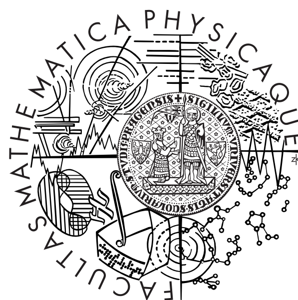
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Tomáš Pop

Kryptografie a její použití při zabezpečeném přenosu datových souborů

Název katedry či ústavu ??

RNDr. Drahomíra Doležalová-Spoustová, Ústav formální a aplikované lingvistiky

Informatika: Programování

2006

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne Tomáš Pop

Obsah

1 Úvod **7**
- 1.1 Historie . . . 7
- 1.2 Základní pojmy . . . 7
  - 1.2.1 Pro ujasnění . . . 7
  - 1.2.2 Entropie . . . 8
  - 1.2.3 Redundance, absolutní poměr jazyka . . . 8
- 1.3 Otázka obtížnosti řešení problému . . . 9
- 1.4 Otázka správné implementace . . . 10
- 1.5 Pohledy hodnocení kryptosystémů . . . 10
- 1.6 Útoky na kryptografické algoritmy . . . 11
- 1.7 Využití při přenosu souborů . . . 12
- 1.8 Poznámky . . . 12

2 Malé povídání o kryptografii **13**
- 2.1 Dělení šifer . . . 13
- 2.2 Volba a uchování klíčů . . . 14
- 2.3 Algoritmy asymetrické kryptografie . . . 14
  - 2.3.1 RSA . . . 15
  - 2.3.2 El Gamal . . . 16
  - 2.3.3 Ostatní Asymetrické šifry . . . 17
- 2.4 Algoritmy symetrické kryptografie . . . 18
  - 2.4.1 Vernamova šifra . . . 18
  - 2.4.2 Feistelovy sítě, s-boxy . . . 18
  - 2.4.3 DES a 3DES . . . 18
  - 2.4.4 Blowfish . . . 19
  - 2.4.5 Twofish . . . 19
  - 2.4.6 IDEA . . . 20
  - 2.4.7 Rijndael . . . 20

2.4.8 SkipJack, aneb jak to vidí úředníci 20
2.4.9 Zástupce streamových – RC4 20
2.5 Módy vstupních dat 21
2.6 Hašovací funkce 22
2.6.1 MD-stregthening 22
2.6.2 Algoritmus MD5 22
2.6.3 Algoritmy rodiny SHA-X 23
2.6.4 Klíčované jednocestné šifrovací systémy (MAC) . . . 23
2.7 Zmínka o kryptografických protokolech 23

3 Současné nástroje přenosu dat 25
3.1 Úvod do ssh 25
3.1.1 terminologie 25
3.1.2 Základy ssh 25
3.1.3 Autentifikace v ssh 25
3.1.4 Slabiny mechanismů ssh 26
3.1.5 Souvislost ssh, scp a sftp 27
3.1.6 sftp zblízka 27
3.1.7 Současné implementace SSH 27
3.1.8 Verze scp1, scp2, rychlosti při jednotlivých algoritmech 27
3.2 SSL 28
3.2.1 Autentifikace v SSL 29
3.2.2 Výhody a nevýhody SSL 29
3.3 Ostaní nástroje 29

4 Moje aplikace 30
4.1 Platforma 30
4.2 Terminologie 30
4.3 Motivace vzniku aplikace 30
4.4 Zaměření aplikace, specifikace 31
4.4.1 Cílová skupina uživatelů 31
4.4.2 Schéma ochrany dat 31
4.4.3 Synchronizace činnosti uživatelů 31
4.4.4 Další požadavky na aplikaci 32
4.5 Nástroje a knihovny potřebné k překladu aplikace 32
4.6 Vybaveni požadované pro chod aplikace 32
4.6.1 Proč SSH 33
4.7 Principiální fungování aplikace 33

4.8 Aplikace z pohledu uživatele 34
4.8.1 Popis zprovoznění aplikace 34
4.8.2 Konfigurační soubor 34
4.8.3 Start aplikace 36
4.8.4 Uživatelské rozhraní 36
4.8.5 Výpisy klienta 37
4.8.6 Příklad session 39
4.8.7 Licence 40
4.9 Technická řešení 41
4.9.1 Spuštění serveru a schéma propojení s ssh 41
4.9.2 Rozdělení práce na projektu 41
4.9.3 Protokol komunikace mezi klientem a serverem 41
4.9.4 Popis adresářové struktury a schéma uložení dat na serveru 43
4.9.5 Model serveru 43
4.9.6 Získávání dat ze serveru 43
4.9.7 Kryptovcí jádro aplikace 44
4.9.8 Synchronizace práce uživatel 45
4.10 Zhodnocení aplikace 47
4.10.1 Možné útoky na aplikaci 47

5 Závěr **50**

6 Přílohy **51**
6.1 Zdrojové kódy aplikace 51

Literatura **52**

Název práce: Kryptografie a její použití při zabezpečeném přenosu datových souborů
Autor: Tomáš Pop
Katedra (ústav): ??
Vedoucí bakalářské práce: Mgr. Drahomíra Doležalová-Spoustová, Ústav formální a aplikované lingvistiky
e-mail vedoucího: Drahomira.Spoustova@mff.cuni.cz

Abstrakt: Předložená práce se zabývá aplikovanou kryptografií. Cílem práce není v žádném případě studovat algoritmy a protokoly po stránce matematické a statistické, ale je spíše jakýmsi pohledem na kryptografii z pohledu programátora. Případnému čtenáři by měla usnadnit výběr vhodného algoritmu a postupu pro jeho produkt. V příkladu, který je součástí práce je také dána přednost čitelnosti, a srozumitelnosti před ideální implementací z hlediska rychlosti.

