym kľúčom S_E, ktorý vlastní iba A. V poslednom kroku komunikácie (6) pošle A nadobudnuté číslo r_2 nezabezpečeným kanálom entite B. B potom vie, že A obdržal zvyšné časti kľúča. V tomto momente majú obe strany vštky hodnoty (k_1, k_2, k_3) potrebné na získanie symetrického kľúča pomocou dohodnutej funkcie.
Ak by boli kompromitované súkromné kľúče entít A a B, tak by účasník nevedel získať k_2 potrebnú na zostavenie kľúča. Vo výsledku tento protokol nezaručuje len výmenu kľúča a vzájomnú autentifikáciu účastníkov, ale splňuje aj forward secrecy.
Pre ilustráciu sa pozrieme na Obr. 1, na ktorom vidno jednotlivé zmeny pri vylepšovaní Needham-Schroederovho protokolu rozlíšené farebne.



Obrázok 1: Protokoly

1.2.3 McElieceov kryptosystém

McEliecov kryptosystém sa zaraďuje do asymetrickej kryptografie [10]. Preto je vhodným kandidátom pre protokoly na výmenu symetrického kľúča. Je založený na dekódovacom probléme, ktorý sa radí medzi NP-úplné problémy. Originálny algoritmus využíva Goppa kódy, ktoré sú ľahko dekódovateľné aj vďaka Pattersonovmu algoritmu. Na rozdiel od iných kryptosystémov vyvužívajúcich matematické problémy, ktoré sú zlomiteľné na

kvantovom počítači, dekódovací problém by mal byť bezpečný. Tejto šifre sa avšak nevenuje veľa pozornosti kvôli veľkosti jej verejného kľúča.

Generovanie kľúčov

Pri generovaní kľúčového páru sa zvolia tri parametre:

- t - počet chýb, ktoré dokáže dekódovací algoritmus opraviť,
- n - veľkosť konečného poľa, určuje úroveň bezpečnosti,
- k - dimenzia podpriestoru.

Algoritmus 4 Generovanie kľúčov

1: Zvolí sa generačná matica G veľkosti $n \times k$, ktorá dokáže opraviť t chýb
2: Náhodne sa vygeneruje nesingulárna matica S veľkosti $k \times k$
3: Náhodne sa vygeneruje permutačná matica P veľkosti $n \times n$
4: Vypočíta sa $\hat{G} = SGP$

Ako vidno v Alg. 4 popísanom v [2], generujú sa dohromady tri matice. Tieto matice tvoria spolu súkromný kľúč (S, G, P). Verejný kľúč je odvodený zo súkromného kľúča. Je to $(\hat{G}, t)$.

Šifrovanie

Alg. 5 popisuje zašifrovanie správy m. V McElieceovom kryptosystéme je šifrovanie najrýchlejší proces. Zašifrovaný text je binárny vektor c, ktorý vznikne vynásobením s maticou G a pridaním chybového vektora v.

Algoritmus 5 Šifrovanie

1: Nech správa m reprezentuje bitový reťazec dĺžky k
2: Zvolíme náhodný vektor v dĺžky n, kde $hwt(v) = t$
3: Vyrátame $c = m\hat{G} + v$

Dešifrovanie

Alg. 6 popisuje dešifrovanie správy c. Dešifrovanie je pomalšie ako šifrovanie. Jeho podstatou je dekódovací algoritmus, ktorý dekóduje vektor $\hat{c}$. Príkladom takéhoto algoritmu je napr. Pattersonov dekódovací algoritmus. Dešifrovanou správou je potom m, ktoré

Algoritmus 6 Dešifrovanie

1: Nech P^{-1} je inverzná matica k matici P a S^{-1} je inverzná matica k matici S
2: Vyrátame $\hat{c} = cP^{-1}$
3: Použijeme dekódovací algoritmus a dekódujeme $\hat{c}$ na $\hat{m}$
4: Vyrátame $m = \hat{m}P^{-1}$

vznikne vynásobením P^{-1} a výstupu z Pattersonovho algoritmu.

Voľba parametrov

V pôvodnom článku [10], kde McEliece uviedol tento kryptosystém, sú parametre nastavené na hodnoty: $n = 1024$, $t = 50$, a $k = 524$. Podľa novších výskumov z [2] sú odporúčané parametre: $n = 1024$, $t = 38$, $k = 644$. Ako bolo spomínané, tento kryptosystém sa neujal v praxi kvôli veľkosti verejného kľúča. V takomto prípade je veľkosť kľúča 2^{19}.

CCA2-bezpečná verzia McElieceovho kryptosystému

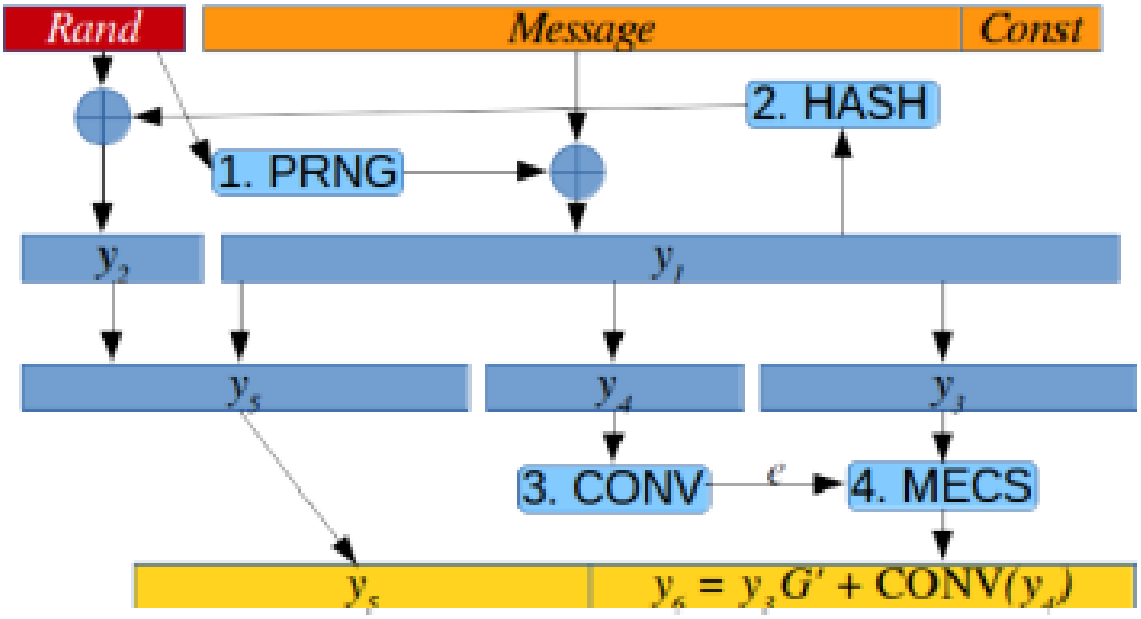
Pôvodný McElieceov kryptosystém nie je úplne bezpečný. Je zraniteľný voči niektorým útokom. Preto sa k pôvodnému algoritmu pridávajú rozšírenia, ktoré eliminujú jeho zraniteľnosti. Bezpečná verzia môže byť docielená konverziou pôvodnej správy na náhodný reťazec bitov. Tento reťazec je potom vstupom do pôvodného algoritmu [9].

Kobara-Imai konverzia

Kobara a Imai vytvorili konverzie McElieceovho kryptosystému, ktoré sú CCA2-bezpečné. Vychádzajú z nej aj ďalšie konverzie prezentované Pointchevalom a taktiež Fujisaki s Komamotom. Tieto dve schémy avšak pridávajú veľký nadbytok dát do zašifrovaného textu. Je to až n-bitov ako sa uvádza v [4].

Kobara-Imai schéma sa snaží tento nadbytok dát redukovať, ba dokonca aby bol menší než v pôvodnom kryptosystéme. Vyžaduje iba pridanie $(n-k+s)$ bitov v závislosti od nastavenia bezpečnostnej úrovne s, ktorá závisí od voľby parametrov kryptosystému. Táto konverzia môže byť aplikovaná na viaceré kryptosystémy fungujúce na podobnom systéme ako McEliece. Jedným z nich je napríklad Niederreiterov kryptosystém ako sa uvádza v [6]. Táto schéma ako prvá pridáva náhodnosť do algoritmu. To činí zašifrovaný text nerozlíšiteľným od pôvodnej správy. Vďaka tomu je šifra odolná voči viacerým útokom, ktoré využívajú vzťah medzi zašifrovaným textom a správou.

Definícia 2 [4] Zlomenie nerozlíšiteľnosti v CCA2 modele s využitím spomenutých konverzií je rovnako ťažké ako zlomiť pôvodný McElieceov kryptosystém.



Obrázok 2: Kobara-Imai schéma [9]

Šifrovanie

Na Obr.2 je zobrazený algoritmus šifrovania správ popísaný v [9]. Do algoritmu vstupuje náhodný kľúč *Rand*, ktorý definuje úroveň bezpečnosti symetrického šifrovania. K správe je pripočítaná verejne známa konštanta *Const*, ktorá definuje úroveň bezpečnosti autentifikácie. *Rand* vstupuje do $PRNG$, čo je pseudo-náhodný generátor ekvivalentný symetrickému šifrovaniu. *Conv* je konverzná fukncia, ktorá inverzne počíta chybový vektor e s $hwt(e) = t$. Tento vektor taktiež vstupuje do $MECS$, ktorý reprezentuje klasický McElieceov kryptosystém.

Dešifrovanie

Vstupom do dešifrovacej funkcie je správa c, verejne známa *Const* a náš súkromný kľúč. Pomocou klasického dešifrovaciho algoritmu pre McElliece z Alg. 6 získame y_3 a y_4, respektíve $y_4 = CONV^{-1}(e)$. Inverzne k Obr č.2 ďalej dostaneme y_1 a y_2. Potom správu m a konštantu *Const* dostaneme vypočítaním $PRNG(y_2 + HASH(y_1)) + y_1$. Integrita správy sa dá zistiť porovnaním nadobudnutej konštanty *Const* [9].

