Středoškolská odborná činnost

Obor SOČ: 18.: Informatika

Kryptograficky bezpečný komunikační nástroj

Jan Jirman

Jan Polák

kraj Královéhradecký

Náchod 2016

Středoškolská odborná činnost

Obor SOČ: 18.: Informatika

Kryptograficky bezpečný komunikační nástroj

Autoři:	Jan Jirman
	Jan Polák
Škola:	Jiráskovo gymnázium Náchod
Kraj:	Královéhradecký
Konzultant:	Ing. Dalibor Vích

Náchod 2016

Prohlášení

Prohlašujeme, že jsme svou práci SOČ vypracovali samostatně a použili jsme pouze podklady (literaturu, projekty, SW atd.) uvedené v seznamu vloženém v práci SOČ.

Prohlašujeme, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemáme závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Náchodě dne ..... podpisy: .....

.....

Poděkování

Za pomoc při tvorbě této práce bychom rádi poděkovali Ing. Daliboru Víchovi, který s námi konzultoval postup tvorby této práce.

Anotace

Naším cílem bylo vytvořit nástroj pro zabezpečenou komunikaci v reálném čase. Přestože existují některé programy a systémy které zdánlivě splňují tyto požadavky, každému z nich chybí nějaká kritická funkce. Proto jsme vytvořili vlastní komunikační platformu, která je narozdíl od ostatních bezpečná, robustní, rozšiřitelná a multiplatformní.

Klíčová slova: Komunikační nástroj, chatovací aplikace, kryptografie

Annotation

Our aim was to create a tool for secure realtime communication. There already are some systems which seemingly match this criteria, but they all lack some key features. Therefore we created our own communication platform, which is, opposed to others, secure, robust, extensible and multiplatform.

Keywords: Communication tool, Chat app, Cryptography

Obsah

1. Úvod

V dnešní době je zabezpečení informací velkým tématem. V mnoha zemích se aktivně jedná o legislativě, která v lepším případě umožní, nebo zakáže používání kryptografických šifer. Masové sledování občanů je rozšířené v celém civilizovaném světě a je často tiše podporováno i velkými organizacemi. Zároveň roste objem dat které přes Internet proudí. Z toho vyplývá, že soukromí jednotlivých občanů pomalu mizí a stává se takřka veřejnou záležitostí. Nedostatek spolehlivých a prokazatelně zabezpečených komunikačních nástrojů nás tedy vedlo k tvorbě tohoto programu a systému, který není nepodobný existujícím nástrojům jako například "Skype" nebo "WhatsApp" ale narozdíl od nich je zabezpečený a rozšiřitelný, takže ho můžeme použít jako platformu k dalším projektům.

1.1 Slovníček

- **uživatel**: Jeden registrovaný uživatel na jednom serveru. Také používáno jako "Všichni clienti jednoho uživatele".
- **server**: Jeden server na kterém jsou zaregistrováni uživatelé. Tito uživatelé si mohou navzájem spolu posílat zprávy.
- **client**: Jedno připojení k server za konkrétního uživatele. Může být víc clientů pro jednoho uživatele.
- **client software**: Program, kterým se client může připojit k serveru. Ukládá soukromá data uživatele (viz [2.1]).
- **člověk**: Člověk používající client software *nebo* automatizovaný bot, implementovaný nad *client software*.
- **bot**: Speciální *client software*, který dovede odpovídat nebo reagovat na zprávy dle vlastních pravidel, například textová hra, umělá inteligence nebo monitoring externích systémů
- **session**: Komunikace mezi 2 a více uživateli. Má vlastní SessionID a šifovací klíč.

2. Protokol

2.1 Uložená data

Server i klient (v tomto případě client software) ukládají určitá data potřebná k bezpečné komunikaci. Některá jsou veřejná, respektive z pohledu bezpečnosti přenášených informací nevadí když je získá třetí strana, a ostatní jsou tajná. K tajným informacím patří především privátní RSA klíče, který celý systém používá.

Každý uživatel má:

- Uživatelský pár šifrovacích klíčů:
  - Veřejný klíč, distribuován serveru a klientům ostatních uživatelů ze seznamu přátel
  - Privátní klíč, který zná a může znát pouze majitel uživatelského účtu (dal by se přirovnat k heslu)
- `UserID` - identifikační číslo uživatele, přidělené serverem při registraci
- Jedná se o náhodné 64-bitové číslo
- Není tajné, používá se jako anonymní označení pro jednotlivé uživatele na serveru
- Používá se místo jména, aby se jméno dalo měnit a hlavně aby nebylo snadné zjistit, kdo na serveru má nebo nemá účet, protože server poskytne jméno patřící k `UserID` pouze těm, kterým to držitel daného čísla povolí
- Veřejný klíč serveru
- Veřejné klíče a `UserID` všech uživatelů na seznamu přátel

Tyto data jsou uložena v šifrované podobě v *Identity souboru*, který je součístí client softwaru uživatele. Pokud chce člověk nebo bot používat více než jeden client software, například na počítači a na mobilním zařízení, musí tento soubor přenést, protože je využíván místo klasického páru jméno-heslo.