Klíčová slova: kryptografie, ssh, kryptografie, datový přenos.

Title: Cryptography and Its Application in Secure Data-File Transfer
Author: Tomáš Pop
Department: Název katedry či ústavu v angličtině
Supervisor:
Mgr. Drahomíra Doležalová-Spoustová, Institute of Formal and Applied Linguistics, UK MFF Supervisor's e-mail address: Drahomira.Spoustova@mff.cuni.cz

Abstract: In the present work we study applied cryptography… The goal of this work is not to study algorithms from the mathematical or statistical view, instead we are trying to look at cryptography from a poin of view of the „end consumer“ or „programmmer“. It should help the reader to find an appropriate algorithm for his program. The application on the end of the work is written to as readable as possible.

Keywords: cryptography, ssh, cryptography, secure data transfer.

Kapitola 1

Úvod

1.1 Historie

Jeden z nejstarších systémů symetrické kryptografie je pravděpodobně takzvaný Skytale, používaný sparťany v 5. století př. n. l. Tajemství bylo tvořeno hůlkou jistého průměru, na kterou se namotal kožený proužek, tak, že pokrýval povrch hůlky. Poté na něj byla zapsána zpráva ve směru podélné osy. Po odmotání proužku byla zpráva nečitelná.

Kryptografie v podobě takové, jakou si ji představujeme dnes se začala rozvíjet na počátku 20 století, zejména s rozvojem telegrafie. Ještě většího využití došly šifry během druhé světové války. Pro tuto dobu byly typické mechanické šifrovací stroje typu podobného známého systému enigma.

V druhé polovině 20. století dochází ke značnému rozšíření použití v obchodní a finanční sféře. Kryptografie přestává být záležitostí vojáků, ale stává se věcí veřejnou.

Během posledních deset let se stala většina požívajících algoritmů věcí otevřenou, podléhají veřejné diskusi, která často výrazně napomáhá odhalení chyb.

1.2 Základní pojmy

1.2.1 Pro ujasnění

Terminologie v oblasti kryptografie je poněkud uvolněná, jedno slovo podle kontextu často zastává několik významů. V následujícím textu budeme používat následující pojmy:

kryptografie — Věda o tvorbě šifer
kryptoanalýza — Věda o prolamovaní (luštění šifer)
kryptologie — Věda o šifrování, obecné označení pro kryptografii a kryptoanalýzu
otevřený text, plaintext — Originální tvar dat
Zašifrovaný text, ciphertext — Zašifrovaný tvar zprávy
šifrování, kryptování, enkryptování, enciphering — Proces přeměny Plaintextu na Ciphertext
dešifrování, dekryptování, deciphering — Zpětný proces přeměny Ciphertextu Plaintext
šifrovací algoritmus — Popis postupu šifrování
dešifrovací algoritmus — Popis postupu dešifrování
šifra — Popis postupu umožňujícího šifrování a dešifrování
protokol — Sada pravidel umožňující dohodu a výměnu informací mezi více subjekty
kryptosystém — Systém umožňující šifrování a dešifrování

1.2.2 Entropie

Nejprve definujeme co je množsví informace. Nechť je každý význam zprávy M stejně pravděpodobný. Potom *množství informace* definujeme jako počet bitů nutných pro zakódování všech možných významů dané zprávy. M. Množství informace ve zprávě popisuje veličina *entropie*, značíme ji $H(M)$. Je snadné nahlédnou vztah $H(M) = \log_2 n$, který platí za předpokladu stejné pravděpodobnosti všech zpráv, kde n značí počet možných významů zprávy.

1.2.3 Redundance, absolutní poměr jazyka

Přirozené jazyky, které se šifrují, obvykle obsahují méně informace, než kolik by se dalo soudit z jejich zápisu. Je poměrně vhodné stanovit veličinu, která popisuje kolik informace je skutečně obsaženo v nějaké jednotce popisu jazyka. Mějme zprávu délky N znaků. Potom definujeme *poměr jazyka* r jako $r = \frac{H(M)}{N}$. Jednotkou je *bpc*, bit per char. Pro zajímavost anglický jazyk má přibližně $r = 1,3$. V teorii informace je možno dokázat že poměr jazyka závisí nejen na jazyku, ale i na délce zprávy. (Velmi krátké zprávy mají větší poměr).

Absolutní poměr jazyka definujeme jako $R = log_n L$, kde L je počet znaků v abecedě. Motivace je zřejmá, jde o to postihnout, kolik bitů je třeba na zakódování jednoho znaku abecedy.

Redundance je definována jako $D = R - r$. Redundance je tedy v *bpc*. Je zajímavé zavést ještě *poměrnou nadbytečnost* jako podíl redundance a absolutního poměru jazyka, který na první pohled něco vypovídá. Anglický text pak z pohledu teorie informace obsahuje asi 72% nadbytečných znaků.