1.2.4 Advanced Encryption Standard

Táto šifra bola pôvodne jedným z kandidátov na súťaži, ktorá mala určiť nástupcu šifry DES, ktorá už ďalej nebola bezpečná. Súťaž vyhral algoritmus Rijndael, ktorý skonštruovali Belgičania Joan Daemen a Vincent Rijmen. Jedná sa o symetrickú blokovú šifru so substitučno-permutačnou sieťou. Na rozdiel od DESu nemá Feistfalovskú štruktúru. Algoritmus je popísaný podľa [1], v ktorom je popísaný štandard FIPS 197 venujúci sa tejto šifre.

Algoritmus AES šifruje bloky textu veľkosti 128-bitov, pričom veľkosti kľúča môžu byť 128, 192 alebo 256 bitov. Počet kôl sa potom mení so zväčšujúcim sa kľúčom na 10, 12 a 14.

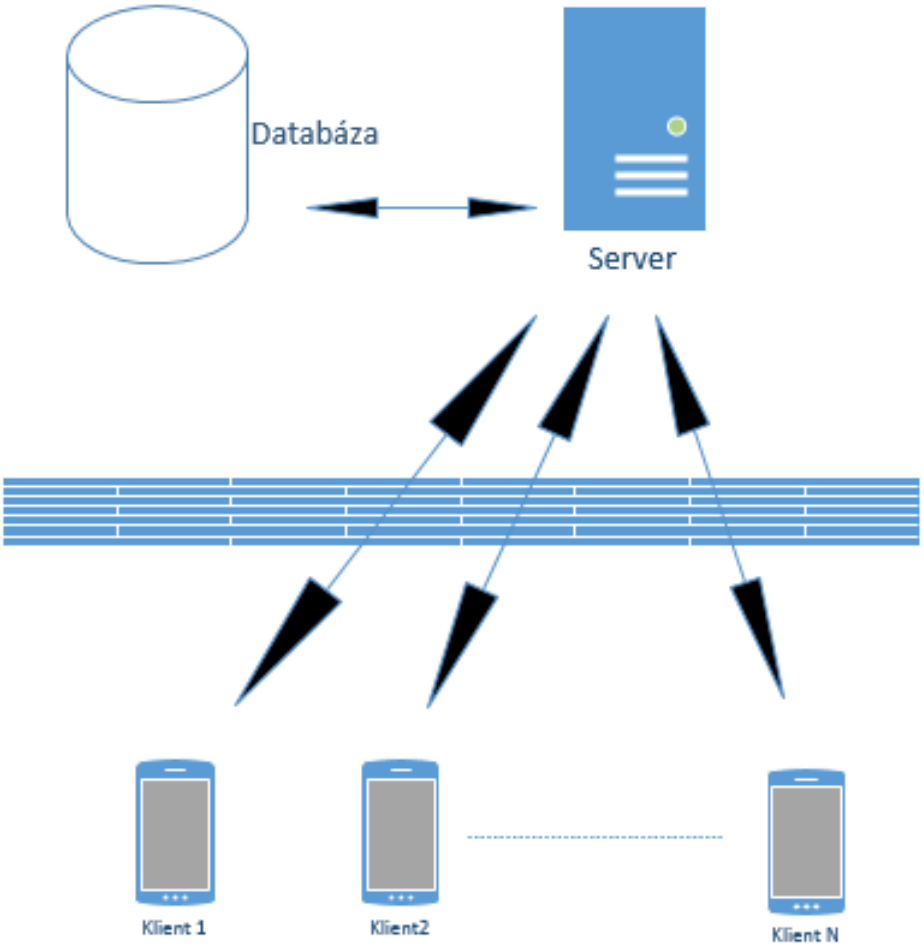
Jednotlivé kolo algoritmu pozostáva z týchto operácií:

- Substitúcia bytov podľa S-boxu,
- rotácia riadkov,
- premiešanie stĺpcov,
- pripočítanie kľúča.

V posledom kole sa vynecháva operácia premiešania stĺpcov. Šifrovanie aj dešifrovanie využíva rovnaký algoritmus, respektívne ich inverzné operácie s miernymi zmenami pri dešifrovaní.

2 Návrh

Úlohou je vytvoriť klient-server aplikáciu ktorá by demonštrovala navrhnutý systém. Aplikáciu bude možné nainštalovať na mobilné zariadenia so systémom Android. Bude sa jednať o aplikáciu na posielanie správ medzi klientmi. Ide o typ komunikačnej aplikácie nazývanej messenger. Komunikáciu medzi klientmi bude sprostredkovávať server. Dáta o užívateľoch budú uchovávané v databáze, ku ktorej bude server pristupovať. Bude sa jednať o klasickú klient-server architektúru z Obr.č.3 používanú v mobilných aplikáciách.



Obrázok 3: Klient-server architektúra

2.1 Zabezpečenie komunikácie

Každý z užívateľov si pri registrácii vygeneruje McElieceov kľúčový pár. Verejný kľúč spolu s identitou užívateľa budú poslané na server a uložené do databázy. Súkromný kľúč bude uložený v telefóne a chránený heslom. Server bude mať taktiež svoj kľúčový pár. Kvôli dôveryhodnosti bude verejný kľúč servera súčasťou nainštalovanej aplikácie. Prvotná komunikácia medzi aplikáciou a serverom nebude zabezpečená. Bude sa jednať o registráciu užívateľa. Po registrácii môže klient nadviazať zabezpečenú komunikáciu s iným klientom. Keďže komunikácia bude zabezpečená cez server, užívateľ nadviaže zabezpečený kanál taktiež so serverom. Komunikácia bude prebiehať šifrovaním správ symetrickým kľúčom, ktorý poznajú len dve komunikujúce strany. Výmena tohoto kľúča bude zrealizovaná niektorým z prezentovaných protokolov využívajúcich verejný kľúč.

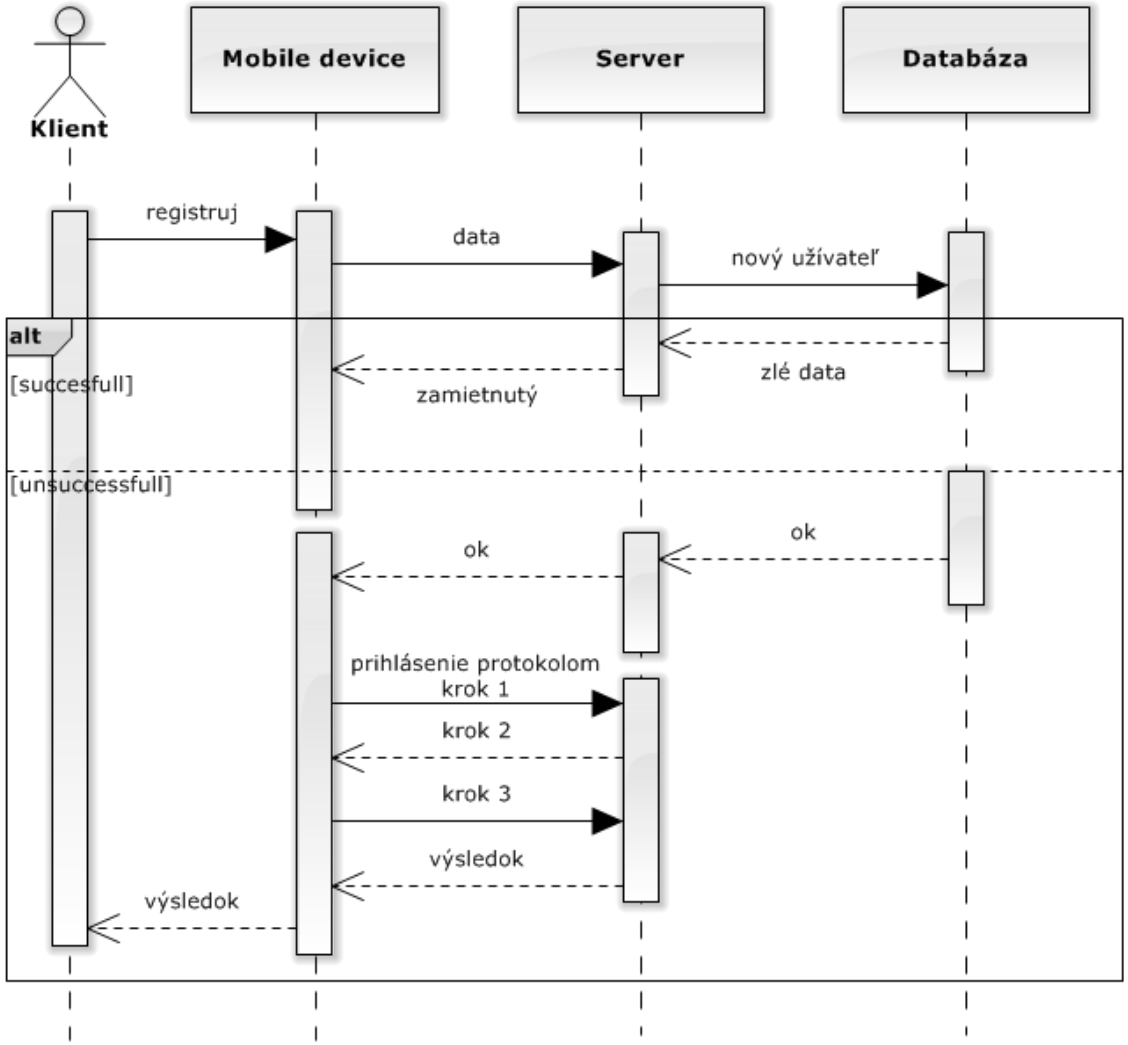
Nadviazanie komunikácie medzi dvomi klientmi je možné len v prípade, že sú obaja prihlásení v aplikácii a majú prístup ku svojmu súkromnému kľúču. V opačnom prípade nie je možné načítať zašifrovaný súkromný kľúč a nadviazať komunikáciu. V prípade ak je online len jeden užívateľ a bola nadviazaná komunikácia už v minulosti, použije sa dohodnutý symetrický kľúč. Tento kľúč sa bude meniť za podmienky, že sú obaja účastníci online.

Prihlásenie na server bude prebiehať vytvorením zabezpečeného kanálu medzi klientom a serverom. Keďže prezentované protokoly zabezpečujú vzájomnú autentifikáciu, prihlasovanie heslom nie je treba.

