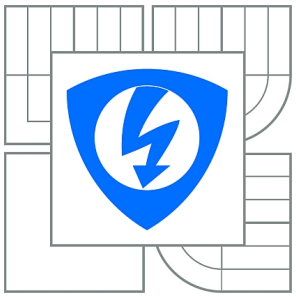
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

DECENTRALIZOVANÝ KOMUNIKAČNÍ NÁSTROJ S GARANCÍ ANONYMITY

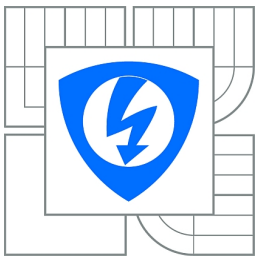
DECENTRALIZED COMMUNICATION TOOL WITH ANONYMITY GUARANTEE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MICHAL LEGÉŇ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN MALÝ
SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Michal Legéň ***ID:*** 78474
Ročník: 2 ***Akademický rok:*** 2009/2010

NÁZEV TÉMATU:

Decentralizovaný komunikační nástroj s garancí anonymity

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte principy anonymních nástrojů pro síťovou komunikaci s decentralizovaných přístupem, zejména s ohledem na metody typu Onion Routing (TOR). Navrhněte a implementujte vlastní systém v jazyce C pomocí jednoduchých konzolových aplikací. Systém by měl splnit kritéria anonymity provozu vzhledem k nezúčastněným jednotkám, nulového centrálního managementu a dynamického provozu vzhledem k přidávání nebo odebírání uzlů za běhu.

DOPORUČENÁ LITERATURA:

[1] Reed M. G., Syverson P. F., Goldschlag D. M.: Anonymous connections and onion routing, IEEE Journal on Selected Areas in Communication Special Issue on Copyright and Privacy Protection, 1998, Vol. 16, Issue 4, pp: 482-494, ISSN: 0733-8716
[2] Donahoo M. J., Calvert K. L.: TCP/IP sockets in C: practical guide for programmers, Morgan Kaufmann, 2001, ISBN: 9781558608269

Termín zadání: 29.1.2010 ***Termín odevzdání:*** 26.5.2010

Vedoucí práce: Ing. Jan Malý

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

V poslednom čase sa do popredia čím viac dostáva otázka zabezpečenia anonymity komunikujúcich strán. Použitím rôznych nástrojov je možné totiž v počítačovej sieti monitorovať činnosť užívateľov, s čím súvisí strata súkromia. Cieľom tejto diplomovej práce je preskúmať možnosti tvorby anonymných systémov, predovšetkým metódy Onion Routing. Práca obsahuje popis tejto metódy spolu s asymetrickým systémom RSA, ako dvoch základných prvkov navrhnutého anonymného systému. Druhá polovica práce je venovaná úvodu do socketového programovania a samotnej implementácií anonymného systému v programovacom jazyku C++. Záver práce je zameraný na analýzu navrhnutého systému z hľadiska bezpečnosti a časových parametrov. Navrhnutý anonymný systém splňuje podmienku anonymity a taktiež decentralizovanosti, vzhľadom na to, že v systéme neexistuje žiadny centrálny prvok a o jeho riadenie sa starajú zúčastnené stanice prostredníctom signalizačných správ.

KLÍČOVÁ SLOVA

anonymita, kryptografia, počítačová bezpečnosť, Onion Routing

ABSTRACT

Anonymity on the internet is becoming a actuall issue nowadays. There are several tools, that can be used to monitor user's activity and it can lead to lose privacy of users. The aim of this master's thesis is to describe different ways of working anonymous systems, especially the method called Onion Routing. The introduction of this work is devoted to the description of this method together with asymmetric cryptosystem RSA. The second part belongs to basics of socket programming and to the implementation of anonymous system in programming language C++. The final part is focussed on analysis of system in terms of security and time complexity. The conditions of anonymity and decentralization are accomplished. There is no presence of central server in the system and the management is handled by signalling messages.

KEYWORDS

anonymity, cryptography, computer security, Onion Routing

LEGÉŇ, Michal *Decentralizovaný komunikační nástroj s garancí anonymity*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2010. 60 s. Vedoucí práce byl Ing. Jan Malý,

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Decentralizovaný komunikační nástroj s garancí anonymity“ 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 jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

Poďakovanie

Ďakujem vedúcemu diplomovej práce Ing. Janu Malému za veľmi užitočnú metodickú pomoc a cenné informácie.
Osobitné poďakovanie patrí mojím rodičom za ich podporu počas celého vysokoškolského štúdia.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Úlohou mojej diplomovej práce bolo naštudovať metódy používané v systémoch, umožňujúcich anonymnú komunikáciu a pokúsiť sa navrhnúť vlastný anonymný systém. Centralizovaný model komunikácie vytvára jednoduchšie podmienky na analýzu a kontrolu dátových tokov v počítačovej sieti, preto jednou z podmienok kladených na systém je decentralizovanosť a dynamickosť systému, umožňujúca bezproblémové pripájanie a odhlasovanie staníc do systému.

V práci bola značná pozornosť venovaná metóde Onion Routing, ktorá je jednou z najznámejších a najpoužívanejších metód v anonymných systémoch. Prvá kapitola práce popisuje v krátkosti princíp tejto metódy v dvoch modifikáciách. Prvá varianta využíva asymetrickú kryptografiu, druhá pracuje pomocou tzv. Information Slicing. Druhá kapitola je zameraná na návrh vlastného anonymného systému využívajúceho Onion Routing v spolupráci s asymetrickou kryptografiou. Pre problémy asymetrickej kryptografie súvisiace s nízkou rýchlosťou šifrovania je vhodné, predovšetkým v prípade veľkých dátových tokov, použiť symetrickú kryptografiu. Cieľom tretej kapitoly je načrtnúť iné možnosti realizácie za účelom nahradiť asymetrickú kryptografiu symetrickou. Súčasťou je tiež popis využitia hashovacích funkcií ako prostriedku na autentizáciu dát. Štvrtá časť diplomovej práce obsahuje podrobný popis algoritmu RSA, vrátane algoritmov ako Miller-Rabinov test prvočísel a Rozšíreného Euklidovho Algoritmu. Súčasťou kapitoly je taktiež vysvetlenie vlastnej implementácie a ukážka jej funkčnosti. Pri programovaní sieťových aplikácií je nutné pracovať so socketmi. Preto základy socketového programovania je obsahom piatej kapitoly. Mnohé z funkcií spomínaných v tejto kapitole bolo použitých pri programovaní výsledného programu. Samotnej implementácií sa venuje kapitola šesť, ktorá obsahuje popis jednotlivých tried. Ako programovací jazyk bol zvolený jazyk C++. Posledná kapitola práce sa venuje analýze výsledného programu z hľadiska bezpečnosti ale aj časových parametrov.

1 SYSTÉMY PRE ANONYMNÚ KOMUNIKÁCIU

1.1 Úvod

Počítačová sieť je skupina prepojených počítačov, umožňujúca jej členom vzájomnú komunikáciu. Dáta sú primárne cez počítačovú sieť prenášané bez ochrany. Boli vyvinuté mnohé šifrovacie algoritmy, umožňujúce transformovať pôvodnú správu do podoby, ktorá je čitateľná iba užívateľovi, disponujúcemu potrebným dešifrovacím kľúčom. Veda o týchto algoritmoch, a teda o metódach utajenia obsahu sa nazýva kryptografia. V súčasnosti používané moderné kryptografické algoritmy sú na vysokej úrovni a zašifrované správy nie sú bez znalosti kľúča výpočtovou technikou v rozumnom čase dešifrovateľné. Kryptografia umožňuje teda utajiť obsah prenášanej správy.

V poslednom čase sa však do popredia čím viac dostáva otázka zabezpečenia anonymity komunikujúcich strán. Použitím rôznych nástrojov je možné totiž v počítačovej sieti monitorovať činnosť užívateľov. Informácie, ktoré sa dajú takýmto spôsobom získať, môžu zahŕňať napríklad zoznam webových stránok ktoré daný užívateľ navštevuje, komu odosiela elektronickú poštu, kde napríklad pracuje a podobne.

Anonymitu na Internete nie je celkom jednoduché zabezpečiť. Súvisí to so spôsobom akým internet pracuje. Využíva pri tom protokolovú sadu TCP/IP, ktorého základom je sieťový protokol IP. Ten k prenášanej správe pridáva IP hlavičku, ktorej súčasťou je IP adresa zdroja a príjemcu správy. Tieto údaje môžu slúžiť k identifikácii komunikujúcich strán. Cieľom je teda chrániť užívateľov pred nežiaducimi osobami, disponujúcimi prostriedkami na monitorovanie komunikácie, vďaka ktorým by boli schopní získať informáciu o identite odosielateľa a príjemcu správy.

Tradičné systémy umožňujúce anonymnú komunikáciu sú podľa [8] označované ako mix-based systémy, ktoré pozostávajú zo sady serverov určených k prenosu správ. Rozšírením týchto systémov sú P2P anonymné systémy, ktoré vznikli aplikáciou mix-based systémov do P2P prostredia, to znamená do prostredia decentralizovaného, kde všetky stanice v rámci systému sú si rovné. Veľká väčšina anonymných systémov využíva metódu označovanú ako Onion Routing [11].

Otázka využiteľnosti anonymných systémov je relatívne jednoduchá. Je výhodné ich použiť v oblastiach kde požadujeme anonymitu a súkromie. Anonymný systém sa dá použiť napríklad na ochranu komunikácie pred nežiaducimi osobami, anonymné prehliadanie webových stránok, pre vyhľadávanie citlivých tém, prístup na

blokované servery apod. Práve posledne menovanému účelu sa viaže aj praktický príklad z [13]. Článok sa venuje cenzúre internetu v Iráne po voľbách. Irán, aby zabránil úniku informácií z a do krajiny, zaviedol niekoľko opatrení. To najpodstatnejšie bolo, že začal filtrovať prístup na zahraničné servery, ktoré sa snažili nezávisle informovať o situácií v Iráne, nie len na základe webových adries ale aj na základe obsahu. Nahrával mu v tom aj fakt, že prístup z Iránskeho internetu smerom von má silný centralizovaný charakter a prechádza cez jedinú centrálnu bránu. Bol zatknutý veľký počet blogerov, ktorí boli odhalení cez IP adresy uchovávané v publikovaných správach. Práve problémy s odhaľovaním užívateľov na základe IP adresy viedlo k nárastu používania anonymného systému TOR [3], ktorý je označovaný aj ako Onion Routing druhej generácie.

1.2 Onion Routing

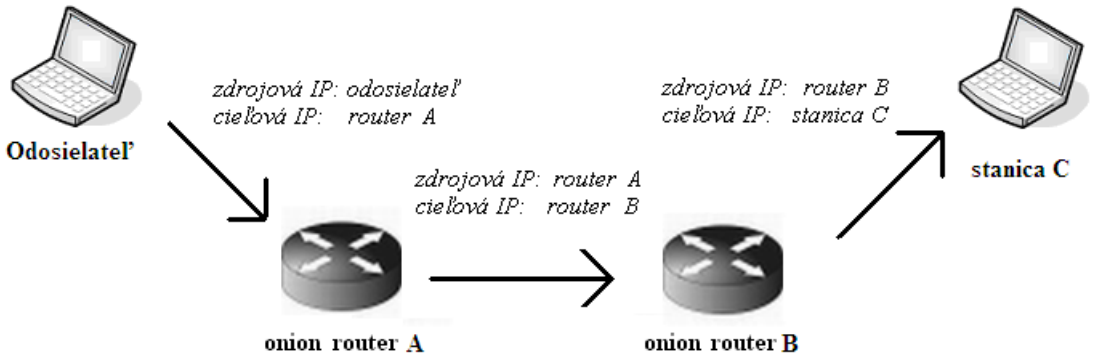
Táto časť podá základný popis metódy Onion Routing, ktorá je základom väčšiny súčasných anonymných systémov. Pri popise sa bude vychádzať z [11]. Onion Routing je metóda pre zaistenie anonymnej komunikácie cez počítačovú sieť. Cieľom analýzy komunikácie je odhaliť kto komunikuje s kým. Metóda Onion Routing poskytuje anonymné spojenia, ktoré sú odolné voči tomuto typu hrozby, to znamená, že pre pozorovateľa je veľmi náročné na základe prenášaných dát presne identifikovať komunikujúce strany.

Princíp metódy Onion Routing spočíva v tom, že namiesto vytvárania priameho spojenia s plánovaným príjemcom dát, vytvorí konkrétna vysielajúca aplikácia spojenie prostredníctvom série uzlov nazývaných onion router. Sieť skladajúca sa z onion routerov umožní, aby odosielateľ a príjemca zostali pre tretiu stranu anonymní. V prípade, že odosielateľ chce zostať anonymný aj pre príjemcu musí z odosielaných dát odstrániť všetky identifikačné informácie.

Onion routre sú navzájom prepojené prostredníctvom trvalých spojení, cez ktoré sú prichádzajúce anonymné spojenia multiplexované a prenášané. Avšak čo je podstatné je, že každý z tejto série routerov pozná iba predchádzajúci a nasledujúci router na ceste a nepozná celú trasu. Preto ani v prípade, ak niektorý z onion routerov zúčastňujúcich sa prenosu je ovládaný treťou, napríklad nedôveryhodnou, stranou nie je táto tretia strana schopná určiť odosielateľa a príjemcu dát.

Dátová jednotka sa označuje onion a má vrstevnatý charakter. Každá vrstva je určená jednému z onion routerov na trase. Prvému onion routru na trase je určená okrajová vrstva onionu. Každá vrstva určuje nasledujúci router na trase. Každý router, ktorý príjme onion, odstráni a dešifruje pomocou svojho súkromného kľúča okrajovú vrstvu a na základe jej analýzy určí následujúci router na trase, ktorému

upravený onion pošle. Posledný onion router zúčastňujúci sa na prenášaní dát, pošle dáta pôvodnému príjemcovi.

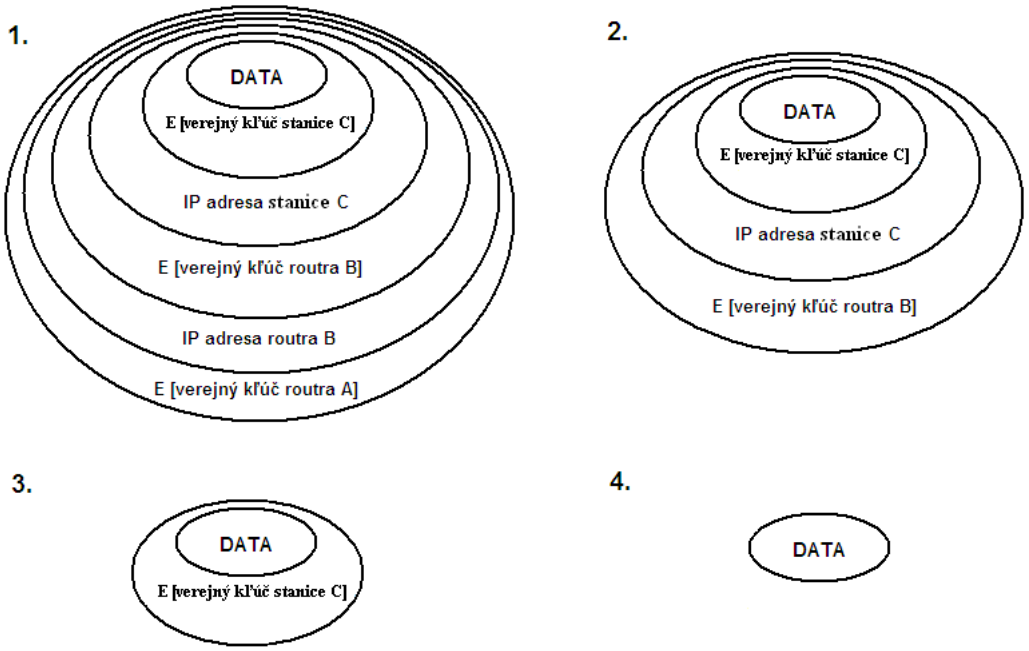


Obr. 1.1: Príklad zmeny IP adries pri onion routingu.

Myšlienka prenosu a zmeny zdrojovej a cieľovej IP adresy je ilustrovaná na obrázku 1.1. Obrázok 1.2 zobrazuje pre tento konkrétny prípad ako vyzerá onion pri prenose. Obrázky zobrazujú situáciu v prípade anonymného spojenia medzi odosielateľom a stanicou C, prechádzajúceho routrami A, B. Operácia E[kľúč] predstavuje asymetrické šifrovanie. Prvý podobrázok predstavuje dátovú jednotku, ktorá je vytvorená na vysielacej strane a bude prenesená postupne obidvoma routrami, začínajúc routrom A. Ten dešifruje pomocou svojho súkromného kľúča prvú vrstvu a dostane sa k IP adrese následujúceho skoku. Tým je router B, ktorý podobne využije svoj súkromný kľúč na dešifrovanie informácie o IP adrese stanice C.

