VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

SYSTÉM PRO ANONYMNÍ PŘEDÁVÁNÍ ZPRÁV

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

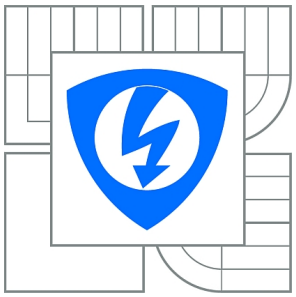
AUTOR PRÁCE
AUTHOR

Bc. JAN KISLINGER

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

SYSTÉM PRO ANONYMNÍ PŘEDÁVÁNÍ ZPRÁV

SYSTEM FOR ANONYMOUS TRANSMIT OF MESSAGES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

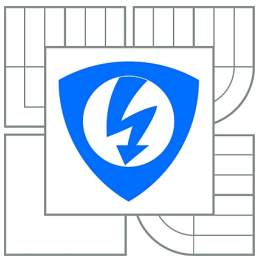
AUTOR PRÁCE — Bc. JAN KISLINGER

AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR LEŽÁK

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Jan Kislinger	***ID:***	125243
Ročník:	2	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Systém pro anonymní předávání zpráv

POKYNY PRO VYPRACOVÁNÍ:

Realizujte systém pro anonymní předávání zpráv pomocí protokolu popsaného v článku "Využití Diffie-Hellmanova protokolu pro anonymní autentizaci". Systém se bude skládat ze serveru a klientů, vzájemná komunikace bude probíhat po síti. Jednotliví klienti se budou moci na server zaregistrovat a zanechat anonymní zprávy pro ostatní klienty. Vhodným způsobem je nutné zobrazit průběh síťové komunikace.

DOPORUČENÁ LITERATURA:

[1] LEŽÁK. Využití Diffie-Hellmanova protokolu pro anonymní autentizaci. [online]. 2013, roč. 2013, č. 1, s. 5 [cit. 2013-10-04]. Dostupné z: http://www.elektrorevue.cz/cz/clanky/informacni-technologie/0/vyuziti-diffie-hellmanova-protokolu-pro-anonymni-autentizaci-1/

[2] BONEH, Dan. The decision Diffie-Hellman problem. In: Third Algorithmic Number Theory Symposium: Lecture Notes in Computer Science. Springer-Verlag, 1998 [cit. 2012-11-27]. Vol. 1423. Dostupné z: http://crypto.stanford.edu/~dabo/pubs/abstracts/DDH.html

Termín zadání:	10.2.2014	***Termín odevzdání:***	28.5.2014

Vedoucí práce: Ing. Petr Ležák
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Diplomová práce se zabývá anonymním předáváním zpráv pomocí protokolu pro anonymní autentizaci. Nejprve obsahuje teoretické seznámení s problematikou a popis protokolu pro anonymní autentizaci a popis jeho vlastností. Dále je popsán návrh komunikace mezi klientem a serverem. V závěru obsahuje popis vytvořeného systému pro anonymní předávání zpráv, skládající se ze serveru a klientů, kteří mohou na serveru zanechávat výzvy pro ostatní uživatele a ti je ze serveru pak získají.
V práci je popsáno spuštění i ovládání programu. Dále jsou zde rozebrány metody výpočtů ověřovacích hodnot, šifrování klíčů a zpráv a ověření příjemce.

KLÍČOVÁ SLOVA

anonymní, autentizace, Diffie-Hellmanův protokol, DSA podpis, RMI-IIOP

ABSTRACT

Diploma thesis deals with an anonymous transmit of messages using protocol for anonymous authentication. In first part, we introduce theoretical familiarization to the issues and description protocol for anonymous authentication. Further, it describes the suggestion of the communication between the client and the server. Finally, contains a description of the created system for anonymous transmit of messages, which consists of the server and clients, who can leave challenges on the server for other users and they obtains challenges from the server.
The thesis explains how to start and control program. There are also discussed methods of computing verification values, encryption keys and messages and authentication of receivers.

KEYWORDS

anonymous, authentication, Diffie-Hellman protocol, DSA signature, RMI-IIOP

KISLINGER, Jan *Systém pro anonymní předávání zpráv*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014. 60 s. Vedoucí práce byl Ing. Petr Ležák

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Systém pro anonymní předávání zpráv“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Petru Ležákovi, za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Technicka 12, CZ-61600 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .

(podpis autora)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

V dnešní době je komunikace pomocí zasílání zpráv přes nezabezpečenou síť jedním z nejpoužívanějších způsobů výměny informací na velké vzdálenosti. Při výměně zpráv přes nezabezpečenou síť je jedním z nejdůležitějších parametrů bezpečnost, tedy při zachycení zprávy útočníkem nesmí být útočník schopen zjistit její obsah. V určitých případech je také potřeba skrýt odesílatele zpráv, například při anonymním zanechávání komentářů, k hlasování, a podobně.

Pro zabezpečení zpráv se používají různé šifrovací algoritmy a další způsoby zapezpečení a pro anonymitu se jméno odesílatele různě šifruje nebo skrývá.

Diplomová práce se této problematice věnuje využitím protokolu pro anonymní autentizaci. Jednotliví uživatelé se zaregistrují na serveru a poté mohou vytvářet výzvy pro ostatní uživatele, kteří je ze serveru opět získají. Při vytváření výzvy se vygenerují ověřovací údaje, vytvoří se zašifrovaná zpráva a obojí se zanechá na serveru. Příjemce výzvu získá, zkontroluje, zda je určena pro něj a nakonec se autentizuje vůči serveru, čímž potvrdí svou identitu, a poté je mu zaslána zpráva. Celý tento systém je řešen síťově, to znamená, že server i klienti se nachází na různých strojích.

Ke stanovení společných parametrů a ověřovacích hodnot se využívá Diffie-Hellmanova protokolu a digitálního podpisu. Tyto technologie jsou v práci popsány. Hlavní výhodou je to, že parametry přenášené po síti jsou veřejné nebo zašifrované, a proto pomocí nich útočník nemůže zjistit ověřovací klíč ani dešifrovat zprávu.

1 TEORETICKÉ ZPRACOVÁNÍ PROBLEMATIKY

1.1 Autentizace

Jeden z nejdůležitějších kroků při předávání zpráv je autentizace, neboli ověření uživatele, zda se skutečně jedná o uživatele, za kterého se vydává. Po úspěšné autentizaci probíhá autorizace, neboli udělení oprávnění.

Autentizační metody můžeme rozdělit do několika tříd:

- **autentizace znalostí**
  Metoda založená na znalosti hesla, kódu nebo fráze je nejjednoduším a nejrozšířenějším způsobem autentizace. Hlavní výhodou je jednoduchost a také to, že se nejedná o fyzický objekt, tzn. že ho lze snadno přemisťovat a zadávat do každého zařízení. Heslo by mělo obsahovat 8–12 znaků, složených z malých a velkých písmen, číslic a symbolů, které tvoří netriviální řetězec, který se neobjevuje ve slovnících a databázích. Takovýto řetězec se ovšem obecně špatně pamatuje, a proto lidé často volí krátká a slabá hesla, která se dají snadno prolomit, například útoky hrubou silou nebo slovníkovými útoky.

- **autentizace předmětem**
  V tomto případě uživatel vlastní nějaký předmět „token“ pomocí kterého probíhá autentizace. Nejčastěji to bývá průkaz nebo čipová karta. Tokeny také mohou mít různé vlastnosti (tvar, elektrický odpor, ...) nebo obsahují tajné informace. Výhodou je obtížné kopírování těchto tokenů, ovšem může snáze dojít ke krádeži a poté nastává problém s prokázáním identity.

- **autentizace žadatelem**
  Tato metoda využívá fyziologické nebo behaviorální charakteristiky člověka, tzv. biometriku. Identita se prokazuje zjištěním aktuálních charakteristik a porovnáním s předchozími záznamy. Nejčastěji se používá otisk prstu, obličeje nebo charakteristika hlasu. Výhodou této metody je, že nemůže dojít k zapomenutí nebo ztrátě, ale biometrické informace jsou obtížně měřitelné a závislé na přesnosti. U této metody také může dojít k možným útokům, například modelem prstu.

Každá metoda má své výhody a nevýhody a také kvůli bezpečnosti často dochází ke kombinaci více metod dohromady, v případě dvou metod se jedná o dvoufaktorovou autentizaci.

1.2 Diffie-Hellmanův protokol pro výměnu klíčů

Otázka výměny klíčů byla jedním z prvních problémů řešených pomocí kryptografického protokolu. Diffie-Hellmanův protokol pro výměnu klíčů byl první způsob pro vytvoření sdíleného tajného klíče přes nezabezpečený komunikační kanál. Tento protokol navrhl Whitfield Diffie a Martin Hellman v roce 1976 a je jedním z nejvíce používaných protokolů.

Protokol slouží k výměně společného tajného klíče mezi dvěmi a více uživateli přes nezabezpečenou komunikační síť, kdy není tento klíč přenášen přes komunikační kanál a z přenášených dat není možné klíč zjistit v rozumném čase. To je zajištěno využitím problému diskrétního logaritmu.

Tento protokol neslouží k šifrování a dešifrování zpráv, ale pouze k bezpečné výměně klíče, kterého lze využít pro šifrování a dešifrování.

1.2.1 Princip

Máme 2 uživatele A a B - Alice a Bob.

- Nejprve dojde k volbě velkého prvočísla, což je parametr p.
- Poté nalezneme parametr g, neboli generátor, kde $g \in < 2, p-2 >$.

Tyto parametry jsou společné a veřejně známé.

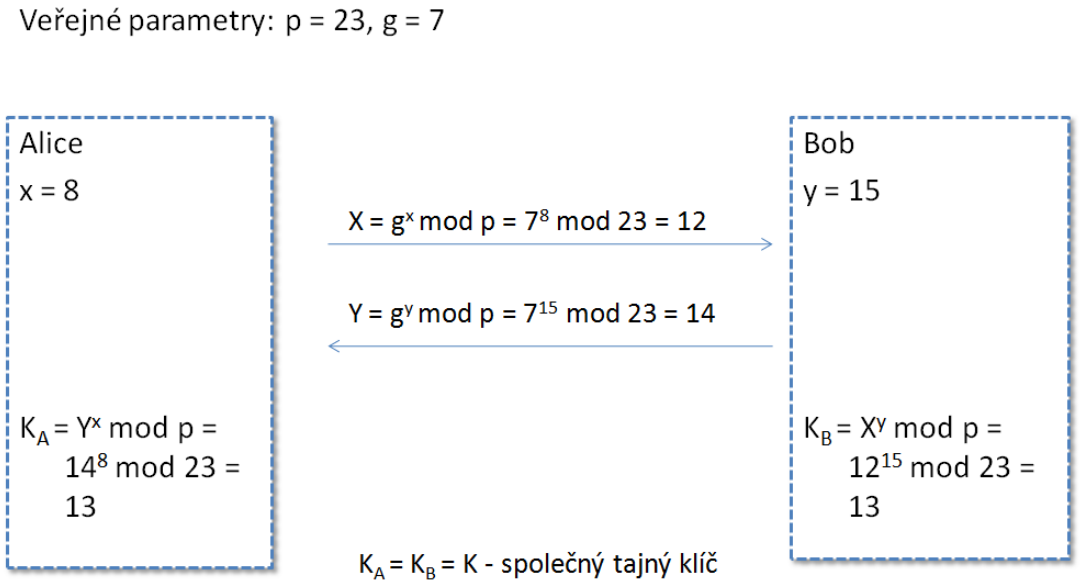
Nyní můžeme přejít ke stanovení klíčů:

- Alice si zvolí svůj soukromý klíč $x \in < 1, p-2 >$ a vytvoří zprávu $X = g^x \bmod p$. Tuto zprávu odešle Bobovi.
- To samé udělá i Bob. Zvolí si svůj soukromý klíč $y \in < 1, p-2 >$ a vytvoří zprávu $Y = g^y \bmod p$. Zprávu odešle Alici.
- Alice si spočítá tajný klíč K_A, pomocí vzorce $K_A = Y^x \bmod p$ a Bob K_B, pomocí $K_B = X^y \bmod p$. Klíče K_A a K_B jsou stejné.

Shodnost klíčů vyplývá z rovnice:

$$K_A = Y^x \bmod p = (g^y)^x \bmod p = (g^x)^y \bmod p = X^y \bmod p = K_B \tag{1.1}$$

Na obrázku 1.1 je ukázán příklad výpočtu tajného klíče.



Obr. 1.1: Ukázka Diffie-Hellmanova protokolu

Pokud by se útočníkovi podařilo zachytit všechny přenášené parametry, tedy: p, g, X, Y, tak není schopen zjistit klíč K, protože by musel vyřešit diskrétní logaritmus. Podle [1] není problém diskrétního logaritmu pro p, o minimální délce 2048 bitů, řešitelný v rozumném čase.