Redundance je nepříjemnou vlastností, která napomáhá útočníkovi. K jejímu snížení můžeme použít:

kompresi, která je obzvláště vhodná, pokud chceme urychlit šifrování.

zmatení (confusion), opakování operací substituce.

rozptýlení (diffusion), operace transpozice.

Zmatení i rozptýlení (pokud jsou použita) jsou zpravidla součástí samotného algoritmu.

1.3 Otázka obtížnosti řešení problému

Prakticky každý algoritmus je založen na nějakém matematickém problému, který (v případě správné implementace a použití) musí případný útočník řešit, pokud chce šifru prolomit.

Polynomiální problémy K řešení takových problémů potřebujeme provést $c \cdot f(n)$, operací kde $f(n)$ je polynomiální funkce, c je vhodná konstanta a n je velikost vstupních dat. Obecně problémy které jdou na strojích v současné době známých řešit lineárním nebo polynomiálním čase, považujeme za výpočetně snadné a pro kryptografické použití většinou nevhodné.

NP-úplné problémy jsou sami o sobě námětem na dlouhou diskusi. To ovšem výrazně přesahuje meze tohoto textu (a jeho autora :-)), omezíme se na tvrzení, že jsou výpočetně složitější než problémy polynomiální. V kryptografii se s nimi setkáváme poměrně často.

Exponenciální problémy jsou problémy, k jejichž přesení potřebujeme provést $c^{f(n)}$, kde n je velikost vstupních dat a f je polynomiální funkce.

Pro ilustraci je zde tabulka 1.1, která popisuje zpracování vstupu délky 10^3:

Tabulka 1.1: Porovnání složitosti problémů

Složitost výpočtu	Počet operací	doba výpočtu
Konstantní	1	1 μs
Lineární	10^3	1 ms
Kvadratická	10^6	1 1s
Kubická	10^9	17 min
Exponenciální	$2 \cdot 10^{119}$	10^{105} let

1.4 Otázka správné implementace

U kryptosystémů je velice důležitá implementace daného algoritmu nebo protokolu. Důkazem může být například to, že jen malá část útoků je vedená na samotný princip algoritmu nebo protokolu. Většina útoků je vedena právě na implementaci, popřípadě na použití. Jako praktickou ukázku můžu zmínit např. tzv. „časový útok “ na šifru RSA, kterou podrobněji popíšu později. Vpodstatě jde o fakt, že klíčem je velké číslo, a princip dešifrování souvisí s umocněním zprávy na toto velké číslo. Z toho plyne, že čas potřebný k dešifrování je při nesprávné implementaci v relaci s klíčem. Kupodivu i takováto „maličkost“ může postačovat ke značnému prořezání prostoru klíčů, a k případnému úspěšnému útoku. Další nebezpečí je ve výběru a správě klíčů viz 2.2.

1.5 Pohledy hodnocení kryptosystémů

Na hodnocení kryptosystémů můžeme nahlížet z mnoha různých stran. Hodnocení může založit na rychlosti výpočtu, bezpečnosti nebo třeba na snadnosti implementace. Ani tento způsob nejde zcela zavrhnout, množství námahy vynaložené na zašifrování dat by mělo být úměrné míře bezpečnosti, kterou požadujeme. Jsou ale určitá pravidla, která by měl kryptosystém splňovat.

- Šifrováním by neměl narůstat objem dat. A pokud naroste, tak jen o konstantní velikost(vycpávky na konci souborů a pod)
- Implementace by měla být únosně složitá
- Rozumná implementace by měla být přiměřeně rychlá

- Šifra by neměla obsahovat žádná omezení na data, na která bude použita.
- Chyby při šifrování by se neměli nepřiměřeně šířit.

V tomto textu se budeme snažit hodnotit šifry podle několika kritérií, hlavním ovšem bude bezpečnost.

Z hlediska bezpečnosti rozdělujeme šifry na:

nepodmíněně bezpečné tj. takové, které nepodávají žádnou informaci o zprávě. Je známa jediná taková šifra a to sice Vernamova šifra s jednorázovým utajeným klíčem. viz 2.4.1.

prolomitelné Všechny ostatní dnes používané systémy. Dají se rozlišit

dokazatelně bezpečné u kterých umíme dokázat, že k jejich prolomení je třeba vyřešit výpočetně složitý problém

výpočetně bezpečné u kterých se předpokládá, že na jejich řešení je třeba vyřešit nereálné výpočetní úsilí, spadá sem většina dnešních algoritmů.

bezpečné u kterých cena kryptoanalýzy převyšuje cenu utajovaných dat, popřípadě kryptoanalýza nemůže proběhnout v dostatečně krátkém čase.

1.6 Útoky na kryptografické algoritmy

Útoky se z hlediska znalostí útočníka dělí na:

Ciphertext only attack, který je nejobtížnější, z Ciphertextu se pokouší zjistit tajemství nebo plaintext. Je účinný jen pro velmi slabé kryptosystémy.

Known plaintext attack, který se z plaintextu a ciphertextu snaží určit tajemství.

Choosen plaintext attack, kterému jsou na vstup přiváděny určité zprávy, a je pozorováno jak se chová, jaké má slabiny.

Je zřejmé, že první je jasně nejobtížnější, a že bychom se v každé aplikaci kde je to možné měli snažit, dát možnost pouze k tomuto typu útoku.