V tomto prípade teda komunikácia prebieha prostredníctvom onion routerov A a B, napriek tomu ani jeden z nich nepozná celú trasu prenosu a tým pádom nie je schopný s určitosťou povedať, kto je odosielateľom a kto príjemcom dát. Z dôvodu vrstevného charakteru onionu, každý z routerov pozná iba predchádzajúci a nasledujúci prvok na trase a ani jeden z nich nemôže s istotou povedať, že uzol, od ktorého onion prijal, je skutočným odosielateľom a uzol, ktorý za ním následuje, je skutočným príjemcom.

Samozrejme popísaná metóda Onion Routing predstavuje základnú a obecnú metódu. Každý anonymný systém implementuje túto metódu v modifikovanej podobe, pričom tieto implementácie sú omnoho zložitejšie. Napríklad pri nasadení do decentralizovaného prostredia sú jednotlivé servery z architektúry vypustené a celý signalizačný systém musí byť zabezpečovaný stanicami v systéme. Princíp prenosu a tvorby onionu je však rovnaký.



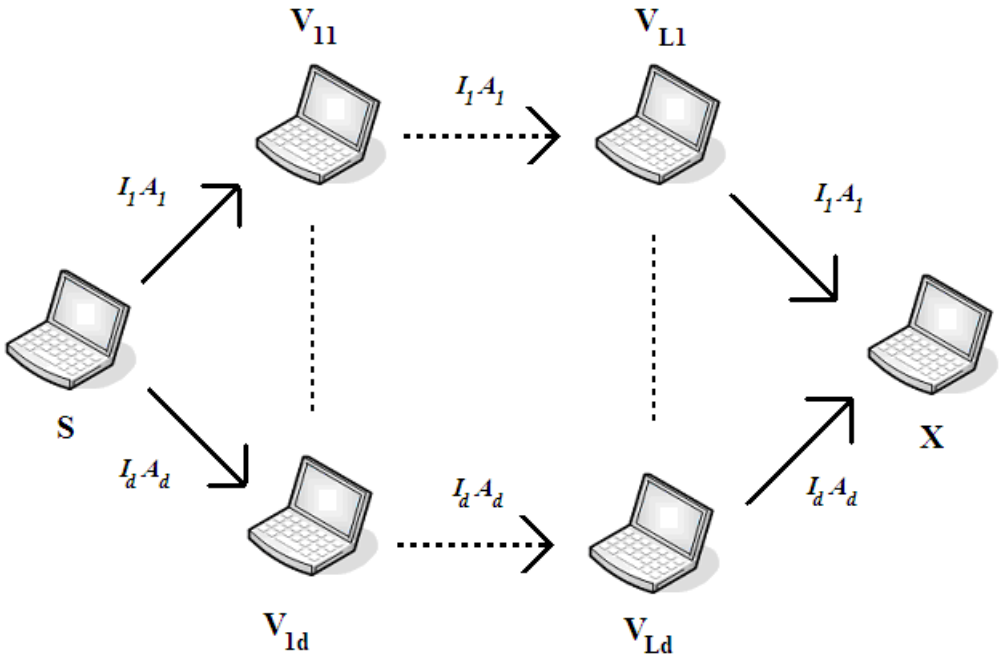
Obr. 1.2: Príklad onionu

1.3 Onion Routing bez použitia asymetrickej kryptografie

Onion Routing je základom veľkej väčšiny súčasných anonymných systémov. Jeho funkcia je podmienená použitím asymetrickej kryptografie. V práci [5] je ukázaná varianta, ktorá nepoužíva asymetrickú kryptografiu. Metóda má však jednu podmienku a tou je, že odosielateľ disponuje viacerými IP adresami. Používajú sa metódy symetrickej kryptografie, ale až v momente prenosu dát. V procese vytvárania anonymného spojenia využíva myšlienku tzv. *Information Slicing*.

Tak ako v klasickom Onion Routingu aj tu, každý uzol zúčastňujúci sa prenosu dátových jednotiek potrebuje pre svoju činnosť individuálny druh informácie [5]. Napríklad uzly na trase potrebujú poznať adresu nasledujúceho uzlu bez toho, aby túto informáciu vedel ktokoľvek iný. Prostredníctvom asymetrickej kryptografie boli tieto informácie šifrované verejnými kľúčmi jednotlivých uzlov, a tak dešifrovať ju bol schopný iba uzol disponujúci patričným súkromným kľúčom. Bez asymetrickej kryptografie je informácia pre jeden z uzlov rozdelená na veľký počet malých častí, ktoré sú nezávislými cestami doručené tomuto uzlu. Tento príjemca po prijatí všetkých častí je schopný spätne poskladať pôvodnú správu a získať tak potrebnú informáciu.

Na obrázku 1.3 je prípad, kedy odosielateľ S chce odoslať správu m uzlu X.



Obr. 1.3: Príklad Information Slicing

Správa sa najprv rozdelí na d blokov m_i. Aby sa jednotlivé správy neposielali v úplne otvorenej podobe, odosielateľ vynásobí vektor $\vec{m} = (m_1, ..., m_\mathrm{i})$, zložený z jednotlivých častí správy, s náhodnou maticou $\mathbf{A}$, ku ktorej musí existovať matica inverzná, a tým vygeneruje d častí ktoré dokopy tvoria náhodnú verziu správy

$$\vec{I} = \mathbf{A} \cdot \vec{m}.$$

Odosielateľ následne vyberie d nezávislých ciest smerom k uzlu X. Po i-tej ceste posiela jednak i-tú časť správy I_i, ale aj $\mathbf{A}_\mathrm{i}$ čo je i-tý riadok matice $\mathbf{A}$. Príjemca po prijatí všetkých častí, dekóduje pôvodnú správu ako

$$\vec{m} = \mathbf{A}^{-1} \cdot \vec{I}.$$

Použitie tohto spôsobu prenosu informácie o adrese nasledujúceho uzlu a iných špecifických informácií smerom ku každému uzlu, ktorý sa zúčastňuje na prenose, je celkom zložité, vzhľadom na veľký počet ciest. Na obrázku 1.4 je príklad riešenia tohto obmedzenia. Odosielateľ má k dispozícii dve IP adresy S a S'. Príjemcom správy je uzol R. Parametre d a L z obrázka 1.3 sú rovné 2 a 3. Cieľom je anonymne oznámiť každému uzlu adresu nasledujúceho uzlu.
Odosielateľ vynásobí IP adresy uzlov s maticou $\mathbf{A}$ veľkosti d x d (2 x 2), ku ktorej, ako bolo spomínané, musí existovať matica inverzná. Napríklad ak Z_l a Z_h sú

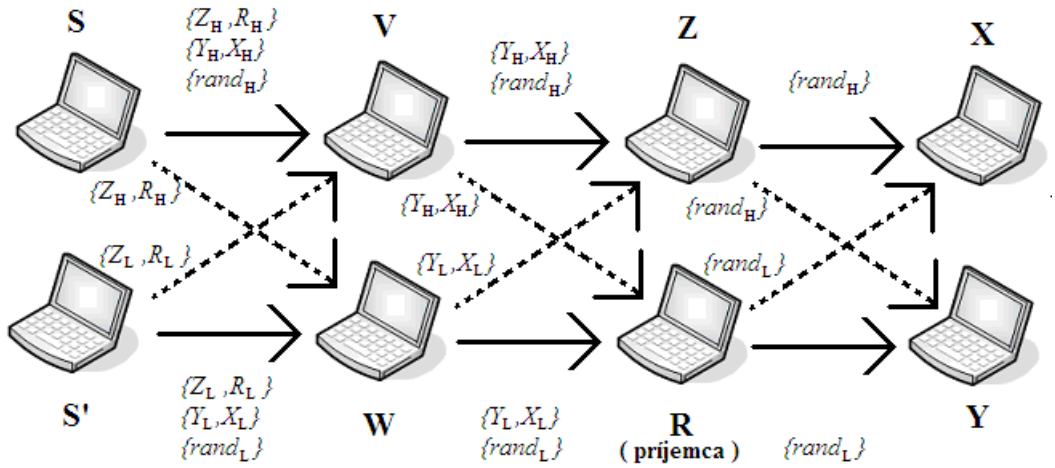
pôvodné časti IP adresy uzla Z, odosielateľ vypočíta

$$\begin{pmatrix} Z_{\mathrm{L}} \\ Z_{\mathrm{H}} \end{pmatrix} = \mathbf{A} \cdot \begin{pmatrix} Z_{\mathrm{l}} \\ Z_{\mathrm{h}} \end{pmatrix},$$

a pošle Z_{L} a Z_{H} dvom rodičovským uzlom, ktorí sa na trase nachádzajú pred uzlom Z, dvomi nezávislými cestami. V tomto prípade sú rodičovské uzly V a W. Na obrázku 1.4 je vidieť prenos všetkých správ takto získaných správ pri vytváraní spojenia. Postup dekódovania týchto správ je podobný. Ako je vidieť z obrázku, uzol V príjme správy { Z_{L}, R_{H} }, { Y_{H}, X_{H} }, { $rand_{\mathrm{H}}$ } od svojho prvého rodiča S. Od druhého rodiča S' príjme { Z_{L}, Z_{H} }. Po prijatí týchto správ uzol V je schopný vypočítať IP adresy nasledujúcich uzlov na trase, teda IP adresy uzlov Z a R, podľa vzťahu

$$\begin{pmatrix} Z_{\mathrm{l}} & R_{\mathrm{l}} \\ Z_{\mathrm{h}} & R_{\mathrm{h}} \end{pmatrix} = \mathbf{A}^{-1} \cdot \begin{pmatrix} Z_{\mathrm{L}} & R_{\mathrm{L}} \\ Z_{\mathrm{H}} & R_{\mathrm{H}} \end{pmatrix}.$$

Čo je ešte podstatné je fakt, že spolu s informáciou o IP adrese nasledujúceho uzla na trase sa prenáša aj kryptografický kľúč pre symetrické šifrovanie a indikátor udávajúci, či uzol je príjemcom dát, pretože príjemcom nemusí byť nutne stanica z poslednej vrstvy prenosového reťazca. V tomto konkrétnom prípade je príjemcom stanica R v predposlednej vrstve. Po prenose a dekódovaní všetkých týchto správ, každý uzol jednak pozná nasledujúci uzol na trase a zároveň zdieľa tajný kľúč s odosielateľom. Odosielateľ môže v tejto fáze použiť klasický algoritmus Onion Routing na prenos samotných dát, šifrovaním jednotlivých vrstiev onionu, použitím tajných kľúčov jednotlivých uzlov na trase, teda použitím symetrickej kryptografie.



Obr. 1.4: Príklad prenosu správ Information Slicing

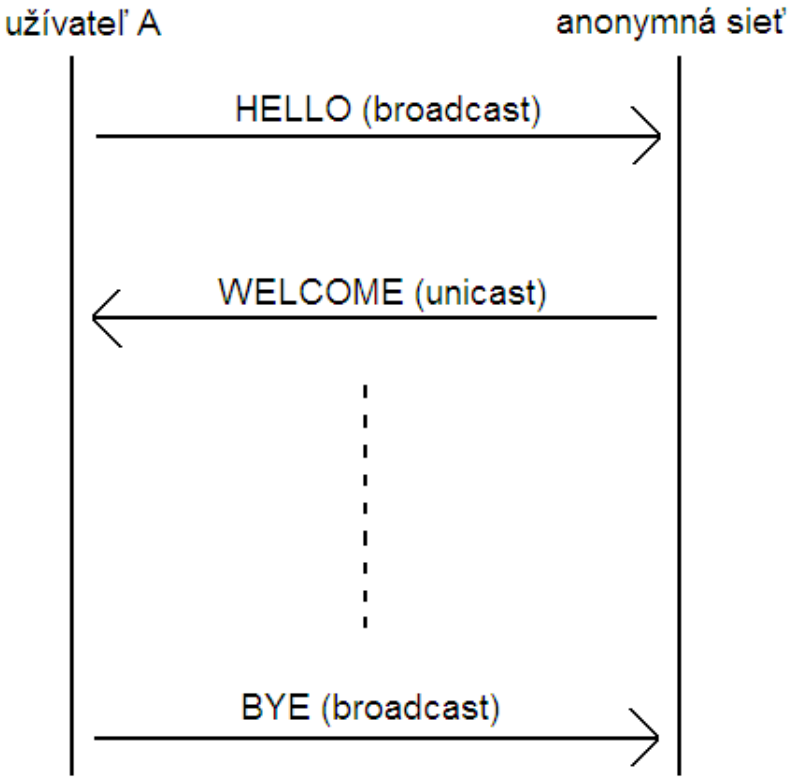
2 NÁVRH ANONYMNÉHO SYSTÉMU

V tejto kapitole bude popísaný návrh decentralizovaného systému využívajúceho metódu Onion Routing spoločne s asymetrickou kryptografiou na zaistenie anonymity užívateľov.

2.1 Prihlasovanie a odhlasovanie

Podstatou decentralizovaného systému je fakt, že v systéme neexistuje externá signalizácia, to znamená, že o správnu činnosť sa starajú zúčastnené jednotky, ktoré musia pomocou navzájom si vymieňajúcich správ externú signalizáciu nahradiť. Jednou z činností, ktoré zúčastnené uzly musia zabezpečiť je ich prihlasovanie a odhlasovanie do systému.

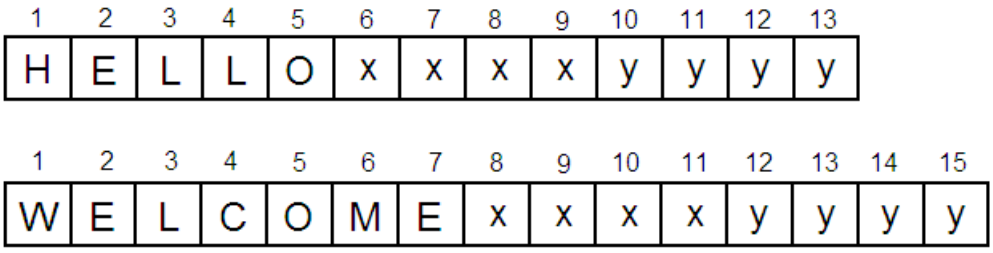
V rámci navrhnutého systému je táto činnosť riešená prostredníctvom troch správ HELLO, WELCOME a BYE, tak ako to ukazuje obrázok 2.1, v prípade prihlásenia a odhlásenia užívateľa A.



Obr. 2.1: Správy pri prihlasovaní a odhlasovaní

Ak sa chce užívateľ A prihlásiť do anonymnej siete vysiela broadcast správu HELLO. Stanice, ktoré sú už súčasťou systému si na základe tejto správy uložia IP adresu užívateľa A a každá stanica vysiela na jeho IP adresu správu WELCOME. Pomocou týchto správ užívateľ A získa zoznam všetkých staníc v systéme. Pri odhlasovaní stačí, ak užívateľ odošle na broadcast adresu správu BYE. Ostatné stanice po prijatí tejto správy vymažú jeho IP adresu zo zoznamu dostupných staníc.

Na správnu funkciu metódy Onion Routing je potrebná asymetrická kryptografia. V systéme bude použitá asymetrická funkcia RSA[1]. Na prenos verejných kľúčov budú použité už spomenuté správy HELLO a WELCOME. Ich štruktúru je vidieť na obrázku 2.2. Obsah správy HELLO má 13 bajtov a WELCOME 15 bajtov. Štyri bajty označené ako x u oboch správ predstavuje verejný kľúč VK a bajty označené písmenom y obsahujú verejný parameter n. Veľkosť týchto správ bude závisieť na dĺžke verejných kľúčov. V tomto ukážkovom prípade sa uvažovali 32 bitové kľúče. Spolu teda s IP adresou si stanica po prijatí správy HELLO ukladá aj verejné parametre systému RSA, obdobne je to aj u správy WELCOME.