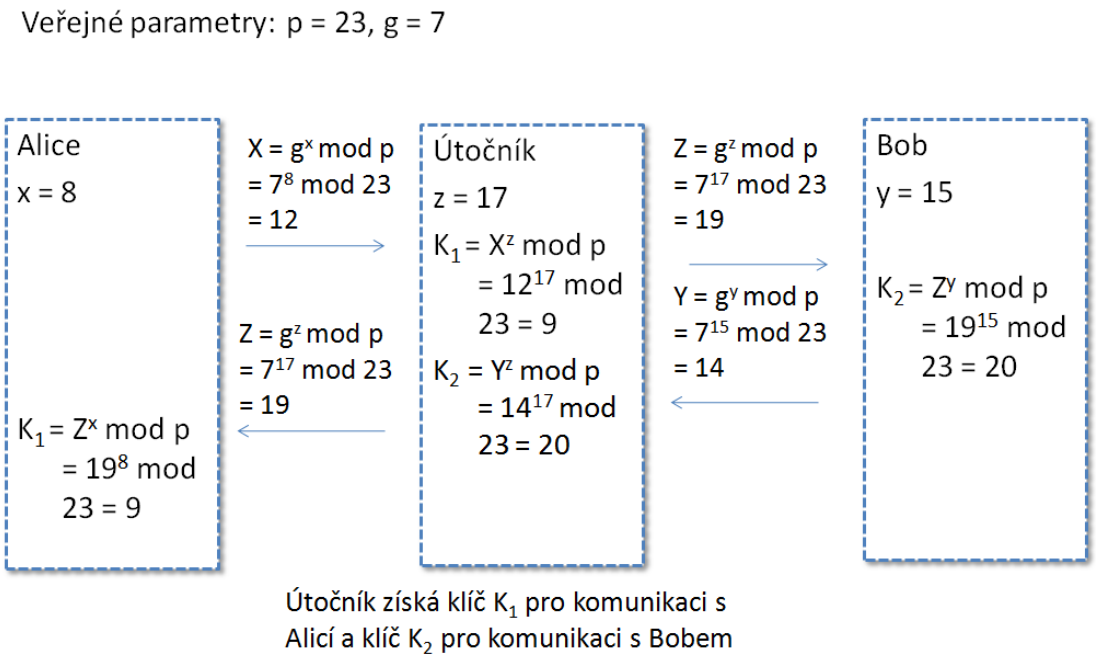
1.2.2 Útok mužem uprosřed

Pokud do komunikace mezi Alicí a Bobem vstoupí útočník a zachytí poslané zprávy X od Alice a Y od Boba, může je pozměnit tak, že do nich vloží vlastní klíče a pošle jim zprávy Z. Poté si oba uživatelé spočítají tajné klíče, ty jsou ovšem stejné jako má útočník.

Tím si útočník vytvoří dva šifrované kanály, bez vědomí Alice a Boba, pomocí nichž bude moci mezi uživateli přeposílat data, která pak bude moci odposlechnout.

To je tak zvaný „útok mužem uprostřed“ neboli „man in the middle“.

Na obrázku 1.2 je ukázán příklad výpočtu klíčů při útoku mužem uprostřed.



Obr. 1.2: Ukázka útoku mužem uprostřed

Ochranou proti útoku mužem uprostřed nejčastěji bývá využití digitálních podpisů nebo certifikátů, kdy si příjemce ověří, zda přijatá zpráva je skutečně od daného odesílatele a nebyla pozměněna.

1.3 DSA - Digital Signature Algorithm

DSA je standard pro podepisování a ověřování zpráv pomocí digitálního podpisu, který je založený na problému diskrétního logaritmu. DSA spolu s RSA (iniciály autorů Rivest, Shamir, Adleman), které je založeno na faktorizaci čísel, patří k nejpoužívanějším digitálním podpisům. Výhodou DSA oproti RSA je rychlejší vytvoření podpisu, ale jeho ověření může trvat déle.

Jak bylo popsáno v [8], tak podpis DSA se skládá ze dvou 160-bitových čísel r a s, kde r je funkce složená z 160-bitového náhodného čísla k tzv. „ephemeral key“, které se mění s každou zprávou. Číslo k musí být opravdu náhodné a dostatečně velké, jinak by bylo možné z veřejných a podpisových parametrů spočítat soukromý klíč podepisovatele.

Parametr s je funkce složená ze zprávy m, soukromého klíče podepisovatele x, parametru r a ephemeralního klíče k.

1.3.1 Princip

- Volba 160-bitového prvočísla q.
- Volba parametru p, které je mezi 512-2048 bity.
  Výpočet p: $p = nq + 1$, kdy $q > p^{\frac{1}{10}}$.
- Výpočet generátoru g: $g = x^n \bmod p$. Pokud se $g = 1$ je potřeba zvolit nové n, dokud $g \neq 1$.

Tím máme stanoveny veřejné parametry (p, q, g). Nyní si uživatel zvolí svůj soukromý podpisový klíč x, kde

$$0 < x < q.$$

Poté spočítáme veřejný klíč y, kde

$$y = g^x \bmod p.$$

K podpisu zprávy m použijeme následující kroky:

- Výpočet haš hodnoty zprávy: $h = H(m)$.
- Volba náhodného ephemerálního čísla k, kde $0 < k < q$.

- Výpočet parametru r:

$$r = (g^k \bmod p) \bmod q.$$

- Výpočet parametru s:

$$s = (h + xr) \cdot k^{-1} \bmod q.$$

Nyní máme ke zprávě m vytvořenou podpisovou dvojci (r, s), kdy tento podpis má délku 320 bitů.

K ověření podpisu (r, s) zprávy m použijeme ověřovací kroky:

- Výpočet haš hodnoty zprávy: $h = H(m)$.
- Výpočet čísla a: $a = h \cdot s^{-1} \bmod q$.
- Výpočet čísla b: $b = r \cdot s^{-1} \bmod q$.
- Výpočet ověřovacího parametru v:

$$v = (g^a y^b \bmod p) \bmod q,$$

kde y je veřejný klíč podepisovatele

Pokud se parametry v a r rovnají, podpis je v pořádku.

Příklad

K ověření správnosti postupu použijeme následující zjednodušený příklad:

- Zvolíme veřejné parametry: $q = 13$, $p = 4q + 1 = 53$, $g = 16$.
- Zvolíme soukromý klíč $x = 3$ a vypočítáme veřejný klíč

$$y = 16^3 \bmod 53 = 15.$$

- Nyní spočítáme haš zprávy, pro jednoduchost např. $h = 5$ a vygenerujeme ephemerální tajný klíč $k = 2$. Poté můžeme přejít k výpočtům:

$$r = (16^2 \bmod 53) \bmod 13 = 5,$$

$$s = (5 + 3 \cdot 5) \cdot 2^{-1} \bmod 13 = 10.$$

Nyní máme podpis ($r = 5$, $s = 10$).

Ověření podpisu:

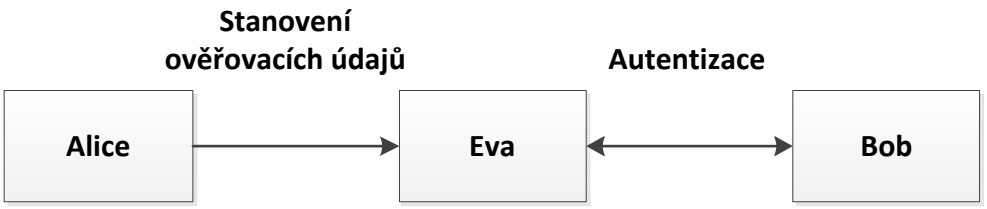
- Výpočet a: $a = 5 \cdot 10^{-1} \bmod 13 = 7$.
- Výpočet b: $b = 5 \cdot 10^{-1} \bmod 13 = 7$.
- Výpočet parametru v:

$$v = (16^7 \cdot 15^7 \bmod 53) \bmod 13 = 5.$$

Zpráva $v = r$, podpis je platný.

1.4 Protokol pro anonymní autentizaci

Základ protokolu je zobrazen na obrázku 1.3. Z toho vyplývá, že Alice umožní Bobovi se autentizovat vůči Evě tak, aby Eva nevěděla, že se autentizuje právě Bob.



Obr. 1.3: Schéma protokolu pro anonymní autentizaci

Z [5] vyplývá, že nejprve Alice vygeneruje ověřovací údaje, s využitím veřejného klíče Boba. Tyto údaje jsou zašifrované do kryptogramu a obsahují zprávu, jejíž součástí je i ověřovací klíč, a podpis zprávy. Tento kryptogram, poté zašle Evě, která s jeho využitím vytvoří výzvu. Výzva je zaslána Bobovi, který pomocí svého tajného klíče dešifruje kryptogram a odpoví na výzvu, čímž potvrdí znalost svého klíče, která je potřeba k dešifrování kryptogramu.

1.4.1 Princip

Generování veřejných parametrů

- Volba prvočísla q.
- Volba $p = nq + 1$, což je prvočíslený modulus, přičemž $q > p^{\frac{1}{10}}$.
- Volba generátoru grupy $g = x^n \bmod p \neq 1$.

Generování klíčů Alice

- Volba náhodného čísla a z rozsahu $< 2; q - 1 >$, což je tajný soukromý klíč Alice.
- Výpočet veřejného klíče

$$A = g^a \bmod p.$$

Alice zveřejní svůj veřejný klíč.

Generování klíčů Boba

- Volba náhodného čísla b z rozsahu $< 2; q - 1 >$, což je tajný soukromý klíč Boba.
- Výpočet veřejného klíče

$$B = g^b \bmod p.$$

Bob zveřejní svůj veřejný klíč. Alice ho využije k výpočtu šifrovacího klíče.

Generování ověřovacích klíčů a vytvoření zprávy

Alice

- Volba soukromého ověřovacího klíče f z rozsahu $< 2; q - 1 >$.
- Výpočet veřejného ověřovacího klíče

$$F = g^f \bmod p.$$

- Vytvoření zprávy $M = (ID_{Alice}||ID_{Eva}||f)$.
- Vytvoření podpisu S ke zprávě M pomocí DSA protokolu využitím soukromého klíče Alice a.

ID_{Alice} a ID_{Eva} jsou jednoznačné identifikátory.

Generování šifrovacího klíče a vytvoření kryptogramu

Alice

- Volba náhodného čísla z z rozsahu $< 2; q - 1 >$.
- Výpočet šifrovacího klíče

$$Z = B^z \bmod p,$$

který slouží k zašifrování a dešifrování kryptogramu C.

- Výpočet hodnoty

$$X_E = g^z \bmod p,$$

což je veřejný parametr sloužící Bobovi k výpočtu šifrovacího klíče *Z'*.

- Vytvoření kryptogramu $C = E(M||S, H(Z))$, kde H je hašovací funkce a *H(Z)* je šifrovací klíč.
- Alice odešle Evě ověřovací údaje (X_E, C a F).

Vytvoření výzvy

Eva

- Volba čísla v z rozsahu $< 2; q - 1 >$.

- Výpočet výzvy
$$V = g^v \bmod p.$$
- Eva odešle Bobovi zprávu (X_E, C, V a *Info*). *Info* je řetězec informující o účelu autentizace.

Ověření a dešifrování

Bob

- Kontrola řetězce *Info*. V případě, že Bob s autentizací nesouhlasí, tak ukončí algoritmus.
- Výpočet šifrovacího klíče
$$Z' = X_E^b \bmod p.$$

  Klíče Z a Z' jsou si rovny neboť:

$$Z = B^z \bmod p = (g^b)^z \bmod p = (g^z)^b \bmod p = X_E^b \bmod p = Z',$$

  zde se využívá Diffie-Hellmanovův protokol.

- Pomocí zjištěného šifrovacího klíče Z' se dešifruje kryptogram C a získáme zprávu M' a podpis S'
$$M'||S' = D(C, H(Z')).$$

  D je dešifrovací funkce, inverzní k funkci E.

- Ze zprávy M' získáme ID_{Alice}, ID_{Eva} a soukromý ověřovací klíč f.
- Díky podpisu S' si ověříme, že zpráva M' je skutečně od Alice. Pokud je podpis neplatný, algoritmus ukončíme.

Výpočet klíče a odpovědi

Bob

- Výpočet klíče
$$K'_{MAC} = V^f \bmod p.$$
- Výpočet odpovědi
$$O' = E(H(ID_{Eva}||Info), K'_{MAC}).$$

  Odpověď je kryptogram vytvořený na základě soukromého ověřovacího klíče f a výzvy V.
- Bob odešle odpověď O' Evě.