Server má:

- Serverový pár šifrovacích klíčů
  - Veřejný klíč serveru - zná ji každý uživatel serveru
  - Privátní klíč serveru - zná ji pouze server

- Veřejné klíče a `UserID` všech registrovaných uživatelů

Veřejné a privátní klíče jsou podle šifrovacího standardu RSA (v našem případě RSA s 2048-bitovým klíčem). Systému asymetrického šifrování, kterým RSA je, umožňuje komunikovat s protějškem bezpečně, bez nutnosti znát jakékoli předchozí tajné informace. Dále umožňuje ověření identity příjemce pomocí digitálního podpisu. Obě tyto funkce náš program využívá.

Veřejné klíče jsou používány k ověření identity jak serveru, tak uživatelů. Jelikož je celý systém decentralizovaný, neexistuje certifikátový řetězec jako má například `TLS` a proto musí uživatelé ověřovat pravost certifikátů manuálně. To sice vyžaduje větší spolupráci od uživatele, ale obchází to potenciální riziko, kdy by kompromitovaný server mohl vydávat falešné klíče a tím umožnit, aby se útočník vydával za někoho kým není.

2.2 Komunikace se serverem

Samotná komunikace mezi serverem a klientem je šifrovaná pomocí `AES` klíče na kterém se spolu domluví pomocí asymetrické `RSA` šifry při spojení, na základě klíčů serveru. client SW má uložený hash veřejného klíče serveru a varuje uživatele pokud by se server (nebo útočník) pokusil navázat spojení s jiným klíčem.

Když se naváže šifrovaný tunel, může začít samotná komunikace. Můžou nastat 2 scénáře:

1. **Uživatel ještě není registrovaný**
   Client SW vygeneruje svůj pár klíčů a veřejnou část pošle serveru, spolu s uživatelským jménem. Server odpoví jestli pokus o registraci přijal nebo zamítnul (majitel serveru může určit vlastní důvody k zamítnutí, například protože jméno je nevhodné nebo protože server má málo místa v databázi). Pokud je registrace přijata, server si uloží veřejný klíč k sobě do databáze, přiřadí uživateli unikátní náhodné `UserID` a pošle ho klientovi zpět. Od této chvíle se uživatel bude autorizovat pomocí svého klíče a `UserID`.
2. **Uživatel se již dříve zaregistroval**
   Client SW načte přidělené `UserID` a pokusí se autorizovat u serveru. Server odpoví s náhodnými daty a požádá client SW aby ho podepsal svým privátním klíčem. Server zkontroluje pomocí uloženého veřejného klíče že klient opravdu zná svůj privátní klíč (nejedná se tedy o útočníka) a umožní klientovi přístup do systému.

Pokud tyto kroky proběhnou správně, je klient autorizován a může začít komunikovat s ostatními uživateli.

2.3 Posílání zpráv

Než může dojít k samotné komunikaci mezi dvěma (nebo více) uživateli, je nutné aby o sobě věděli a navzájem si povolili odesílání zpráv. Takovým uživatelům říkáme, že jsou přátelé. Server udržuje ve své databázi kdo je přítel s kým a povoluje jen přátelům aby navzájem viděli své veřejné klíče a uživatelská jména.

Komunikace probíhá vždy v rámci “session” (“diskuze”, něco jako chatovací místnost). Každý session má vlastní symetrický klíč pomocí kterého jsou zašifrovány všechny odesílané zprávy a dovedou ho přečíst jen účastníci diskuze, nikdo jiný, ani server. Tím je zajištěna bezpečnost a tajnost zpráv i když je server ovládán útočníkem. Přo vytvoření session musí zakládající uživatel znát všechny ostatní účastníky a být s nimi přitel. Seznam účastníků se po vytvoření nedá změnit.

Toto je příklad vytvoření sessionu mezi Alicí a Bobem[1], který ukazuje všechny potřebné kroky. Body 7 až 9 ukazují posílání zpráv:

1. Alice náhodně vygeneruje symetrický šifrovací klíč
2. Alice zašifruje tento klíč veřejnými klíči všech zúčastněných uživatelů, tedy v tomto případě klíči Alice a Boba
3. Alice pošle požadavek serveru aby vytvořil session, kterého se bude účastnit Alice i Bob
4. Zároveň pošle kombinovaný symetrický klíč z bodu **2** jako první zprávu do tohoto nově vytvořeného sessionu
5. Bobovi přijde od serveru informace že Alice vytvořila session a Bob je jeho účastníkem
6. Bob si stáhne první zprávu a rozšifruje ji svým privátním klíčem. Tím získá symetrický klíč kterým jsou šifrovány všechny následující zprávy.
7. Alice zašifruje zprávu `Hello Bob!` symetrickým klíčem z bodu **1** a odešle ji serveru do založeného sessionu
8. Bob si stáhne zašifrované údaje a klíčem který zjistil v bodu **6** ji rozšifruje
9. Bob si přečte zprávu `Hello Bob!`

Tímto způsobem je zaručeno, že se k symetrickému klíči server ani žádný jiný potensiální útočník nikdy nedostane, protože nezná privátní klíč ani Alice, ani Boba, a tudíž nemůže rozšifrovat žádné zprávy.