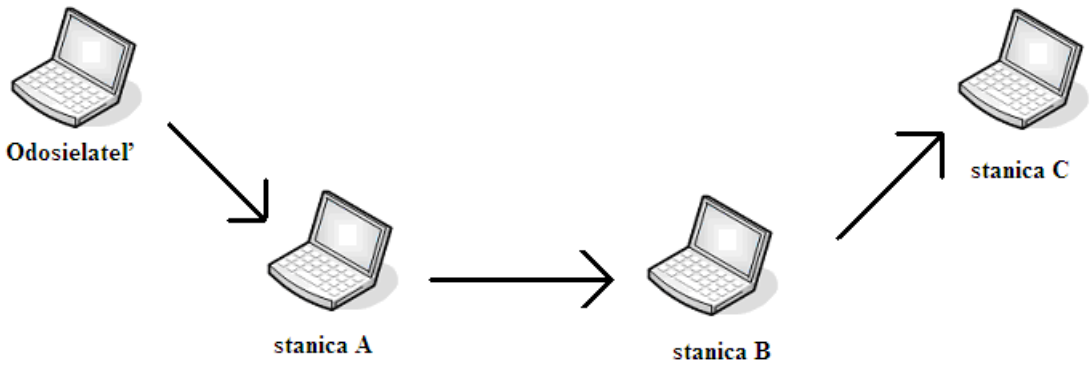
Obr. 2.2: Obsah správ HELLO a WELCOME

2.2 Prenos a dátové jednotky

V systéme bude pri prenose správ umožnené použiť metódu Onion Routing. Priebeh prenosu a vytvárania onionu bude vysvetlený na nasledujúcom príklade. Odosielateľ chce odoslať textovú správu „text“ užívateľovi C s IP adresou IP_{C}. Chce pri tom použiť metódu Onion Routing a správu smerovať postupne cez uzly A a B s IP adresami IP_{A} a IP_{B}, tak ako je to zobrazené na obrázku 2.3.

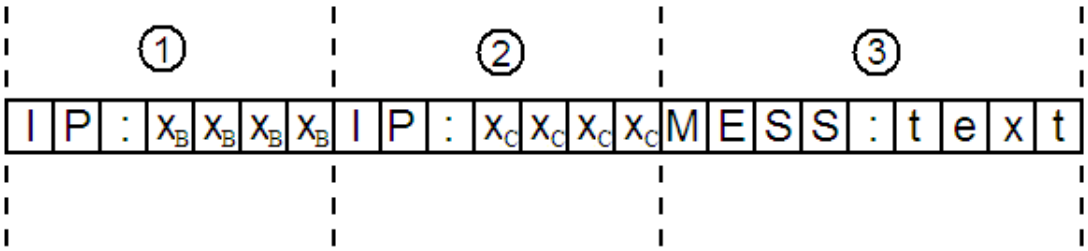
Najprv sa vytvorí nezašifrovaná dátová jednotka, ktorú je vidieť na obrázku 2.4. Skladá sa z troch častí. Počet týchto častí je rovný poradovému číslu príjemcu na

[1]Rivest, Shamir, Adleman



Obr. 2.3: Príklad trasy

trase. Stanica C ako príjemca je tretia v poradí preto práve 3 časti. Každá časť je určená pre jednu stanicu na trase. Časť označená číslom 3 obsahuje samotný text správy a je určená pre konečného príjemcu C. Bajty „MESS:“, ktoré sú súčasťou tejto časti slúžia na to, aby po jej dešifrovaní užívateľ vedel, že ide o konečnú časť. Obdobne je to s bajtami obsahujúcimi „IP:“, ktoré sú súčasťou častí 1 a 2. Tie staniciam hovoria, že správu májú poslať stanici, ktorej IP adresa nasleduje. Bajty $X_BX_BX_BX_B$ a $X_CX_CX_CX_C$ vyjadrujú IP adresu staníc IP_B a IP_C.

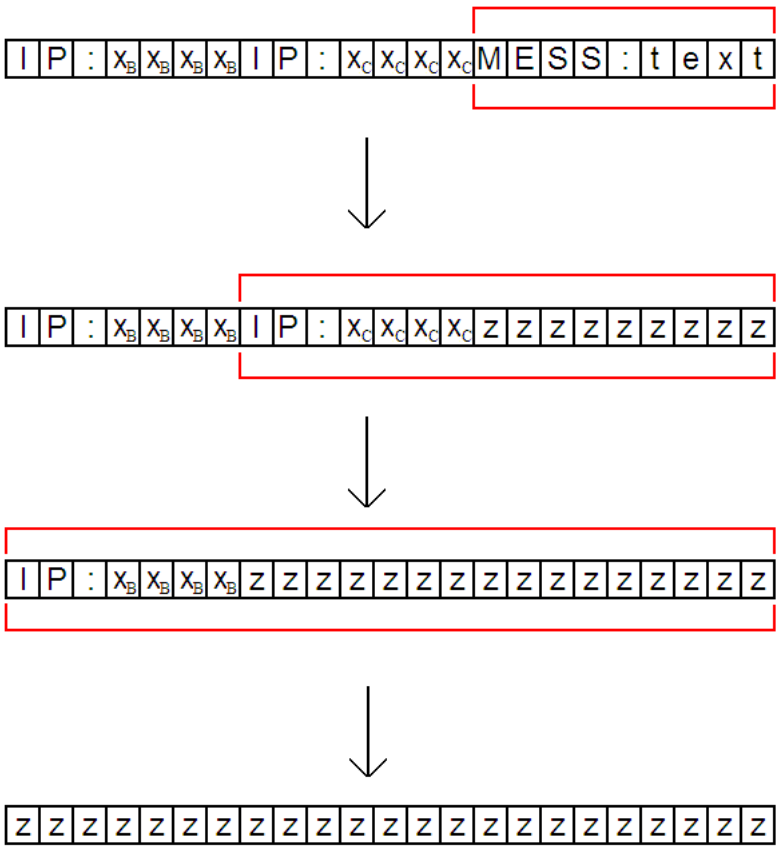


Obr. 2.4: Nezašifrovaný onion

Z tejto dátovej jednotky je postupným šifrovaním vytvorená dátová jednotka onion, ktorá je poslaná prvej stanici na trase a to stanici A s IP adresou IP_A. Proces postupného šifrovania je zobrazený na obrázku 2.5. Ako je vidieť najprv dôjde k zašifrovaniu 3. časti dátovej jednotky pomocou verejného kľúča užívateľa, ktorému chceme textovú správu poslať a to je stanica C. Bajty označené ako z predstavujú už zašifrované bajty. V ďalšom kroku je pomocou verejného kľúča stanice B zašifrovaná dokopy časť 2 a už predtým zašifrovaná časť 3. Na koniec, zašifrujeme časť 1 spolu s výsledkom predchádzajúceho kroku pomocou verejného kľúča stanice A. Takto

vytvorený onion je poslaný stanici A.

Proces postupného dešifrovania je opačný. Stanica A po prijatí onionu ho svojím súkromný kľúčom dešifruje a na základe bajtov obsahujúcich „IP:“ zistí, že nie je konečným príjemcom. Odstráni dešifrovanú vrstvu a pošle onion na adresu ktorú vyčíta z bajtov $X_BX_BX_BX_B$. Podobne postupuje aj stanica B. Rozdiel je u konečnej stanici C, ktorá na základe bajtov obsahujúcich „MESS:“ zistí, že je konečným príjemcom a proces končí.



Obr. 2.5: Postupné šifrovanie onionu

3 INÉ MOŽNOSTI REALIZÁCIE

Návrh komunikačného modelu z kapitoly 2 je pri použití správneho asymetrického systému s dostatočnou dĺžkou šifrovacieho kľúča bezpečný. Problémom však môže rýchlosť. Asymetrické kryptosystémy pracujú s veľkými číslami, preto je asymetrická krptografia na rozdiel od tej symetrickej relatívne pomalá. V navrhnutom anonymnom systéme sa predpokladá prenos krátkych textových správ medzi užívateľmi, bez možnosti prenosu veľkých dátových súborov. Použitie asymetrického kryptografického systému na šifrovanie prenášaných dát je v takomto prípade vhodné. V prípade prenosu veľkých dátových objemov by bolo potrebné zvoliť iný prístup, ktorý by umožňoval použitie symetrickej kryptografie na prenášané dáta respektíve na zašifrovanie a dešifrovanie onionu. Vtedy je však potrebné bezpečne doručiť kľúče k všetkým zúčastneným uzlom. V tejto kapitole budú popísané dva základné postupy ako toto docieliť. Prvým je využitie Diffie-Hellmanovho protokolu, a tým druhým použitie asymetrického kryptosystému. Záver kapitoly je venovaný možnosti využitia hashovacích funkcií v anonymnom systéme ako prostriedok na autentizáciu dát.

3.1 Diffie-Hellmanov protokol

Diffie-Hellmanov protokol patrí do skupiny kryptosystémov založených na probléme nájdenia diskrétneho logaritmu [2]. Jeho primárnou úlohou je bezpečné vytvorenie kľúčov pre symetrický kryptosystém cez prenosové médium. Jeho účelom teda nie je šifrovanie prenášaných správ. Veľkou výhodou je predovšetkým skutočnosť, že ak útočník monitoruje výmenu správ, nie je schopný zistiť aký kľúč bol ustanovený.

Celý proces Diffie-Hellmanovho protokolu je vidieť na obrázku 3.1. Parameter p je veľké prvočíslo a g je označovaný ako primitívny koreň. Obidva parametre musia byť verejné. Algoritmus začína tým, že užívateľ X vygeneruje náhodné veľké číslo a, užívateľ Y číslo b. Tieto čísla však musia zostať utajené. Užívateľ X následne vypočíta a odošle užívateľovi Y číslo $A = g^a \bmod p$, opačným smerom putuje správa s číslom $B = g^b \bmod p$. Obidve strany teraz majú k dispozícií hodnoty, pomocou ktorých sú schopní vypočítať rovnakú hodnotu kľúča. Užívateľ X vypočíta kľúč ako

$$K = B^a \bmod p\,, \tag{3.1}$$

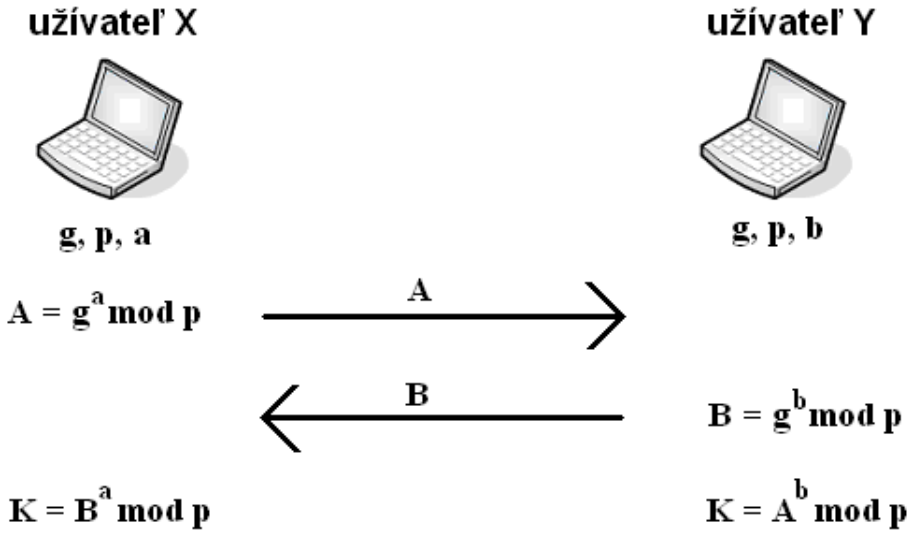
obdobne užívateľ Y určí jeho hodnotu podľa vzťahu

$$K = A^b \bmod p\,. \tag{3.2}$$

Fakt, že obidve strany získajú rovnakú hodnotu kľúča prostredníctvom odlišných vzorcov vyplýva z rovnosti

$$K = A^b \bmod p = (g^a)^b \bmod p = (g^b)^a \bmod p = B^a \bmod p\,. \tag{3.3}$$

Vzťahy pre výpočet šifrovacích kľúčov obsahujú utajené čísla a, b. Bez znalosti týchto údajov nie je možné hodnotu šifrovacieho kľúča vypočítať. Ak chce útočník tieto čísla na základe prenášaných hodnôt A, B vypočítať, musí vyriešiť problém diskrétneho logaritmu. Táto úloha je podľa [2] pre prakticky používané hodnoty parametru p v rozsahu hodnôt 2^{768} až 2^{1024} v súčasnej dobe prakticky neriešiteľná.



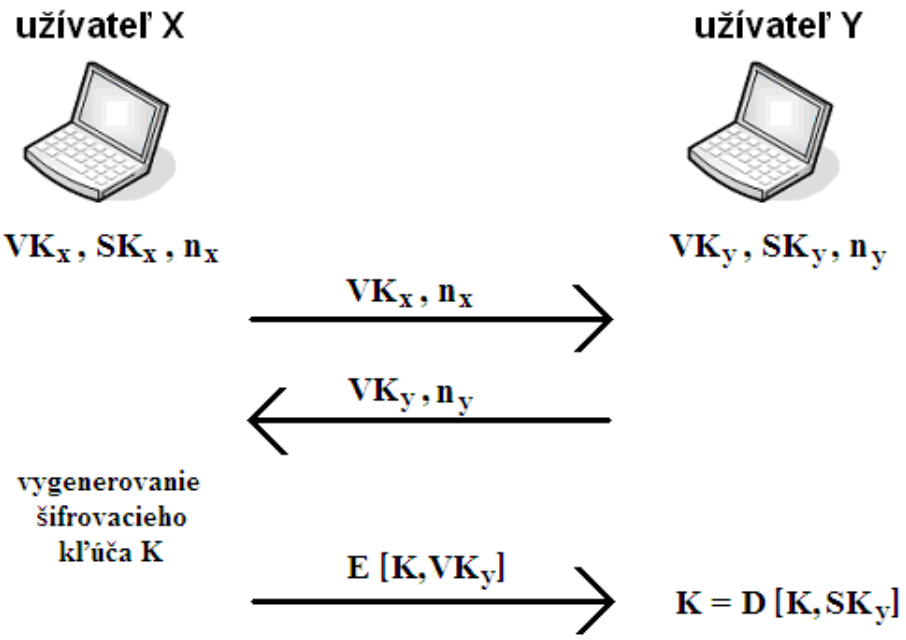
Obr. 3.1: Diffie-Hellmanov protokol

3.2 Využitie asymetrickej kryptografie

Vzhľadom na to, že systémy asymetrickej kryptografie musia pracovať s veľmi veľkými číslami, je ich rýchlosť šifrovania oproti symetrickým systémom omnoho menšia. Preto ich hlavnou doménou nie je šifrovanie dát, predovšetkým ak sa jedná o veľké objemy dát, ale autentizácia dát, digitálny podpis alebo fáza ustanovenia šifrovacích kľúčov pre symetrickú kryptografiu.

Obrázok 3.2 ukazuje príklad bezpečného prenosu šifrovacieho kľúča pre symetrický kryptosystém pomocou asymetrického. Najprv si užívatelia anonymného systému vymenia verejné parametre kryptosystému. Užívateľ X vygeneruje šifrovací kľúč K a zašifruje ho pomocou verejného kľúča užívateľa Y. Po prijatí tejto správy ju užívateľ Y dešifruje svojím súkromným kľúčom. Keďže nikto iný nie je schopný

dešifrovať túto správu, užívatelia X a Y sú jedinými vlastníkmi tajného kľúča K, ktorý môžu následne použiť na symetrické šifrovanie pri tvorbe onionu. Medzi najznámejšie kryptosystémy patrí systém RSA, ktorému je venovaná kapitola 4.



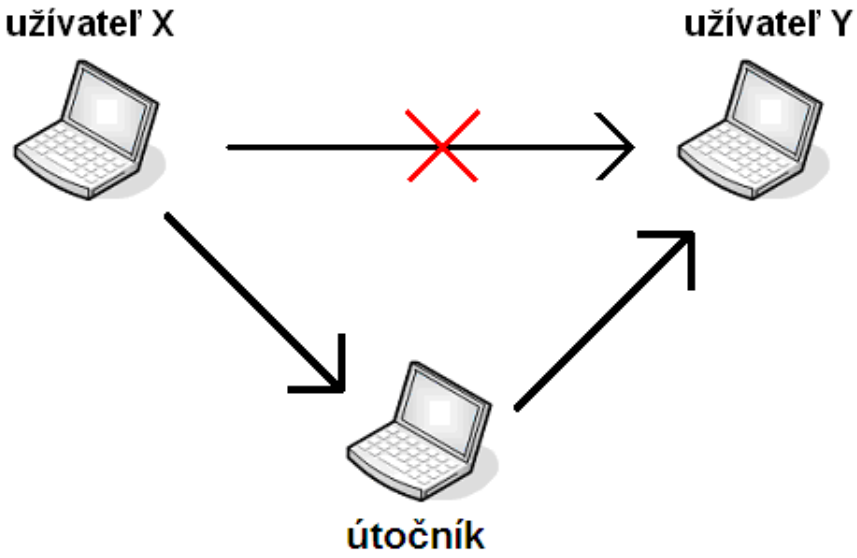
Obr. 3.2: Výmena šifrovacieho kľúča pomocou asymetrického kryptosystému

3.3 Infraštruktúra verejných kľúčov

Použitím vyššie opísaných postupov na bezpečný prenos kľúčov pre symetrické šifrovanie sa ale dostávame k otázke bezpečného doručenia verejných parametrov k druhému účastníkovi. To znamená k problému ako overiť, či prijaté verejné parametre skutočne patria udávanej osobe a nie nejakému útočníkovi, ktorý sa za danú osobu vydáva [2]. Tento typ útoku sa môže označovať ako *man-in-the-middle attack* a jeho princíp vidieť na obrázku 3.3. Pôvodná komunikácia je pri ňom vedená cez útočníka, ktorý pre príjemcu vystupuje ako odosielateľ a naopak.