Ověření

Eva

- Výpočet klíče

$$K_{MAC} = F^v \bmod p.$$

- Výpočet odpovědi

$$O = E(H(ID_{Eva}||Info), K_{MAC}).$$

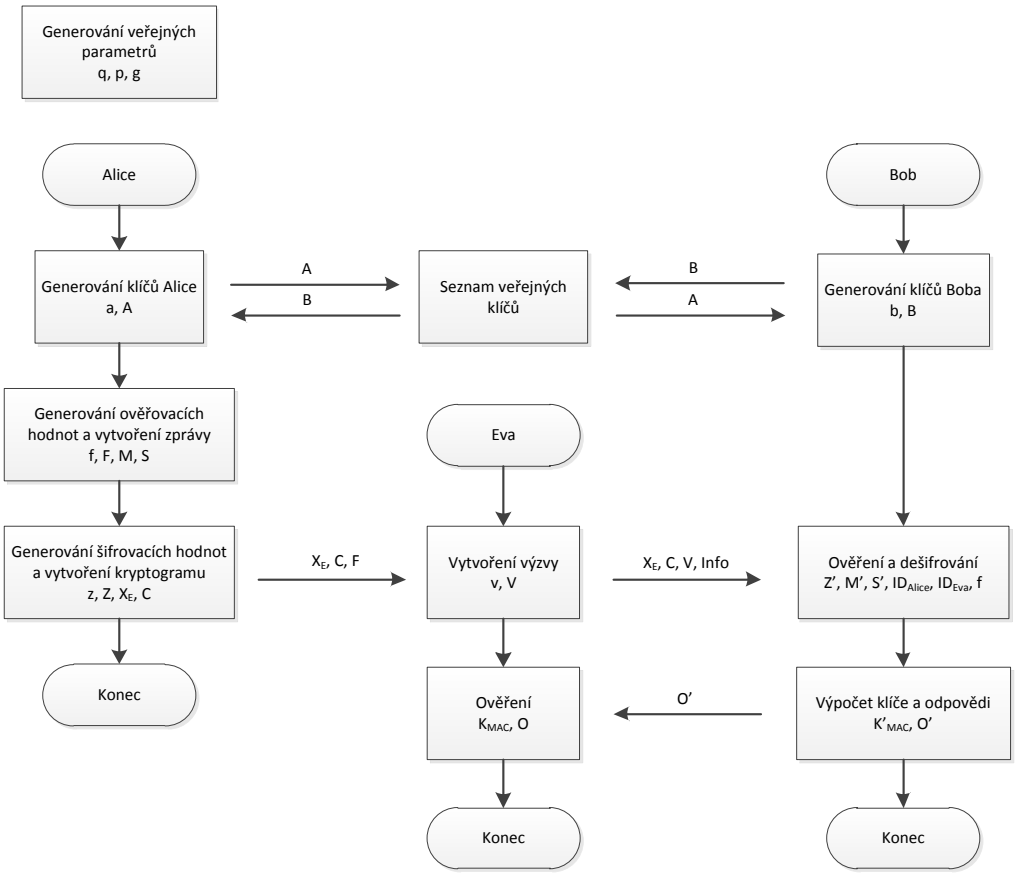
- Pokud $O = O'$, tak je autentizace v pořádku.

Klíč $K_{MAC} = K'_{MAC}$, opětovné využití Diffie-Hellmanova protokolu.

$$K_{MAC} = F^v \bmod p = (g^f)^v \bmod p = (g^v)^f \bmod p = V^f \bmod p = K'_{MAC}$$

Komunikace

Na obrázku 1.4 je znázorněna komunikace mezi jednotlivými bloky protokolu, a také parametry, které jsou přenášeny.



Obr. 1.4: Komunikace mezi jednotlivými bloky protokolu

1.4.2 Vlastnosti

Z výše popsaného postupu vychází několik základních vlastností tohoto protokolu. Jsou to:

- **Neexistence přímé komunikace mezi Alicí a Bobem**
  Generování ověřovacích hodnot Alicí probíhá pouze na základě znalosti veřejného klíče Boba a doménových parametrů, proto mezi nimi neprobíhá žádná komunikace.

- **Korektnost protokolu**
  Byla prokázána při popisu, ve kterém $Z \equiv Z'$, $K_{MAC} \equiv K'_{MAC}$ a také $O \equiv O'$.

- **Autentičnost odpovědi**
  K výpočtu odpovědi O musíme znát klíč K_{MAC}, který je vypočítaný pomocí Diffie-Hellmanova protokolu. Tento klíč spočítáme pouze ze znalosti alespoň jednoho soukromého klíče – náhodného čísla v nebo ověřovacího klíče f. Tento klíč je však zašifrován a číslo v není nikde uloženo. K autentizaci je tedy potřeba spočítat dešifrovací klíč Z. K výpočtu dešifrovacího klíče Z potřebujeme náhodné číslo z, které opět není nikde uloženo, nebo soukromý klíč Boba b. Z toho vyplývá, že autentizovat se může pouze Bob, ostatní klíče potřebné k autentizaci nejsou nikde uloženy nebo jsou zašifrovány.

- **Anonymita Alice a Boba**
  Pouze Alice a Bob mohou zjistit, že se autentizuje Bob a ověřovací hodnoty generuje Alice, proto mezi nimi nesmí být žádná spojitost. Odpověď Boba závisí, kromě doménových parametrů a informací od Evy, pouze na soukromém ověřovacím klíči f, proto odpověď nemá žádnou souvislost se soukromým klíčem Boba.
  Na identitě Alice závisí pouze ID_{Alice} a S, které jsou zašifrované v kryptogramu, takže jsou ostatním kromě Boba nečitelné, čímž je identita Alice skryta. Pokud mají identifikátory uživatelů konstantní délku, pak má konstantní délku i kryptogram, proto není z jeho délky možné zjistit žádné informace.

- **Netestovatelnost identity Boba**
  Žádný uživatel nemá možnost vygenerovat ověřovací údaje jménem Alice, aby otestoval, zda se autentizuje právě Bob. Toho je dosaženo dvojím zabezpečením, při němž jsou ověřovací hodnoty digitálně podepsány Alicí a navíc Bobova odpověď není závislá na jeho identitě.
  Pokud by například Eva chtěla otestovat, kdo se autentizuje, přiměla by k tomu Boba, tak musí nejprve zvolit čí identitu bude testovat. Poté by zjistila, zda se tato identita autentizuje nebo ne. Eva tak má možnost otestovat pouze jednu identitu za běh protokolu.

1.5 RMI - Remote Method Invocation

Java RMI je technologie programovacího jazyka java sloužící ke vzdálenému volání metod, tedy že z jednoho virtuálního stroje lze volat metody objektů na jiném virtuálním stroji, který se nachází na jiném počítači.

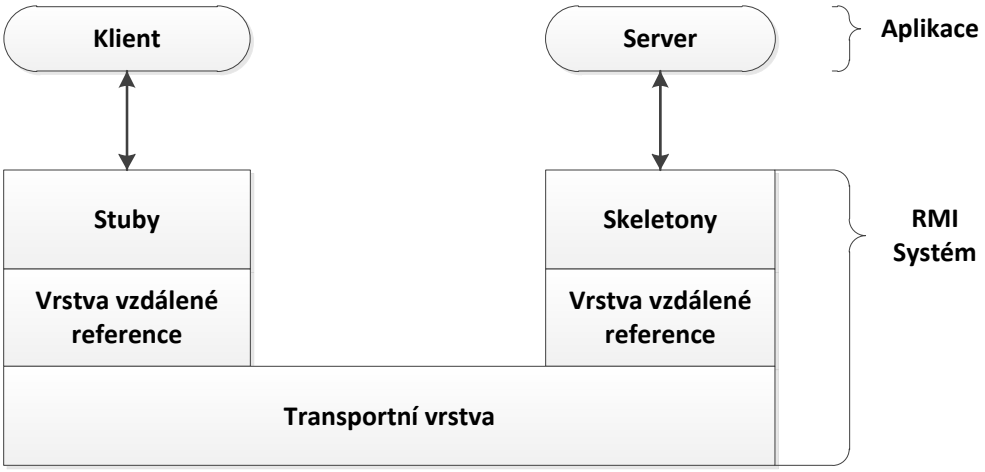
RMI byla využívána i jinými aplikacemi psanými například v jazyce C++, proto vznikla nová verze RMI-IIOP (Java Remote Method Invocation over the Internet Inter-Orb Protocol), která je nezávislá na jazyce a platformě. ORB (Object Request Broker) je zprostředkovatel objektových služeb, který zahrnuje mechanismy pro vyhledávání požadovaného objektu, generování a přenos požadavků, parametrů a výsledků na úrovni komunikace mezi systémy.

RMI-IIOP vychází z architektury CORBA (Common Object Request Broker Architecture), která je určena k usnadnění komunikace mezi systémy založenými na různých platformách. Tato technologie tak v sobě spojuje nejlepší vlastnosti technologie CORBA a RMI.

Při použití RMI-IIOP, nebo jiných distribuovaných aplikací založených na technologii Java, neexistuje jednoznačné rozhraní pro komunikaci mezi vrtsvami RMI systému. Toto rozhraní je pak definováno pomocí IDL (Interface Definition Language).

1.5.1 Architektura RMI

Architektura je typu klient-server a skládá se ze 3 vrstev: vrstvy stub a skeleton, vrstva vzdálené reference a transportní vrstva. Tyto vrstvy jsou zobrazeny na obrázku 1.5.



Obr. 1.5: Architektura RMI

Na straně klienta se nachází stub a na straně serveru skeleton. Klient vyvolává metody na stubu, na kterém se provede navázání spojení se vzdáleným virtuálním strojem, který obsahuje vzdálený objekt. Poté se vytvoří tzv. marshall stream (marshalling), který obsahuje informace o vzdáleném objektu, název volané metody, a ostatní informace a odešle ho na server.

Skeleton, který je na straně serveru, přijme marshall stream, rozbalí ho (unmarshalling) a vyvolá danou metodu. Poté zachytí návratovou hodnotu, případně výjimku, a vytvoří návratový marshall stream, který odešle zpět klientovi.

Na stubu se návratový marshall stream rozbalí a vrátí klientovi návratovou hodnotu nebo výjimku.

Vrstva vzdálené reference (Remote Reference Layer) je využívána pro přenos dat mezi stubem a skeletonem.

Transportní vrstva (Transport Layer) je dělena na 2 části. Ve spodní části se nachází protokol TCP/IP (Transmission Control Protocol/Internet Protocol) a v horní části je JRMP (Java Remote Method Protocol), což je proprietární protokol, který se stará o komunikaci mezi klientem a serverem. JRMP byl v nové verzi nahrazen IIOP, který nevyžaduje použití třídy skeleton.

K nalezení vzdálené služby klient využívá jméno nebo adresář služeb, které jsou provozovány na známé adrese a portu. RMI využívá RMI registr, který se nachází na každém stroji.

1.5.2 Komunikace RMI-IIOP

Jak bylo popsáno v předchozí kapitole, tak technoligie RMI-IIOP využívá IIOP, který je definován jako implementace protokolu GIOP (General Inter-ORB Protocol), což je obecný protokol pro komunikaci mezi různými ORB nad protokolem TCP/IP. GIOP definuje formát požadavků a odpovědí.

Při komunikaci klient iniciuje spojení, zatímco server je pasivní a pouze čeká na navázání spojení od klienta.

Podle [4] je mezi klientem a serverem definováno celkem 7 typu zpráv:

Tab. 1.1: Typy zpráv GIOP

Typ zprávy	**zdroj**	**popis**
Žádost (Request)	klient	volání metody vzdáleného objektu
Odpověď (Reply)	server	výsledek metody
Zrušení požadavku (CancelRequest)	klient	zrušení předcházejícího požadavku
Vyhledávání (LocateRequest)	klient	zjištení umístění vzdál. objektu
Odpověď na vyhledávání (LocateReply)	server	indikuje, zda server implementuje objekt, nebo předá volání dále
Uzavření spojení (CloseConnection)	server	uzavření spojení
Chyba (MessageError)	oba	reakce na předchozí zprávu

Request Message

Žádost slouží klientovi k provádění operací na vzdáleném serveru. Zpráva obsahuje všechny potřebné informace jako je identifikace objektu, jméno operace, atd. Jelikož žádost využívá IDL rozhraní, tak formát podporuje všechny syntaxe, které se mohou v IDL objevit.

Zpráva žádosti je rozdělena na hlavičku a tělo. Formát hlavičky je znázorněn tabulkou 1.2.

Tab. 1.2: Formát hlavičky žádosti

GIOP header	service_contexts	request_id	response_expected	reserved

	object_key	operation	requesting_principal

Popis jednotlivých polí:

- Pole *service_contexts* specifikuje kontext služby. To je využíváno službou CORBA.
- Pole *request_id* se používá pro přidělení jednoznačného identifikátoru žádosti.
- *Response_expected* indikuje, zda se jedná o obousměrnou komunikaci, nebo ne. Většinou nabývá hodnoty TRUE - očekává se odpověď.
- Další pole je rezerva pro budoucí použití.
- *Object_key* je použit k identifikaci objektu na straně serveru, který je volán.
- *Operation* je název volané operace.
- Poslední pole *requesting_principal* identifikuje uživatele, který žádost vyvolal.

Reply Message

Zpráva odpovědi je zasílána ze serveru jako odpověď na žádost. Většinou obsahuje návratovou hodnotu volané operace nebo vyjímku.

Hlavička se skládá ze 3 následujících částí:

- *Service_context* je stejné jako u žádosti.
- *Request_id*, což je identifikátor žádosti na kterou se odpovídá.
- *Reply_status* vrací stav odpovědi. V případě normální odpovědi (NO_EXCEPTION) tělo obsahuje návratovou hodnotu. Pokud nastane problém, tak vrací chybovou hlášku (USER_EXCEPTION \ SYSTEM_EXCEPTION \ LOCATION_FORWARD) a tělo obsahuje detailnější informace o chybě.