3. Implementace

Veškerý kód je psán v jazyku Java a to takovým způsobem, aby se dal spustit na počítači i na mobilních zařízeních. Tento jazyk jsme zvolili protože s ním máme dostatek zkušeností a splňuje všechna kritéria:

- Rychlý: i přes svoji pověst se Java řadí mezi rychlé jazyky
- Čitelný: ze zkušenosti víme, že velké projekty zůstávají čitelné a je snadné udržet čitelnou strukturu
- Dostatek nástrojů: existuje mnoho špičkových IDE (editorů), které výrazně usnadňují práci s velkými projekty
- Multiplatformní: programy napsané v tomto jazyku dovedeme spustit bez výraznách změn na všech cílových platformách

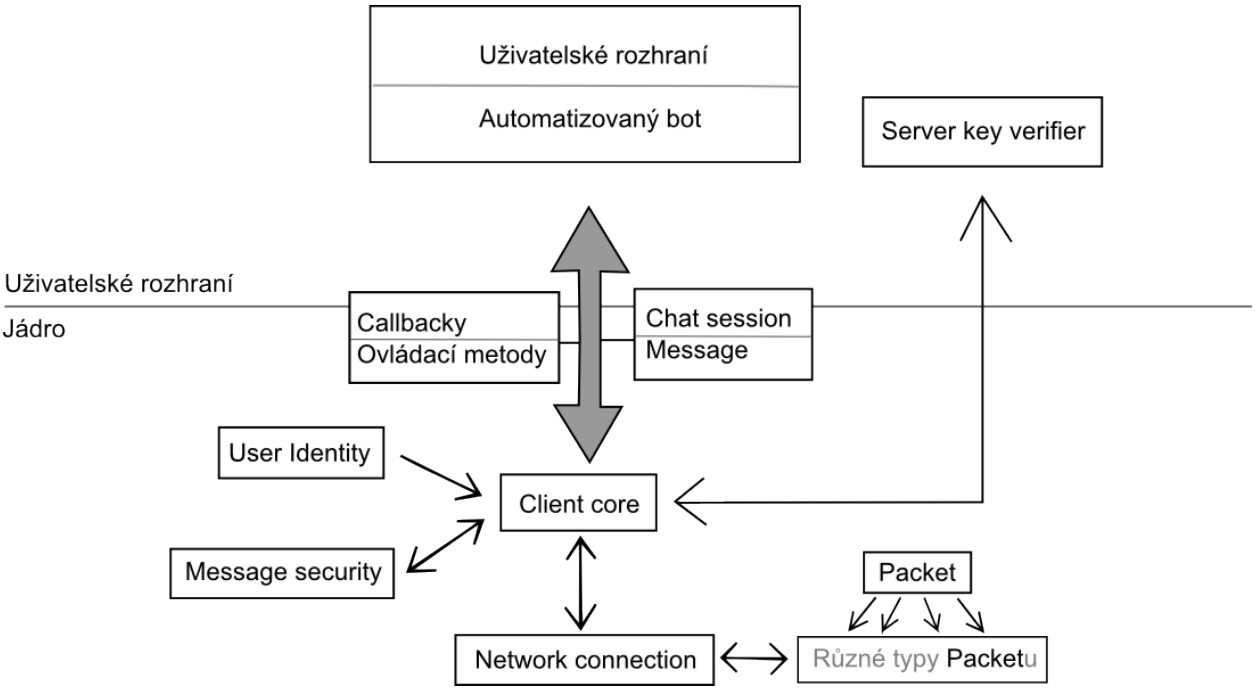
3.1 Architektura

Projekt je rozdělen na několik modulů, na sobě závislých. Těmi jsou, seřazeno podle vzájemné závislosti:

- `shared` Sdílený kód mezi serverem a klientem, jako jsou například kryptografické abstrakce, definice síťových protokolů, diagnostické nástroje apod.
  - `server` Veškerý kód potřebný pro provoz vlastního serveru
  - `client` Obsahuje kód společný všem implementacím client SW, včetně oficiálního uživatelského rozhraní
    * `bot`
      · Pomocný kód pro ty, co chtějí vytvořit vlastního bota
      · Obsahuje i několik námi vytvořených botů používaných pro testování systému
    * `desktop`
      · Kód který je potřeba k zavedení uživatelského rozhraní pro osobní počítače
    * `ios`
      · Kód který je potřeba k zavedení uživatelského rozhraní pro zařízení s operačním systémem iOS
    * `android`