Tento problém rieši infraštruktúra verejných kľúčov tzv. PKI, prostredníctvom certifikátov. Certifikát je niečo ako elektronický dokument, obsahujúci verejné parametre, schválený dôveryhodnou stranou. Ak chcú účastníci X a Y komunikovať, tak si navzájom vymenia certifikáty, následne si musia overia ich platnosť u certifikačnej autority a na základe verejných parametrov obsiahnutých v certifikáte môžu bezpečne komunikovať s vedomím, že verejné parametre patria skutočne udávanej

osobe. Certifikát môže obsahovať jednak verejný kľúč asymetrického kryptosystému, ale taktiež môže obsahovať verejné parametre Diffi-Hellmanovho protokolu.



Obr. 3.3: Man-in-the-middle attack

3.4 Využitie hashovacích funkcií

Hashovacie funkcie sú neoddeliteľnou súčasťou modernej kryptografie. Ich úlohou je vytvárať pre ľubovoľne dlhé správy jedinečné digitálne otisky nazývané tiež hash. Tieto výstupy hashovacích funkcií majú presne definovanú dĺžku. Tabuľka 3.1 obsahuje prehľad najznámejších hashovacích funkcií s dĺžkou ich digitálneho otisku.

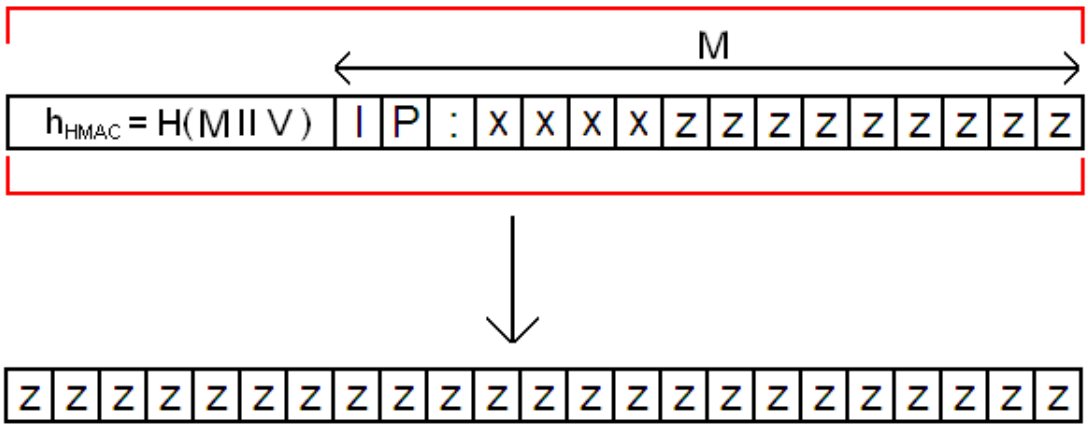
Na hashovacie funkcie je kladených niekoľko požiadaviek na to, aby ich bolo možné bezpečne používať. Patria k ním jednosmernosť, ktorá znamená, že z otisku h nemôže byť inverzným postupom získaná pôvodná správa Z. Ďalšou veľmi dôležitou vlastnosťou je bezkolíznosť. Táto vlastnosť zaručuje, že neexistujú dve správy, ktorých otisk by bol rovnaký. Samozrejme kolízie určite existujú, vzhľadom na to, že množina digitálnych otiskov u každej hashovacej funkcie je konečná a vstupných správ je teoreticky nekonečne veľa. Preto musí byť minimálne výpočetne nezvládnuteľne nájsť dve správy s rovnakým digitálnym otiskom. Mnohé hashovacie funkcie však túto vlastnosť už nesplňujú. Kolízie sa ešte delia na kolízie prvého a druhého radu. Kolízia prvého radu znamená nájdenie dvoch ľubovoľných správ, ktorých digitálny otisk je rovnaký. Jej zložitosť vychádza z tzv. narodeninového paradoxu a je rovná $2^{n/2}$, kde n je dĺžka digitálneho otisku. Kolízia druhého radu je schopnosť nájsť k existujúcej správe inú správu tak, že ich digitálny otisk je rovnaký. Jej zložitosť

sa udáva ako 2^n, kde n je dĺžka digitálneho otisku. U niektorých funkcií, napríklad MD5 alebo SHA-1, bola zložitosť nachádzania kolízií vplyvom kryptoanalytických metód znížená a nepovažujú sa za bezpečné.

Hashovacia funkcia		**Dažka digitálneho otisku**
MD5		128 bitov
SHA-1		160 bitov
SHA-2	SHA-224	224 bitov
	SHA-256	256 bitov
	SHA-384	384 bitov
	SHA-512	512 bitov

Tab. 3.1: Hashovacie funkcie

Hashovacie funkcie sa používajú napríklad ako prostriedok na ukladanie hesiel do databáz, sú základom digitálneho podpisu a umožňujú kontrolu integrity prenášaných dát. Digitálny otisk $h = H(M)$ závisí iba na správe M, to znamená, že otisk h môže spočítať ktokoľvek kto pozná typ použitej hashovacej funkcie H a správu M. K správe sa však môže pridať nejaká tajná hodnota V, tým dostaneme otisk $h_{\text{HMAC}} = H(M \parallel V)$[1]. Výsledný digitálny otisk h_{HMAC} potom závisí aj na tejto tajnej hodnote V a spočítať ho môže len osoba, ktorá ňou disponuje. V tomto prípade sa hashovacia funkcia mení na tzv. autentizačnú funkciu HMAC [2], slúžiacu na overenie integrity dát u osoby, ktorá pozná hodnotu V.



Obr. 3.4: Jedna vrstva onionu s použitím hashovacej funkcie

[1]Operácia $\parallel$ znamená spojenie reťazcov.

Vzhľadom na to, že dátová jednotka onion pri prechode sieťou je spracovávaná vo viacerých uzloch, je možné pre zvýšenie bezpečnosti implementovať do systému autentizačnú funkciu. Predpokladajme, že odosielateľ zdieľa s každou stanicou inú tajnú hodnotu V. Táto hodnota mohla byť opäť ustanovená jednak prostredníctvom Diffie-Hellmanovým protokolom alebo pomocou asymetrickej kryptografie. Pri vytváraní onionu potom odosielateľ pridá do každej vrstvy jednak IP adresu nasledujúceho uzlu, tak ako je to vidieť na obrázku 3.4, ale aj otisk $h_{\mathrm{HMAC}} = H(M \parallel V)$. V je tajná hodnota zdieľaná so stanicou pre ktorú je daná vrstva určená a M je celá aktuálna vrstva onionu. Ako onion putuje sieťou, každý uzol po prijatí dešifruje jednu vrstvu. Následne spočíta vlastný otisk h_{HMAC} na základe svojej hodnoty V. Ak sa takto získaný otisk zhoduje s otiskom, ktorý je súčasťou vrstvy onionu, všetko je v poriadku a onion je odoslaný ďalšiemu uzlu na trase. Ak sa však nezhodujú, niekde došlo k chybe, poprípade bol onion na trase úmyselne upravený a je nutné na to vhodným spôsobom zareagovať, napríklad zahodením onionu a vyslaním chybovej správy.

4 ASYMETRICKÝ KRYPTOSYSTÉM RSA

Obsah kapitoly bude zameraný na popis algoritmu a implementácie asymetrického kryptosystému RSA, ktorý tvorí základ navrhnutého anonymného systému. Pri implementácií boli využité bohaté možnosti voľne dostupnej matematickej knižnice NTL[1]. Pre implementáciu bol zvolený programovací jazyk C++. Pri kompilácií zdrojových kódov bolo použité vývojové prostredie wxDev-C++.

4.1 Algoritmus RSA

Asymetrický kryptosystém RSA patrí do skupiny označovanej ako faktorizačné systémy. Tie sú založené na probléme faktorizácie čísla, čo znamená problém rozloženia čísla na súčin mocnín prvočísel [2]. Samotný algoritmus RSA pozostáva z niekoľkých krokov:

1. Prvým krokom je stanovenie dvoch veľkých prvočísel p, q (100 až 200 dekadických čísel).
2. Výpočet čísla $n = p \cdot q$.
3. Výpočet čísla $r = (p-1) \cdot (q-1)$.
4. Ďalší krok spočíva vo voľbe verejného kľúča VK. Voľba musí spĺňať dve podmienky. Prvá podmienka je $1 < VK < r$. Druhou je fakt, že VK a r musia byť nesúdeliteľné, čo znamená, že tieto čísla nemôžu mať žiadneho spoločného deliteľa okrem 1.
5. Predposledným krokom je výpočet súkromného kľúča SK z podmienky $VK \cdot SK \bmod r = 1$.
6. Posledným krokom je zverejnenie verejných parametrov, ktoré tvoria VK a n. Parameter SK ako súkromný kľúč musí zostať utajený.

Pred samotným šifrovaním dát, sa správa rozdelí na bloky M rovnakej dĺžky. Na každý blok M pozeráme ako na číslo, pričom musí platiť $M < n$. Šifrovanie prebieha podľa vzťahu

$$C = M^{VK} \bmod n\,, \tag{4.1}$$

a dešifrovanie podľa

$$M = C^{SK} \bmod n\,. \tag{4.2}$$

Ak chce útočník dešifrovať zašifrované dáta, zo vzťahu 4.2 vyplýva nutnosť mať k dispozícií súkromný kľúč SK. Ten sa dá získať prostredníctvom čísla $r = (p-1) \cdot (q-1)$. Neznáme prvočísla p, q môže útočník získať z verejného parametru

[1]NTL je voľne dostupná matematická knižnica, umožnujúca prácu s ľubovoľne dlhými číslami. Je dostupná z [9].

n respektíve jeho rozkladom na súčin prvočísel. Číslo n nadobúda bežne hodnoty 2^{768} až 2^{2048}. Podľa [2] je rozklad takto veľkých čísel v súčasnej dobe nereálny.

4.2 Voľba prvočísel p, q

Prvočísla p, q by mali byť približne rovnako dlhé a zároveň ich voľba by mala byť náhodná. Na overenie toho, či vygenerované náhodné číslo je skutočne prvočíslo sa v tomto kroku používa tzv. Miller-Rabinov test [7]. Ide o pravdepodobnostný algoritmus, používaný na overenie toho, či zadané číslo je prvočíslo.

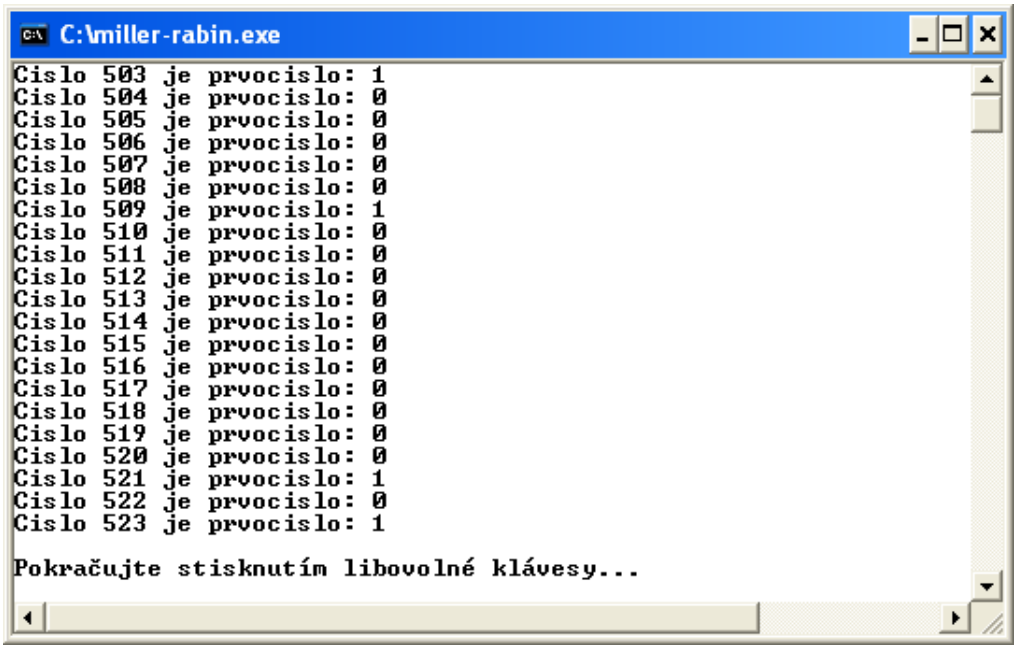
Predpokladajme, že máme číslo n, ktoré chceme otestovať Miller-Rabinovým testom.

1. Prvým krokom je rozklad čísla $n - 1 = 2^b \cdot m$. Musí platiť, že m je nepárne číslo, tým pádom b udáva najväčší násobok 2, ktorým je číslo $n - 1$ možné deliť.
2. Zvolí sa náhodné číslo a, ktoré je menšie ako n.
3. Vypočíta sa hodnota $z = a^{2^K m} \bmod n$, postupne pre $0 \leq K \leq b - 1$, pričom aby n bolo prvočíslo musia byť splnené tieto podmienky:
   - ak $K = 0$ a $z = 1$ alebo $z = n-1$, potom n môže byť prvočíslo a pokračuje sa bodom 2
   - ak $K \neq 0$ a $z = n-1$, potom n môže byť prvočíslo a pokračuje sa bodom 2
   - ak $K \neq 0$ a $z = 1$, potom n určite nie je prvočíslo a cyklus sa ukončí

Čím väčší bude počet generovaných náhodných čísel, tým bude pravdepodobnosť správneho záveru, že číslo n je prvočíslo, väčšia.

Priloha A.1 ukazuje funkciu implementujúcu Miller-Rabinov algoritmus. Sú využité funkcie knižnice NTL. Funkcia `NTL::IsOddm()` zisťuje či vstupné číslo je liché, funkcia `NTL::RandomBnd(n)` generuje náhodné číslo v intervale od 0 do $(n-1)$, funkcia `NTL::PowerMod(a, m, n)` počíta hodnotu $a^m \bmod n$. Premenné typu `NTL::ZZ` sú ľubovoľne dlhé čísla. Vstupnými parametrami funkcie `isPrime(NTL::ZZ n, int num)` je skúmané číslo n a parameter num udávajúci počet generovaných náhodných čísel a. Ak je výstup funkcie rovný 0, číslo n určite nie je prvočíslo. Ak je naopak výstup rovný 1, číslo n je pravdepodobne prvočíslo. Čím je väčší je parameter num tým je táto pravdepodobnosť vyššia.

Obrázok 4.1 zobrazuje výstup funkcie `isPrime` pre čísla v intervale 503 až 523. Je vidieť, že v tomto intervale sú 4 prvočísla: 503, 509, 521 a 523. Funkcia `isPrime` sa bude pri implementácií RSA používať na generovanie prvočísel q, p a generovanie verejného kľúča VK. Verejný kľúč však nemusí byť nutne prvočíslo, postačí ak sú



Obr. 4.1: Ukážka funkcie isPrime

s parametrom r nesúdeliteľné. Vygenerovanie prvočísla q je uskutočnené pomocou zdrojového kódu:

```
q = NTL::RandomLen_ZZ(keylength_bits/2);
while(!isPrime(q, 100)) q++;
```

4.3 Výpočet súkromného kľúča SK

Súkromný kľúč SK sa určí z podmienky

$$(VK \cdot SK) \bmod r \ = 1. \tag{4.3}$$

Na výpočet SK sa používa tzv. *Rozšírený Euklidov Algoritmus* (REA) [4]. Klasický Euklidov Algoritmus sa používa na nájdenie najväčšieho spoločného deliteľa dvoch čísel a, b, označuje sa ako $\gcd(a,\ b)$. REA okrem najväčšieho spoločného deliteľa zistí koeficienty x, y, tak, že platí

$$d = \gcd(a, b) = a \cdot x + b \cdot y\ . \tag{4.4}$$

Dôležité je, že ak je $\gcd(a, b) = 1$, tak pre parametre a, x, b, y platia vzťahy

$$(a \cdot x) \bmod b = 1 \tag{4.5}$$

$$(b \cdot y) \bmod a = 1 \tag{4.6}$$

Použitím indexov RSA môžeme vzťah 4.4 prepísať na

$$d = \gcd(r, VK) = r \cdot x + VK \cdot SK\ . \tag{4.7}$$

Ak platí $\gcd(r, VK) = 1$, tak získame potrebný SK, ktorý splňuje podmienku 4.3.