2 NÁVRH SYSTÉMU PRO ANONYMNÍ PŘEDÁVÁNÍ ZPRÁV

Cílem diplomové práce je realizace systému pro anonymní předávání zpráv založená na protokolu pro anonymní autentizaci. Tento stav je zobrazený na obrázku 2.1.

Obr. 2.1: Anonymní předávání zpráv

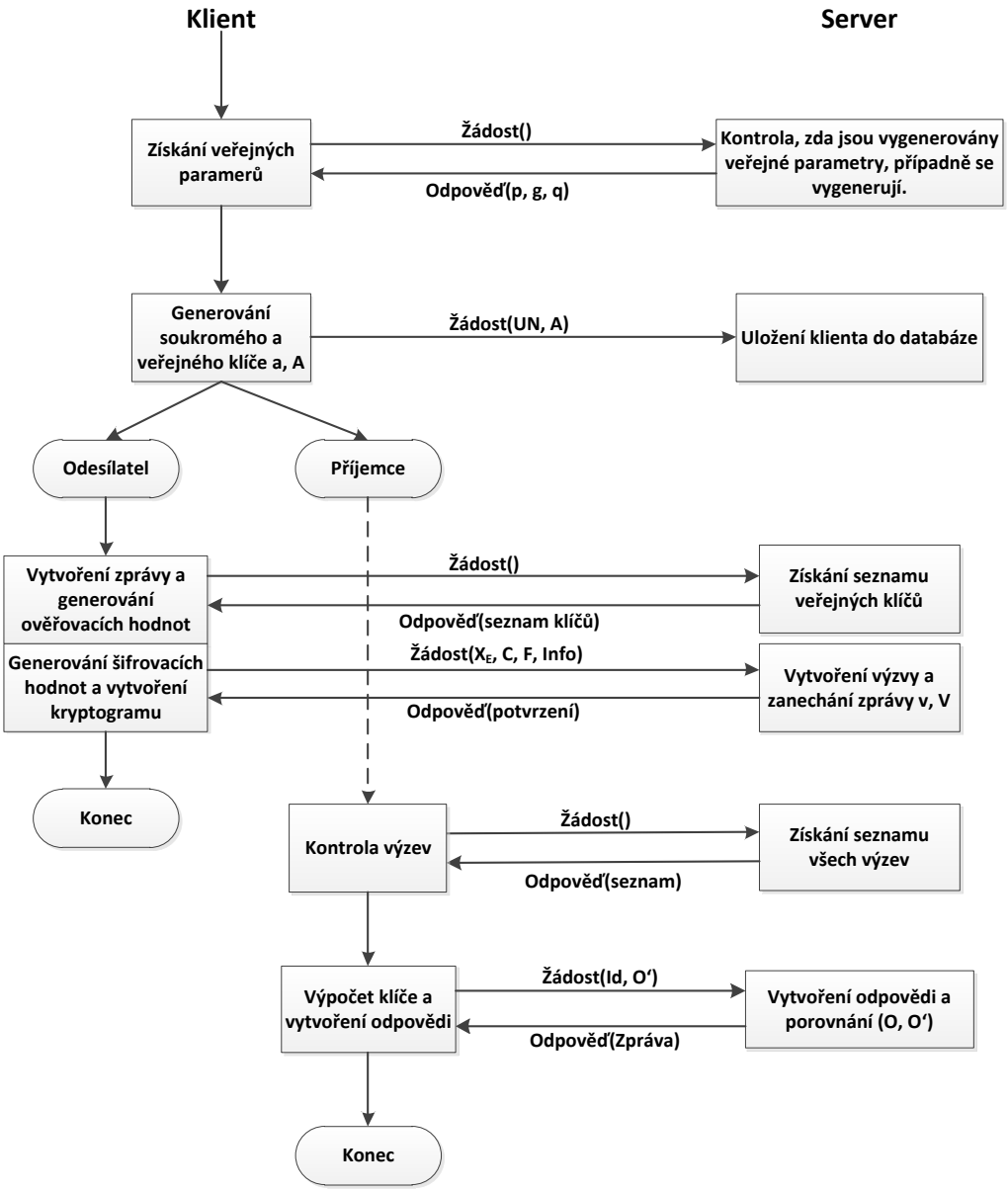
Odesílatel vystupuje jako Alice, tzn. že vytvoří zprávu, zašifruje ji a připojí k ní ověřovací hodnoty. Zprávu a ověřovací hodnoty odešle na server – uživatel Eva. Příjemce – Bob tyto hodnoty ze serveru získá a autentizuje se vůči serveru. Pokud je vše v pořádku, je mu zaslána zpráva, kterou dešifruje a zobrazí.

2.1 Návrh komunikace mezi klientem a serverem

Pro realizaci systému pro anonymní předávání zpráv jsem se rozhodl použít programovací jazyk java vzhledem k jeho jednoduchosti a přenositelnosti. Také umožňuje používání různých balíčků, čímž si ulehčíme práci například při programování síťové komunikace.

Při návrhu komunikace mezi klientem a serverem jsem použil technologii RMI-IIOP, jež je popsána v kapitole 1.5. Jak z textu vyplývá, dochází k situaci, při které klient vzdáleně volá metody uložené na serveru. Proto veškerá iniciativa probíhá ze strany klienta. Z tohoto důvodu využijeme k přenosu hodnot mezi klientem a serverem tyto metody. Pokud potřebujeme přenášet hodnoty od klienta k serveru, využijeme vstupní parametry metod. Zatímco pro opačný směr zvolíme návratové hodnoty metod.

Na obrázku 2.2 je zobrazen diagram komunikace mezi klientem a serverem. Jedná se o základní princip pro jednoho uživatele.



Obr. 2.2: Schéma komunikace mezi klientem a serverem

Na začátku každý klient potřebuje získat veřejné parametry. Proto zavolá metodu na serveru, na kterém se zkontroluje, zda jsou již tyto paramatry vygenerovány. Pokud ne, tak se vygenerují a pošlou se zpět klientovi. Klient pomocí parametrů vygeneruje soukromý a veřejný klíč. Veřejný klíč, spolu se svým uživatelským jménem, zašle na server, na kterém se tyto údaje uloží do databáze.

Nyní si klient zvolí, zda bude odesílatel či příjemce.

V případě odesílatele klient vytvoří zprávu a na základě veřejného klíče příjemce vygeneruje ověřovací hodnoty. Poté vygeneruje šifrovací hodnoty a vytvoří kryptogram. Následně tyto hodnoty společně s řetězcem *Info* zašle na server, na kterém se pomocí nich vytvoří výzva. Tímto jeho práce končí.

Příjemce ze serveru získá seznam všech výzev a pokusí se vůči nim autentizovat. Pokud se mu to podaří, znamená to, že daná výzva je určená pro něj. Poté zkontroluje řetězec *Info*, na jehož základě se rozhodne, zda zprávu přijme nebo ne. V případě nepřijetí ukončí svou činnost, v opačném případě vytvoří odpověď, pomocí které proběhne ověření uživatele na serveru a poté je mu zaslána zašifrovaná zpráva.

2.2 Popis programu pro komunikaci využívající RMI-IIOP

Základní části

Základem síťové komunikace pomocí RMI-IIOP je vzdálené rozhraní, které je v jazyce java definováno jako instance třídy. Vzdálené rozhraní *Interface.java* definuje všechny metody, které se budou volat ze vzdálených strojů.

Další částí je implementace daného rozhraní *Implementace.java*. Tato třída obsahuje těla metod definovaných v Inteface a konstruktor.

Třída *Server.java* vytváří instance implementací vzdálených objektů neboli referenci na vzdálený objekt. Reference se poté registruje na serveru pomocí inicializačního kontextu. Nyní klient může vyhledat objekt podle názvu (pomocí jmenné služby). V našem případě budeme používat „Naming Service“, která je součástí ORB Daemon.

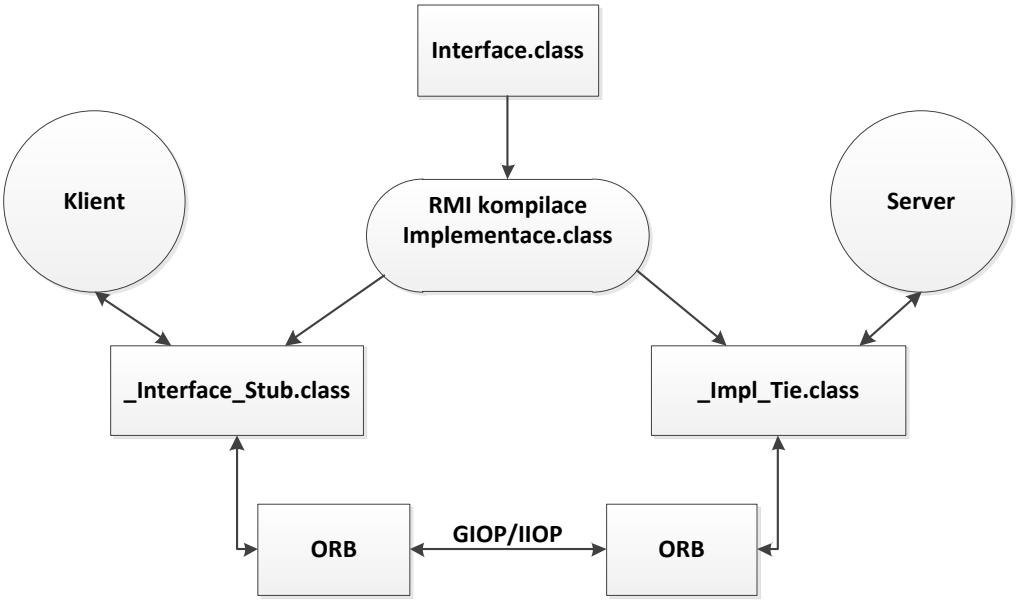
V třídě *Klient.java* se nejprve získá reference k vzdálenému objektu pomocí Naming Service a poté se vytvoří odkaz na Interface, pomocí kterého se dané metody volají.

Kompilace

Pro spuštění jednotlivých částí je potřeba tyto zdrojové java soubory zkompilovat. Ke kompilaci použijeme *javac* neboli java kompilátor, čímž nám vzniknou *class* soubory.

Nyní musíme vytvořit stub soubor pro klienta a tie soubor pro server (obdoba skeleton souboru). K vytvoření spustíme rmic, což je RMI kompilátor, s parametrem -iiop pomocí příkazu *rmic-iiop Implementace*.

Výsledný stav a komunikace je pak znázorněn na obrázku 2.3.



Obr. 2.3: Schéma komunikace pomocí IIOP

Spuštění

Server

Na počítači, na němž poběží server, nejprve spustíme Naming Service pomocí ORB Daemon s parametry InitialPort - používá se port 1050 a InitialHost - jméno počítače se serverem:

orbd -ORBInitialPort 1050 -ORBInitialHost servermachinename

Poté spustíme samotný server:

java Server -ORBInitialPort 1050

zde parametr InitialHost můžeme vynechat, protože Server běží na stejném počítači jako Naming Service.

Klient

Na klientském počítači spustíme klienta obdobně jako server:

java Klient -ORBInitialHost nameserverhost -ORBInitialPort 1050

3 REALIZACE SYSTÉMU PRO ANONYMNÍ PŘEDÁVÁNÍ ZPRÁV

Celý systém je rozdělen na 2 části - *Klient* a *Server*.

Na serverovém stroji běží Naming service, která slouží ke komunikaci se serverem, a také samotný server. K tomuto serveru se pak připojují jednotliví klienti. Pro klienta bylo vytvořeno jednoduché grafické rozhraní pro lepší ovládání programu.

3.1 Struktura programu

3.1.1 Server

Serverová část se skládá z komunikačního rozhraní a jeho tříd.

Třída Server.java

Třída Server.java obsahuje pouze hlavní metodu, která vytvoří instanci pro implementaci vzdáleného rozhraní a poté se zaregistruje v Naming service.

Rozhraní ComInterface.java

Toto rozhraní slouží ke komunikaci mezi klientem a serverem a obsahuje pouze hlavičky metod volaných na serveru.

Třída ComImpl.java

Zde se nachází implementace jednotlivých metod, které jsou volány na serveru. Tyto metody slouží ke kontrole spojení se serverem, získání, případně vygenerování veřejných parametrů. Dále jsou zde metody pro vytvoření a získání výzev, ověření příjemce a odstranění výzvy. V této třídě se také nachází seznam připojených uživatelů a jejich veřejných klíčů a seznam všech výzev a zpráv.

Třída User.java

Tato třída definuje strukturu jednotlivých uživatelů. Obsahuje jejich uživatelské jméno a veřejný klíč.