1.7 Využití při přenosu souborů

Kryptografie může být použita na různých úrovních, implementovaná může být hardwarově nebo softwarově. Hardwarová implementace je obvyklá na linkové úrovní a je pro uživatele transparentní. V tomto textu se ovšem budeme zabývat více softwarovou implementací. Ta se obvykle na aplikační úrovni (nebo velmi těsně pod ní).

1.8 Poznámky

Kryptografie je jeden z oborů, kde člověk (ať už jako návrhář nebo jako uživatel) musí trvale trpět jistou dávkou paranoy. Otázek je stále mnoho, u většiny algoritmů jejich složitost není formálně dokázána (!). Technologie, které jsou použity při kryptoanalýze se prudce rozvíjejí, a je těžké odhadnout, co vlastně útočník nasadí proti použitým technikám.

Navíc rozvoj je značně bržděn špatně fungující zpětnou vazbou. Jednotlivci i organizace často tají z různých důvodů jak objevené algoritmy tak slabiny algoritmů již existujících.

Kapitola 2

Malé povídání o kryptografii

2.1 Dělení šifer

Současné šifry se rozdělují většinou podle dvou různých kritérií, a to podle množství dat, které je šifra schopna zpracovat najednou (šifry blokové a šifry streamové) a podle toho, zda odesilatel a příjemce musí sdílet nějaké tajemství (šifry symetrické a asymetrické).

blokové šifry pracují s celými bloky dat, obvykle ve velikosti 8 - 128 B.

streamové šifry pracují s jednotlivými bity zprávy zvlášť. Jsou obecně považovány za méně bezpečné, jsou pomalejší než šifry blokové.

symetrické šifry Odesilatel i příjemce sdílí jedno tajemství nutné k dešifrování a šifrování zprávy.

asymetrické šifry Odesilatel a příjemce šifrují a dešifrují zprávu různými klíči Nemusí spolu sdílet žádné tajemství. Každá operace realizovatelná pomocí symetrické kryptografie jde realizovat pomocí asymetrické kryptografie. Její nevýhodou je, že je o několik řádů pomalejší než symetrická kryptografie.

V přenosu souborů se požívá především algoritmy blokové a symetrické, proto se budeme zabývat především jimi. Asymetrické šifry mají uplatnění v autentizaci odesilatele a příjemce, popřípadě v podpisech zpráv, streamové jsou vhodné pro šifrování krátkých souborů.

2.2 Volba a uchování klíčů

Klíče a jejich volba jsou zcela stěžejní částí zabezpečení dat. Ocelová vrata jsou jistě skvělá, ale naprosto zbytečná bezpečnostní pomůcka, pokud jde jejich zámek otevřít šroubovákem. A podobně je to i s klíči. sebelepší algoritmus je zcela bezmocný, pokud je jako tajemství použito např. Vaše jméno, nebo datum narození Vaší sousedky. Sociální útoky jsou velmi rozšířenou a nečekaně účinnou metodou lámání kryptografické ochrany dat

Klíč by měl být přiměřeně dlouhý, pokud je prostor klíčů příliž malý, vystavujeme se útoku hrubou silou. Je třeba stále pamatovat na to, že výpočetní síla strojů neustále roste, a šifra dnes bezpečná, za několik let vůbec bezpečná být nemusí.

Dalším problémem je generování náhodných „dočasných" klíčů. Na generátory ze standardních knihoven jsou vedeny často útoky. Je proto náhodná data dobré získávat od uživatele, nebo je odvozovat alespoň od dostatečně stochastického stavu systému (zařízení random na linuxoých strojích).

Některé algoritmy jako např. RSA potřebují ke své činnosti klíče značně specifické, v případě RSA velká prvočísla. Generovat takovéto klíče je již samo o sobě netriviální úloha, a navíc algoritmy používané v dnešní době nám dávají záruky správného výsledku pouze s určitou (ikdyž prakticky libovolnou) pravděpodobností.

Věčnou a často aktuální otázkou je, jestli volit jeden delší klíč, nebo několik kratších a včas je měnit.

Navíc je dobré si rozmyslet kdy a jak dlouho bude klíč držen v paměti. I z paměti programu (a to jak za běhu tak po skončení) může útočník leccos poznat.

2.3 Algoritmy asymetrické kryptografie

Jak už bylo řečeno, symetrické algoritmy jsou takové, kde odesilatel a příjemce nesdílí žádné tajemství... Po úvaze jsem se rozhodl popsat asi nejznámější algoritmus RSA trochu podrobněji, ostatní méně. V následujících odstavcích budou použita některé věty algebry, Většinu z nich nebudu explicitně ani zmiňovat, jindy budou uvedeny bez důkazu. Nejde mi ani tak o naprostou korektnost, jako spíš o to ukázat, že se za kryptografií často schovává pěkná matematika.

2.3.1 RSA

Matematický podtext

Algoritmus RSA je založen na poměrně netriviálních výsledcích algebry. Už proto mi přijde jako docela pěkné zavést ho o trochu formálněji než zbytek tohoto textu. Uvádím tu alespoň ty nejdůležitější poznatky. Důkazy najdou případní zájemci třeba v [4].