V takomto type aplikácie by mali byť správy doručované v reálnom čase. Na takéto účely boli pre systém Android vytvorené služby ako GCM [8]. Využitie nedôveryhodnej tretej strany by však nebolo kryptograficky bezpečné. Tieto služby využívajú na nadviazanie tzv. real-time spojenia sockety. Preto sa vždy po prihlásení do aplikácie nadviaže spojenie pomocou socketov. V systéme Android je možné toto spojenie udržovať aj pri nepoužívaní aplikácie vďaka Android servisom. Ide o proces bežiaci na pozadí, ktorý dokáže vykonávať úlohy ako je udržovanie spojenia so serverom.

Proces naviazania spojenia s so serverom je znázornený na diagrame v Obr.4. Užívateľ sa registruje zadaním potrebných osobných údajov a vygenerovaním kľúčového páru McElieceovho kryptosystému. Dáta sú spracované na serveri a uložené do databázy. V prípade zle zadaných údajov alebo nesprávneho kľúča, server pošle adekvátnu odpoveď zobrazenú do mobilnej aplikácie.

V opačnom prípade sa s užívateľom vytvorí zabezpečené spojenie, ktoré taktiež nahradzuje prihlásenie heslom, keďže použitý protokol podporuje vzájomnú autentifiká-



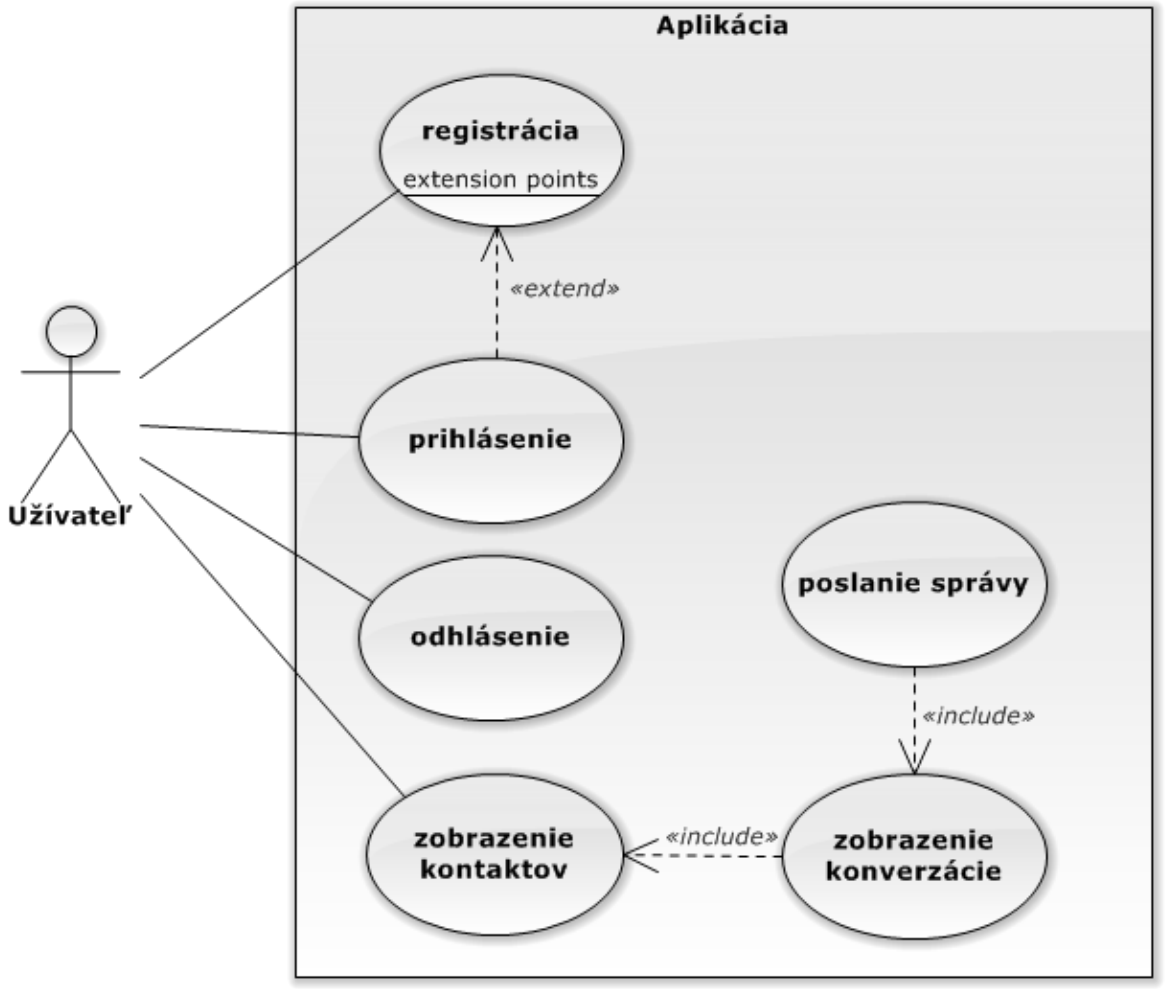
Obrázok 4: Sekvenčný diagram registrácie a prihlásenia užívateľa

ciu. Verejný kľúč užívateľa už je aktualizovaný v databáze na strane servera. Môžu sa teda simulovať jednotlivé kroky spomínaného protokolu. Ďalej môže užívateľ ľubovoľne používať aplikáciu.

2.2 Aplikácia

Pri nainštalovaní aplikácie je od užívateľa vyžadovaná registrácia. Pre užívateľa je vytvorený nový pár kľúčov. Po vyplnení registračných údajov sú dáta poslané na server. Aplikácia bude môcť pridávať do zoznamu užívateľov, s ktorými sa bude dať komunikovať. Užívateľ by mal byť schopný v aplikácii vykonávať aktivity z Obr. č.5.

Užívateľ sa bude môcť prihlásiť až po vykonaní registrácie pozostávajúcej z vyplnení



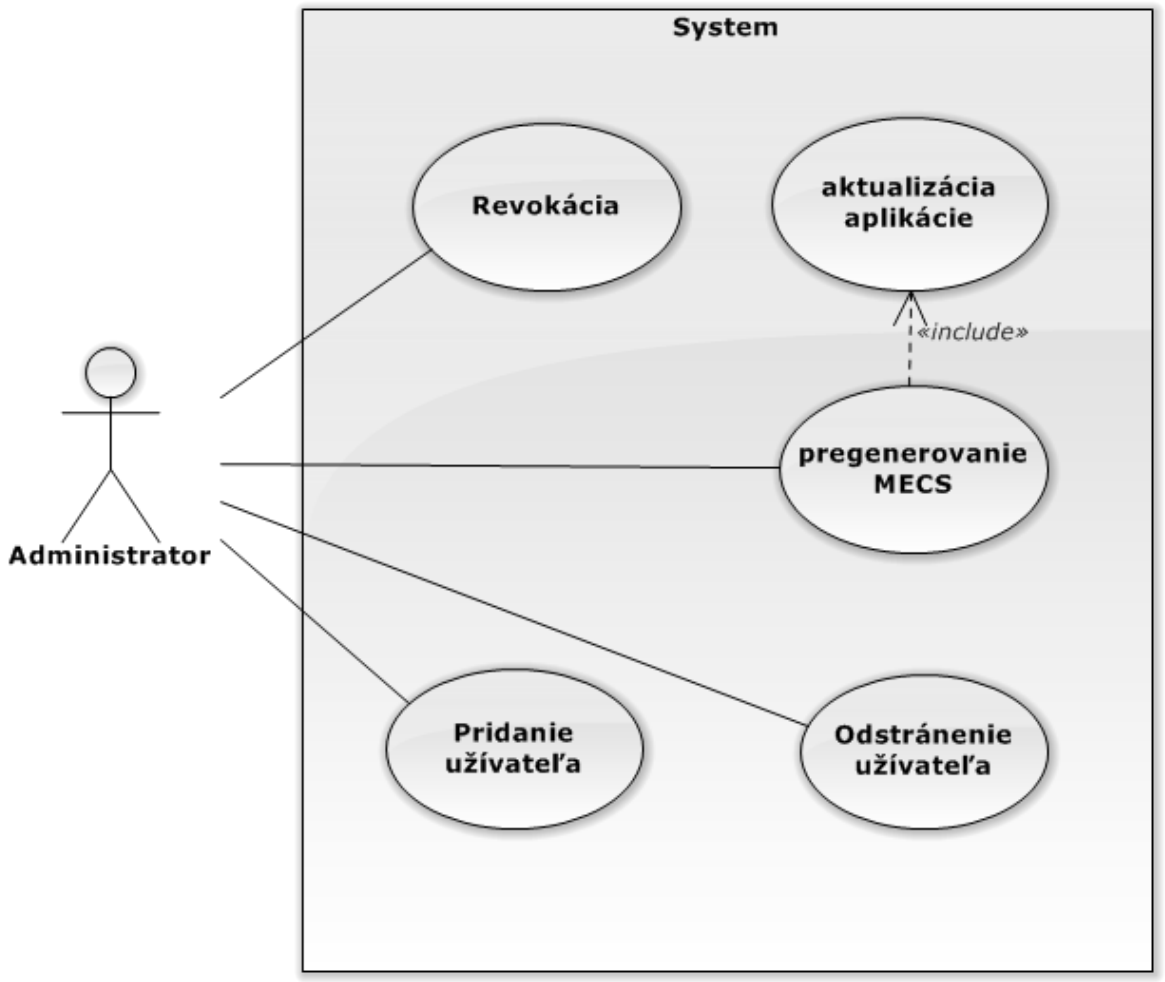
Obrázok 5: Use-case diagram pre aplikáciu

užívateľského mena a zaslaním na server spolu s verejným kľúčom. Klient sa môže odhlásiť a znovu registrovať, keďže súkromný kľúč bude pri odhlásení vymazaný. Po prihlásení do aplikácie sa užívateľovi zobrazí zoznam kontaktov. Konverzácia medzi užívateľmi sa zobrazí po kliknutí na daný kontakt. V tomto okne sa taktiež bude dať posielať nová správa.

2.3 Server

Server vytvára spojenia medzi užívateľmi a spravuje ich pomocou databázy. Server by mal byť schopný vykonávať aktivity z Obr. č.6. Na serveri sa bude môcť dať upravovať databáza, čiže pridávať alebo odstraňovať užívateľov. Nový užívateľ je pridaný na základe jeho registračnej žiadosti, v ktorej by nemal chýbať jeho verejný kľúč a užívateľské meno.