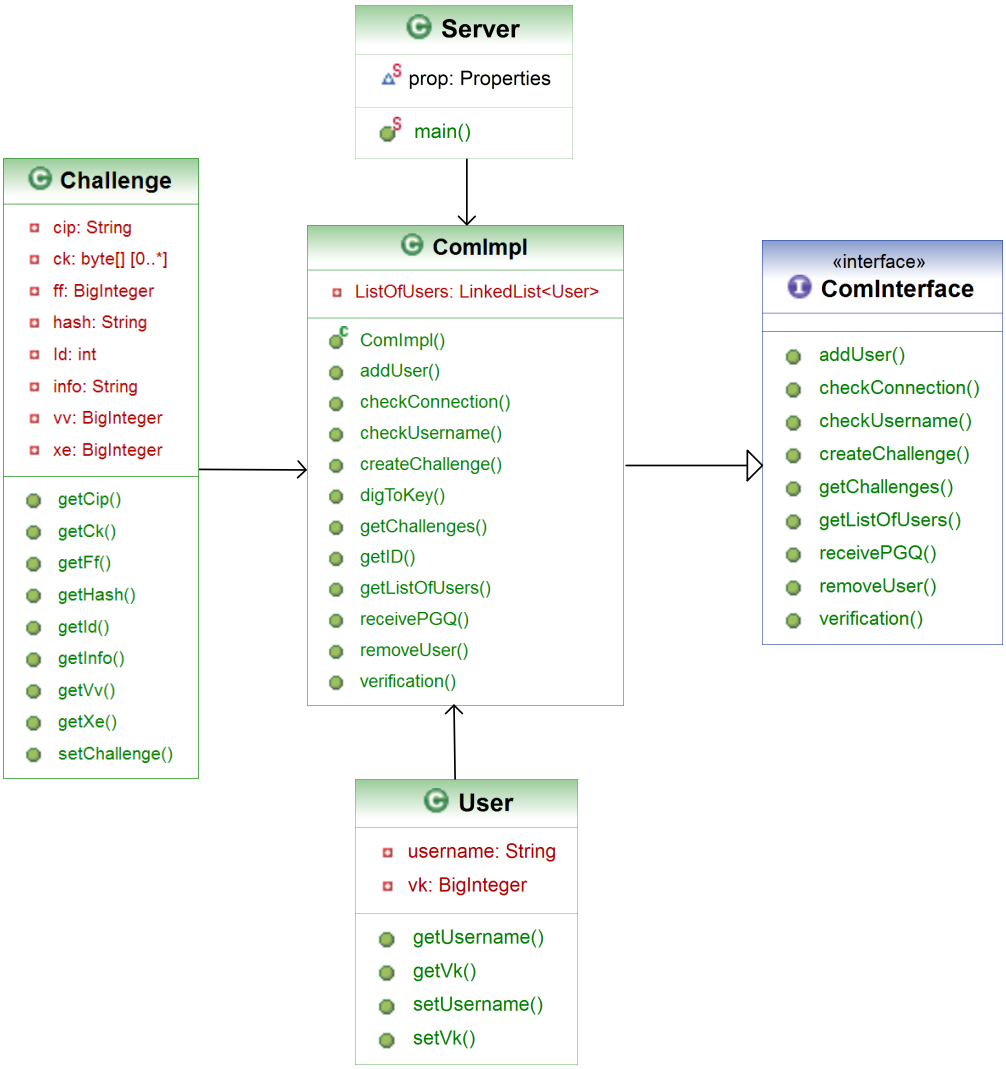
Třída Challenge.java

A tato třída definuje strukturu jednotlivých výzev. Každá výzva se skládá z následujících hodnot:

- *Id* - identifikátor výzvy
- *Xe*, *F* - parametry pro dešifrování,
- *V* - parametr výzvy,
- *ck* - zašifrovaný klíč, *info* - řetězec info,
- *cip*, *hash* - typ použité hašovací a šifrovací funkce.

UML diagram - Server

Na obrázku 3.1 je zobrazen UML (Unified Modeling Language) diagram tříd, který znázorňuje vztahy mezi třídami a rozhraním. Přesnější informace o jednotlivých třídách a metodách jsou popsány v dokumentaci přiložené na CD.



Obr. 3.1: UML diagram serveru

3.1.2 Klient

Klientská aplikace se opět skládá z několika tříd a grafického rozhraní.

Třída Main.java

Tato třída slouží ke spuštění klientské aplikace, čímž dojde k zobrazení grafického rozhraní a načtení dat z konfiguračního souboru.

Grafické rozhraní Gui.fxml

Toto rozhraní slouží k registraci uživatelů a vytváření nebo zobrazování výzev. Skládá se z tlačítek, textových polí a popisků.

Třída Graphics.java

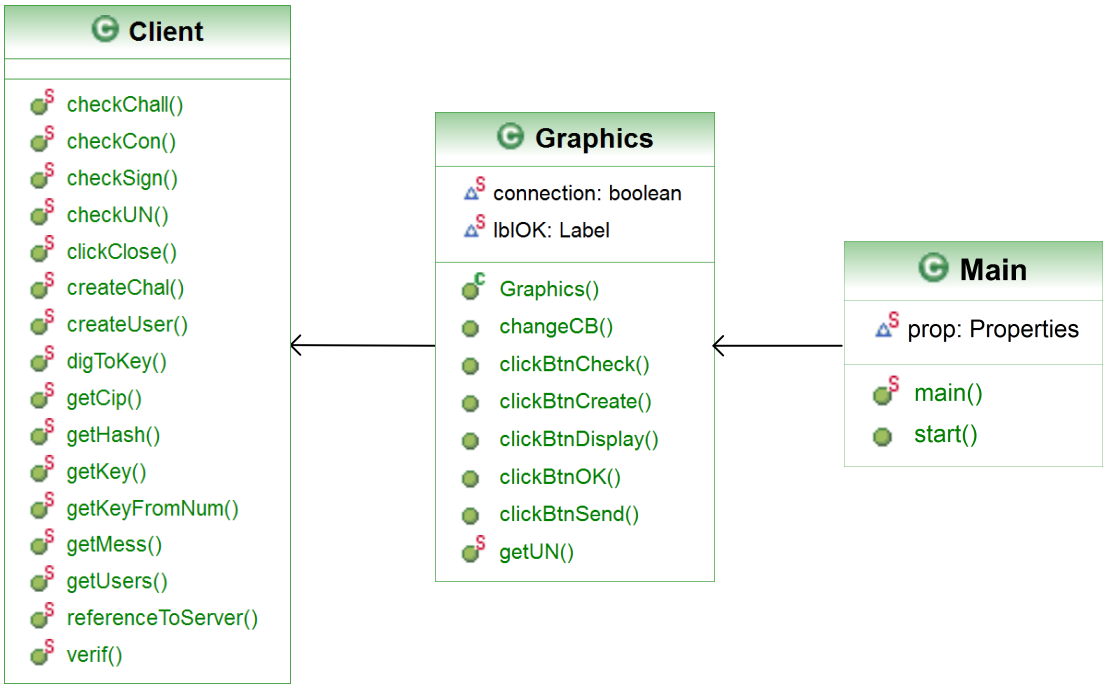
Pomocí této třídy spolu komunikuje grafické rozhraní a třída Client. Jedná se o metody spuštěné při akci provedené v grafickém rozhraní jako je stisk tlačítka nebo výběr položky z výběrového pole.

Třída Client.java

Jedná se o hlavní klientskou třídu, která komunikuje se serverem. V této třídě jsou prováděny všechny důležité operace jako je volba klíčů a parametrů, šifrování a dešifrování zpráv, tvorba a kontrola podpisu zprávy, atd.

UML diagram - Klient

Na obrázku 3.2 je ukázán UML diagram tříd pro klienta.



Obr. 3.2: UML diagram klienta

3.2 Časový diagram klientské aplikace

Celý průběh aplikace je znázorněn pomocí časového diagramu na obrázku 3.3. V tomto diagramu je znázorněna posloupnost jednotlivých metod různých tříd, provedené akce v grafickém rozhraní i komunikace se serverem.

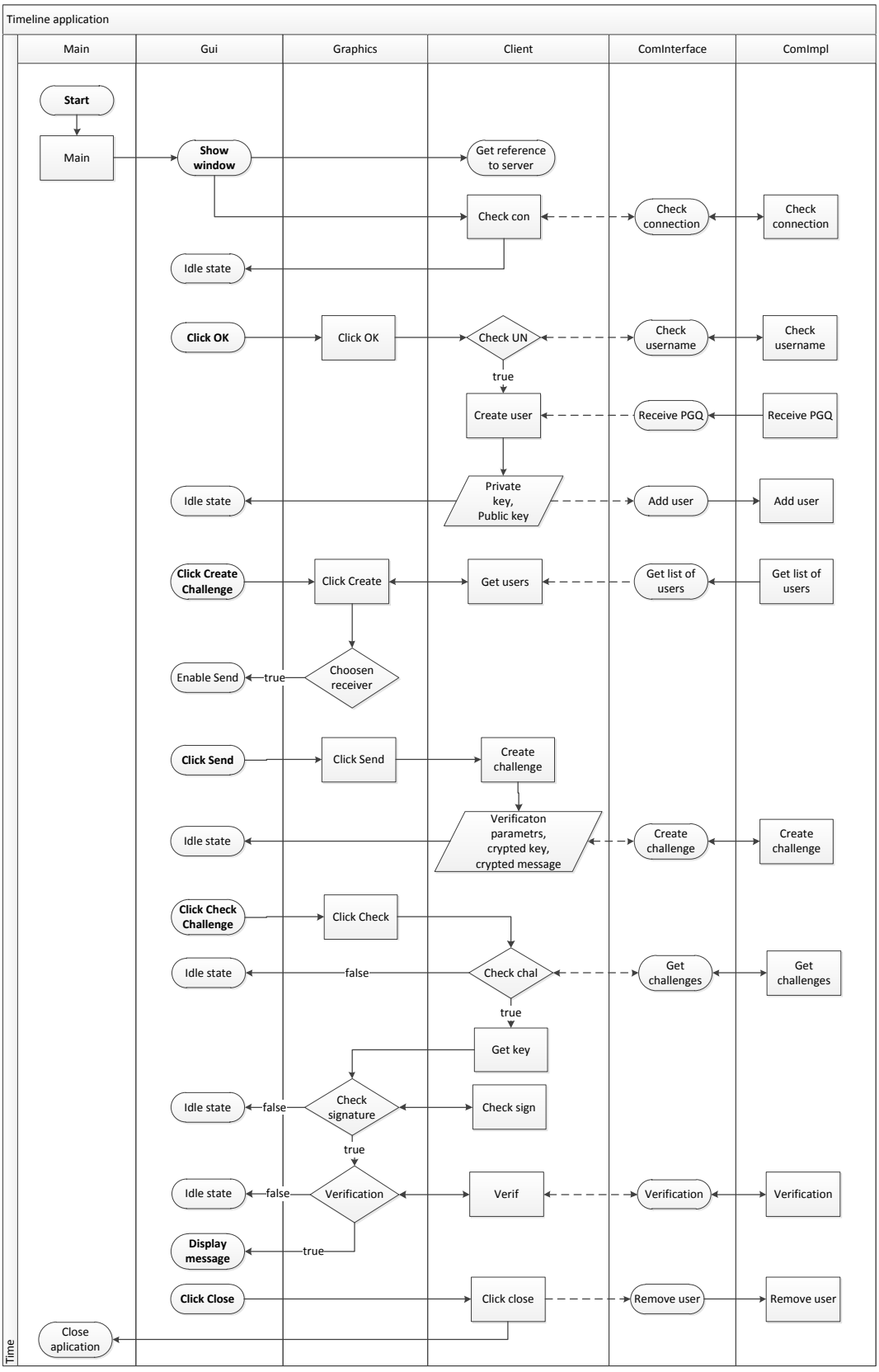
Spuštění klientské aplikace probíhá pomocí metody *Start* ve třídě Main, která zobrazí grafické rozhraní. Dále následuje metoda *main* sloužící k načtení dat z konfiguračního souboru. Po zobrazení se vytvoří reference pro komunikaci se serverem a dané spojení se zkontroluje. Poté aplikace přejde do idle stavu, což je stav kdy nic nedělá, jen čeká na akci uživatele v grafickém rozhraní. V tomto případě čeká na stisk tlačítka *OK*.

Po vložení jména a stisku tlačítka *OK* se uživatelské jméno (Username – UN) zkontroluje a ověří na serveru, případně se zadá nové. Ze serveru se načtou veřejné parametry „p“, „g“ a „q“ pomocí metody *Receive PGQ*. Tyto parametry se použijí k tvorbě soukromého a veřejného klíče. Dále se na serveru vytvoří nový uživatel, který se uloží do seznamu uživatelů a aplikace přejde do idle stavu.

Při stisku tlačítka *Create Challenge*, se ze serveru načte seznam uživatelů (List of users) do výběrového pole. Poté co dojde k výběru příjemce, povolí se tlačítko *send*. Po odeslání zprávy se ze seznamu uživatelů získá veřejný klíč příjemce, vygenerují se potřebné parametry a vytvoří se výzva, která se společně se zprávou odešle na server. Poté se opět přejde do idle stavu.