Definice 1 *(Kongruence) Mějme následující relaci definovanou na okruhu celých čísel.* $\theta_m = \{< a, b >; a = b + t \cdot m, t \in \mathbf{R}, m \in \mathbf{N}\}$ *Potom tuto relaci nazveme kongruence modulo m. Tato relace je ekvivalence.*

Definice 2 *(Třída rozkladu) Třídou rozkladu budeme nazývat* $[a]_m = \{b; < a, b > \in \theta_m\}$.. *Symbolem* $\mathbb{Z}/\theta_m = \{[a]_m; a \in \mathbb{Z}\}$ *budeme nazývat systém tříd rozkladu. Zajímavé pro nás ovšem bude, že je možné na* $\mathbb{Z}/\theta_m$ *jde zavést operace*

$$[a]_m + [b]_m = [a + b]_m, [a]_m \cdot [b]_m = [a \cdot b]_m.$$

Definice 3 *(Eulerova* φ*-funkce) Zobrazeni* $\varphi : \mathbb{N} \to \mathbb{N}$*, které každému* $n \in \mathbb{N}$ *zobrazuje na počet menších nesoudělných čísel nazveme Eulerova* φ*-funkce. Formálně ji můžeme popsat* $\varphi(n) = Card\{k; k \in \mathbb{N}, k < n, nsd(k, n) = 1\}$

Lemma 1 *Pokud p,q* $\in \mathbb{N}$ *jsou prvočísla, je* $\varphi(p) = p - 1$ *a* $\varphi(pq) = (p - 1) \cdot (q - 1)$.

Zcela zásadní je zejména následující:

Věta 2 *(Fermatova-Eulerova) Nechť* $q \in \mathbb{Z}, n \in \mathbb{N}$ *jsou nesoudělná. Potom platí. Potom platí*

$$\left[q^{\varphi(n)}\right]_n = [1]_n.$$

Princip algoritmu

Nejprve jsou zvolena náhodně dvě prvočísla p a q. Potom je spočten jejich součin, $n = p \cdot q$. Potom zvolíme dvě libovolná celá čísla e, d tak aby platilo $[e \cdot d]_{\varphi(n)} = [1]_{\varphi(n)}$. Nyní označíme plain–text zprávu v zakódovanou zprávu w. Šifrování probíhá podle vztahu $[w]_n = [v^e]_n$, dešifrování podle vztahu $[v]_n = [w^d]_n$. Klíči jsou tedy dvojice $< e, n >$ a $< d, n >$. Pokud by někdo chtěl z jednoho klíče dopočítat druhý, musel by řešit kongruenci $[e \cdot d]_{\varphi(n)} = [1]_{\varphi(n)}$, což v současné době vyžaduje rozklad čísla n, a právě v tomto kroku se skrývá „asymetrie“ tohoto algoritmu.

Korektnost algoritmu

Pokud jsou splněny všechny předpoklady z předchozího odstavce a zpráva je zašifrována popsaným způsobem. Půjde nám o to ukázat, že $[v]_n = [w^d]_n$.

Z požadavku na čísla e, d plyne, že $e \cdot d = 1 + r \cdot \varphi(n), r \in \mathbb{Z}$. Pokud vztah, který popisuje šifrování umocníme dostaneme $\left[w^d\right]_n = \left[v^{e \cdot d}\right]_n$. Z těchto dvou poznatků tedy plyne, že stačí dokázat vztah $[v]_n = \left[v^{e \cdot d = 1 + r \cdot \varphi(n)}\right]_n$.

Pokud je v s n nesoudělné, máme vyhráno. Platí totiž $\left[v^{1 + r \cdot \varphi(n)}\right]_n = [v]_n \cdot \left[v^{r \cdot \varphi(n)}\right]_n$, a druhý člen tohoto sočinu je v důsledku Fermatovy-Eulerovy věty ve stejné třídě jako 1.

Pokud v s n soudělné, tak musí být buď $v = a \cdot p$ nebo $v = b \cdot q$. Nechť platí (BÚNO) např. první. Jelikož je ovšem v nesoudělné s q, (q je prvočíslo) platí $v = a \cdot p$ a tedy

$$\left[v^{r \cdot \varphi(n)}\right]_q = \left[v^{\varphi(n)}\right]_q^n = \left[v^{(p-1) \cdot (q-1)}\right]_q^r = \left[v^{(q-1)}\right]_q^{r \cdot (p-1)} = [1]_q^{r \cdot (p-1)} .$$

Tento fakt jde jinými slovy vyjádřit tak, že $v^{r \cdot \varphi(n)} = 1^r + t \cdot q$. Pokud tento vztah vynásobíme v a uvážíme, že $n = p \cdot q$, dostaneme vztah $v^{1 + r \cdot \varphi(n)} = v + (at)n$, což je přesně vztah, který jsme chtěli dokázat.

Poznámky

Mnoho věcí by chtělo dodat. Například, to, že jde dokázat, že ke každému e, n náleží pouze jedno d. Chtělo by také doplnit, jak se k d dopočítat, nebo jak efektivně generovat prvočísla, ale to je mimo možnosti tohoto textu.

Co se týká bezpečnosti algoritmu, je považován při volbě dostatečně dlouhého klíče za bezpečný. Jeho nevýhodou je, že vzhledem k stále dokonalejším algoritmům rozkladu čísel na prvočísla (tzv. sítům) je třeba za bezpečnou délku klíč považovat nejméně 1024 bitů, nebo lépe ještě delší.

Volba takto dlouhého klíče bude zjevně výrazně prodlužovat dobu výpočtu i legitimního uživatele.