· Kód který je potřeba k zavedení uživatelského rozhraní pro zařízení s operačním systémem Android

Vnitřní struktura klienta pak vypadá takto:



Obrázek 3.1: Blokové schéma architektury

3.2 Použité knihovny a nástroje

Celý projekt využívá několik externích open-sourcových projektů:

- **libGDX** [2] poskytuje abstrakce nad rozhraním OpenGL, vykreslováním grafiky a textu, a základní abstrakce nad cílovými platformami. Během vývoje jsme do tohoto projektu několikrát přispěli vlastním kódem.
- **kryo a kryonet** [3] pro serializaci dat a network
- **Joda-Time** [4] pro práci s časem
- **Bouncy Castle** [5] pro implementaci všech kryptografických algoritmů, jako RSA, AES a SHA
- **slf4j** [6] pro logging, ale používáme vlastní backend
- Server využívá **H2** [7] databázi pro ukládání svých dat a **DBUtils** [8] pro abstrakci nad přístupem do ní
- iOS backend využívá **RoboVM** [9] pro AOT kompilaci Javy pro iOS

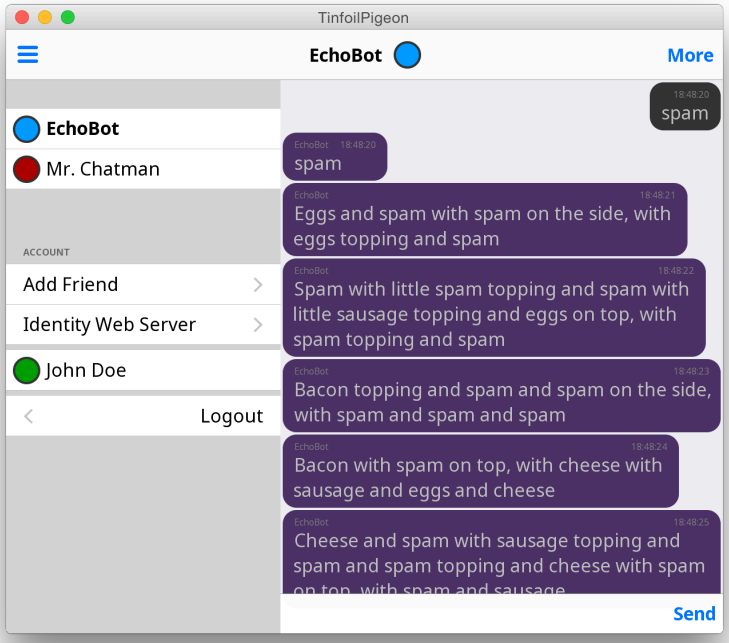
Vše je kompilováno pomocí build systému **sbt** [10], který využívá několik dalších pluginů, pro práci se zdrojovou grafikou a kompilací pro Android a iOS. Plugin pro práci se zdrojovou grafikou byl částečně vyvíjen podle potřeb tohoto projektu jedním z autorů této práce **ResourcePacker** [11] a je k dispozici volně ke stažení jako open-sourcový projekt.

3.3 Uživatelské rozhraní

Většině uživatelů ale bude stačit oficiální client SW pojmenovaný "TinfoilPigeon", který je kompatibilní se všemi velkými operačními systémy, tj. Microsoft Windows, OS X, GNU/Linux, Android a iOS. Ale způsob, kterým je celý projekt vytvořený umožňuje každému naprogramovat si vlastního klienta nebo bota (a tím i vlastní uživatelské rozhraní) s využitím našeho jádra.

Systém uživatelského rozhraní (*UI*) je celý vystavěn pomocí vlastních bloků na základě `scene2d.ui` z knihovny libGDX, vykreslovaných pomocí OpenGL. Nevyužívá tedy žádné UI systémy poskytnuté operačním systémem a je tedy bez problému přenosný mezi platformami. Na druhou stranu to ale znamená, že veškeré funkce a elementy které už jsou na většině platformách hotové, musíme vystavět znovu. V některých místech se bez spolupráce s cílovou platformou neobejdeme (například vstup textu), ale obecně se dá říct, že jsme vytvořili systém, který je velmi přenosný a vypadá dobře na všech platformách.

Mezi problémy na které jsme narazili při tvorbě UI systému, byla podpora obrazovek s vysokou hustotou pixelů ("Retina"), které používají některé počítače a mobilní zařízení Apple a v budoucnu snad i ostatní systémy. Pro vyřešení této výzvy bylo nutné vytvořit speciální nástroje, které dovedou s těmito hustotami pracovat, což se nám po několika experimentech povedlo.