V prípade skompromitovania súkromného kľúča užívateľa by mal server obsahovať zoznam nedôveryhodných užívateľov a prestať s nimi komunikovať. Princíp je rovnaký ako pri CRL [3], ktorý sa využíva na podobné účely pri práci s certifikátmi.



Obrázok 6: Use-case diagram pre systém

Ak by bol odcudzený privátny kľúč servera, v takomto prípade by mal byť vytvorený nový kľúčový pár a vykonaná aktualizácia aplikácie obsahujúca nový verejný kľúč servera. Server bude mať teda k dispozícii aplikáciu na generovanie nového kľúčového páru.

3 Implementácia

V tejto časti je popísaná implementácia servera, mobilnej aplikácie, databáza a iné. Prevažne sa budeme venovať použitej knižnici pre kryptografické účely a použitým technológiám. Aplikácia aj serverová časť je naprogramovaná v jazyku java.

3.1 Kryptografická časť

V knižnici SpongyCastle sa dá použiť šifrovanie viacerých CCA2-bezpečných konverzií McElieceovho kryptosystému. Knižnica žiaľ neposkytuje prostriedky na manipuláciu s kľúčovým párom. Nedá sa teda zapísať do čitateľnej formy a ukladať či prenášať ako dáta medzi aplikáciami. Táto knižnica sa avšak stále rozvíja a jej nové komponenty sú dostupné na [12] ako beta verzia. Ide však o projekt BouncyCastle, ktorý ešte nie je dostupný v SpongyCastle knižnici. Tieto McEliece časti boli odskúšané aj na Android zariadeniach a sú teda vhodné aj pre tento účel. Použité sú ako na strane servera, tak aj na strane klienta (Android zariadenie).

3.1.1 McEliece

Generovanie kľúčov

BouncyCastle knižnica poskytuje provider, ktorý pridáva potrebné algoritmy do java security balíku. Pomocou tohto balíka sa ďalej robia všetky veci súvisiace so šifrovaním. Najskôr je treba nastaviť inštancie všetkých potrebných tried na zvolený algoritmus. Taktiež treba zvoliť parametre pre kryptosystém. Ide o veľkosť generačnej matice a počet chýb, ktoré dokáže kód opraviť.

```
Security.addProvider(new
    org.bouncycastle.pqc.jcajce.provider.BouncyCastlePQCProvider());

kpg = KeyPairGenerator.getInstance("McElieceKobaraImai", "BCPQC");
cipher = Cipher.getInstance("McElieceKobaraImai", "BCPQC");
params = new McElieceCCA2KeyGenParameterSpec(11, 50);
keyFactory = KeyFactory.getInstance("McElieceKobaraImai");
```

Funkcia na generovanie kľúčov potom vyzerá nasledovne.

```
public KeyPair generateKeyPair(){
   keyPair = kpg.genKeyPair();
return keyPair;}
```

Trieda KeyFactory ostane inicializovaná pre manipuláciu s kľúčmi. Cipher sa ďalej využíva na šifrovanie a dešifrovanie.
Kľúče sú uchovávané ako reťazce v base 64 kódovaní, ktoré prepisuje binárne dáta do ASCII znakov. Toto kódovanie je súčasťou BouncyCastle knižnice. Konverzia kľúčov na reťazce vyzerá nasledovne.

```
Base64.toBase64String(keyPair.getPublic().getEncoded());
Base64.toBase64String(keyPair.getPrivate().getEncoded());
```

Šifrovanie

Pred šifrovaním verejným kľúčom ho treba zostaviť z reťazca do premennej určenej pre verejné kľúče. Táto časť nebola možná v nekompletnej implementácii Spongy Castle. Je teda dostupná len v beta verzii. Najskôr sa kľúč dekóduje späť na pole bytov. Tie sú vstupom do X509 špecifikácie pre verejné kľúče, z ktorej je kľúč vygenerovaný. V java security balíčku sa nastavujú jednotlivé módy šifier. Nastaví sa teda šifrovanie a vykoná sa finálna fáza. Výstupom z šifrovania je pole bytov zašifrovaného textu. S ním sa dá taktiež pracovať pomocou base 64 kódovania.

```
byte[] publicBytes = Base64.decode(publicK);
X509EncodedKeySpec keySpec = new X509EncodedKeySpec(publicBytes);
PublicKey pubKey = keyFactory.generatePublic(keySpec);

cipher.init(Cipher.ENCRYPT_MODE, pubKey, params);
 byte[] cBytes = cipher.doFinal(plaintext.getBytes());
```

Dešifrovanie

Dešifrovanie je takmer identické ako šifrovanie, s tým rozdielom, že sa dešifruje súkromným kľúčom, ktorý má PKCS8 špecifikácie. Mód šifry sa nastaví na dešifrovanie. Výstupom dešifrovanie je taktiež pole bytov.

```
byte[] privateBytes = Base64.decode(privateK);
PKCS8EncodedKeySpec privSpec = new PKCS8EncodedKeySpec(privateBytes);
PrivateKey privKey = keyFactory.generatePrivate(privSpec);

cipher.init(Cipher.DECRYPT_MODE, privKey, params);
byte[] dBytes = cipher.doFinal(Base64.decode(ciphertext));
```

3.1.2 Advanced Encryption Standard

Keďže McElieceov kryptosystém slúži v našom prípade k dohode na symetrickom kľúči, je v implementácii použitý aj AES. Táto šifra je bežne dostupná v java balíčkoch a nie je potreba použitia externej knižnice. Je používaný 128-bitový kľúč dĺžky 16 ASCII znakov, ktoré sú zostavené na základe spomínaného protokolu na výmenu symetrického kľúča. Použitie je podobné ako pri McElieceovej šifre s menšími rozdielmi. Ide o definovanie kľúča, ktorý je v tomto prípade len jeden.

Šifrovanie

Pri špecifikácii kľúča ide o celý 16-znakový reťazec v bytoch a názov šifry. Výstupom z šifrovania, ako ja dešifrovania je pole bytov. S ním pracujeme taktiež pomocou base 64 kódovania.

```
SecretKeySpec keySpecification = new
    SecretKeySpec(encryptionKey.getBytes("UTF-8"), "AES");
Cipher cipher = Cipher.getInstance("AES");
cipher =cipher.init(Cipher.ENCRYPT_MODE, keySpecification);

byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
```

Dešifrovanie

Dešfrovanie je iba s jedným rozdielom a to je zmena módu šifry.

```
cipher =cipher.init(Cipher.DECRYPT_MODE, keySpecification);
```

3.2 Server

Implementácia servera sa skladá z troch hlavných častí. Ide o Web-service, ktorý predstavuje nezabezpečenú komunikáciu. Ďalej tu je správa socketov, ktoré komunikujú s klientmi zabezpečeným kanálom. A v neposlednej rade aj prepojenie s databázou, v ktorej sú uchovávané dáta o užívateľoch.

3.2.1 Web-service

Na prvotnú komunikáciu medzi klientom a serverom je používaný Web-service. Ide o RESTful Web-service, ktorý slúži primárne na výmenu dát.

RESTful Web-service - ide o architektonický štýl určený k tomu aby fungoval čo najlepšie na Webe. Je to docielené použitím správnych dátových typov a komponentov vrámci použitého protokolu (typicky HTTP). Podľa [13] je založený na týchto princípoch:

- Identifikácia zdrojov pomocou URI - jednotlivé zdroje sú sprístupnené klientovi cez odkazy.
- Jednotné rozhranie - je definované týmito metódami:
  - PUT - slúži na vytvorenie nových zdrojov,
  - GET - získava aktuálny stav zdrojov,
  - POST - aktualizuje zdroje,
  - DELETE - slúži na vymazanie nejakých zdrojov.
- Seba-opisujúce správy - ide o definovanie obsahu jednotlivých správ. Môže to byť napr. XML, čistý text, PDF, JSON ...
- Stavové interakcie prostredníctvom hypertextových odkazov - stav môže byť vložený do odpovede aby na niečo poukázal.

Náš Web-service je spustený na aplikačnom serveri Glasfish 4.1. Obsahuje tri metódy slúžiace na registráciu, prihlásenie a vyhľadávanie užívateľov. Tieto metódy sú typu POST, ktorá v tomto prípade posiela dáta na server v JSON formáte. Ide o špeciálne upravený reťazec, ktorý mapuje dáta k tzv. kľúčom.

Každá z metód je definovaná jej typom, adresou odkazu a definíciou vstupných dát. Metódy môžeme popísať nasledovne:

- registrácia

```
@POST
@Path("/register")
@Consumes(MediaType.APPLICATION_JSON)
```

  Pokračuje sa naprogramovanou funkcionalitou. V tomto prípade sa skontroluje či zaslané meno už existuje. Ak nie vytvorí sa nový záznam v databáze spolu so zaslaným verejným kľúčom.

- prihlásenie

```
@POST
@Path("/login")
@Consumes(MediaType.APPLICATION_JSON)
```

Jedná sa iba o kontrolu užívateľského mena. Pri jeho správnosti sa vytvorí socket spojenie s klientom, v ktorom prebehne prihlásenie protokolom.

- vyhľadávanie

```
@POST
@Path("/search")
@Consumes(MediaType.APPLICATION_JSON)
```

Metóda vyhľadáva v databáze zadané užívateľské meno a vracia všetky nájdené výsledky.