Iteratívny postup výpočtu REA je založený na tom, že v každom kroku i sa počíta hodnota výrazu

$$r_{\mathrm{i}} = a \cdot x_{\mathrm{i}} + b \cdot y_{\mathrm{i}}. \tag{4.8}$$

Hodnota r_{i} sa vypočíta ako

$$r_{\mathrm{i}} = r_{\mathrm{i-2}} - r_{\mathrm{i-1}} \left\lfloor \frac{r_{\mathrm{i-2}}}{r_{\mathrm{i-1}}} \right\rfloor\ . \tag{4.9}$$

Výraz 4.9 predstavuje vlastne vzťah pre výpočet zvyšku po delení

$$r_{\mathrm{i}} = r_{\mathrm{i-2}} \bmod r_{\mathrm{i-1}}\ . \tag{4.10}$$

Výraz 4.8 sa v i-tom kroku algoritmu vypočíta ako

$$r_{\mathrm{i}} = a \left(x_{\mathrm{i-2}} - \left\lfloor \frac{r_{\mathrm{i-2}}}{r_{\mathrm{i-1}}} \right\rfloor x_{\mathrm{i-1}} \right) + b \left(y_{\mathrm{i-2}} - \left\lfloor \frac{r_{\mathrm{i-2}}}{r_{\mathrm{i-1}}} \right\rfloor y_{\mathrm{i-1}} \right)\ . \tag{4.11}$$

Algoritmus končí v okamžiku keď r_{i} je rovno 0. Tabuľka 4.1 ukazuje postup výpočtu REA čísel 139 a 41.

krok i	$\left\lfloor \frac{r_{\mathrm{i-2}}}{r_{\mathrm{i-1}}} \right\rfloor$	$r_{\mathrm{i}} = r_{\mathrm{i-2}} - r_{\mathrm{i-1}} \left\lfloor \frac{r_{\mathrm{i-2}}}{r_{\mathrm{i-1}}} \right\rfloor$	$r_{\mathrm{i}} = a \cdot x_{\mathrm{i}} + b \cdot y_{\mathrm{i}}$
1	–	139	$139 = 139 \cdot 1 + 41 \cdot 0$
2	–	41	$41 = 139 \cdot 0 + 41 \cdot 1$
3	3	$16 = 139 - 41 \cdot 3$	$16 = 139 \cdot 1 + 41 \cdot (-3)$
4	2	$9 = 41 - 16 \cdot 2$	$9 = 139 \cdot (-2) + 41 \cdot 7$
5	1	$7 = 16 - 9 \cdot 1$	$7 = 139 \cdot 3 + 41 \cdot (-10)$
6	1	$2 = 9 - 7 \cdot 1$	$2 = 139 \cdot (-5) + 41 \cdot 17$
7	3	$1 = 7 - 2 \cdot 3$	$1 = 139 \cdot 18 + 41 \cdot (-61)$
8	2	$0 = 2 - 1 \cdot 2$	–

Tab. 4.1: Príklad výpočtu Rozšíreného Euklidovho Algoritmu

Pre potreby REA algoritmu bola vytvorená funkcia `REA()`. Jej prototyp má tvar:

```
NTL::ZZ REA(NTL::ZZ r, NTL::ZZ VK);
```

Funkcia vracia hodnotu súkromného kľúča SK.

4.4 Implementácia RSA

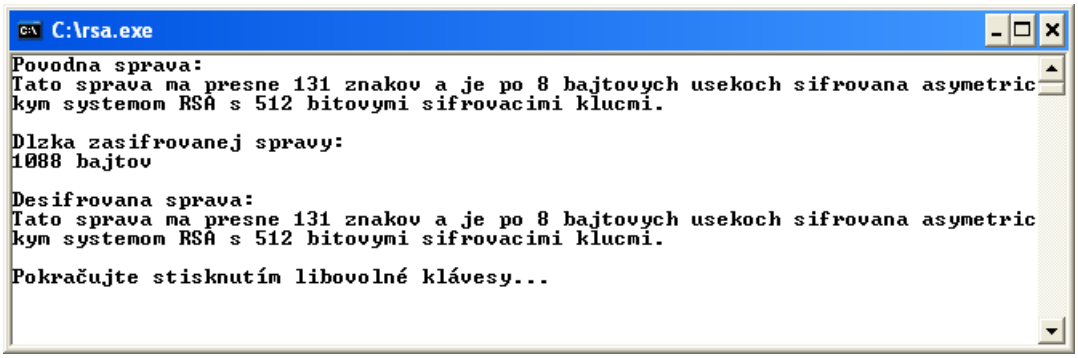
Pri implementácií kryptosystému RSA bola vytvorená trieda `Crsa`. V prílohe A.2 je vidieť jej deklaráciu.

Dôležitá je funkcia `init_crypto_keys()`, ktorá slúži na vygenerovanie kryptografických kľúčov dĺžky `keylength_bits`. Vstupná správa je rozdelená na bloky konštantnej veľkosti a parameter `length_decr_block` udáva práve veľkosť tohto bloku v bajtoch. Posledným parametrom funkcie je hodnota `seed`. Tá sa používa na nastavenie počiatočnej hodnoty pseudonáhodného generátora, ktorý je súčasťou knižnice NTL.

Funkcia `add_padding()` je určená na zarovnanie vstupnej správy na celočíselný násobok hodnoty `length_decr_block` pridaním nulových bajtov. Funkcia sa volá pred samotným šifrovaním. Obdobne `delete_padding()` odstraňuje z konca dešifrovanej správy nulové bajty. Funkcie `crypt()` a `decrypt()` už podľa názvu je zrejmé, že slúžia k šifrovaniu a dešifrovaniu správy.

Všetky štyri posledne spomenuté funkcie majú dva vstupné parametre. Prvým je samotná správa. Druhým parametrom je ukazovateľ na premennú typu int, v ktorej je udržovaná veľkosť spracovávanej správy. Je to z toho dôvodu, lebo funkcia `add_padding()` pridáva do správy nulové bajty. Nulové bajty sa však môžu vyskytnúť v správe aj vplyvom šifrovania. Tieto bajty neumožňujú získanie dĺžky správy pomocou klasickej funkcie `strlen`, preto je nutné aktuálnu dĺžku správy udržiavať v nejakej premennej, aby pri následnom posielaní pomocou soketov bolo možné definovať koľko bajtov je treba poslať. Všetky funkcie vracajú ukazovateľ na výstupnú správu.

Zdrojový kód v prílohe A.3 ukazuje vytvorenie objektu triedy Crsa a postupnosť jednotlivých funkcií pri šifrovaní a dešifrovaní správy.



Obr. 4.2: Ukážka implementácie kryptosystému RSA

Na obrázku 4.2 je ukážka implementácie RSA pri šifrovaní správy s dĺžkou 131

znakov. Jeden znak predstavuje jeden bajt. Správa bola šifrovaná po 8 bajtových úsekoch a boli použité kryptografické kľúče dĺžky 512 bitov. Dĺžka zašifrovanej správy je 1088 bajtov. Je to z toho dôvodu, že dĺžka 131 nie je celočíselným násobkom 8. Preto sa správa doplní 5 nulovými bajtmi na dĺžku 136. Tá predstavuje presne 17 blokov dĺžky 8 bajtov. Pretože sú použité 512 bitové kľúče, čo je 64 bajtov, každý zašifrovaný blok bude mať taktiež 64 bajtov. Výsledná správa je teda rovná $64 \cdot 17 = 1088$ bajtov.

5 SOCKETOVÉ PROGRAMOVANIE

Pri programovaní sieťových aplikácií je potrebné pracovať so socketami. Táto kapitola obsahuje úvod do sieťového programovania v programovacom jazyku C++. Mnohé z popísaných funkcií boli využité pri tvorbe výsledného programu.

5.1 Socket

Internetový socket môžeme podľa [10] považovať za definíciu dátového toku, ktorá nám popisuje typ siete a spôsob sieťovej komunikácie. Socket býva definovaný dvoma položkami:

- IP adresou
- cielovým portom.

Podľa typu a spôsobu práce sa dajú sockety rozdeliť do troch základných skupín:

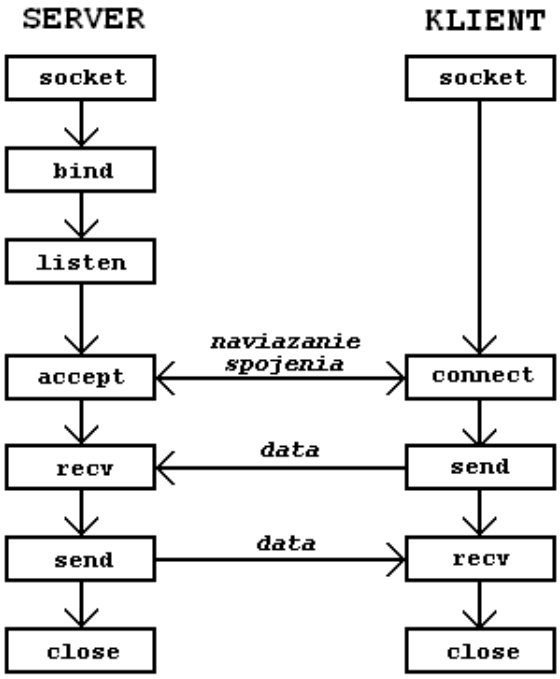
1. **Spojovo-orientované sockety (stream sockety).** Využívajú transportný protokol TCP, preto tento typ zabezpečuje spoľahlivú obojsmernú komunikáciu a dátové jednotky sú príjemcovi doručené v poradí, v akom boli odvysielané.
2. **Nespojovo-orientované sockety (datagram sockety).** Na rozdiel od predošlého typu využívajú služieb transportného protokolu UDP, tým pádom sa jedná o menej spolahlivý prenos bez predchádzajúceho naviazania spojenia.
3. **Surové sockety (raw sockety)**. Umožnujú úpravu informácií v hlavičkách protokolov IP, ICMP ale aj TCP resp. UDP.

5.2 Funkcie pre prácu so socketami

Funkcie pre prácu so socketmi sú súčasťou knihovny s názvom *Windows Sockets* (*WinSock*). V súčasnosti existuje vo verzii 2.2. Na začiatku je potreba knihovnu *WinSock* vždy inicializovať príkazom:

```
WSAData wsadata;
int state = WSAStartup(MAKEWORD(2,2),&wsadata);
if(state!=0) cout << "Nepodarilo sa inicializovat knihovnu WinSock.";
```

Pre prácu so socketmi existuje niekoľko funkcií. Postupnosť ich jednotlivých volaní závisí na type socketu a na tom, či sa jedná o klientskú alebo serverovú stranu komunikácie. V prípade, že sa jedná o stream sockety, je postupnosť jednotlivých funkcií videť na obrázku 5.1. U datagram socketov prebieha komunikácia spôsobom zobrazeným na obrázku 5.2.



Obr. 5.1: TCP socket

5.2.1 Vytvorenie socketu

Na vytvorenie socketu je určená funkcia `socket()`. Jej prototyp má tvar:

```
SOCKET socket(int network, int type, int protocol);
```

Vstupnými parametrami sú:

- `network` udáva typ používanej siete. Príklady tohoto parametru je vidieť v tabuľke 5.1.

network	význam
AF_INET	IPv4 protokol
AF_INET6	IPv6 protokol
AF_UNSPEC	nešpecifikóvaná sieť

Tab. 5.1: Parameter network

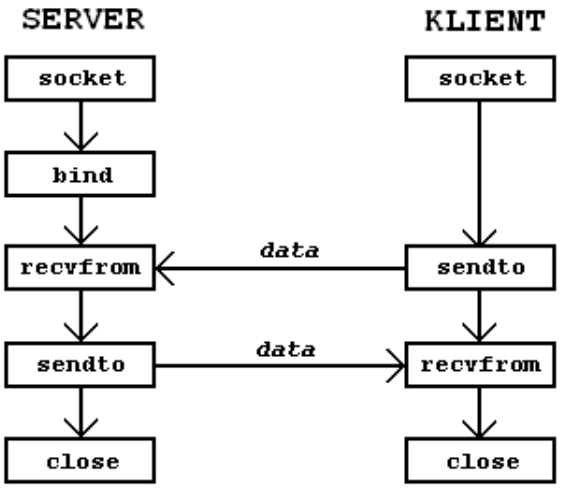
- `type` udáva typ použitého socketu a tým aj typ spojenia. Možnosti sú v tabuľke 5.2.

type	význam
SOCK_DGRAM	nespojovo-orientované sockety
SOCK_STREAM	spojovo-orientované sockety
SOCK_RAW	surové sockety

Tab. 5.2: Parameter type

- `protocol` udáva používaný protokol, na ktorom sa socket vytvára. Implicitná hodnota je 0, v tom prípade bude typ protokolu nastavený automaticky.

Funkcia vytvorí socket a vráti jeho deskriptor typu `SOCKET`. Pri ukončení práce by mal byť socket zrušený pomocou funkcie `closesocket()`.



Obr. 5.2: UDP socket

5.2.2 Spojenie socketu

Po vytvorení socketu je potrebné jeho spojenie s portom na lokálnej stanici, na ktorom bude server naslouchat. To sa uskutočňuje funkciou `bind()`:

```
int bind(SOCKET socket, sockaddr *addr, int addrlen);
```

Zároveň je možné prostredníctvom štruktúry `sockaddr` špecifikovať z akých adries má server naslouchat. Buď sa jedná o konkrétnu IP adresu alebo o všetky IP adresy, v prípade použitia parametru `INADDR_ANY`. Funkcia je používaná výhradne na strane

serveru.
Vstupnými parametrami sú:

- `socket` predstavuje deskriptor nespojeného socketu
- `addr` je ukazovateľ na štruktúru typu `sockaddr`
- `addrlen` veľkosť štruktúry `addr` v bajtoch.

5.2.3 Asynchrónne sockety

Po vytvorení socketu sa jedná o blokujúci socket, to znamená, že pri čakaní na výskyt udalostí zablokuje beh programu a nedovoľuje uživateľovi vykonať žiadnu akciu. Pre operačný systém Windows boli zavedené tzv. asynchrónne sockety. Windows zároveň umožňuje udalostmi riadené programovanie. Ide o to, že pri akejkoľvek udalosti v systéme sú jednotlivým oknám poslané správy, ktoré tieto udalosti charakterizujú. Príkladom takejto udalosti môže byť napríklad stlačenie klávesy, kliknutie myšou, ale napríklad aj nejaká udalosť spôsobujúca zmenu stavu socketu. Pomocou funkcie `WSAAsyncSelect()` je možné pre okno `hwnd` zaregistrovať správu `wMsg`, ktorá bude vyvolaná po určitom type udalosti, ktorý prebehne nad daným socketom. Potom stačí obslúžiť zachytenie predanej zprávy. Podľa [10] sa dá výsledok tejto funkcie reprezentovať ako vlákno, ktoré má na starosti sledovanie špecifických udalostí, ktoré nastanú. V prípade, že udalosť nastane a je zachytená, oknu je poslaná špecifická správa, na základe ktorej je možné na udalosť vhodným spôsobom zareagovať. Funkcia `WSAAsyncSelect()` má tvar:

```
int WSAAsyncSelect(SOCKET socket, HWND hwnd, int wMsg, long event);
```

Vstupnými parametrami sú:

- `socket` predstavuje deskriptor socketu
- `hwnd` je identifikátor okna
- `wMsg` identifikuje špecifikovanú zprávu, ktorá bude po zachytení udalosti poslaná oknu `hwnd`, napríklad `WM_USER+1`[1]
- `event` je identifikátor udalosti, na ktorú chceme reagovať. Prehľad možných hodnôt je v tabuľke 5.3.

S funkciou `WSAAsyncSelect()` úzko súvisí použitie funkcie `accept()`:

```
SOCKET accept(SOCKET socket, struct sockaddr *addr, int addrlen);
```

Je používaná pri naviazaní spojenia metódou asynchronného naslouchání, ktoré bolo vytvorené práve funkciou `WSAAsyncSelect()`, a volá sa po zistení požiadavku na

[1]Konštanta `WM_USER` predstavuje hranicu medzi správami definovanými v systéme (rozsah 0 až WM_USER–1) a správami, ktoré môžu byť použité aplikáciami k definícií vlastných správ (rozsah WM_USER až 32767).

event	význam
FD_READ	oznámenie prichádzajúcich dát
FD_WRITE	oznámenie pripravenosti odoslať data
FD_ACCEPT	oznámenie prichádzajúcej žiadosti o naviazenie spojenia (nastáva po spojení `connect()` z klientskej strany, viac 5.2.4)
FD_CONNECT	oznámenie o úspešnosti naviazenia spojenia
FD_CLOSE	oznámenie o zrušení socketu

Tab. 5.3: Možné hodnoty parametru event

spojenie pre socket, ktorý naslouchá, to znamená podľa tabuľky 5.3 sa jedná o zprávu s príznakom `FD_ACCEPT`. Vstupný parameter `addr` je ukazovateľ na štruktúru `sockaddr`, ktorá popisuje stanicu snažiacu sa naviazať spojenie s lokálnou stanicou. Funkcia vracia identifikátor na novo-vytvorený socket, identifikujúci nové spojenie.