2.3.2 El Gamal

Algoritmus El Gamal byl vlastně první asymetrický algoritmus, který fungoval. (Myšlenka asymetrického šifrování pochází od Diffieho a Helmana, kteří ovšem žádný funkční algoritmus nevymysleli). V jistém smyslu není tak pěkný elegantní, jako algoritmus RSA, na první pohled je jednodušší, je založen na složitosti výpočtu diskrétního algoritmu. Jeho největší nevýhodou

je, že prodlouží šifrovaná data na dvojnásobnou délku. Možná proto dnes není používán tak masivně jako RSA. Je ovšem nutné konstatovat, že na výpočet diskrétního logaritmu je v současné době, vzhledem k již zmíněným sítům na rozklad prvočísel považován za problém, který dává potenciálnímu útočníkovi menší šanci na úspěch.

Fungování algoritmu

Nechť je zvolen veřejně známá čísla q, tzv modul a g co nejvyššího řádu. I-tý účastník si volí svůj tajný klíč y_i a vypočte veřejný klíč k_i^{pub} jako $g^{y_i} \bmod q$ Pokud potom posílá A zprávu P uživateli B (zpráva musí být menší než q) tak komunikace probíhá podle následujícího schématu.

- je zvoleno náhodné číslo k.
- A spočte $g^k \bmod q$ a $Q = P \circ (g^{y_B})^k \bmod q$ a obě tato čísla pošle uživateli B.
- Uživatel B spočte $(q^k)^{y_B} \bmod q$ a k tomuto číslu určí inversní prvek (vzhledem k operaci $\circ$).
- Uživatel B spočte zprávu P jako $P = Q \circ ((q^k)^{y_B})^{-1}$.

Korektnost algoritmu

S využitím vět algebry za daných předpokladů platí:

$$(P \circ (q^{y_B})^k \circ ((q^k)^{y_B})^{-1}) \bmod q = (P \circ (q^{y_B})^k \circ ((q^{y_B})^k)^{-1}) \bmod q = P$$

2.3.3 Ostatní Asymetrické šifry

Poměrně často využívána v asymetrické kryptografii „finta“. Tato finta je založena na následující myšlence: V přirozených číslech toho známe hodně, a je tedy snadnější algoritmy prolamovat, na základě jistých charakteristik prvočísel. Ale algoritmus RSA není na přirozených číslech přímo závislý. Pokud tedy najdeme jiné vhodné těleso, kde vhodně definujeme potřebné operace a obdoby uvedených vět, můžeme algoritmus používat někde, kam za námi „útočníci nemůžou“. Podobným pokusem je např. těleso eliptických křivek(ECC, Eliptic Kurve Cryptosystem). Tato vize má ovšem slabinu, že je velmi pravděpodobné, že je jen otázkou času, než budou ony prolamující

algoritmy nalezeny i v těchto tělesech, a proto je mnoho lidí s použitím podobných systémů opatrných.

2.4 Algoritmy symetrické kryptografie

Algoritmy symetrické mají sice proti asymetrickým menší možnosti využití (vše, co jsme schopni udělat symetrickou kryptografií, jsme schopni zajistit i asymetrickou, ale naopak to neplatí), ale zato jsou o několik řádů rychlejší.

2.4.1 Vernamova šifra

Tato šifra, která pochází z počátku 20. století je jediná o které je formálně prokázáno, že je bezpečná. Důkaz provedl až Shannon, ikdyž už Vernam si byl podle svých slov jist, že jde o šifru absolutně bezpečnou. Jediná její chyba je v tom, že je nepoužitelná. Jde o to, že se ke zprávě operací xor připojí klíč stejné délky, jako zpráva sama a tím se problém přesune z přenesení zprávy na přenesení klíče.

2.4.2 Feistelovy sítě, s-boxy

velká třída dnes používaných šifer je dnes realizována jako feistelova síť. Asi nejvýstižněji daný mechanismus zachycuje obrázek. Princip spočívá v tom, že se několikrát v cyklu data rozdělí na dvě poloviny, jedna polovina je modifikována klíčem a je nějakým reversibilním mechanismem promíchána s druhou polovinou.

Druhým důležitým pojmem jsou tzv. s-boxy. Obecně je s-box mechanismu, který ke vstupu o délce n bitů vrací výstup dékly m bitů. s-boxy bývají zpravidla implementovány jako tabulky a to buď předem známé, jak je tomu u DESu a nebo vypočítávané z klíče, jak je tomu u šifer Blowfish a Twofish.

2.4.3 DES a 3DES

Des je velice stará šifra. Jedná se vpodstatě o klasickou ukázku feistelova systému. Převádí 64 bitový vstupní blok na 64 bitový výstupní blok. Nejprve je ze vstupního líče délky 64 bitů vyrobeno 16 subklíčů délky 48 bitů. Data projdou vstupní permutací - ta nemá prakticky žádný vliv na bezpečnost.

V návrhu DESu je z technických důvodů efektivity přístupu k datům na původních čipech, které DES realizovaly.