3.2.2 Socket komunikácia

Komunikácia medzi klientom a serverom je sprostredkovaná pomocou socketov. Ide o internetové spojenie medzi dvoma bodmi [14]. Na serveri je vytvorený koncový bod, ktorý čaká na pripojenie klienta. Toto pripojenie je definované IP adresou a vybraným portom na serveri. Socket sa na serveri vytvorí ako:

```
ServerSocket listener = new ServerSocket(PORT);
```

Kde PORT je číslo portu, na ktorom počúva socket. Ďalej sa čaká na pripojenie od klienta príkazom:

```
Socket socket = socketlistener.accept();
```

Každé nové spojenie je riadené nezávisle na sebe. Po pripojení je vytvorené nové vlákno pre každého klienta zvlášť. Najskôr prebehne prihlásenie protokolom ak je treba. Ďalej sa v cykle spracúvajú správy, ktoré sú rozposielané medzi klientmi vstupno/výstupným tokom:

```
socket.getInputStream();  //->vstup
socket.getOutputStream(); //->výstup
```

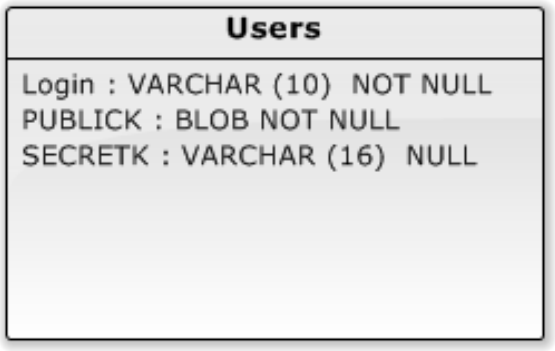
Po stratení spojenia sa socket uzavrie:

```
socket.close();
```

3.2.3 Databáza

Ide o identifikačné údaje registrovaných užívateľov. Databáza je vytvorená pomocou Java DB (Derby) v Netbeans IDE. Databáza na serveri predstavuje jednu tabuľku. Jej UML diagram môžme vidieť na Obr. č. 7. Jednotlivé atribúty sú:

- LOGIN - unikátne užívateľské meno zadané pri registrácii. Je uložené do premennej VARCHAR maximálnej veľkosti 10 znakov.
- PUBLICK - verejný kľúč užívateľa zadaný pri registrácii. Veľkosť kľúča je 137 kB. Táto hodnota sa nedá uložiť ako klasický reťazec, pretože je príliš veľký. Preto je uložený do BLOB súboru. Ide o prepis do binárnych dát používaných pre veľké reťazce alebo obrázky.
- SECRETK - symetrický kľúč šifry AES, ktorým je šifrovaná komunikácia medzi užívateľom a serverom. Kľúč je predvolene nastavený ako null. Doplnený je po prihlásení užívateľa protokolom. Je uložený do premennej VARCHAR veľkosti 16 znakov, keďže veľkosť kľúča je 128 bitov.



Obrázok 7: ER diagram pre databázu

Pripojenie k databáze je realizované pomocou Apache Derby knižnice. Údaje sú do databázy vkladané a vyhľadávané pomocou SQL dopytov prostredníctvom spomínanej knižnice. V kóde sú priamo využívané tieto dopyty:

- Vloženie nového užívateľa,
- aktualizácia AES kľúča,
- načítanie AES kľúča,
- načítanie verejného kľúča,
- overenie duplicity užívateľského mena,
- vyhľadávanie užívateľov so zadaným menom.

3.2.4 Generovanie kľúčov

Tak isto ako aj klienti, aj server potrebuje svoj pár kľúčov McElieceovho kryptosystému. Tie sú vygenerované v nezávislej aplikácii. Jej výstupom sú dva textové súbory: "PUBLIC KEY.txt" a "PRIVATE KEY.txt". V každom sa nachádza jeden kľúč. Súkromný kľúč sa uchováva iba na serveri a je používaný na dešifrovanie správ od klienta. Verejný kľúč je vložený do klientskej aplikácie už v inštalačnom balíčku. Tým sa dosiahne autenticita kľúča, keďže ten nie sťahovaný z externých zdrojov. Pri skompromitovaní servera sa aktualizuje aplikácia s novým verejným kľúčom.

3.3 Klientská aplikácia

Prezentačnú časť tvorí aplikácia pre mobilné zariadenie Android nazvaná McChat. Jedná sa o tzv. "messenger", ktorý slúži na posielanie "real-time" správ medzi užívateľmi. Bezpečnosť Android aplikácií je založená na povoleniach, ktoré aplikácia vyžaduje. Tieto povolenia sú definované v manifeste aplikácie, ktorá definuje jej základné prvky. V našom prípade sú to tieto povolenia:

- android.permission.INTERNET,
- android.permission.WAKE_LOCK.

Tieto povolenia sú potrebné pre pripojenie na internet a funkčnosť aplikácie v režime spánku. Používateľ musí schváliť tieto povolenia pri inštalácii aplikácie.

Po spustení aplikácie sa vyžaduje prihlásenie alebo registrácia užívateľa. Užívateľ začne registráciu zadaním svojho užívateľského mena. Na mobilnom zariadení sa začne generovať súkromný a verejný kľúč nového užívateľa. Verejný kľúč spolu s menom sú zaslané na server. Pri úspešnej registrácii je súkromný kľúč uložený v telefóne a môže prebehnúť prihlásenie. Tieto údaje sa v telefóne ukladajú do tzv. "Shared preferences".

Shared preferences - ide o úložný priestor aplikácie v mobilnom zariadení. Každá aplikácia má svoj vlastný. Iné aplikácie k nej nemajú prístup ak nie je povolený samotnou aplikáciu. Z toho dôvodu do neho môžeme ukladať aj súkromné dáta.

V našom prípade sú do neho ukladané tieto hodnoty:

- Súkromný kľúč,
- užívateľské meno,
- AES kľúč pre server spojenie,
- čas vytvorenia AES kľúča.

3.3.1 Prihlásenie (klient-server spojenie)

Po vytvorení socket spojenia prebehne prihlásenie protokolom, ktoré môžme vidieť na Obr. č. 8. Aplikácia skontroluje, či je v telefóne platné heslo so serverom. Heslo má nastavenú expiráciu na sedem dní. V prípade neplatného hesla sa na server pošle správa s popisom. Ak je heslo platné, zašifruje sa ním reťazec "Hello server". Server dešifruje reťazec kľúčom, ktorý má uložený v databáze. Ak je všetko v poriadku, nevyžaduje sa prihlásenie protokolom a klientovi sa zobrazí chat.

Protokol začína server. Ten si načíta z databázy verejný kľúč klienta, ktorým šifruje prvú protokolovú správu. Na strane servera sa vygeneruje dočasný McEliecov kľúčový pár, ktorým sa šifruje časť kľúča kvôli spomínanej "forward-secrecy". Tento proces je výpočtovo náročný, preto sa robí na serveri. Taktiež vygeneruje prvú časť kľúča k1, ktorá má 6 znakov a náhodné číslo r1. Spolu to pošle klientovi. Pre jednoduchosť a prehľadnosť sú tieto časti pozabaľované do JSON objektov.

Klient všetko dešifruje svojím súkromným kľúčom a uloží do premenných. Ďalej vygeneruje Ďalšie časti definované protokolom a pošle späť na server zašifrované jeho verejným kľúčom. Ten sa načíta z /assets priečinka, v ktorom je predvolene uložený ako "PUBLIC KEY.txt".

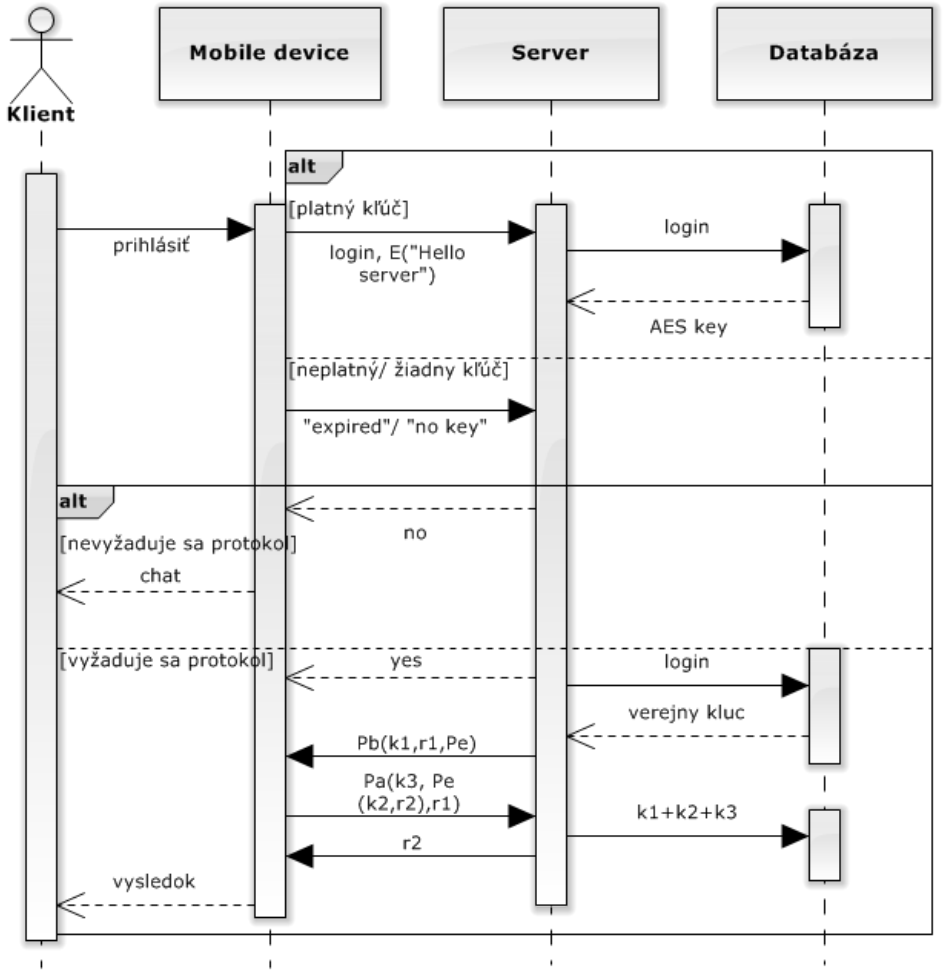
Po zbehnutí posledného kroku sa skontrolujú na oboch stranách vygenerované čísla. Ak je všetko v poriadku, server si aktualizuje databázu s novým AES kľúčom. Klient si ho uloží do úložného priestoru aplikácie. Ako bolo spomínané, tento protokol zaručuje vzájomnú autentifikáciu, preto nie je potreba prihlásenie zadávaním hesla. Ďalej sú všetky správy medzi serverom a klientom šifrované týmto heslom.

3.3.2 Štruktúra komunikácie

Po prihlásení, server primárne preposiela správy medzi klientmi. Server musí vedieť komu správu preposlať a tak isto je správa v niektorých prípadoch smerovaná pre server. Aby vedel server tieto správy rozlíšiť, sú vkladané do JSON objektov. Môžeme tieto správy zaradiť do týchto skupín:

- Pre server - ide o vyžiadanie verejného kľúča nejakého užívateľa,
- pre užívateľa - server iba preposiela,
  - bežná správa,
  - protokolová správa.