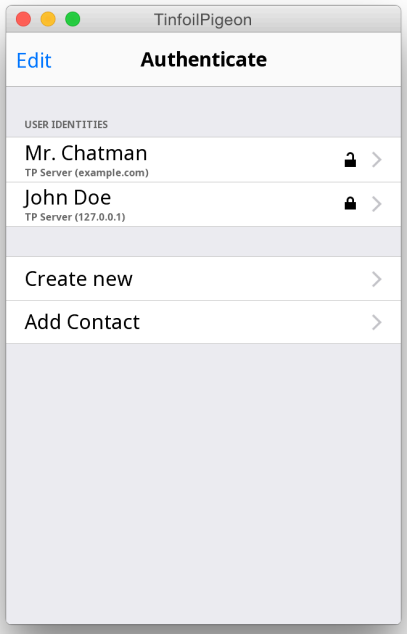


Obrázek 3.2: Snímek obrazovky při otevřené konverzaci

3.3.1 Unikátní problémy a jejich řešení

Existuje již mnoho programů pro posílání zpráv, takže většina vzorů UI již je vyřešená a není zde co vymýšlet, ale nám se do cesty postavil problém, jak uživatelům intuitivně předložit koncept souborů Identity, který naše schéma využívá. Jak již bylo popsáno, tyto soubory obsahují veškeré identifikační údaje, které uživatel má (a jsou prakticky nezapamatovatelné: šifrovací klíče, `UserID` a navíc některá uložená data), místo známého páru jméno-heslo známého z ostatních systémů a služeb. (Podobný systém používají i některé banky, kdy přístupové informace jsou uloženy v souboru nebo na kartě.) Tento soubor může být (a je to doporučeno) navíc šifrován heslem, aby ho útočník nemohl využít pokud ho nějak získá.

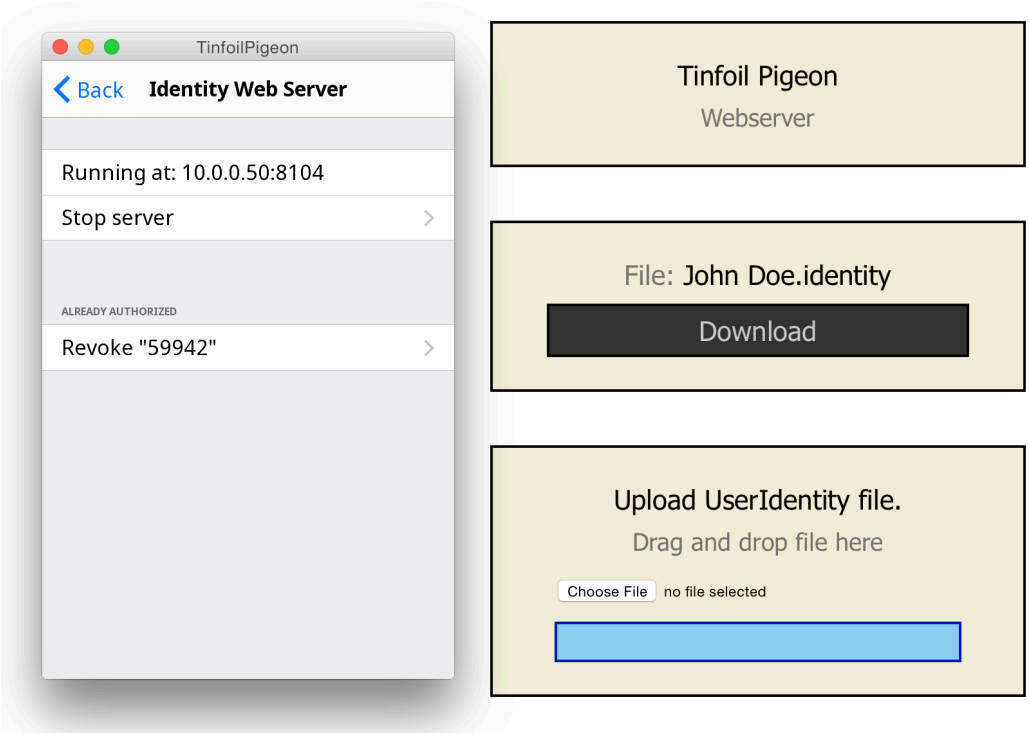
Při zapnutí klienta se zobrazí všechny Identity soubory, které klient nalezl. Uživatel nějaký z nich zvolí a zadá heslo. Pokud se soubor povede rozšifrovat a není poškozený, použije se pro přihlášení. V opačném případě je oznámena uživateli chyba.



Obrázek 3.3: Snímek přihlašovací obrazovky

Tento systém je oproti páru jméno-heslo o něco méně praktický, protože při každém přihlášení musí uživatel tento soubor mít. Může ho ale s sebou nosit na nějakém přenosném datovém zařízení, například flash disku, nebo ho mít uložený na vzáleném úložišti. Pokud je na Identity souboru nastaveno heslo, neohrožuje to výrazně bezpečnost systému.

Pro jednodušší přenos těchto souborů na mobilní zařízení, je do klienta zabudovaný webový server, který umožní přenos Identity souboru na lokální síti mezi jakýmikoli zařízeními, samozřejmě s náležitou bezpečtností.