Při stisku tlačítka *Check Challenge* se ze serveru získají všechny výzvy a zjišťuje se, zda je některá z nich určena pro daného uživatele, pomocí metody *Check chal* (Check challenge – kontrola výzvy). V případě, že ano, tak se vypočítá ověřovací klíč a proběhne ověření podpisu klíče. Když je i ten v pořádku, vytvoří se odpověď, která se ověří na serveru. Pokud je ověření v pořádku, je uživateli zaslána zpráva. Nakonec se zpráva dešifruje a zobrazí se.

Po zavření aplikace se odstraní záznamy o uživateli ze serveru a aplikace se ukončí.



Obr. 3.3: Časový diagram klientské aplikace

3.3 Spuštění programu

Pro spuštění programu stačí mít nainstalován JRE (Java Runtime Environment) verze 1.6 nebo vyšší, který je určen koncovým uživatelům ke spouštění java aplikací.

Nejprve je potřeba spustit Naming Service a server. Poté se mohou začít připojovat jednotliví uživatelé.

3.3.1 Serverová část

Služba Naming service se spouští z příkazového řádku pomocí „orbd“ s parametry, které definují IP (Internet Protocol) adresu a port serverového stroje. Server se spouští také z příkazové řádky se stejnými parametry. Pro jednoduší spuštění byl vytvořen dávkový soubor „Server-run.bat“, který se postará o spuštění Naming Service i serveru. Jeho obsah je znázorněn na obrázku 3.4.

```
@ echo off
start /b orbd -ORBInitialHost 192.168.1.31 -ORBInitialPort 1050
@ echo Naming service: Ready...
start java ComPackage.Server 192.168.1.31:1050
@ echo on
```

Obr. 3.4: Spuštění serveru

Před spuštěním je potřeba v daném souboru nastavit správné IP adresy a porty. Poté může být soubor spuštěn.

Pokud je vše v pořádku, tak se zobrazí 2 konzole – naming service a server. V první je zobrazeno: „Naming service: Ready...“ a ve druhé: „Server: Ready...“. V konzoli pro server se zobrazují kontrolní výpisy o počtu uživatelů.

3.3.2 Klientská aplikace

Nyní se mohou začít připojovat jednotliví klienti. Pro jednoduchost byl vytvořen spustitelný soubor *Client-run.jar*, který v sobě obsahuje všechny potřebné knihovny.

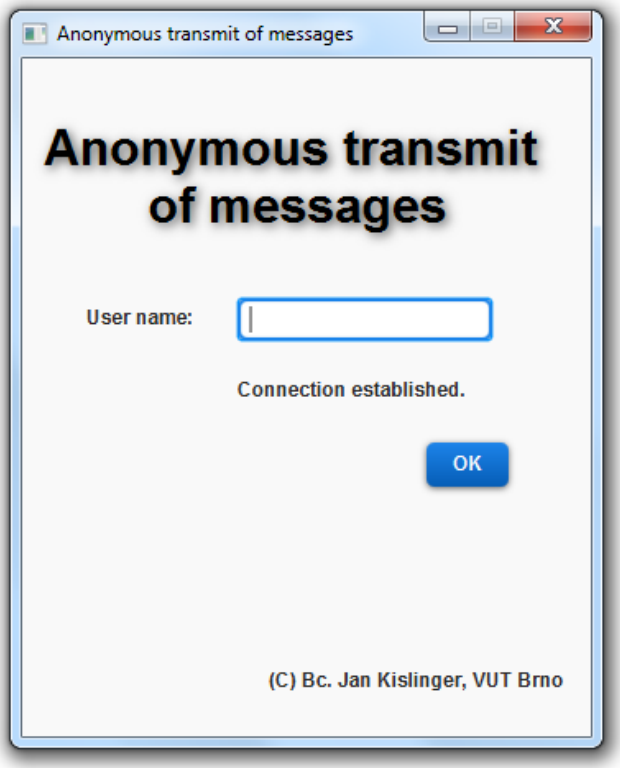
Ke spuštění aplikace slouží hlavní klientská třída Main.java. Tato třída obsahuje 2 metody a to metodu Main a metodu Start. Main slouží k načtení konfiguračních dat z textového souboru „Config.txt“. Tento soubor se musí nacházet ve stejné složce

jako je *Client-run.jar* soubor. Konfigurační soubor definuje IP adresu a port serveru a také slouží k volbě hašovací funkce a šifrovací funkce (pro kontrolu jsou vypsány možné hodnoty a jejich bitové délky). To je ukázáno na obrázku 3.5.

```
# config file for Main
#
test=ok
serverIPaddress=192.168.1.31
port=1050
hash=md5
cipher=desede


# Possible values of cipher and hash
#
# BLOWFISH               : 128bit
# PBEWITHSHA1ANDDESEDE   : 128bit
# AESWRAP                : 128bit
# DESEDE                 : 192bit
# DES                    :  64bit
# AES                    : 128bit
# DESEDEWRAP             : 128bit
# ARCFOUR                : 128bit
# RC2                    : 128bit
# PBEWITHMD5ANDDES       : 128bit
# PBEWITHSHA1ANDRC2_40   : 128bit
# PBEWITHMD5ANDTRIPLEDES: 128bit
#
#
# SHA-256                : 256bit
# SHA-512                : 512bit
# SHA                    : 160bit
# SHA-384                : 384bit
# MD5                    : 128bit
# MD2                    : 128bit
```

Obr. 3.5: Obsah konfiguračního souboru

Metoda Start slouží ke spuštění a zobrazení grafického rozhraní tzv. gui. Toto rozhraní se načítá ze souboru Gui.fxml, který pro svůj běh potřebuje knihovnu javafx. Tato knihovna je již zahrnuta v .jar souboru. Pro nastavení vzhledu aplikace byly použity kaskádové styly, neboli css (cascading style sheets). Tyto styly jsou načteny ze souboru App.css. Po spuštění již dojde k navázání spojení se serverem, kontrole spojení a zobrazení samotné aplikace. Výsledek spojení je zobrazen pod textovým polem, což je zobrazeno na obrázku 3.6.



Obr. 3.6: Po spuštění aplikace

Pokud je vše v pořádku, dojde k výpisu „Connection established.“ . V případě, že někde nastane problém, tak se červeným písmem vypíše „ERROR: Connection refused!“ a při stisku tlačítka *OK*, dojde k jeho zablokování a aplikace vás nepustí dál.

V případě problému zkontrolujte: správné spuštění serveru, dostupnost počítače, na kterém se server spouští, IP adresy a port nebo nastavení firewallu. Poté zkuste aplikaci znovu spustit.

Poslední část metody Main obsahuje funkci, která se spuští při ukončení grafického rozhraní. Tato funkce zkontroluje, zda je uživatel registrován na serveru. Pokud ano, tak je ze serveru odstraněn. Poté je již aplikace řádně ukončena.

3.4 Registrace uživatelů

Pokud je spojení v pořádku, aplikace vás nejprve vyzývá k zadání uživatelského jména.

Po stisku tlačítka „OK“ dojde ještě ke kontrole uživatelského jména, zda se toto jméno již na serveru nevyskytuje nebo zda nebyl zadán prázdný řetězec. Případně zadejte jiné uživatelské jméno.

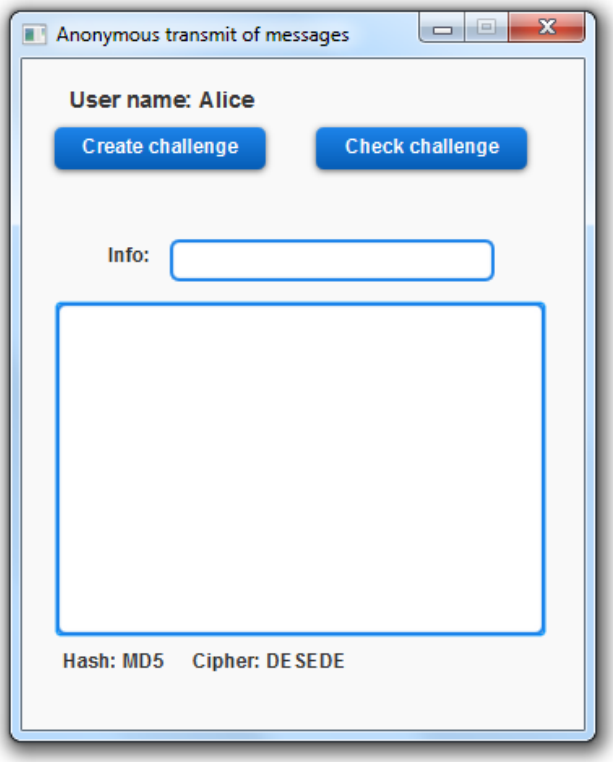
Před přechodem do hlavního okna však probíhá několik funkcí a výpočtů. Každý klient si musí vytvořit soukromý a veřejný klíč, proto potřebuje získat ze serveru veřejné paramatry. Na serveru se nejrve zkontroluje, zda jsou parametry vygenerovány, pokud nejsou, tak dojde k jejich generování. To je znázorněno následujícím zdrojovým kódem:

```
/**
 * Gets public parameters 'p', 'g' and 'q'.
 * @return array of big integers, [0] = p, [1] = g and [2] = q
 */

public BigInteger[] recivePGQ() throws java.rmi.RemoteException {
        if (p.intValue() == 0) {
                p = BigInteger.probablePrime(512, rnd);
                q = BigInteger.probablePrime(160, rnd);

                BigInteger h = BigInteger.probablePrime(512-160,
                    rnd);
                do      {
                        h = h.add(BigInteger.ONE);
                        p = h.multiply(q).add(BigInteger.ONE);
                        g = h.modPow(h, p);
                } while (!p.isProbablePrime(100) ||
                    g.equals(BigInteger.ONE));
        }

        BigInteger PGQ[] = new BigInteger[3];
        PGQ[0] = p;
        PGQ[1] = g;
        PGQ[2] = q;
        return PGQ;
}
```

Nyní již může být zvolen soukromý a veřejný klíč a klient zaregistrován na serveru.

Dále jsou do programu načteny zvolené hašovací a šifrovací funkce. Pokud by byla zadána nedovolená nebo neexistující funkce, tak se zvolí výchozí hodnota, což je pro haš „MD5“ a pro šifru „AES“.

Poté je již zobrazeno hlavního okno aplikace. To je ukázáno na obrázku 3.7.

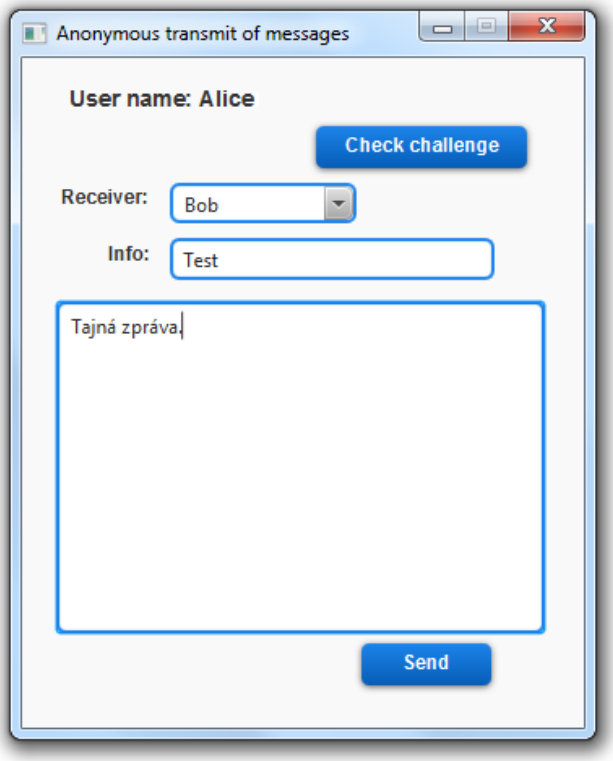


Obr. 3.7: Hlavní okno aplikace

Nyní se uživatel může rozhodnout, zda vytvoří novou výzvu nebo zkontroluje, zda se na serveru nachází nějaká výzva určená pro něj.

3.5 Vytvoření výzvy

Po stisku tlačítka „Create challenge“ se zobrazí volby pro vytvoření výzvy, což je ukázáno na obrázku 3.8.



Obr. 3.8: Vytvoření výzvy

Zde si uživatel vybere příjemce výzvy, čímž dojde k povolení tlačítka „send“. Pokud k serveru není nikdo kromě nás připojen, nelze výzvu odeslat. V opačném případě již můžeme vyplnit řetězece info a zpráva a výzvu odeslat. Při vyplňování je potřeba brát zřetel na to, že řetězec info je přenášen v čisté podobě, to znamená, že útočník, který zachytí komunikaci, tento řetězec uvidí. Naproti tomu řetezec zpráva je šifrovaný, proto jeho obsah nikdo, kromě správně autentizovaného příjemce, nezjistí.

Po správném odeslání a vytvoření výzvy na serveru se zobrazí: „Challenge created.“. V případě, že se výzva nevytvoří, zobrazí se chybová hláška: „ERROR: Challenge uncreated!“.