5.2.4 Naviazanie spojenia

Naviazanie spojenia je inicializované na klientskej strane, a to iba v prípade stream socketov `SOCK_STREAM`. Slúži k tomu funkcia `connect()`:

```
int connect(SOCKET socket, sockaddr *addr, int addrlen);
```

Vstupnými parametrami sú:

- `socket` predstavuje deskriptor nenaviazeného socketu
- `addr` identifikuje stanicu, s ktorou chceme spojenie naviazat
- `addrlen` veľkosť štruktúry `addr` v bajtoch.

5.2.5 Prenos dát

K účelu prenosu dát existuje dvojica funkcií, `recv()` pre príjem a `send()` pre vysielanie. Funkcia `send()` má tvar:

```
int send(SOCKET socket, char * msg, int len, int flags);
```

Vstupnými parametrami sú:

- `socket` predstavuje deskriptor socketu, prostredníctvom ktorého sa dáta odošlú.
- `msg` je ukazovateľ na odosielané dáta

- `len` dĺžka odosielaných dát
- `flags` doplnkové príznaky.

Pokiaľ sa nevyskytne chyba, tak funkcia vráti počet odoslaných bajtov. Opakom je funkcia `recv()`:

```
int recv(SOCKET socket, char *msg, int len, int flags);
```

Význam paramterov je podobný ako u predchádzajúcej funkcie. Príma dáta zo socketu `socket` do pola znakov `msg`, parameter `len` udáva dĺžku tohoto pola a `flags` sú ďalšie nastavenia. V prípade, že nenastane chyba, funkcia vráti počet prijatých bajtov.

Predchádzajúce dve funkcie sa obecne používajú v spojení s TCP stream socketmi. Pre UDP datagram sockety sa používajú funkcie `recvfrom()` a `sendto()`. Vstupné parametre majú rovnaké ako funkcie `recv()` a `send()`. Vzhľadom ale na to, že datagram sockety nie sú na rozdiel od stream socketov pripojené k vzdialnej stanici, je potrebné špecifikovať u funkcie `sendto()` cieľovú adresu pomocou štruktúry `sockaddr`. U funkcie `recvfrom()` je taktiež vstupným parametrom štruktúra `sockaddr`, ktorá je po prijatí dát naplnená IP adresou a portom odosielacej stanice.

5.2.6 Úprava vlastnosti socketov

Vlastnosti socketov sa dajú upraviť pomocou funkcie `setsockopt()`:

```
int setsockopt(SOCKET socket, int level, int optname, char * optval,
               int optlen);
```

Pred samotnou úpravou je vhodné najprv zistiť aktuálne vlastnosti socketu prostredníctvom funkcie `getsockopt()`:

```
int getsockopt(SOCKET socket, int level, int optname, char * optval,
               int *optlen);
```

Obidve funkcie majú rovnaké vstupné parametre:

- `socket` predstavuje deskriptor socketu, ktorého vlastnosť chceme zmeniť
- `level` určuje úroveň pre vlastnosť `optname`, kedy pre každú úroveň sa dajú meniť iné vlastnosti
- `optname` identifikátor požadovanej vlastnosti socketu
- `optval` je ukazovateľ na buffer pre výstupné resp. vstupné informácie
- `optlen` je ukazovateľ na dĺžku položky `optval`.

Príloha A.4 ukazuje príklad použitia týchto dvoch funkcií. Konkrétne sa jedná o zistenie možnosti prenosu broadcast zpráv využitím funkcie `getsockopt()`. Pokiaľ je táto možnosť vypnutá, tak sa pomocou funkcie `setsockopt()` zapne.

6 IMPLEMENTÁCIA SYSTÉMU

Cieľom tejto kapitoly je popísať výslednú implementáciu anonymného systému. Ako programovací jazyk bol zvolený objektovo orientovaný jazyk C++ a vývojové prostredie wxDev-C++. Pri implementácií bolo vytvorených niekoľko tried, ktorých stručný popis bude obsahom následujúcich strán. Pre veľký rozsah zdrojových kódov, budú uvádzané iba najdôležitejšie časti, deklarácie tried a funkcií.

6.1 Triedy Cuser, Clist a štruktúra Skeys

Za účelom komunikácie s ostatnými užívatelmi v systéme, je potrebné poznať verejné kľúče pripojených staníc. Verejný kľúč sa skladá z dvoch RSA parametrov, a to z parametru n a VK, viac v kapitole 4. Pre prácu s dvojicou týchto parametrov bola vytvorená štruktúra `Skeys`, ktorej deklarácia vyzerá následovne:

```
struct Skeys
{
    NTL::ZZ VK;
    NTL::ZZ n;
};
```

Premenné typu `NTL` sú ľubovolné veľké čísla a boli spomínané v kapitole 4. Pre úplnosť treba poznamenať, že pre prácu s knihovnou NTL je potrebné do projektu pripojiť knihovnu `NTL.a`. Podobne pre prácu so socketmi knihovnu `libws2_32.a`[1].

Každá stanica musí udržiavať informácie o ostatných pripojených staniciach. Informácie o jednej stanice sú združené v objekte triedy `Cuser`:

```
class Cuser
{
    private:
        unsigned long ul_ip;
        Skeys keys;
    public:
        Cuser(unsigned long ul_ip, Skeys keys);
        ~Cuser(void){}
        unsigned long get_ip(void);
        Skeys get_keys(void);
};
```

[1]Knihovna `libws2_32.a` je súčasťou vývojového prostredia wxDev-C++, knihovnu `NTL.a` je možné získať z [9].

Trieda obsahuje informáciu o IP adrese danej stanice vo formáte `unsigned long` a verejné kľúče danej stanice v štruktúre `Skeys`. Tieto hodnoty sa nastavujú pri inicializácií objektu využitím konštruktoru a získavajú sa z objektu pomocou funkcií `get_ip()`, respektíve `get_keys()`.

Pre samotnú prácu s objektmi `Cuser` slúži trieda `Clist`:

```
class Clist
{
    private:
        list <Cuser*> list_of_users;
    public:
        bool add(char * ip, Skeys keys);
        bool erase(char * ip);
        void erase_all(void);
        Skeys get_keys(char * ip);
        unsigned long get_ip(int index);
        int get_size(void);
};
```

Objekty triedy `Cuser` sú ukladané do STL[2] kontajneru typu `list`, ktorý je implementovaný ako obojsmerný zoznam. Dôležitá je funkcia `add()`, ktorá do zoznamu pridá informácie o stanici s IP adresou `ip` a s verejnými kľúčmi `keys`. Funkcia vráti hodnotu true ak uloženie informácií prebehlo v poriadku, respektíve zoznam neobsahuje ešte záznam s IP adresou `ip`, v opačnom prípade vráti false. K vymazaniu záznamu o konkrétnej stanici slúži funkcia `erase()`. Návratová hodnota true signalizuje, že záznam bol úspešne vymazaný. Na druhej strane hodnota false znamená, že záznam neexistoval, preto nebol ani vymazaný. Následujúci zdrojový kód je ukážkou implementácie funkcie `get_keys()`, ktorá vráti verejné parametre na základe predanej IP adresy:

```
Skeys Clist::get_keys(char * ip)
{
    list<Cuser*>::iterator it;
    unsigned long ul_ip = inet_addr(ip);
    for (it=list_of_users.begin(); it!=list_of_users.end(); it++)
    {
        if ((*it)->get_ip() == ul_ip) return (*it)->get_keys();
    }
    Skeys keys;
```

[2]Štandardná knihovna šablón STL (Standard Template Library).

```
    keys.VK = 0;
    keys.n = 0;
    return keys;
}
```

S použitím iterátora sa prechádza postupne celý zoznam a hladá sa objekt reprezentujúci objekt s požadovanou IP adresou. V prípade, že zoznam neobsahuje záznam o danej IP adrese, hodnoty verejných parametrov v návratovej štruktúre `Skeys` sú nulové.

6.2 Triedy Constants a Clayer

Trieda `Constants` je trieda združujúca konštanty použité pri implementácií.

```
#define KEY_LENGTH_BITS 512
#define BLOCK_LENGTH 63
#define IP_LAYER_PREFIX "IP:"
#define MESS_LAYER_PREFIX "MESS:"
```

Významy jednotlivých konštánt sú zjavné z ich názvu. Hodnoty `BLOCK_LENGTH` a `KEY_LENGTH_BITS` súvisia s algoritmom RSA, konkrétne triedou `Crsa`, ktorá bola popísaná v kapitole 4.

- `KEY_LENGTH_BITS` definuje dĺžku šifrovacích kľúčov systému RSA
- `BLOCK_LENGTH` určuje veľkosť bloku, po ktorom bude zpráva šifrovaná
- `MESS_LAYER_PREFIX` a `IP_LAYER_PREFIX` sú konštanty použité na rozlíšenie vrstiev onionu, tak ako je to vidieť na obrázku 2.4.

K tvorbe a zpracovaniu vrstiev onionu je určená trieda `Clayer`. Obsahuje tri statické metódy, to znamená, že metódy sú dostupné aj bez vytvorenia instancie triedy:

```
class Clayer
{
    public:
        static char * add_ip_prefix(char * message, int * length,
                                    unsigned long ul_ip);
        static char * add_mess_prefix(char * message, int * length,
                                      unsigned long ul_ip);
        static char * delete_prefix(char * message, int * length,
                                    unsigned long * ul_ip,
                                    bool * end_of_transmission);
};
```

Funkcia `add_ip_prefix()` pridá k zpráve `message` vrstvu identifikujúcu následujúcu stanicu, s IP adresou `ul_ip`, na trase. Podobnú funkciu má `add_mess_prefix()`, ktorá k zpráve pridáva vrstvu určenú konečnému príjemcovi. Narozdiel od predchádzajúcej funkcie v tomto prípade IP adresa `ul_ip` nereprezentuje IP adresu následujúcej stanice na trase ale pôvodného odosielateľa, z dôvodu aby príjemca po prijatí vedel, kto správu odoslal a teda aj pre potreby obojsmernej komunikácie.

Poslednou funkciou je `delete_prefix()`, ktorá je použitá v momente, keď stanica príjme správu, dešifruje ju a chce rozhodnúť na základe aktuálnej vrstvy o tom, či je ona sama konečným príjemcom alebo je nutné správu ďalej poslať. Vstupné parametre, dešifrovaná správa `message` a jej dĺžka `length`, sú doplnené o ďalšie dva parametre, ktoré spolu súvisia a po ukončení funkcie môžu nastať dva prípady:

1. ak platí podmienka `(end_of_transmission==true)`, tak daná stanica je konečným príjemcom a ukazovateľ `ul_ip` ukazuje na IP adresu pôvodného odosielateľa. Výstupom je ukazovateľ na správu, ktorá je následne predaná užívateľovi.
2. ak platí podmienka `(end_of_transmission==false)`, tak daná stanica nie je konečným príjemcom a správa má byť preposlaná ďalšej stanici na prenosovej trase, ktorej IP adresa je získana pomocou ukazovateľa `ul_ip`. Výstupom je ukazovateľ na správu, ktorá je následne poslaná ďalej.

6.3 Trieda Cmessages

Podobne ako trieda `Clayer` popísaná v predchádzajúcej kapitole, obsahuje trieda `Cmessages` iba statické metódy, a preto nie je nutné vytvárať instanciu tejto triedy. Funkcie tejto metódy súvisia s vytváraním a spracovaním signalizačných správ používaných pri pripojovaní do systému, viac v kapitole 2.1. Prototypy funkcií majú následujúci tvar:

```
class Cmessages
{
    public:
        static char * create_hello_message(Skeys keys, int * size);
        static char * create_welcome_message(Skeys keys, int * size);
        static Skeys decode_hello_message(char * hello_message);
        static Skeys decode_welcome_message(char * welcome_message);
};
```

Funkcia `create_hello_message()` vytvorí správu HELLO, ktorú stanica pri prihlásení do systému odvysiela na broadcastovú adresu. Vstupnými parametrami je

štruktúra `Skeys` obsahujúca verejné kľúče danej stanice, spolu s ukazovateľom na veľkosť výslednej správy. Výstupným parametrom je ukazovateľ na vytvorenú správu HELLO. Všetky stanice, ktoré príjmu túto broadcastovú správu HELLO, použijú na jej spracovanie funkciu `decode_hello_message()`. Vstupným parametrom je samotná správa. Výstupom je štruktúra `Skeys`, ktorá obsahuje verejné kľúče prihlasovanej stanice, teda stanice, ktorá správu HELLO odoslala. Zvyšné dve funkcie triedy `Cmessages` sú podobné, s tým rozdielom, že slúžia na vytváranie a spracovanie správ WELCOME. Vytvorená správa WELCOME je poslaná unicastovo na IP adresu prihlasovej stanice, jedná sa teda o dozvu na správu HELLO.

6.4 Trieda Cnet

Najdôležitejšou triedou v rámci projektu je trieda `Cnet`. Obsahuje niekoľko funkcií, ktoré zaisťujú veškerú funkčnosť systému. Využívajú funkcie tried popísaných v predchádzajúcich kapitolách.

Prvou významnou funkciou je funkcia `prepare_listen()`:

```
void prepare_listen(void);
```

Táto funkcia je volaná okamžite po zapnutí programu, ešte skôr ako dojde k prihláseniu do systému prostredníctvom správ HELLO a WELCOME. Funkcia pripravuje globálne deklarované sockety `lisSocket_uni` pre unicast a socket `lisSocket_broad` pre broadcast. Nad týmito socketmi umiestni asynchrónny model a pre každý z nich zaregistruje správu, ktorá bude volaná pri zmene stavu socketu. U socketu `lisSocket_uni` ide o správu `WM_USER+1`, u broadcast socketu `lisSocket_broad` sa jedná o `WM_USER+2`.

Pre ďalší popis je nutné zadefinovať funkciu, ktorá je zodpovedná za reakcie na zachytávané správy generované operačným systémom. Ide o funkciu `WindowProcedure()`. Časť, ktorá je dôležitá pre tento konkrétny prípad:

```
case WM_USER + 1:
  net.listen_proc_unicast(wParam, lParam);
  break;
case WM_USER + 2:
  net.listen_proc_broadcast(wParam, lParam);
  break;
```

Je vidieť, že v momente ako je zachytená správa `WM_USER+1`, to znamená, že došlo k nejakej zmene u unicast socketu `lisSocket_uni`, je zavolaná funkcia, ktorá má za úlohu na tento stav vhodným spôsobom zareagovať. Konkrétne sa jedná o funkciu

`listen_proc_unicast()`. Môžu nastať dva prípady, pričom rozhodnutie závisí od príznaku predávaného tejto správe:

1. Ak je príznak rovný FD_ACCEPT, znamená to, že prišla žiadosť o spojenie, a dôjde k naviazaniu spojenia.
2. Ak je príznak rovný FD_READ, signalizuje to, že prichádzajú dáta. Po prijatí sa najprv otestuje či sa jedná o WELCOME správu. Ak áno, tak je správa predaná funkcií `decode_hello_message()` z predchádzajúcej kapitoly. V opačnom prípade, je správa pomocou súkromného kľúča dešifrovaná a následne predaná funkcií `delete_prefix()` 6.2. Na základe výsledku tejto funkcie, konkrétne hodnoty premennej `end_of_transmission`, sa rozhodne či je daná stanica koncovým príjemcom alebo je potrebné dátovú jednotku odoslať následujúcej stanice na trase.

Ďalšou funkciou je `listen_proc_broadcast()`, obsluhujúca zmeny, ktoré sa vyskytnú nad broadcast socketom `lisSocket_broad`. Po prijatí správy sa najprv zistí druh prijatej správy. Broadcast správy existujú v systéme iba dve. Ide o správy HELLO a BYE.

1. Ak sa jedná o správu HELLO, predá sa funkcií `decode_hello_message()`, popísanej v kapitole 6.3. Následne sa pomocou funkcie `create_welcome_message()` vytvorí správa WELCOME a odošle sa na IP adresu, z ktorej bola správa HELLO prijatá.
2. Ak sa jedná o správu BYE, dôjde k odstráneniu záznamu o stanici, ktorá danú správu odoslala.