Obrázek 3.4: Snímek obrazovky ovládání webserveru (nalevo) a stránka z webového prohlížeče (napravo)

Do budoucna máme v plánu experimentovat s centralizovaným systémem, který bude fungovat jako úložiště Identity souborů, a bude je vydávat na základě jména a hesla, čímž se práce s naším programem přiblíží klasickým chatovým programům. Použití tohoto systému ale nikdy nebude povinné a bude záviset pouze na uživateli, pokud ho bude chtít použít či nikoli.

3.4 Komunikace přes internet

Výměna dat po síti probíhá pomocí serializovaných datových tříd. Používáme serializátor Kryo[3].

Příklad datové třídy, která je použita pro odesílání zpráv:

```
public final class MessageSendPacket implements Packet {

    // Zde je několik proměnných ve kterých jsou uloženy informace o odeslané zprávě

    public long chatRoom = 0;
    public byte ackID = 0;
    public byte messageType = 0;
    public long messageTypeMetadata;
```

```
    public byte[] message = null;

    // Níže můžeme vidět vlastní serializaci a deserializaci této třídy
   // Serializovaná forma zpráv je optimalizovaná aby velikost posílaných dat byla co nejmenší

    @Override
    public void write(Kryo kryo, Output output) {
        output.writeLong(chatRoom);
        output.writeByte(ackID);
        output.writeByte(messageType);
        if(messageType == MessageTypes.MSG_TYPE_SUCCESSORY) {
          output.writeLong(messageTypeMetadata);
        }
        kryo.writeObject(output, message);
    }

    @Override
    public void read(Kryo kryo, Input input) {
        chatRoom = input.readLong();
        ackID = input.readByte();
        messageType = input.readByte();
        if(messageType == MessageTypes.MSG_TYPE_SUCCESSORY) {
          messageTypeMetadata = input.readLong();
        }
        message = kryo.readObject(input, byte[].class);
    }
}
```

3.5 Ukládání dat na serveru

Pro ukládání všech potřebných dat momentálně používáme databázi H2[7]. V případě potřeby ji v budoucnu budeme moci vyměnit za nějakou výkonější, například MySQL[12], ale zatím to nebylo nutné.

Naprogramovali jsme si vlastní ORM ("object-relational mapping"), což je zjednodušeně řečeno způsob jak z objektově orientovaného jazyka (jako je Java) přistupovat velice jednoduše do SQL databáze. Přestože již existuje několik ORM systémů, všechny byly buď příliš těžkopádné, špatně čitelné nebo málo flexibilní, proto jsme vytvořili tento.

Zde je příklad jak vypadá objekt korespondující jednomu řádeku dané tabulky v databázi:

```
@DB.Table(name = "clowns", constraints = {"PRIMARY KEY (lastname)"})
public static final class Clown extends DBEntry<Clown> {
```

```
    @DB.Column()
    public String firstname;

    @DB.Column(sqlSpecial = "NOT NULL DEFAULT 'Potter'")
    public String lastname;

    @DB.Column(name = "funFactor", sqlSpecial = "NOT NULL DEFAULT 999.9")
    public float funnyness;

    @DB.Column(sqlSpecial = "NOT NULL DEFAULT 66")
    public byte color;

    protected Clown(Database db) {
        super(db);
    }
}
```

Můžeme zde vidět použití anotací pro označení proměnných, které budou uloženy/načteny z databáze. Každá anotace `@DB.Column` může a nemusí mít následující parametry:

- **name**: Název sloupce, ve kterém je uložena daná hodnota. Pokud tato proměnná není nastavena, použije se jméno proměnné v daném objektu.
- **sqlType**: Datový typ pro daný sloupec. Pokud tato proměnná není nastavena je typ určen automaticky.
- **sqlSpecial**: Speciální vlastnost proměnných v daném sloupci, například "NOT NULL" nebo "DEFAULT". Pokud tato proměnná není nastavena, je použita hodnota "NOT NULL"

Anotace `@DB.Table` nám říká, že tato třída je určená pro hodnota v tabulce. Anotace má zase několik parametrů:

- **name**: Název tabulky. Tato proměnná musí být definována.
- **createTable**: SQL příkaz použitý na vytvoření této tabulky v databázi, pokud neexistuje. Pokud tato proměnná není nastavena, SQL příkaz se vygeneruje automaticky, tak aby odpovídal daným `@Column` anotacím
- **constraints**: Pokud proměnná `createTable` není nastavena, hodnota proměnné `constraints` je přidána k automaticky vygenerovanému příkazu pro vytvoření tabulky.

4. Ostatní

4.1 Vlastní URI protokol

Protože uživatelé nejsou identifikováni jménem ale svým `UserID` a adresou serveru na kterém jsou zaregistrováni, není předávání kontaktů triviální. A proto jsme vyvinuli speciální URI protokol pro jednoduché a bezpečné předávání kontaktů na uživatele a servery.