V prípade, že chcú dvaja klienti naviazať zabezpečené spojenie **(klient-klient spojenie)**, musia si zo servera vyžiadať ich verejné kľúče aby mali čím šifrovať protokolové



Obrázok 8: Sekvenčný diagram prihlásenia užívateľa

správy. Štruktúra JSON objektu je potom nasledovná.

```
{"WHAT":"GET_PUBLIC","TO":"SERVER","PROT":"POA","OF":"user_name"}
```

Pole PROT definuje, ktorá časť protokolu sa má vykonať pri prijatí protokolovej správy. Pre príklad si ukážeme ešte jednu protokolovú správu.

```
{"TO":"user_name","PROT":"P1B","MESSAGE":"DDXSXCRgmrsTilpS0WSpi4Qms7Iwxay0..."}
```

Môžeme vidieť, že správa je zašifrovaná verejným kľúčom príjemcu. Po úspšenom zbehnutí protokolu vyzerá štruktúra JSON objektu nasledovne. Klient dešifruje hodnotu poľa MESSAGE, v ktorom je správa pre užívateľa.

```
{"TO":"user_name","PROT":"NO","MESSAGE":"KeSJfzP3f3rpgitMAWTOoA=="}
```

3.3.3 GUI

Užívateľ začne pridaním ďalšieho užívateľa do "priateľov" stlačením tlačidla na úvodnej obrazovke. Zobrazí sa mu obrazovka na vyhľadávanie užívateľov ako na Obr. č. 9. Po pridaní sa mu zobrazí v zozname s nezabezpečeným spojením. Kliknutím na neho začne prebiehať protokol medzi klientmi.

Obrázok 9: Ukážka aplikácie - vyhľadávanie

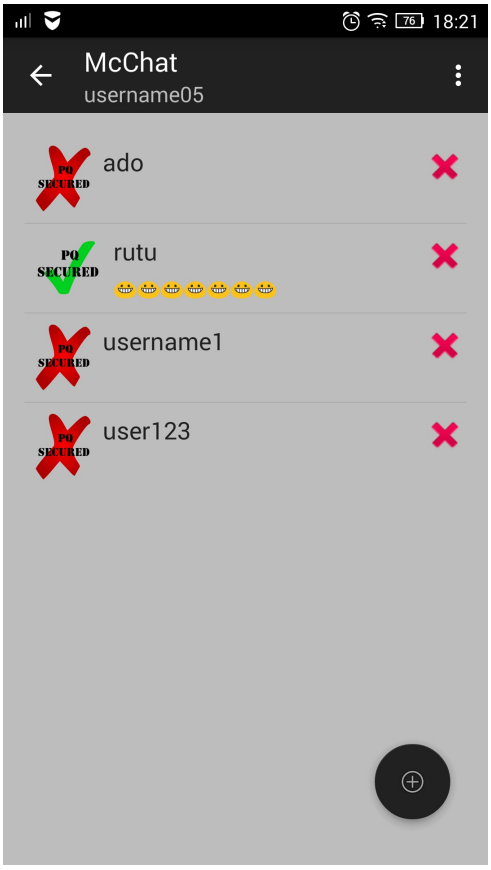
Po zbehnutí protokolu sa dáta uložia do SQLite databázy, ktorá je súčasťou Android zariadení. Pridaní užívatelia sú v tabuľke "friends", ktorá obsahuje atribúty:

- LOGIN - užívateľské meno. Je uložené do premennej VARCHAR.
- KEY - AES kľúč, ktorým sa šifruje toto spojenie. Je uložený do premennej VARCHAR.
- CREATED - dátum poslednej aktualizácie kľúča. Je uložený do premennej INTEGER.

Pred začatím konverzácie sa vždy kontroluje dátum poslednej aktualizácie kľúča. Po vypršaní 7 dní sa vyžaduje nový.

Užívatelia so zabezpečeným spojením sú označení zelenou fajkou. Naopak nezabezpečené spojenie je označené červeným krížikom ako na Obr. č. 10. Jednotlivé konverzácie sú ukladané to tabuľky "history". Obsahuje tieto atribúty:

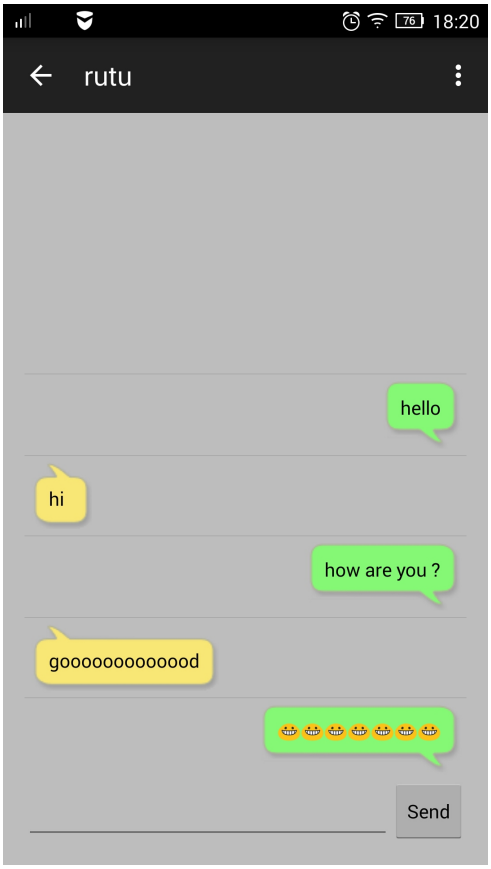
- LOGIN - užívateľské meno, s ktorým sa komunikuje. Je uložené do premennej VARCHAR.
- MESAGE - Správa je uložená do premennej VARCHAR.
- SIDE - definuje, či bola správa odoslaná alebo prijatá. Je uložená do premennej INTEGER.



Obrázok 10: Ukážka aplikácie - konverzácie

Po odoslaní správy je správa spracovaná serverom. Ten zistí, či je daný užívateľ online. Ak je online, tak správu pošle ako jemu, tak aj naspäť odosielateľovi, aby vedel, že

bola odoslaná. Správy odchytáva "service" aplikácie. Ide o komponent Android aplikácií, ktoré bežia na pozadí a nemajú žiadne používateľské rozhranie. Ten ukladá prijaté správy do databázy alebo posiela odpovede na protokolové požiadavky. Po prijatí správy ukáže v mobilnom zariadení notifikáciu a pošle broadcast do celej aplikácie, aby sa mohli aktualizovať jednotlivé komponenty. Komunikačné GUI je zobrazené na Obr. č. 11.



Obrázok 11: Ukážka aplikácie - chat

4 Testovanie

Testovanie mobilnej aplikácie prebiehalo na dvoch mobilných zariadeniach:

- HTC One X+

```
971 MB RAM
ARMv7 Processor rev9
Android v. 4.3.1
```

- Lenovo Vibe X2

```
2 GB RAM
Octa-core (Little: 1.69GHz, Big: 2.0GHz)
Android v. 4.4.2
```

Aplikácia najskôr nefungovala na mobilnom zariadení HTC. Zistilo sa, že príčinou je nedostatočná veľkosť pamäte priradenej aplikácii. Pre HTC je jej počiatočná hodnota 64 MB. Pre Lenovo je to 195 MB. Jedinou možnosťou ako priradiť aplikácii viacej pamäte na OS Android je priradenie tohoto riadku do manifestu aplikácie:

```
android:largeHeap="true"
```

Ani toto nastavenie nezaručuje dostatočne veľkú pridelenú pamäť, ale pre testovacie telefóny to stačí. Údaje o veľkosti pridelenia pamäte sa dajú získať priamo z kódu a to príkazmi:

```
ActivityManager am = (ActivityManager) getSystemService(ACTIVITY_SERVICE);

int memoryClass = am.getMemoryClass();
int largeMemoryClass = am.getLargeMemoryClass();
```

Jednotlivé hodnoty pre testované telefóny sú potom:

- HTC One X+

```
memoryClass: 64 MB
largeMemoryClass: 256 MB
```

- Lenovo Vibe X2

```
memoryClass: 195 MB
largeMemoryClass: 512 MB
```

4.1 Merania

Testované boli viaceré hodnoty (RAM, čas, CPU) pre rôzne časti aplikácie zvlášť. Merania boli zamerané na vykonávanie protokolovej časti aplikácie, v ktorej sa vykonáva generovanie kľúčov, šifrovanie a dešifrovanie McElieceovým kryptosystémom. Protokol je mierne rozdielny pre klient-server a pre klient-klient spojenie. Klient-server protokol je vykonaný na jeden raz, pretože bez neho sa nie je možné prihlásiť do aplikácie. Klient-klient protokol je vykonávaný vo viacerých častiach, ktoré medzi klientmi preposiela server. Taktiež je dôležité, ktorá strana protokol vyžaduje ako prvá. Na strane žiadateľa sa vykonáva výpočtovo najnáročnejšie generovanie dočasných kľúčov. Preto sú merané jednotlivé časti protokolu pre oba scenáre. Parametre pre McElieceov kryptosysystém boli nastavené na hodnoty $n = 50$, $t = 11$. Tieto hodnoty sú nemenné vrámci použitej implementácie.

Merané časti protokolu v závislosti od McElieceového kryptosystému sú:

- Step 1 A - generovanie dočasného kľúčového páru, 1x šifrovanie údajov
- Step 2 A - 2x dešifrovanie údajov
- Step 1 B - 1x dešifrovanie údajov
- Step 2 B - 2x šifrovanie údajov

V jednotlivých krokoch sa taktiež vykonávajú aj iné úlohy, ktoré môžeme zanedbať pri meraniach. Sú to napr. práca s JSON objektom, AES úlohy(generovanie častí kľúča, šifrovanie, dešifrovanie)... V poslednej časti protokolu nie sú vykonávané úlohy s McElieceovým kryptosystémom, preto ju nezahrňujme do merania.

4.1.1 Časová náročnosť

Boli merané časy jednotlivých častí protokolu ako vidno v Tab. č. 1 a 2. Merania boli vykonané v oboch prípadoch,t.j. keď zabezpečenie spojenia vyžaduje mobilné zariadenie Lenovo a naopak. Celkový čas A, resp. B predstavuje trvanie celého cyklu protokolu na danom telefóne. V tomto meraní je zahrnuté čakanie na odpoveď z druhej strany a taktiež oneskorenie pri komunikácii so serverom.