Poté je vstupních 64 bitů rozděleno na dva 32-bitové díly. V 16 cyklech se opakuje následující postup: Jedna polovina je expandována na 48 bitů (expanzní permutací), je operací xor složena se subklíčem příslušným danému cyklu. Předposlední fází je substituce fixním s-boxem. Na závěr dojde ještě k permutaci, poloviny jsou prohozeny, a začíná další cyklus. Po posledním cyklu dojde ještě k jedné permutaci, která ovšem opět není pro bezpečnost důležitá. Dešifrování probíhá obdobně, jen s jinými klíči. DES je ukázkou dobře navržené šifry, vpodstatě byla zlomena až speciálním zařízením (tzv. DES cracker) hrubou silou. Jeho největší slabinou je nevhodnost softwarové implementace a malý prostor klíčů (ze 64 bitů klíče se efektivně využívá jen 56, zbylé jsou paritní). 3DES (TDES, Triple DES) je obdobou, kdy jsou data zašifrována dešifrována a zašifrována různými klíči. O 3DESU odborníci tvrdí, že je bezpečný, ovšem je poměrně pomalý.

2.4.4 Blowfish

Blowfish je šifra, která byla vytvořena, aby nahradila zastaralý DES. Funguje v jistém smyslu podobně jako DES. Transformuje také 64 bitů na 64 bitů, jedná se také o feistelovu síť.Hlavní změnou je, že s-box je generován z klíče proměnné délky (32-448 bitů) netriviální a výpočetně poměrně složitou cestou. Ač se to může jevit jako nevýhoda, je to jeden z pilířů bezpečnosti tohoto algoritmu. Značně znesnadňuje útok hrubou silou, jen spočtení s-boxů na „odzkoušení“ jednoho klíče je časově velice náročné. Přitom prostor klíčů je dostatečný. Další výhodou je, že byl navržen s ohledem na softwarovou implementaci na dnešních procesorech a že není licencován.

2.4.5 Twofish

Je konstrukčně podobný algoritmu Blowfish, odstraňuje některé jeho nedostatky. Operuje nad bloky délky 128 bitů. Zmiňuji ho, protože je považován za jeden z nejkvalitnějších současných šifer (Jeden z finalistů AES), a jeho tvůrce se rozhodl ho poskytnout volně k použití (není licencován). Délka klíče je nejvýše 256 bitů.

2.4.6 IDEA

Jedná se o patentovaný algoritmus, délka bloku je 64 bitů a délka klíče je 128 bitů. Algoritmus probíhá v 8 kolech. V první fázi je vytvořeno pomocí bitových posunů 52 subklíčů délky 16 bitů (šest na každé kolo a 4 na závěrečnou transformaci). Vstupní slovo je rozděleno na 16 bitové subbloky. V každém kroku se provádí několik operací násobení a operací xor, schéma je poměrně složité. V [1] autor píše, že algoritmus IDEA považuje za nejbezpečnější tehdejší blokovou šifru, což jistě svědčí o její značné kvalitě. (Podotýkám, že kniha je starší a dnes se za bezpečnější považují jiné, třeba Rijndael).

2.4.7 Rijndael

Je vítěz soutěže o Advanced Cryptography Standart, a je v současné době považován za „krále" blokových algoritmů. Jeho specifikem je jeho formální definice, v matematickém stylu (je definován nad tělesem polynomů). V poslední době se objevují znepokojivé zprávy o možnostech útoku na AES, nicméně, prakticky se nepodařilo žádný realizovat. Algoritmus je sice velice složitý, jde však elegantně převést na v počítači snadno implementovatelné a rychlé procesy. Pro zájemce je k dispozici podrobný popis algoritmu, včetně zmíněného matematického podtextu, zde: http://www.macfergus.com/pub/rdalgeq.pdf Já osobně jsem se do studia tohoto algoritmu nepustil.

2.4.8 SkipJack, aneb jak to vidí úředníci

Tento algoritmus uvádím jen jako zajímavost. Jeho specifikum spočívá v tom, že jsou v jeho konstrukci implementována tzv. zadní vrátka, neboli, že na soudní příkaz může být šifra zlomena. Tento fakt je důvodem (spolu s licencí), že se algoritmus prakticky kromě státní sféry v USA příliš neuplatnil.

2.4.9 Zástupce streamových – RC4

Bylo by nefér zcela vynechat streamové šifry, ikdyž v přenosu datových souborů se nevyužívají tak často, protože jsou obvykle pomalejší než šifry blokové. To ovšem neplatí vždy, např. RC4 je asi desetkrát rychlejší než DES. RC4 je na první pohled velice jednoduchý, přesto je považován za bezpečný a je (nebo alespoň byl) součástí např. implementace databáze Secure-Oracle.

Tabulka 2.1: Generování pseudonáhodných bytů

```
i = i + 1 mod 256;
j = j + S[i] mod 256;
swap(S[i], S[j]);
output(S[S[i]+S[j]] mod 256)
```

Nejprve je pole S naplněno čísly 0-255, $S[i] = i$, a pole S_2 naplněno klíčem podle schématu $S_2[i] = key[i \bmod strlen(key)]$. Potom je pole S zamícháno:

```
j=0;
for(i=0;i<256;i++){
 j+=S[i]+ S2[i] mod 256;
 swap(S[i], S[j]);
}
```

Dále je používáno jen pole S. Schéma 2.1 ukazuje, jak se v RC4 generuje byte po bytu, se kterým je xorován plaintext a vzniká ciphertext (nebo opačně při dešifrování). Jedinou známou slabinou RC4 je fakt, že na začátku náhodné sekvence je více nul než jedniček. Po asi 100 bytech tento nedostatek zmizí.