URI protokoly jsou technologie, kterou většina uživatelů zná z prostředí Internetu, například URL "http://example.com" je ukázkou URI protokolu "`http`". `http` URL (URL je subset URI) je používán k přenosu adresy a názvu dokumentu na serveru. My potřebujeme přenést jiné informace a to:

- `UserID`, identifikační číslo uživatele
- Hashe veřejných klíčů uživatele a serveru pro ověření jejich identity po spojení (tím odpadá manuální kontrola)
- Adresa a port kde se server nachází

Takové URI se pak dá relativně bezpečně poslat přes další kanály, jako třeba e-mail, link na webových stránkách nebo převést do QR kódu, který uživatel přečte v aplikaci třetí strany a ta URI otevře v naší aplikaci.

Formát:

```
tp-contact://UserID&UserPubKeyHash@ServerAddress:Port&ServerPubKeyHash/
```

Kde:

- **UserID**: Identifikační číslo (`UserID`) uživatelé na daném serveru, kódované způsobem popsaným níže.
- **UserPubKeyHash**: Hash veřejné části klíče daného uživatele.
- **ServerAddress**: IP adresa serveru nebo jeho hostname. Pokud je použita IPv6 adresa, musí být v hranatých závorkách (`[::1]`)
- **Port**: Port serveru (dvojtečka a číslo). Pokud není definován, je použit defaultní port.
- **ServerPubKeyHash**: Hash veřejné části klíče serveru.

Hashe klíčů nejsou povinné pokud je délka limitována, ale jsou doporučené pro vyšší bezpečnost. Celý blok o uživatelských informacích může chybět, pak se jedná jen o kontakt na server, ne na uživatele na serveru. Takový link by se pak mohl nacházet třeba na stránkách provozovatele serveru.

V praxi pak takový URI link se všemi informacemi vypadá třeba takto:

```
tp-contact://U2BALCXXSZCXXD3&_qHF1FnDHC77ibXiaXXWuJsouejxbE5BZoAadX31_WQ=@localhost&
AK0rjwRVbT26MjLwvSEri2us353c8xUh3oGY5tiG5BM=/
```

A při převedení na QR kód takto: [Example QR image]

4.2 Zápis UserID

'UserID' je nezvykle dlouhé číslo (64 bitů, až 19 číslic v desítkové soustavě) které se ale často opisuje a předává. Proto jsme vytvořili systém pro jejich zápis, který je při opisu rychlejší než desítková nebo hexadecimální soustava a méně náchylný k chybám. Je inspirovaný systémem Base32[13] ale upravený pro naše potřeby.

Formát:

- Začíná písmenem 'U', který slouží k identifikaci formátu.
- Následně je číslo rozděleno na 5 bitové bloky, od nejméně významného bitu, které jsou transformovány na znaky soustavy o základu 32 a postupně připojovány do řetězce, dokud nezbývají žádné nebo jen nulové bloky.
- Nakonec je zařazen kontrolní znak, který je získán XOR operací na všechny předchozí bloky

Znaky soustavy o základu 32 jsou v pořadí tyto:
`ABCDEFGHIJKLMNOPQRSTVWXYZ2345679`
Z běžného abecedního pořadí jsou vyřazeny ty znaky, které jsou si příliš podobné, jako například "B" a "8" nebo "I" a "1", tak aby z nich zůstal jen jeden. Pokud při dekódování program narazí na některý vyřazený znak ("8") automaticky předpokládá že uživatel myslel ten jemu podobný ("B"). Kód není závislý na velkých a malých písmenech a při prezentaci uživateli je pravidelně prokládán mezerami, aby bylo snadné ho manuálně přepisovat nebo diktovat, podobně jako u telefoních čísel.

Například číslo `7515995504831728126` je převedeno na `U7P C4O L5C ORT QGJ`, což je výrazně čitelnější.

5. Rozšíření

Na našem softwaru je stále ještě co vylepšovat a přidávat. Zde je výčet těch nejzajímavějších:

- **Podpora formátovacího jazyku markdown[14] ve zprávách**
  Pomocí markdownu bude možné mít v textu například tučné písmo, jiné barvy nebo styly. To umožní mnohem lepší komunikaci mezi lidmi, ale i mezi člověkem a botem. Představme si, že někdo udělal bota, který se chová jako server textové hry. Díky markdownu bude mnohem jednodušší orientace mezi zprávami. V budoucnu bychom chtěli tento systém i rozšířit interaktivními prvky, například tlačítka nebo formuláře. Takové funkce by navíc byly zajímavé v kombinaci s dalším bodem.
- **Interpretace skriptovacího jazyka přimo v chatu**
  Zatím nejsme rozhodnuti o jaký jazyk by šlo, uvažujeme nad jazykem Lua[15], BeanShell[16] nebo vlastním bytekódem. Zde je riziko zneužití, takže interpretace skriptů by fungovala pouze pokud by uživatel povolil spouštění skriptů od ostatních, respektive konkrétních přátel. Pomocí tohoto skriptovacího jazyku rozšíříme náš software nejen na chatovací program, ale i na platformu pro vykreslovaní grafiky, jednoduché hry a utility, které budou fungovat na PC i mobilních zařízeních. A to vše na bezpečně šifrovaném základu.
- **Streamy dat**
  Jednalo by se o šifrovaný datový tok mezi všemi členy sessionu, který není ukládán na server (narozdíl od většiny standardních zpráv). Takový datový stream se pak dá použít například pro hlasový hovor, video hovor nebo na nějaký specifický účel při komunikaci mezi boty.
- **Posílání souborů a obrázků**
- **Peer to peer spojení**
  Peer to peer spojení pro data streamy by se automaticky navázalo za pomoci serveru, tam kde to bude možné. P2P tok dat ulehčí serveru a urychlí přenos dat, takže například hlasové hovory budou plynulejší.
- **Identity server**
  Server, veřejně dostupný na internetu, na který bude možné nahrát Identity soubor a když bude třeba tak ho zase stáhnout a použít pro přihlášení na jiném klientovi. Použití Identity Serveru je naprosto dobrovolné a není vyžadováno pro funkčnost celého systému, nebo jakékoli jeho části, protože s sebou nese drobné bezpečnostní riziko, a to že uživatelův Identity soubor je uložen někde mimo jeho přímou kontrolu a je tedy teoreticky přístupný vládním organizacím na základě zákonů země kde se server nachází, samotnému poskytovateli IdentityServeru, nebo v nejhorším případě útočníkům,

pokud provozovatel takového serveru selže. Alternativně by na základě Identity serveru šel vybudovat certifikátový řetězec aby nebylo nutné kontrolovat klíče manuálně, ale o tom zatím pro dohlednou dobu neuvažujeme.

6. Závěr

Naprogramovali a otestovali jsme chatovací aplikaci, která je bezpečná k používání a to i za předpokladu, že se serveru nedá věřit, a to v tom smyslu, že by server četl osobní zprávy uživatelů.

Naučili jsme se kryptografické zásady a získali spoustu zkušeností s vývojem komplexních aplikací, programováním pro různé platformy a jejich specifické systémy a optimalizace, vykreslování složité grafiky a mnoho dalšího. Přestože má nyní celý projekt kolem 27 tisíc řádků kódu, stále máme co vylepšovat do budoucna. Avšak i bez těchto vylepšení je software plně připraven k použití a plánujeme ho začít testovat a používat i mezi přáteli a postupně i širokou veřejností. Už ve své nynější podobě si troufáme říct, že je příjemnější a spolehlivější než některé programy, které jsme k těmto účelům používali doposud.

Během vývoje jsme zjistili zajímavý poznatek o tom, že čím víc se snažíte kód optimalizovat aby byl méně náročný na procesor nebo pamět, tím víc práce si do budoucna přiděláte, protože se pravděpodobně bude muset, dříve nebo později, stejně přepsat tak, aby využíval o něco víc procesoru nebo paměti, ale byl čitelnější, čistější a dal se snadno rozšířit dál.

Seznam obrázků

Použité zdroje

1. Alice and Bob - two commonly used placeholder names
   WikimediaFoundation, 2001- [cit. 2015-01-29]. Dostupné z:
   `https://en.wikipedia.org/wiki/Alice_and_Bob`
2. LibGDX. [cit. 2016-03-31]. Dostupné z:
   `https://libgdx.badlogicgames.com`
3. EsotericSoftware / kryonet
   `https://github.com/EsotericSoftware/kryonet`
4. Joda time [cit. 2016-03-31]. Dostupné z:
   `http://www.joda.org/joda-time/`
5. Bouncy Castle [cit. 2016-03-31]. Dostupné z:
   `http://bouncycastle.org`
6. Simple Logging Facade for Java [cit. 2016-03-31]. Dostupné z:
   `http://www.slf4j.org`
7. H2 Database [cit. 2016-03-31]. Dostupné z:
   `http://h2database.com/html/main.html`
8. DbUtils [cit. 2016-03-31]. Dostupné z:
   `https://commons.apache.org/proper/commons-dbutils/`
9. Robo VM [cit. 2016-03-31]. Dostupné z:
   `https://robovm.com`
10. SBT [cit. 2016-03-31]. Dostupné z:
    `http://www.scala-sbt.org`
11. ResourcePacker [cit. 2016-03-31]. Dostupné z:
    `https://github.com/Darkyenus/ResourcePacker`
12. MySQL [cit. 2016-03-31]. Dostupné z:
    `https://www.mysql.com/`

13. Markdown
WikimediaFoundation, 2001- [cit. 2016-03-31]. Dostupné z:
https://en.wikipedia.org/wiki/Markdown

14. Base 32 [cit. 2016-03-31].
https://tools.ietf.org/html/rfc4648#section-6

15. Lua [cit. 2016-03-31]. Dostupné z:
http://www.lua.org/

16. Beanshell [cit. 2016-03-31]. Dostupné z:
http://www.beanshell.org/