O vytvoření výzvy se stará metoda „createChal“, která se skládá z několika částí:

- **Volba šifrovacích a ověřovacích klíčů**
  Zde je zvolen soukromý f a veřejný F (v kódu vyjádřen jako ff) ověřovací klíč a šifrovací klíče z, Z (zz) a Xe. Popis těchto klíčů byl rozebrán v části 1.4.1. Výpočet je znázorněn následujícím zdrojovým kódem:

```
final BigInteger f = new BigInteger(bitLength, rnd);
final BigInteger ff = g.modPow(f, p);

final BigInteger z = new BigInteger(bitLength, rnd);
final BigInteger zz = b.modPow(z, p);
final BigInteger xe = g.modPow(z, p);
```

- **Výpočet podpisových parametrů klíče**
  Následující zdrojový kód ukázuje výpočet podpisových parametrů r a s, které jsou spočítány z haše klíče f, parametru k a veřejných parametrů p, g, q. Postup výpočtu byl detailně rozebrán v 1.3.1.

```
dig.update(f.toByteArray());
final BigInteger h = new BigInteger(dig.digest());
final BigInteger k = new BigInteger(64, rnd);

final BigInteger r = (g.modPow(k, p)).mod(q);
final BigInteger tmp3 = (h.add(r.multiply(sk)).mod(q));
final BigInteger s = (tmp3.multiply(k.modInverse(q))).mod(q);
```

- **Zašifrování ověřovacího klíče**
  Přenášený klíč *key* se skládá z podpisových parametrů, veřejného klíče odesílatele a samotného ověřovacího klíče f, které jsou odděleny pomocí oddělovacího znaku „#“.
  Z šifrovacího klíče Z se vypočítá jeho haš hodnota pomocí zvolené hašovací funkce. K tomuto účelu byla vytvořena funkce *getKeyFromNum*. Z této haše se vytvoří tajný klíč *pk*, kterým se zašifruje přenášený klíč *key*, čímž vznikne kryptogram *ck* (crypted key). To je znázorněno zdrojovým kódem.

```
final SecretKey pk = getKeyFromNum(dig, cip, zz);
c.init(Cipher.ENCRYPT_MODE, pk);
final String key = r.toString() + "#" + s.toString() + "#" +
    vk.toString()+ "#" + f;
final byte[] ck = c.doFinal(key.getBytes());
```

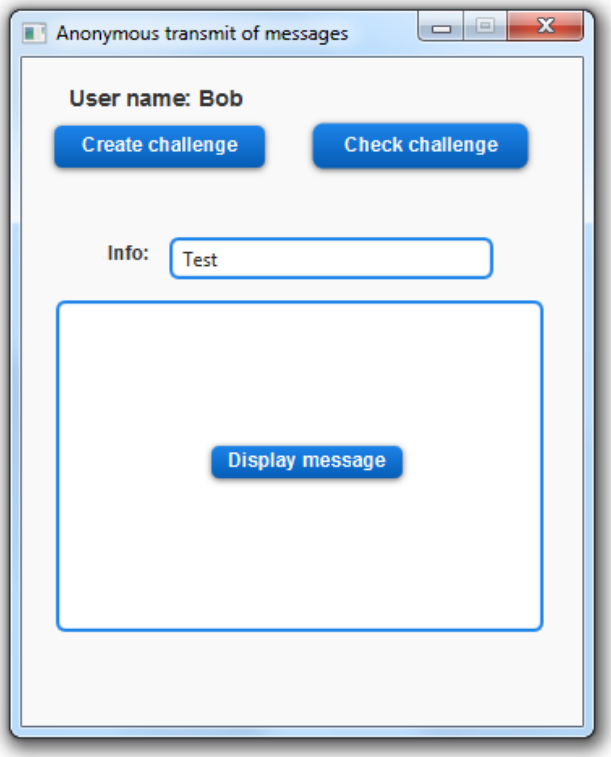
- **Zašifrování zprávy**
  Zašifrování zprávy probíhá stejným způsobem, jako byl zašifrován klíč. Zpráva je zašifrována stejným klíčem f a vznikne kryptogram cm (crypted message).

Vytvořená výzva je spolu se zašifrovanou zprávou zaslána na server. Na serveru se výzvě přiřadí identifikátor Id, zvolí se soukromý parametr výzvy v, dopočítá se veřejný parametr V a poté je vše uloženo do seznamu výzev. Zašifrovaná zpráva se spolu se svým identifikátorem, který je stejný jako má výzva, uloží do seznamu zpráv.

3.6 Získání výzvy

Při stisku tlačítka „Check challenge“ se ze serveru získá seznam všech výzev. Poté uživatel prochází jednotlivé výzvy a zkouší se vůči nim autentizovat. Pokud se mu to podaří, znamená to, že daná výzva je určena pro něj. Z této výzvy získá ověřovací klíč a podpisové parametry, k ověření pravosti klíče. Pomocí ověřovacího klíče se na serveru ověří identita uživatele. V případě kladného vyhodnocení, je uživateli zaslána zpráva a zobrazí se stav ukázaný na obrázku 3.9.

V případě, že žádná výzva není určena pro uživatele, zobrazí se „You have no challenges.“.



Obr. 3.9: Získání výzvy

Uživatel zde má zobrazen řetězec info, podle kterého se může rozhodnout, zda si zprávu zobrazí, či nikoliv. Poté je výzva a zpráva ze serveru odstraněna.

Před získáním zprávy samozřejmě probíhá několik dalších operací. Jsou to:

- **Kontrola výzev**

  Ze serveru je získán seznam všech výzev (*ListOfChals*) a poté se prochází jednotlivé výzvy (*chal*). U každé výzvy se nejprve zjistí, zda už byla kontrolována v předchozích průchodech. Identifikátory již zkontrolovaných výzev jsou uloženy v seznamu *checked*. Pokud výzva nebyla zkontrolována, tak se přejde do bloku *try* (zkusit).

  Zde se pomocí metody *getKey* (popsána dále) vypočítá dešifrovací klíč, s jehož pomocí se zkusí dešifrovat zašifrovaný kryptogram *ck* (získaný pomocí metody *chal.getCk()*). V případě, že se dešifrování podaří, znamená to, že daná výzva je určena pro nás – máme správný soukromý klíč, a můžeme ji dál zpracovávat. V opačném případě nastane vyjímka, kterou zachytí blok *catch* (chytit) – zadali jsme špatný klíč, proto víme, že výzva není pro nás. Její identifikátor je přidán do seznamu zkontrolovaných výzev. Poté můžeme zkontrolovat další výzvu.

  Pokud není žádná výzva pro nás, metoda vrací hodnotu *null*. Kontrola výzev je ukázána následujícím kódem:

```
ListOfChals = ci.getChallenges();
        for (Challenge chal: ListOfChals) {
                if (!checked.contains(chal.getId())){
                        try {
                                Cipher c =
                                    Cipher.getInstance(chal.getCip());
                                c.init(Cipher.DECRYPT_MODE,
                                    getKey(chal));
                                byte[] checkKey =
                                    c.doFinal(chal.getCk());
                                return chal;

                        } catch (BadPaddingException |
                            NoSuchPaddingException |
                            InvalidKeyException |
                            IllegalBlockSizeException e) {
                                checked.add(chal.getId());
                        }
                }
        }
        return null;
}
```

- **Získání dešifrovacího klíče – metoda *getKey***
  Pomocí svého soukromého klíče *sk* a parametru *Xe* vypočítáme dešifrovací klíč *Z*. To je potvrzeno v 1.4.1. Z tohoto klíče spočítáme jeho haš hodnotu a pomocí ní vytvoříme dešifrovací klíč *pk*. To je ukázáno následující částí kódu:

```
final BigInteger zz = chal.getXe().modPow(sk, p);
final SecretKey pk = getKeyFromNum(dig, Scip, zz);
return pk;
```

- **Kontrola podpisu klíče**
  V další části se pomocí dešifrovacího klíče získá přenášený řetězec *dk*, který se pomocí oddělovacího znaku rozdělí na jednotlivé parametry. Poté se ověří správnost získaného klíče pomocí jeho podpisu. Při ověření se z haše přijatého klíče *f*, podpisových parametrů *r*, *s* a veřejného klíče odesílatele *Svk* vypočítá parametr *ver*. Pokud *ver* odpovídá *r*, pak je podpis v pořádku a tudíž ověřovací klíč nebyl změněn. Rozdělení a ověření je ukázáno na následujícím kódu:

```
String[] splitter = dk.split("#");

if (splitter.length == 4) {

        final BigInteger r = new BigInteger(splitter[0]);
        final BigInteger s = new BigInteger(splitter[1]);
        final BigInteger Svk = new BigInteger(splitter[2]);
        final BigInteger f = new BigInteger(splitter[3]);

        dig.update(f.toByteArray());
        final BigInteger h = new BigInteger(dig.digest());
        final BigInteger tmp3 = s.modInverse(q);
        final BigInteger a = tmp3.multiply(h.mod(q)).mod(q);
        final BigInteger b = tmp3.multiply(r.mod(q)).mod(q);

        final BigInteger ver = ((g.modPow(a,
            p)).multiply(Svk.modPow(b, p))).mod(p).mod(q);
        if (ver.equals(r)) return f;
}
return BigInteger.ZERO;
```

- **Ověření uživatele**
  Po získání ověřovacího klíče f vypočítáme *kmacc* klíč. Přesný postup ověření na straně klienta i serveru je popsán v 1.4.1. K ověření se používá tzv. odpověď o, která vznikne zašifrováním haše řetězce *Info*. K zašifrování použijeme klíč *pk*, který opět vytvoříme pomocí metody *getKeyFromNum*, kdy vytvoříme haš hodnotu z klíče *kmacc*.
  Odpověď je zaslána na server, kde se obdobným způsobem také vypočítá odpověď. Pokud jsou stejné je ověření v pořádku a uživateli je zaslána zašifrovaná zpráva. Po získání zprávy proběhne její dešifrování a zobrazení. Poté je výzva i zpráva ze serveru odstraněna. Výpočet odpovědi na straně klienta je ukázán následujícím kódem:

```
final BigInteger kmacc = chal.getVv().modPow(f, p);
final SecretKey pk = getKeyFromNum(dig, Scip, kmacc);
dig.reset();
dig.update(chal.getInfo().getBytes());
c.init(Cipher.ENCRYPT_MODE, pk);
final byte[] o = c.doFinal(dig.digest());
return (ci.verification(chal.getId(), o));
```

  Pokud je autentizace neplatná, tak se uživateli zobrazí hláška: „Authentication is rejected!“ a výzva se zprávou zůstane dál na serveru.

3.7 Testování

Program byl testován na operačních systémech Windows XP a Windows 7.

Testování probíhalo pomocí virtuálních strojů, na kterých byl operační systém Windows XP (anglická verze). U této verze nastal problém se zobrazením českých znaků, jako jsou písmena s háčkem nebo čárkou. Problém jsem vyřešil nastavením podpory pro český jazyk a restartováním stroje. Komunikace mezi systémy Windows 7 a Windows XP také nebyla žádným problémem.

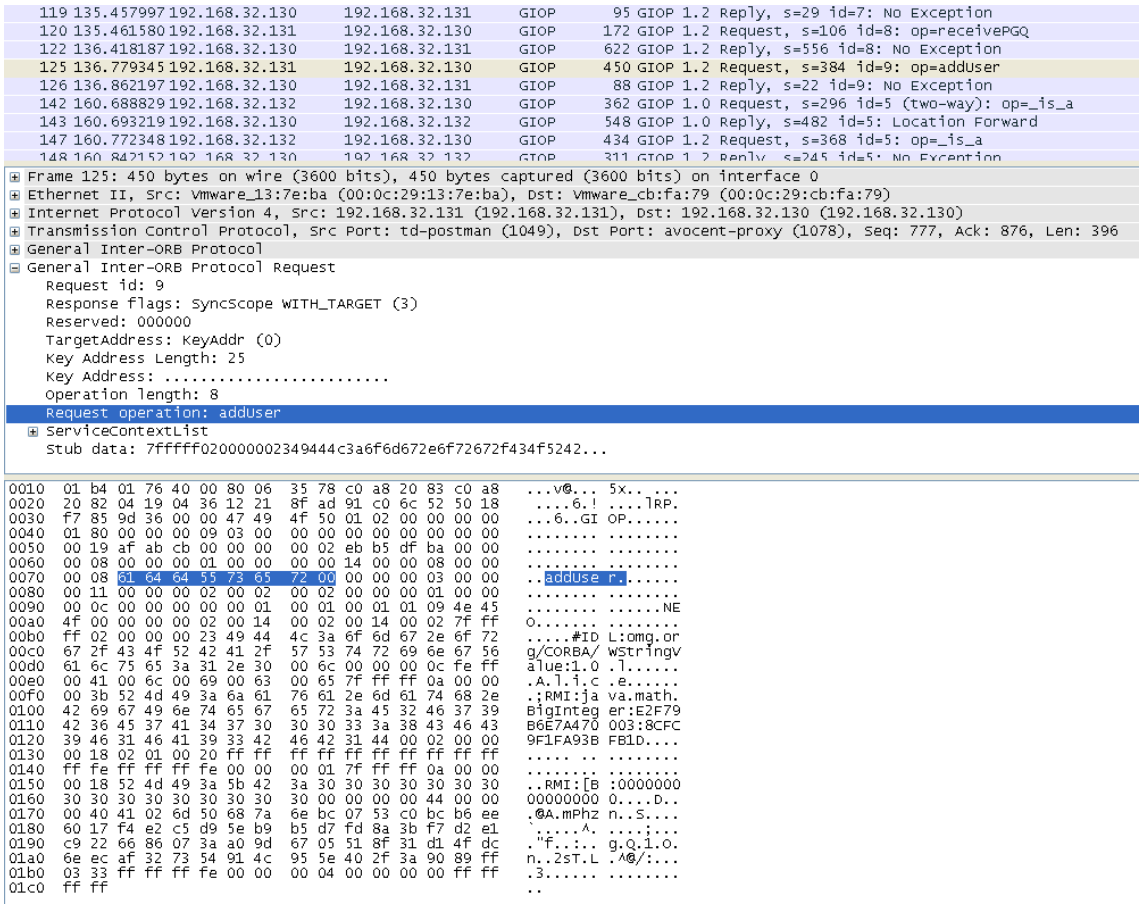
Během tohoto testování byl zachycen přenos dat pomocí programu Wireshark, který slouží ke sledování a analýze síťového provozu. Zachycený přenos, který probíhal na serveru je zobrazen na obrázku 3.10, na kterém je znázorněn pomocí tzv. „Flow graph“.