K samotnému odosielaniu správ slúžia funkcie:

```
void send_broadcast(CHAR *message, int length);
void send_unicast(CHAR *message, CHAR *IP, int length);
```

Rozdiel medzi nimi je zrejmý z ich názvu. Obidve majú vstupný parameter správu, ktorá sa má odoslať, a jej dĺžku. U funkcie `send_unicast()` je potrebné ešte definovať IP adresu kam má byť správa poslaná. Cieľovým portom u oboch druhov prenosu je 10012.

K vygenerovaniu prenosovej trasy, ktorou bude správa od odosielateľa k príjemcovi smerovaná, slúži funkcia `generate_path()`:

```
void generate_path(stack <unsigned long> *listIP, int n)
```

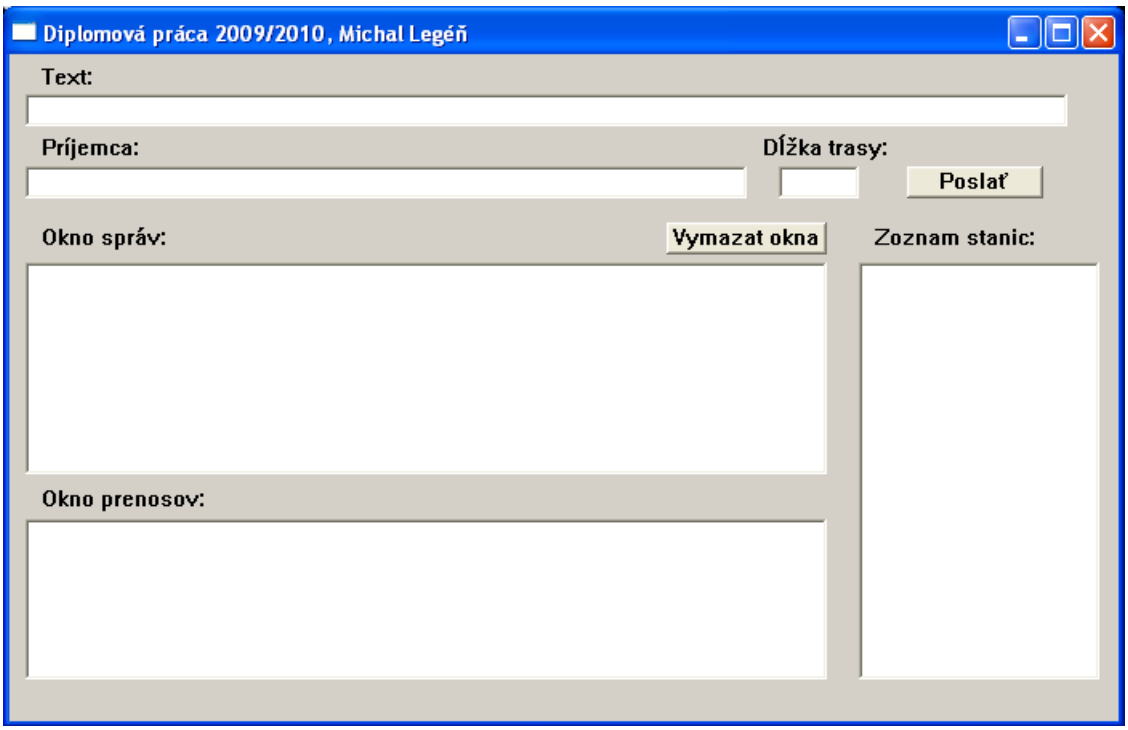
Funkcia má dva vstupné parametre. Prvým je odkaz na zásobník, do ktorého sa budú ukladať IP adresy vo formáte `unsigned long`, druhým parametrom je počet staníc na trase. Tento parameter si môže užívateľ zvoliť v rámci GUI[3].

[3]Grafické užívatelské rozhranie (Graphical User Interface)

Po vygenerovaní trasy je potrebné vytvoriť dátovú jednotku onion, ktorá bude touto trasou smerovaná. Tým pádom poslednou významnou funkciou triedy `Cnet` je funkcia:

```
char * create_onion(stack <unsigned long> listIP, char * text,
                    int * length, unsigned long * receiver_ip,
                    unsigned long * first_hop_ip)
```

Úlohou tejto funkcie je vytvoriť vrstvovo šifrovanú dátovú jednotku onion spôsobom zobrazeným na obrázku 2.5. Prvým vstupným parametrom je zásobník `listIP`, obsahujúci IP adresy staníc na trase, tak ako boli vygenerované funkciou `generate_path()`. Ďalšími parametrami sú samotná odosielaná správa a ukazovateľ na jej dĺžku pred šifrovaním. Po skončení funkcie bude ukazovateľ ukazovať na dĺžku vytvoreného onionu. Výstupom sú taktiež dva ukazovateľe na adresu konečného príjemcu `receiver_ip` a na adresu prvej stanice na trase `first_hop_ip`. V rámci funkcie neplnia významnú funkciu, ich zmysel je v tom aby informácia o IP adresách týchto dvoch staníc bola dostupná ja mimo funkcie. IP adresa `first_hop_ip` bude použitá pri odosielaní, pretože je to prvá stanica na trase. IP adresa `receiver_ip` je použitá iba ako informácia v užívateľovom GUI o konečnom príjemcovi.



Obr. 6.1: Grafické užívatelské prostredie

6.5 Grafické užívatelské prostredie

Grafické užívatelské prostredie naprogramovaného anonymného systému je vidieť na obrázku 6.1. Skladá sa z niekoľkých častí. Vstup **Text** umožnuje zadať správu, ktorá sa odošle príjemcovi s IP adresou, ktorú je možno zadať v časti **Príjemca**. Pomocou textboxu **Počet staníc** sa dá definovať počet staníc, cez ktoré bude dátová jednotka prechádzať. Ak obsahuje hodnotu 0, tak správa bude odoslaná priamo príjemcovi. **Okno správ** zobrazuje prijaté a odoslané správy. **Okno prenosov** zobrazuje prenosy, pre ktoré je daná stanica len jednou zo staníc zúčastnujúcich sa prenosu. Tvar týchto výpisov je následovný:

```
predchádzajúca stanica -> JA -> následujúca stanica
```

Tlačítko **Vymazať okna** vymaže okná **Okno správ** a **Okno prenosov**. Listbox **Zoznam staníc** obsahuje automaticky aktualizovaný zoznam IP adries pripojených staníc v systéme.

7 ANALÝZA SYSTÉMU

Každý navrhnutý systém je potrebné objektívne zhodnotiť a analyzovať z rôznych hľadísk. Výsledom by mala byť správa o tom, na koľko sa stanovené podmienky podarilo splniť. V rámci tejto záverečnej kapitoly bude uskutočnená analýza implementovaného anonymného systému, čo sa týka časových parametrov a bezpečnosti, respektíve možnosti narušenia.

7.1 Pravdepodobnosť prerušenia prenosu

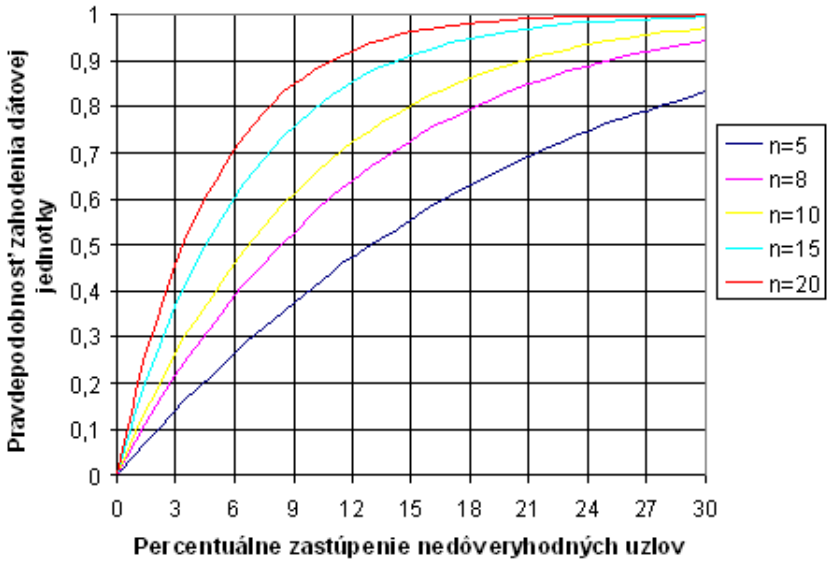
Nie každá stanica zapojená v systéme môže byť dôveryhodného charakteru. Nedôveryhodnosťou stanice je myslená stanica snažiaca sa nejakým spôsobom fungovanie systému prerušiť alebo aspoň narušiť. Jednou z takýchto činností môže byť úmyselné zahadzovanie dátových jednotiek. Dôsledkom tohoto kroku je prerušenie aktuálne prebiehajúceho prenosu. Zavedieme premennú p reprezentujúcu pravdepodobnosť výberu nedôveryhodných staníc v systéme. Vzhľadom na to, že výber staníc je náhodný, a stanice sa môžu na trase opakovať je pravdepodobnosť výberu aspoň jednej nedôveryhodnej stanice P je rovný:

$$P = 1 - (1 - p)^n. \tag{7.1}$$

Parameter n predstavuje dĺžku trasy. Bude sa predpokladať, že nedôveryhodná stanica po prijatí dátovej jednotky ju zahodí, a preto čo i len jedna stanica takáto stanica na trase znamená prerušenie dátového prenosu. Tým pádom premenná P je zároveň pravdepodobnosť zahodenia dátovej jednotky pri prenose. Obrázok 7.1 zobrazuje závislosť pravdepodobnosti zahodenia dátovej jednotky na percentuálnom zastúpení nedôveryhodných staníc v systéme, pri trasách dlhých $n = 5, 8, 10, 15, 20$ staníc. Z grafu je vidieť, že čím je dĺžka prenosovej trasy väčšia, tým je väčšia aj pravdepodobnosť, že jej súčasťou bude nedôveryhodna stanica, ktorá prenos ukončí.

7.2 Bezpečnosť a časová zložitosť systému na bázi RSA

Ďalšou skupinou staníc snažiacich sa o narušenie funkčnosti systému môžu byť stanice usilujúce sa o nalomenie kryptosystému RSA. Tento kryptosystém je podľa [1] považovaný vo verzii s 1024 bitovými kľúčmi do konca roku 2010 za bezpečný a zároveň riziko prelomenia je malé najmenej do konca roku 2014.



Obr. 7.1: Graf pravdepodobnosti zahodenia dátových jednotiek

Bezpečnosť celého kryptosystému RSA je založená na probléme rozkladu veľkého čísla n na dve prvočísla. Zatiaľ najrýchlejším spôsobom je faktorizácia. Podľa [6] najjednoduchším spôsobom faktorizácie je postupné delenie čísla n číslami 2, 3, 4 až $\sqrt{n}$. Zložitosť tohoto spôsobu je $O(\sqrt{n})$. Ďalšou možnosťou je podľa [6] použitie tzv. Pollard ρ metóda, ktorej binárna zložitosť je $O(\sqrt[4]{n}\log_2^3 n)$. Existujú aj snahy o paralelizáciu faktorizácie procesu. Napríklad v práci [12] je prezentovaná vylepšená metóda MPQS[1]. Autorom sa podarilo pomocou tejto metódy faktorizovať číslo dlhé 100 číslic použitím 32 počítačov za 6 dní a 14 hodín.

Zložitosť samotného šifrovacieho procesu kryposystému RSA závisí na operácií modulo, tak ako je to vidieť zo vzorcov pre šifrovanie 4.1 a dešifrovanie 4.2. Ak budeme predpokladať k-bitové čísla a, b a c, potom zložitosť vzťahu $(a^b \bmod c)$ je rovná $O(k^3)$. Jedná sa teda o polynomickú zložitosť.

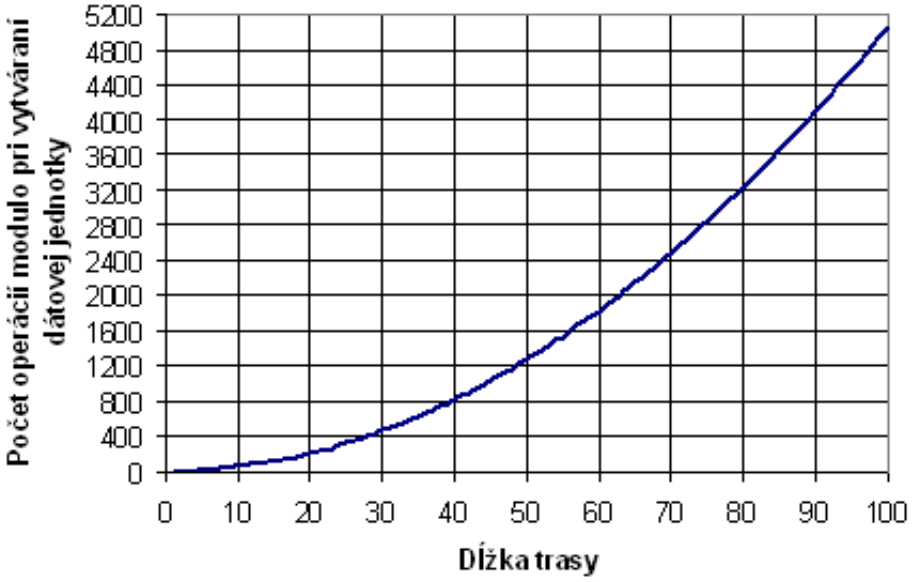
Zamerajme sa na proces vytvárania dátovej jednotky u odosielateľa. Cieľom je zistiť počet operácií modulo nutných k vytvoreniu dátovej jednotky. Predpokladajme kryptosystém RSA s k-bitovými kľúčmi a správu dlhú l bitov, tak že platí $l \leq k$. To znamená, že celá správa bude zašifrovaná v jednom kroku algoritmu RSA pomocou verejného kľúča príjemcu. Vznikne tak po prvom kole zašifrovaná správa dlhá k bitov. Odosielateľ ale následne pridá, k takto zašifrovanej správe, krátku vrstvu dlhú j bitov určenú stanici pred samotným príjemcom. Takto vytvorenú správu zašifruje pomocou jej verejných kľúčov. Kedže ale táto správa bude o j bitov dlhšia ako blok

[1]Multiple Polynomial Quadratic Sieve

po ktorom sa uskutočnuje šifrovanie, bude správa zašifrovaná pomocou dvoch operácií modulo. Takto sa proces n-krát opakuje, kde n je počet staníc na trase. Samotný proces prebieha podľa obrázku 2.5. Počet operácií modulo nutných k zašifrovaniu dátovej jednotky pre trasu dlhú n staníc je rovný:

$$x_n = \sum_{i=1}^{n} x_i \,. \tag{7.2}$$

Obrázok 7.2 ukazuje závislosť počtu operácií modulo pro šifrovaní na dĺžke trasy od 1 po 100 užívateľov. Je vidieť, že počet operácií modulo, a tým aj zložitosť šifrovacieho procesu sa pri zvyšovaní počtu staníc na trase zväčšuje rýchlejšie ako lineárne.

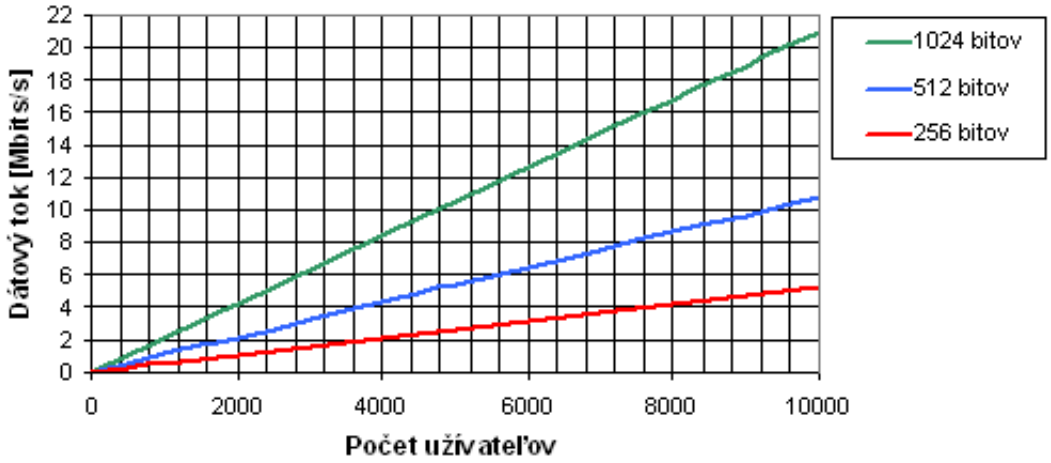


Obr. 7.2: Závislosť počtu operácií modulo pro šifrovaní na počte užívateľov na trase

7.3 Záplavový útok

Zložitosť prihlasovacieho procesu do systému je závislý na počte užívateľov, viac v kapitole 2.1. Každý účastník v systéme odpovedá na broadcast správu HELLO, unicastovou správou WELCOME, preto zložitosť prihlasovacieho procesu je lineárna $O(n)$, kde n je počet staníc v systéme. Vzhľadom na malé zataženie systému pri broadcastovej správe HELLO, nebude sa jej existencia vo výpočtoch zohladovať.