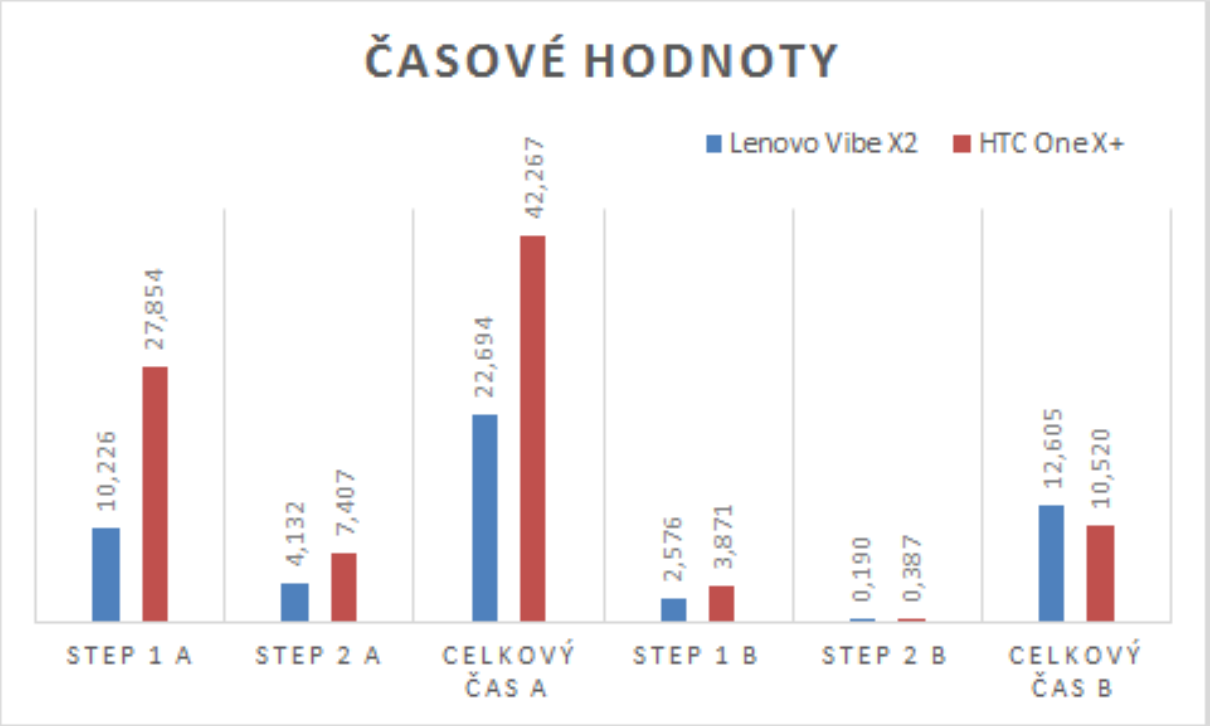
	Časové údaje (v sec)					
	Lenovo Vibe X2			HTC One+		
Meranie	Step 1 A	Step 2 A	Celkový čas A	Step 1 B	Step 2 B	Celkový čas B
1	11.259	4.213	24.107	4.137	0.472	11.11
2	6.746	4.003	18.665	3.92	0.335	10.2
3	15.092	4.044	27.381	3.901	0.409	10.339
4	9.664	4.01	22.168	3.891	0.326	10.895
5	9.822	4.401	21.446	3.697	0.342	10.768
6	8.775	4.121	22.394	3.681	0.436	9.807
Priemer	10.226	4.132	22.6935	3.871	0.39	10.52

Tabuľka 1: Časové merania prípad č.1

	Časové údaje (v sec)					
	HTC One+			Lenovo Vibe X2		
Meranie	Step 1 A	Step 2 A	Celkový čas A	Step 1 B	Step 2 B	Celkový čas B
1	31.984	8.033	45.643	2.567	0.201	11.924
2	27.741	7.242	41.502	2.578	0.233	11.196
3	57.593	7.592	70.923	2.589	0.173	12.127
4	16.822	7.4	33.224	2.56	0.191	14.423
5	17.277	7.223	32.854	2.533	0.135	13.692
6	15.708	6.951	29.456	2.628	0.207	12.267
Priemer	27.854	7.407	42.267	2.576	0.19	12,605

Tabuľka 2: Časové merania prípad č.2

Z výsledkov môžeme vidieť, že mobilné zariadenie Lenovo zvláda jednotlivé výpočty rýchlejšie ako HTC. Vyplýva to z hardvérového vybavenia oboch zariadení. Ako vidno z tabuliek, najrýchlejšou časťou je Step 2 B. Ide o šifrovanie údajov, ktoré je v McElieceovom kryptosystéme rýchlejšie ako dešifrovanie údajov. To môžeme vidieť v stĺpcoch Step 2 A a Step 1 B, ktoré predstavujú dešifrovanie údajov. Šifrovanie je niekoľkonásobne rýchlejšie. Najzdĺhavejšou časťou je Step 1 A, v ktorom prebieha generovanie údajov. Ide o výpočtovo najnáročnejšiu časť protokolu. V tejto časti sa nachádzajú mierne výkyvy pri meraniach, ktoré môžu byť spôsobené systémovými procesmi v mobilných zariadeniach.



Obrázok 12: Graf porovnania časových meraní

V grafe na Obr. č. 12 môžme vidieť dôležitosť použitého hardvéru, ktorá sa prejavuje najmä pri generovaní a šifrovaní údajov.

4.1.2 Využitie RAM

Jednotlivé merania využitia pamäte RAM z Tab. č. 3 a 4 boli merané súvisle s časovými meraniami. Ich hodnoty sú adekvátne časovým údajom, t.j. časové výkyvy vyžadujú viacej pamäte RAM, resp. menej.

Využitie pamäte RAM nezávisí od výpočtového výkonu telefónu. Pri porovnaní tabuliek môžeme vidieť, že mobilné zariadenia využívajú približne rovnako veľké množstvo

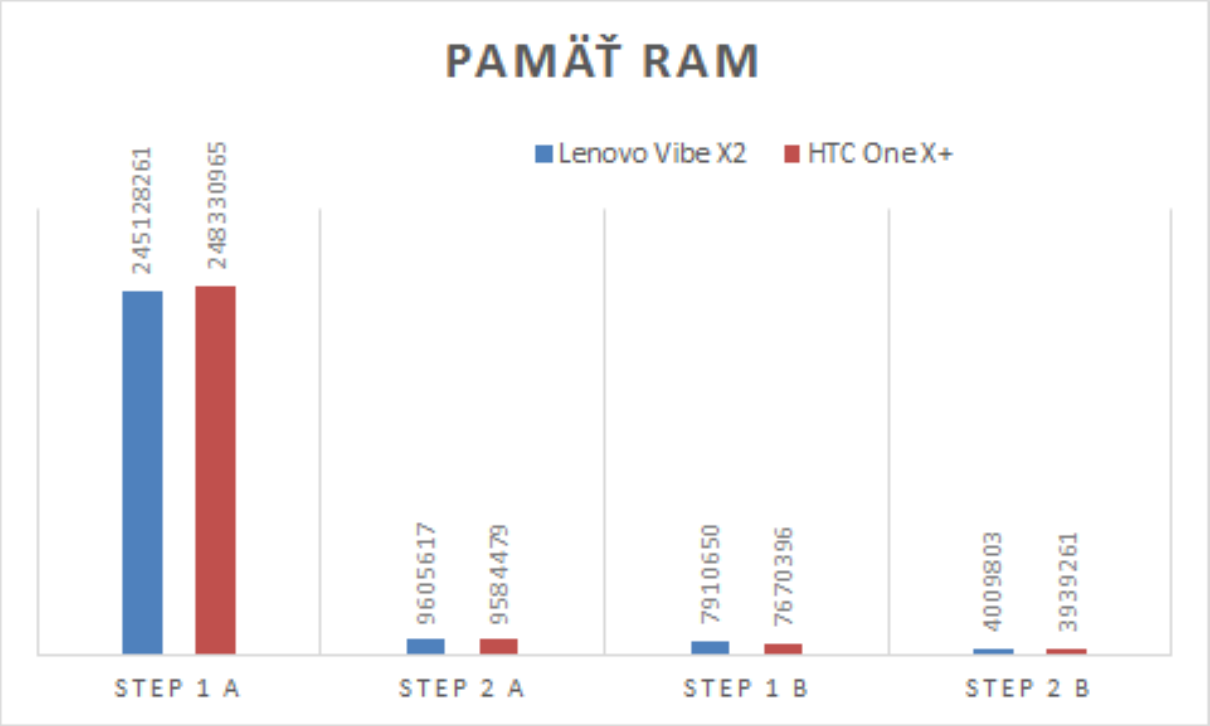
	využitie RAM (v B)			
	Lenovo Vibe X2		HTC One+	
Meranie	Step 1 A	Step 2 A	Step 1 B	Step 2 B
1	258123947	9606461	8194757	3944450
2	113049747	9605679	7617375	3938440
3	444740008	9606323	7574023	3938018
4	231837085	9603931	7540636	3938203
5	233466938	9605884	7542218	3938102
6	189551842	9605421	7553368	3938351
Priemer	245128261	9605617	7670396	3939261

Tabuľka 3: Merania RAM prípad č.1

	využitie RAM (v B)			
	HTC One+		Lenovo Vibe X2	
Meranie	Step 1 A	Step 2 A	Step 1 B	Step 2 B
1	278268047	9561662	8436317	4014844
2	270243785	9563258	7822818	4009186
3	608604114	9561175	7966864	4008212
4	136241198	9564165	7625021	4008743
5	96069063	9566267	7861460	4008870
6	119775807	9563523	7751419	4008960
Priemer	251533669	9563342	7910650	4009803

Tabuľka 4: Merania RAM prípad č.2

pamäte. Mobilné zariadenie HTC má pritom pridelenú menšiu časť pre používanie aplikácie. To môže byť jedna z príčin pomalších výpočtov. Jednotlivé časti protokolu, ktoré sú zdĺhavejšie, využívajú väčšie množstvo pamäte. Najviac pamäte využíva generovanie kľúčov Step 1 A.



Obrázok 13: Graf porovnania využitia pamäte RAM

Z grafu na Obr. č. 13 je vidno, že veľkosť využívanej pamäte nezávisí od použitého zariadenia. Taktiež si môžme všimnúť výpočtovú náročnosť generovania parametrov, ktorá je výrazne vyššia od ostatných častí protokolu v závislosti od využitia pamäte RAM.