Obr. 3.10: Zachycená komunikace

Z obrázku vyplývá, že komunikace probíhala mezi třemi stroji, při které se na adrese 192.168.32.130 nacházel server a k němu se připojili 2 uživatelé 192.168.32.131 jako Alice a 192.168.32.132 jako Bob. Jak bylo popsáno v kapitole 1.5, tak komunikace probíhá pomocí žádostí a odpovědí, a všechny žádosti jsou generovány ze stran klientů.

Dále zde můžeme vidět, že nejprve proběhlo připojení k serveru a následné zaregistrování uživatele. Dále Alice vytvořila na serveru výzvu. Bob ji přijal, zkontroloval a poté proběhlo úspěšné ověření Boba a zaslání zprávy. Nakonec se oba uživatelé odhlásili ze serveru. V komentářích jsou zobrazeny jednotlivé metody, které se v komunikaci vyskytují.

Na obrázku 3.11 se podíváme detailněji na obsah přenášených dat.



Obr. 3.11: Zachycená data při registraci uživatele

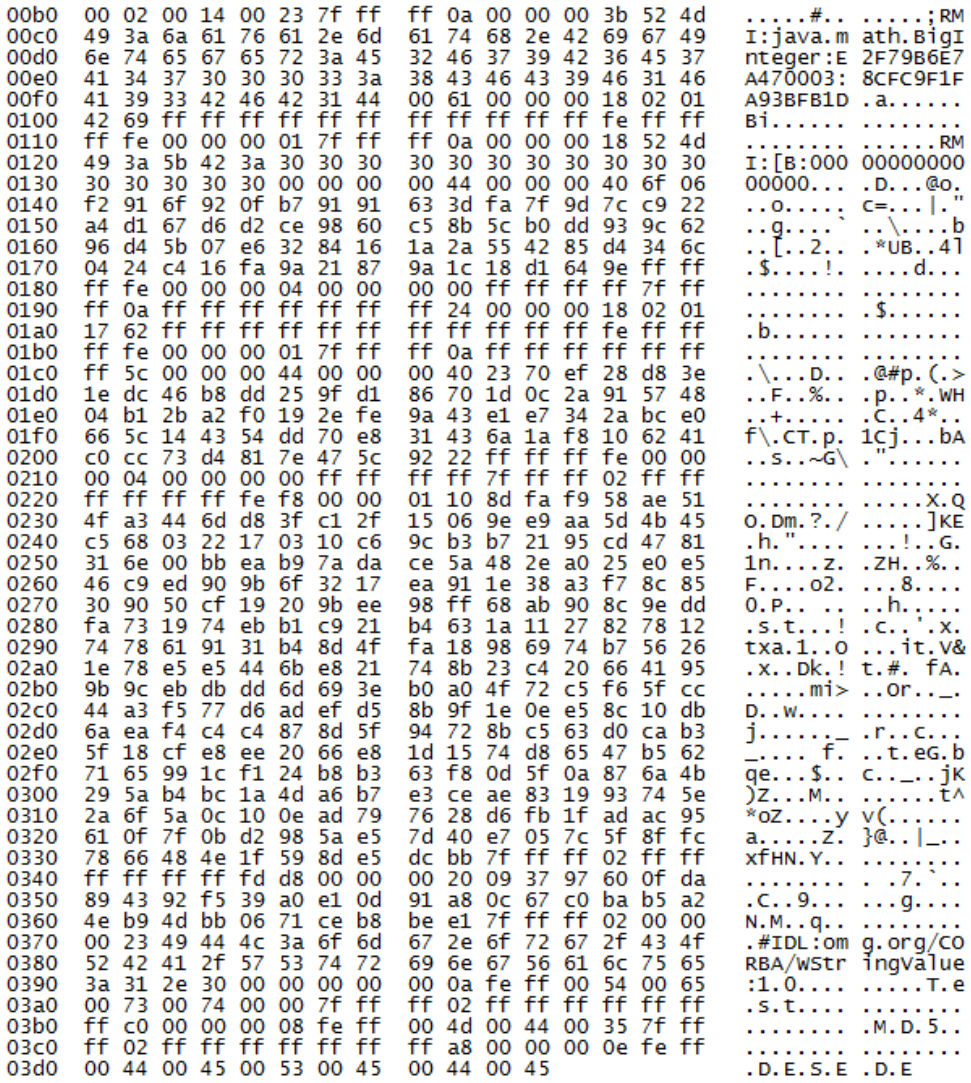
Z těchto dat můžeme vyčíst následující parametry:

- GIOP – protokol použitý pro komunikaci
- addUser – název metody volané na serveru
- IDL:omg.org/CORBA/WStringValue: . . . Alice – definuje typ přenášený přes rozhraní a jeho hodnotu – přenáší uživatelské jméno
- RMI:java:math:BigInteger: E2F79B6E7A470003:8CFC9F1FA93BFB1D – přenos čísla typu BigInteger – přenáší veřejný klíč

Přesnější popis je v kapitole 1.5.2, kde je struktura hlavičky detailněji rozepsána.

Nyní se zaměříme na bezpečnost přenášených dat, což je zobrazeno na obrázku 3.12.

Zde jsou zobrazena data zachycená při vytváření výzvy, ze kterých zjistíme veřejný parametr *Xe*, řetězec info „Test“, typ použité hašovací funkce „MD5“ a typ šifrovací funkce „DESEDE“. Obsah zprávy nebo ověřovací klíč z těchto dat nijak nevyčteme, to je bezpečně zašifrováno.



Obr. 3.12: Zachycená data při vytváření výzvy

Dále probíhalo i síťové testování, kdy na stolním počítači běžel server a na 2 noteboocích klientské aplikace. Vše probíhalo bez problémů, jen je potřeba povolit komunikaci ve firewallu.

4 ZÁVĚR

V teoretické části diplomové práce jsme se nejprve seznámili s autentizací a obecným ověřením uživatelů. Poté jsme se zaměřili na Diffie-Hellmanův protokol pro výměnu zpráv, pomocí něhož dochází ke stanovení společného klíče a také ho využíváme k bezpečnému přenosu hodnot přes nezabezpečenou síť. Pro zvýšení bezpečnosti Diffie-Hellmanova protokolu využíváme digitální podpis DSA.

V další části byl popsán protokol pro anonymní autentizaci na jehož základě budeme náš systém realizovat. V práci je popsán přesný postup protokolu, matematické výpočty i přenášené parametry. Také jsou potrvzeny vlastnosti protokolu.

Po nastudování teoretické části jsem provedl návrh komunikace mezi klientem a serverem a volbu technologie použité pro přenos dat přes síť, při které jsem zvolil technologii pro vzdálené volání metod, neboli RMI-IIOP. Tato technologie byla v práci také popsána, i se zaměřením na typy přenášeých zpráv a jejich formát.

Dále následovala realizace celého systému, který se skládá ze serveru a klientské aplikace. Tento systém jsem vytvořil v programovacím jazyce java.

Princip spočívá v tom, že jednotliví klienti se nejprve zaregistrují na serveru, pomocí svých uživatelských jmen a veřejných klíčů. Poté mohou vytvářet zprávy pro ostatní klienty v podobě výzev, které umístí na server. Ostatní klienti získají ze serveru seznam všech výzev a poté prochází jednotlivé výzvy a zkoušejí se vůči nim autentizovat. V případě, že se uživateli podaří autentizovat se, znamená to, že daná výzva je určena pro něj. Poté z výzvy získá ověřovací klíč a podpisové parametry, s jejichž pomocí zkontroluje platnost klíče. Poté pomocí klíče vytvoří odpověď pro ověření na serveru. Při kladném ověření je uživateli zaslána zašifrovaná zpráva, kterou si dešifruje a zobrazí.

Po vytvoření systému probíhalo testování, při kterém server i klientské aplikace běžely na různých fyzických strojích. Při testování byl zachycen způsob komunikace i rozebrán obsah přenášené zprávy při vytváření výzvy, u které byla prokázána bezpečnost systému.

LITERATURA

[1] BURDA, Karel. *Bezpečnost informačních systémů* [učebnice]. Brno: FEKT VUT v Brně, 2005 [cit. 2013-12-21].

[2] ČLUPEK, Vlastimil. *Kryptografický protokol výměny klíčů Diffie-Hellman.* Brno: FEKT, VUT v Brně, 2010. Dostupné z: <https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=30264>. Bakalářská práce. VUT v Brně. Vedoucí práce Jiří Sobotka.

[3] KRHOVJÁK, Jan a Václav MATYÁŠ. *Autentizace a identifikace uživatelů.* Zpravodaj ÚVT MU [online]. 2007, č. 1, s. 5 [cit. 2013-12-21]. Dostupné z: <http://www.ics.muni.cz/bulletin/articles/560.html>.

[4] LAMPA, Petr. *CORBA a IIOP* [online]. Ústav informatiky a výpocetní techniky, Fakulta elektrotechniky a informatiky, VUT v Brně [cit. 2014-3-20]. Dostupné z: <http://www.fit.vutbr.cz/~lampa/papers/corba.html>.

[5] LEŽÁK, Petr. *Využití Diffie-Hellmanova protokolu pro anonymní autentizaci.* Elektrorevue [online]. 2013, s. 6 [cit. 2013-12-21]. Dostupné z: <http://www.elektrorevue.cz/cz/clanky/informacni-technologie/5/vyuziti-diffie-hellmanova-protokolu-pro-anonymni-autentizaci-1/> ISSN 1213-1539.

[6] NASH, Simon et al. JAVAWORLD. *RMI over IIOP.* [online]. 1999 [cit. 2013-12-21]. Dostupné z: <http://www.javaworld.com/article/2076547/soa/rmi-over-iiop.html>.

[7] ORACLE. *Tutorial: Getting Started Using RMI-IIOP.* [online]. 2013 [cit. 2013-12-21]. Dostupné z: <http://docs.oracle.com/javase/7/docs/technotes/guides/rmi-iiop/tutorial.html#5180>.

[8] SMART, Nigel. *Cryptography: An Introduction* [online]. [cit. 2013-12-21]. 3rd Edition. Dostupné z: <www.cs.bris.ac.uk/~nigel/Crypto_Book/>.

[9] ŠURKOVSKÝ, Martin. *RMI - Distribuované objekty v Javě.* [online]. 2009 [cit. 2013-12-21]. Dostupné z: <http://homel.vsb.cz/~sur096/skola/pds/ref/pds_ref.html>.

SEZNAM ZKRATEK

CORBA	architektura pro ORB komunikaci – Common Object Request Broker Architecture
DSA	algoritmus digitálního podpisu – Digital Signature Algorithm
GIOP	obecný protokol pro ORB komunikaci – General Inter-ORB Protocol
IDL	jazyk pro definici rozhraní – Interface Definition Language
JRMP	java protokol vzdálených metod – Java Remote Method Protocol
ORB	zprostředkovatel žádostí objektu – Object Request Broker
RMI	vzdálené volání metod – Remote Method Invocation
RMI-IIOP	vzdálené volání metod pomocí Internet Inter-ORB protokolu – Remote Method Invocation over the Internet Inter-Orb Protocol
RSA	asymetrický kryptosystém
TCP/IP	Transmission Control Protocol/Internet Protocol

SEZNAM PŘÍLOH

A OBSAH CD

- Složka **Client** obsahuje klientskou aplikaci.
- Složka **Development** obsahuje celý java projekt – klient i server.
- Složka **Documentation** obsahuje dokumentaci k projektu.
- Složka **Server** obsahuje soubory potřebné pro spuštění serveru.
- Soubor **Diplomová práce – Jan Kislinger.pdf**, což je elektronická verze diplomové práce.
- Soubor **README.txt** obsahuje zjednodušený návod ke spuštění serveru i klienta.