Veľkosti správ HELLO a WELCOME sú závislé na veľkosti kryptografických kľúčov, ktoré prenášajú. Obrázok 7.3 zobrazuje závislosť veľkosti dátového toku na



Obr. 7.3: Dátový tok pri prihlasovacom procese

počte užívateľov pri rozličných veľkostiach použitých kryptografických kľúčov. Je vidieť, že dátový tok pri prihlásení stanice do systému môže nadobúdať pomerne veľkých hodnôt. Tohoto faktu sa dá využiť v sieťach s veľkým počtom užívateľov alebo v sieťach s malou šírkou pásma. Útočník bude prihlasovací umelo vyvolávať v krátkych intervaloch, a tým pádom môže dôjsť k zahlteniu siete.

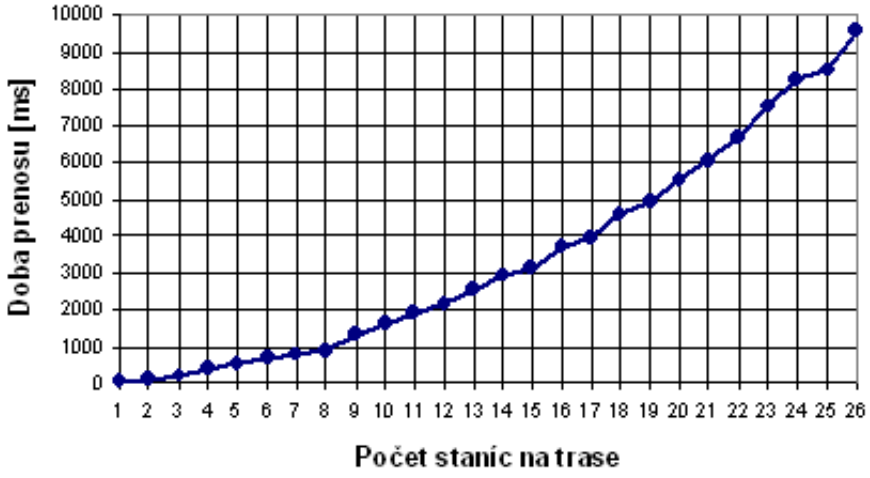
Ochranou voči tomuto javu by mohlo byť použitie vyrovnávajúcej pamäte na strane príjemcu. Do tejto pamäte by sa po určitú dobu, napríklad 500ms, ukladali prijaté správy HELLO. Po uplynutí tejto doby, by stanica na jednotlivé správy odpovedala správami WELCOME. Pokiaľ by sa však v pamäti nachádzali dve alebo viac správ HELLO odoslané jednou stanicou, správu WELCOME by odoslala iba jeden krát. Následné by sa pamäť vypráznila a postup by sa opakoval.

7.4 Praktické meranie doby prenosu

Do systému bola implementovaná funkcia umožnujúca získanie doby prenosu medzi odosielateľom a príjemcom správy. Snahou bolo získať závislosť doby prenosu na dĺžke trasy. Tento doplnok do systému funguje tak, že odosielateľ po odoslaní správy do siete spustí časovač. Ten sa zastaví v momente priatia potvrenia ACK od konečného príjemcu. Potvrdenie ACK je posielané unicastovo od príjemcu pôvodnej správy k jej odosielateľovi. V klasickom anonymnom systéme by implementované nemala byť.

Program bol následne upravený tak, aby do textového súboru sa vypisovali informácie o jednotlivých prenosoch a ich dobe prenosu, pre potreby analýzy. Dĺžku

trasy sa postupne zvyšovala od 1 po 26 staníc. Výsledný graf závislosti doby prenosu na dĺžke trasy zobrazuje obrázok 7.4. Je vidieť, že v prípade vyššieho počtu staníc na trase sa táto doba dostáva k relatívne vysokým hodnotám.



Obr. 7.4: Závislosť doby prenosu na dĺžke trasy

7.5 Splňenie vstupných kritérií systému

Jednou z nutných podmienok kladených na systém bola **anonymita** voči nezúčastneným jednotkám. Toto kritérium je splnené. Vzhľadom na šifrovanie onionu, má táto dátová jednotka význam len pre stanicu disponujúcu príslušnými dešifrovacími kľúčmi. Všetky stanice v systéme nemusia byť dôveryhodného charakteru. Preto je výhodné, že systém založený na metóde Onion Routing zaručuje anonymitu aj vzhľadom na jednotky, ktoré sa zúčastňujú prenosu. Je to dané tým, že každá jednotka prenášajúca onion pozná len predchádzajúcu a nasledujúcu stanicu na trase. Celkovú trasu prenosu pozná iba odosielajúca stanica. Je dokonca na odosielateľovi, či poskytne konečnému príjemcovi informáciu o svojej identite. Ak však očakáva obojsmernú komunikáciu mal by tak učiniť.

Ďalším kritériom bola **decentralizovanosť systému**, teda nulový centrálny manažment. Táto podmienka je taktiež splnená, vzhľadom na to, že v systéme nefiguruje žiadny centrálny prvok, ktorý by zabezpečoval chod systému. O funkčnosť systému sa starajú zúčastnené jednotky prostredníctvom signalizačných správ HELLO, WELCOME a BYE. Každá jednotka je pomocou nich autonómna a získava informácie zo svojho okolia, ktoré si uchováva a na základe nich sa samostatne rozhoduje.

Kritériom, ktoré je splnené len z časti je **kritérium dynamickosti systému**. Táto podmienka určuje možnosť dynamického prihlasovania a odpojovania staníc do systému bez toho, aby bola funkčnosť systému nejakým spôsobom obmedzená alebo narušená. Problém nastáva predovšetkým v momente, keď sa stanica určená na prenos onionu, odpojí zo systému krátko pred jeho prijatím a spracovaním. V tomto prípade má odhlásenie stanice priamy vplyv na funkčnosť systému. Podobným problémom je úmyselné zahadzovanie dátových jednotiek niektorou zo staníc určených na prenos.

Problémom by mohla byť taktiež skutočnosť, že keď onion putuje sieťou a stanice postupne dešifrujú a odstraňujú jeho vrstvy, tak by sa dal na základe zmenšujúcej sa veľkosti onionu odhadnúť zostávajúci počet staníc na trase. Jednou možnosťou je doplniť onion na vysielacej strane na náhodnú dĺžku, respektíve pridať k onionu výplň náhodnej dĺžky. Druhou možnosťou je použitie onionu s konštantou dĺžkou. S tým samozrejme súvisí povinnosť staníc zúčastňujúcich sa na prenose po dešifrovaní a odstránení okrajovej vrstvy, doplniť onion náhodnými hodnotami rovnakej dĺžky ako odstránená vrstva.

Obmedzením navrhnutého systému je taktiež systém výmeny verejných parametrov asymetrického kryptosystému, ktoré sú prenášané ako súčasť správ HELLO a WELCOME. V praxi by bolo vhodnejšie zvoliť spôsob založený na prenose a overovaní prostredníctvom tzv. certifikátov (viď 3.3), aby stanice si boli isté tým, že verejné parametre skutočne patria udávanej osobe.

8 ZÁVER

V poslednom čase sa do popredia čím viac dostáva otázka zabezpečenia anonymity na internete. Použitím rôznych nástrojov je možné totiž v počítačovej sieti monitorovať činnosť užívateľov, s tým samozrejme súvisí strata súkromia. Anonymitu na internete nie je celkom jednoduché zabezpečiť. Je to dané IP protokolom, na ktorom princípe internet pracuje. Súčasťou prenášaných dátových jednotiek totiž býva IP adresa odosielateľa aj príjemcu. Tieto údaje môžu slúžiť na jednoznačnú identifikáciu komunikujúcich strán.

Hlavným cieľom mojej diplomovej práce bolo naštudovať metódy používané v anonymných systémoch a navrhnúť vlastný anonymný systém spĺňujúci požadované podmienky. V práci je postupne teoreticky opísaná metóda Onion Routing a asymetrický kryptosystém RSA, ktoré tvoria základ navrhnutého anonymného sytému. Pri samotnej implementácií bol zvolený jazyk C++. Druhá časť práce obsahuje popis samotnej implementácie anonymného systému, ktorý bol simulovaný na aplikačnej vrstve prostredníctvom socketov. Následne bol systém analyzovaný, čo sa týka jeho bezpečnosti a časových parametrov. Bola uskutočnená diskusia nad prihlasovaním procesom, ktorý je citlivou časťou systému. Môže byť zneužitý na záplavový útok a tým pádom na útok typu DoS (Denial of Service), ktorého cieľom je zabránenie poskytovaniu služby. Problémom je tiež spôsob výmeny verejných kľúčov, ktoré sú prenášané ako súčasť signalizačných správ. Tento problém sa však dá jednoducho vyriešiť pomocou infraštruktúry verejných kľúčov. Z podmienok kladených na systém, navrhnutý systém zaručuje anonymitu vzhľadom k nezúčastneným jednotkám a zároveň k jednotkám v rámci systému. Každá stanice totiž pozná iba predchádzajúcu a následujúcu stanicu na trase. Systém je plne decentralizovaný, pretože v systéme neexistuje žiadny centrálny prvok a o jeho riadenie sa starajú zúčastnené jednotky pomocou signalizačných správ. Kritérium dynamickosti systému je narušené v situáciach, kedy stanica sa odpojí zo systému krátko pred prijatím dátovej jednotky. Takéto odhlásenie stanice má priamy vplyv na funkčnosť systému. Rovnakým problémom je aj úmyselné zahadzovanie dátových jednotiek niektorou zo staníc na trase.

Hlavný prínos tejto diplomovej práce vidím v načrtnutí možnosti využitia a tvorby anonymných systémov. Vzhľadom na aktuálnosť problematiky anonymity na internete ide o zaujmavú a rýchlo sa rozvíjajúcu oblasť.

LITERATURA

[1] BOS W. J., KAIHARA E. M., KLEINJUNG T., LENSTRA K. A., MONTGOMERY L. P.: *On the Security of 1024-bit RSA and 160-bit Elliptic Curve Cryptography* [online]. [2009] [cit. 2010-03-01]. Dostupný z URL: <https://documents.epfl.ch/users/l/le/lenstra/public/papers/ecdl2.pdf>.

[2] BURDA K.: *Bezpečnost informačních systémů.* Brno: FEKT VUT v Brně, 2005. 104 s.

[3] DINGLEDINE R., MATHEWSON N., SYVERSON P.: *Tor: The Second-Generation Onion Router* [online]. [cit. 2009-12-01]. Dostupný z URL: <http://www.torproject.org/tor-design.pdf>.

[4] *Extended Euclidean algorithm* [online]. [2002-2009] [cit. 2009-12-01]. Dostupný z URL: <http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm>.

[5] KATTI S., KATABI D., PUCHALA K.: *Slicing the Onion: Anonymous Routing Without PKI* [online]. [2005] [cit. 2009-12-01]. Dostupný z URL: <http://dspace.mit.edu/handle/1721.1/30564>.

[6] LASOŇ M.: *Porovnání bezpecnosti kryptosystému RSA a Eliptických krivek* [online]. [2005] [cit. 2009-03-01]. Dostupný z URL: <http://homel.vsb.cz/ las03/ta/rsa-ecc.pdf>.

[7] *Miller-Rabin primality test* [online]. [2003-2009] [cit. 2009-12-01]. Dostupný z URL: <http://en.wikipedia.org/wiki/Miller-Rabin_primality_test>.

[8] NAMBIAR A. R.: *Towards A Stronger Peer-to-peer Anonymous System* [online]. 2006 [cit. 2009-12-01]. Dostupný z URL: <http://repositories.tdl.org/tdl/handle/10106/492>.

[9] *NTL: A Library for doing Number Theory* [online]. [cit. 2009-12-01]. Dostupný z URL: <http://www.shoup.net/ntl/>.

[10] PIRKL, J.: *Síťové programování pod Windows a programování Internetu.* České Budějovice: Kopp, 2001. 357s. ISBN 80-7232-145-5

[11] REED M. G., SYVERSON P. F., GOLDSCHLAG D. M.: *Anonymous connections and onion routing*, IEEE Journal on Selected Areas in Communication Special Issue on Copyright and Privacy Protection, 1998, Vol. 16, Issue 4, pp: 482-494, ISSN: 0733-8716

[12] YEH Y., HUANG T., LIN H., CHANG Y.: *A Study on Parallel RSA Factorization* [online]. [2009] [cit. 2010-03-01]. Dostupný z URL: <http://www.academypublisher.com/jcp/vol04/no02/jcp0402112118.pdf>.

[13] ZANDL, P.: *Írán je dalším pokusem o znásilnění Internetu* [online]. 2009 [cit. 2009-12-01]. Dostupný z URL: <http://www.lupa.cz/clanky/iran-je-dalsim-pokusem-o-znasilneni-internetu>.

SEZNAM PŘÍLOH

A UKÁŽKY ZDROJOVÝCH KÓDOV

A.1 Implementácia Miler-Rabinovho algoritmu

```
bool isPrime(NTL::ZZ n, int num)
{
 NTL::ZZ b, m, z, a;
 bool next;
 b = 0; m = n-1;
 while(NTL::IsOdd(m)==0)
 {
   m = m / 2; b++;
 }
 for (int i = 0; i < num; i++)
 {
   next = 0;
   a = NTL::RandomBnd(n-2); a += 2; // a musí byť väčšie ako 2
   z = NTL::PowerMod(a, m, n);
   if (z==1 || z==n-1) continue;
   for (int K = 1; K < b; K++)
   {
     z = NTL::PowerMod(z, 2, n);
     if (z==1) return 0;
     if (z==n-1)
     {
       next = 1;
       break;
     }
   }
   if (next) continue;
   return 0;
 }
 return 1;
}
```

A.2 Deklarácia triedy `Crsa`

```
class Crsa
{
 private:
   int keylength_bits, keylength_bytes, length_decr_block;
   NTL::ZZ n, VK, SK;
   void init(int keylength_bits, long seed);
   bool isPrime(NTL::ZZ n, int num);
   NTL::ZZ REA(NTL::ZZ r, NTL::ZZ VK);
 public:
   Crsa(void);
   ~Crsa(void);
   void init_crypto_keys(int keylength_bits, int length_decr_block,
                         long  seed);
   NTL::ZZ get_ZZ_VK(void);
   NTL::ZZ get_ZZ_n(void);
   char * add_padding (char * message, int * size);
   char * delete_padding (char * padd_message, int * size);
   char * crypt(char * message, int * size);
   char * decrypt(char * cypher, int * size);
};
```

A.3 Ukážka práce s objektom triedy `Crsa`

```
char mess [] = "nejaka sprava...";
int size = strlen(mess);
Crsa * rsa = new Crsa();
rsa->init_crypto_keys(512, 8, 123);
char * padd_mess = rsa->add_padding (mess, &size);
char * crypt_mess = rsa->crypt(padd_mess, &size);
char * decrypt_mess = rsa->decrypt(crypt_mess, &size);
char * result_mess = rsa->delete_padding (decrypt_mess, &size);
delete rsa;
delete [] crypt_mess; delete [] decrypt_mess;
delete [] padd_mess; delete [] result_mess;
```

A.4 Zapnutie zasielania broadcast zpráv

```
SOCKET socket;
int result, size, state;
size = sizeof(result);
socket = socket(AF_INET, SOCK_DGRAM,0);

state = getsockopt(socket, SOL_SOCKET, SO_BROADCAST,(char*) &result,
                   &size);

if (state == SOCKET_ERROR)
{
 closesocket(socket);
 return;
}

if (result == 0)
{
 result = 1;
 state = setsockopt(socket, SOL_SOCKET, SO_BROADCAST, (char*) &result,
                    size);
 if (state == SOCKET_ERROR)
 {
   closesocket(socket);
   return;
 }
}
```

B OBSAH PRILOŽENÉHO CD

- **Text práce** - obsahuje elektronický text diplomovej práce.
- **Zdrojové kódy** - obsahuje zdrojové kódy implementovaného anonymného systému.
- **Výsledný program** - obsahuje skompilovanú verziu anonymného systému pre 512 bitové šifrovacie kľúče.
- **NTL** - obsahuje použitú knihovnu `NTL.a`.