4.2 Vyhodnotenie

Pri meraniach sa narazilo na viacero podstatných faktov v súvislosti s používaným hardvérom a implementáciou. Boli použité dve mobilné zariadenia s rozdielnym výpočtovým výkonom. Zariadenie s horším hardvérom rapídne zvyšuje dĺžku trvania protokolu. Jedná sa najmä o generovanie kľúčov McElieceovho kryptosystému. Generovanie sa taktiež odlišuje od ostatných častí extrémnou spotrebou pamäte RAM. Táto časť je tiež najnáročnejšia na výkon procesoru. Na zariadeniach s procesorom s nižšou taktovacou frekvenciou, resp. menej jadrami, môžeme pozorovať ‘zamrznutie zariadenia’. Je tiež pod-

statné, ktoré mobilné zariadenie začína komunikáciu. Na druhom zariadení sa tým pádom eliminuje generovanie a nezaťažuje sa toľko hardvér. Taktiež sa potvrdilo, že šifrovanie je rapídne rýchlejšie ako dešifrovanie, prihliadnuc na výpočtovo náročný dekódovací problém.

Záver

Cieľom práce bolo vytvorenie klient-server aplikácie demonštrujúcej využitie McElieceovho kryptosystému v OS Android. Bolo potrebné zanalyzovať dostupnú literatúru k tomuto kryptosystému. Navyše bolo nutné navrhnúť vhodný protokol, ktorý by sa dal použiť k dohode na symetrickom kľúči.

Vytvorili sme prehľad dostupných knižníc, ktoré by sa dali použiť na tento účel. Pri implementácii sme zistili, že vybratá knižnica neobsahuje všetky potrebné komponenty. Neobsahovala časti kódu potrebné na prácu s vygenerovaným kľúčovým párom. Po nahlásení problému vývojárom tejto knižnice boli tieto prvky doplnené vrámci funkčnej beta verzie knižnice. Implementácia je z tohto dôvodu obmedzená len na niektoré parametre kryptosystému. Pomocou tejto knižnice sme vytvorili aplikáciu na 'chatovanie'. Táto aplikácia komunikuje zabezpečeným kanálom ako so serverom, tak aj s ďalšími aplikáciami (klientmi).

Aplikácia bola dôkladne otestovaná na viacerých mobilných zariadeniach. Zistili sme, že implementácia je výrazne závislá od hardvéru mobilného zariadenia. Malý výpočtový výkon markantne spomaľuje celý proces vykonávania protokolu. Jedná sa najmä o generovanie kľúčov, ktoré taktiež využíva najviac pamäte RAM. Preto by sa v budúcnosti malo zamerať najmä na optimalizáciu tohto procesu. Po vyhodnotení vykonaných testov môžeme povedať, že aplikácia je plne funkčná a splňuje podmienky post-kvantovo zabezpečenej komunikácie. Na základe dosiahnutých výsledkov môžeme skonštatovať, že zadanie diplomovej práce sme úspešne splnili.

Zoznam použitej literatúry

[1] Federal information processing standards publication 197. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, (26.9.2015).

[2] Alfred J. Menezes, Paul C. van Oorschot and Scott A. *Handbook of Applied Cryptography, 1996*. 5. CRC Press. s. 816. ISBN: 0-8493-8523-7.

[3] Andrej Boledovič. *Authentication using code based cryptography, 2015*. Bakalárska práca. FEI STU.

[4] Daniel J. Bernstein, Johannes Buchmann, Erik Dahmen. *Post-Quantum Cryptography, 2009*. Springer. s. 136. ISBN: 978-3-540-88701-0.

[5] Encyclopedia, W. T. F. Forward secrecy. https://en.wikipedia.org/wiki/Forwardsecrecy, (10.10.2015).

[6] Frederik Armknecht, Stefan Lucks (Eds.). *Research in Cryptology, 2011*. Springer. s. 136. ISBN: 978-3-642-34158-8, 4th Western European Workshop, WEWoRC.

[7] Inc., G. Android security. https://source.android.com/security/index.html, (2.10.2015).

[8] Inc., G. Google cloud messaging. https://developers.google.com/cloud-messaging, (16.12.2016).

[9] Marek Repka, P. Z. Overview of the mceliece cryptosystem and its security. *Tatra Mountains Mathematical Publications* (2014), 57–83.

[10] McEliece, R. J. A public-key cryptosystem based on algebraic coding theory. *DSN Progress Report 42-44* (1978), 116.

[11] of the Bouncy Castle Inc., L. About the legion of the bouncy castlel. https://www.bouncycastle.org/about.html, (21.9.2015).

[12] of the Bouncy Castle Inc., L. Bouncy castle beta releases. https://downloads.bouncycastle.org/betas/, (16.3.2016).

[13] Oracle. The java ee 6 tutorial. http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html, (13.13.2016).

[14] Oracle. The java™ tutorials, all about sockets. https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html, (11.12.2015).

[15] Rtyley. Spongy castle. http://rtyley.github.io/spongycastle/, (15.9.2015).

[16] F. Uhrecký. *Implementácia kryptografickej knižnice s McEliece kryptosystémom, 2015.* Diplomová práca. FEI STU.

Prílohy

A Štruktúra elektronického nosiča

\Diplomová_práca.pdf	dokument diplomového projektu
\Server	serverové časti
\Server\DiplomaServer	Netbeans projekt servera so zdrojovými súbormi
\Server\GeneratorME	Netbeans projekt pre generovanie kľúčov
\Server\libs	použité knižnice
\Server\chat	Derby DB
\Mobilná aplikácia	aplikačná časť
\Mobilná aplikácia\McEllieceMessengerDP	Android Studio projekt aplikácie so zdrojovými súbormi
\Mobilná aplikácia\app-debug.apk	inštalačný súbor aplikácie

B Manuál

Inštalácia testovacieho prostredia

Pre funkčnosť testovania musí byť server aj mobilné zariadenie pripojené na rovnakej WiFi sieti. V mobilnej aplikácii treba nastaviť IP adresu WiFi spojenia. Nachádza sa v súbore mcelliecemessengerdp/utils/Info pod premennou SERVER_IP.

Server

- Serverová časť sa nainštaluje do vývojového prostredia Netbeans IDE 8.0.2.
- Celý súbor /DiplomaServer sa otvorí ako projekt. Do projektu treba importovať knižnice zo súboru /libs.
- Databáza sa importuje skopírovaním do určeného priečinka. Cesta k tomuto priečinku je v Netbeans IDE pod Services/Databases/JavaDB/Properties. Databáza by sa mala objaviť pod políčkom Databases, kde ju treba spustiť pred pripojením. Databáza môže vyžadovať prihlasovacie údaje (andrej, andrej).
- Po importovaní treba projekt spustiť. Spustí sa iba RESTful service.
- Ďalej treba spustiť súbor /serverSocket/ChatServer.

Server - generovanie

- Pre vygenerovanie nového kľúčového páru otvoríme Netbeans projekt GeneratorME. Importujeme BouncyCastle knižnicu a spustíme.
- V zdrojovom priečinku projektu sa vygenerujú súbory 'PUBLIC KEY' a 'PRIVATE KEY'.
- Súkromný kľúč sa vloží do hlavného priečinka projektu pre server.
- Verejný kľúč sa vloží do priečinka /McEllieceMessengerDP/app/src/main/assets v mobilnej aplikácii.

Aplikácia

- Aplikácia sa nakopíruje do vývojového prostredia Android Studio.

- Celý súbor /McEllieceMessengerDP sa otvorí ako projekt. Ten sa ďalej spustí na pripojenom zariadení.
- Aplikácia sa dá nainštalovať rovno do mobilného zariadenia inštalačným súborom .apk.

Užívateľský manuál

Registrácia a prihlásenie

- Po spustení aplikácie treba kliknúť na textové pole k registrácii.
- Aplikácia si vyžiada užívateľské meno, ktoré nesmie mať viac než 10 znakov.
- Klikneme na tlačidlo 'register'. Proces môže trvať dlhšie kvôli generovaniu parametrov. Treba počkať na odozvu zo servera.
- Po úspešnej registrácii sme presmerovaní k prihláseniu. Prihlásime sa rovnakým užívateľským menom.
- Po kliknutí na tlačidlo 'login' treba počkať na odozvu zo servera, pretože prebieha prihlásenie protokolom.

Pridanie užívateľa

- Klikneme na tlačidlo v pravom dolnom rohu obrazovky.
- Písaním do textového poľa automaticky vyhľadávame užívateľov ako na Obr. č. B.1.
- Kliknutím na užívateľa ho pridáme do zoznamu priateľov.

Obrázok B.1: Vyhľadanie užívateľa

Chatovanie

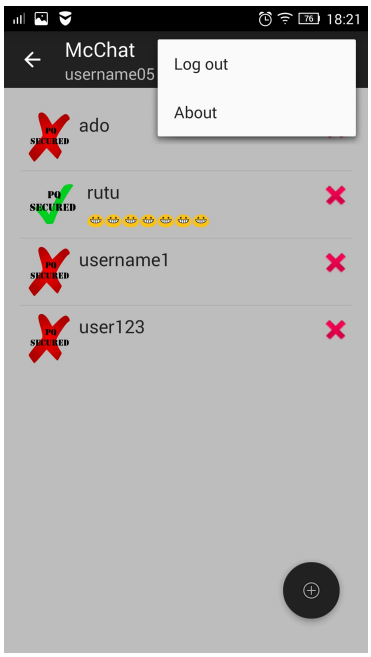
- Užívatelia označení červeným krížikom nie sú zabezpečený protokolom. Užívatelia označení zelenou fajkou naopak sú.
- Kliknutím na nezabezpečeného užívateľa začne bežať protokol. Je to zdĺhavejšia operácia, preto nič nerobíme a čakáme na odpoveď.
- Po nadviazaní spojenia sa môže odoslať správa.
- S užívateľmi označenými zelenou fajkou sa môže hneď 'chatovať'.

Odstránenie užívateľa

- Klikneme na červený krížik v pravej časti pri užívateľovi, ktorého chceme vymazať.

Menu

- Kliknutím na menu v pravej hornej časti z Obr. č. B.2 sa môžme odhlásiť alebo zobraziť informácie o aplikácii.
- Po odhlásení sa vymaže súkromný kľúč užívateľa a treba sa odznova registrovať.



Obrázok B.2: Menu