2.5 Módy vstupních dat

V předchozí části jsem nastínil jak jsou šifrovány bloky dat o velikosti bloku, což bývá od 8 do 128 bitů. Většinou však chceme šifrovat delší data. Nejjednodušší metodou je tzv. ECB mód, neboli data jsou rozdělena na bloky a bloky jsou zašifrovány jeden za druhým. Tatometoda je nejjednodušší, a je odolná proti chybám (pokud dojde k porušení jednoho bloku, nepřijdeme o více, než právě jeden blok).

Mírně složitější je CBC mód, který zavádí zpětnou vazbu na předchozí data. Každý blok je před zašifrováním provázán se zašifrovaným předchozím blokem (například operací xor). Tato cesta znesnadňuje možnosti statistických útoků.

Ještě sofistikovanější je CFB, kde se podle dat zašifrovaných v předchozím bloku mění šifrování (např se změní s-boxy) bloku následujícího. Výsledek je tedy funkcí dat, klíče a předchozího bloku po šifrování.

Módu vstupních dat je samozřejmě více, další variací je míchání částí ciphertextu s plaintextem před zapsáním (CTR) a různé mutace a kombinace již jmenovaných. Více informací může zájemce najít na wikipedii, např zde: http://en.wikipedia.org/wiki/Cipher_Block_Chaining

Mód vstupních dat může mít zásadní vliv na bezpečnost aplikace.

2.6 Hašovací funkce

Hašovací funkce jsou cosi zvláštního. Snažíme se vytvořit funkci, která ze vstupu proměnné délky bude generovat výstup pevné délky (řádově desítky nebo stovky bitů), bude se chovat dostatečně „splašeně“, tj. bude při malé změně vstupu vykazovat velkou změnu výstupu, a pokud známe výstup bude velmi obtížné určit vstup. Jakkoli jsou na první pohled takovéto funkce zbytečné, jsou ve skutečnosti velmi potřebné.

2.6.1 MD-stregthening

Tato technika dnes vpodstatě pokrývá všechny běžně používané hašovací algoritmy. Je založená na následujícím principu: $h_i = f(M_i, h_{i-1})$, kde h_i je i-tá iterace výpočtu kontrolního součtu na i-tém bloku dat M_i. Tato metoda zaručuje rychlé šíření poruch vstupu do celého součtu tím, že zavádí kontextovou vazbu.

2.6.2 Algoritmus MD5

Algoritmus MD5 je jeden z nejpoužívanějších v dnešní době, ikdyž na některé jeho aplikace existují „efektivní“ útoky. V tomto smyslu slovo „efektivní“ značí, že stačí řádově něco okolo 2^{60} operací na nalezení druhé zprávy se stejným součtem ke známé zprávě. Proto by se dnes (pokud není chráněn např. šifrováním, aby útočník nevěděl na co útočí) neměl v aplikacích zabývajících se bezpečností vyskytovat.

Produkuje hash v délce 128b, což je málo. Algoritmus pracuje následujícím způsobem:

1. Zpráva je doplněna na délku dělitelnou 512, tak, že se přidá 1 a potom tolik nul, aby zbylo 64b. do těchto 64 bitů je uložena délka původní zprávy.

2. Poté je rozdělena do bloků po 512 bitech.

3. Každý blok je dále rozdělen na 128b subbloky a ty na 4 32b slova A, B, C, D.
4. Bloky A, B, C, D jsou inicializovány na fixní konstanty
5. Při každém zpracování subbloku docházi v 16 kolech k „promíchání“ bloků A, B, C, D, za využití operací XOR OR AND a bitových posunů.
6. To, co zbude v blocích A, B, C, D na konci výpočtu je seřazeno (v definovaném0m sledu). tím je součet hotov.

Podrobný popis jde najít třeba na http://en.wikipedia.org/wiki/MD5

2.6.3 Algoritmy rodiny SHA-X

Tyto algoritmy jsou přímými pokračovateli myšlenek MD5. Jsou pouze doplněny o větší počet kol na subblok, delší výsledný součet a složitější funkce na šíření chyby (což je nutnost, po delším součtu se chyba musí šířit déle - tj. ve více kolech ve ví nebo sofistikovaněji tj - vylepšenými funkcemi). V dnešních dnech je na spadnutí prolomení algoritmu SHA-1 (160b) a doporučuje se používat alespoň SHA256. Algoritmus má i mutace SHA384 a SHA512.

2.6.4 Klíčované jednocestné šifrovací systémy (MAC)

Systémy MAC zde uvádím jen tak pro zajímavost a vysvětlení pojmu. Jde o funkce kde výsledek není jen funkcí dat, která chceme otisknout, ale také nějakého tajemství, klíče. Jejich využití jsou nejrůznější autorizační a podpisové aplikace.

2.7 Zmínka o kryptografických protokolech

V předchozích odstavcích jsem se zabýval kryptografickými algoritmy. Ty jsou však sami o sobě k ničemu, dohromady je spolu s nějakým rozumným využitím pojí až nějaký protokol.

Pěkný příklad osvětlení tohoto pojmu je v [1]. Pokud pečete koláč je to sice činnost, ale ne protokol. Je to algoritmus. Pokud se s někým dohodnete, že ho po upečení sní, už máte protokol.

V dnešní době je více útoků směřovaných na protokol, než na algoritmy samotné, chyby v protokolech jsou mnohem častější.

Podrobněji se budeme zabývat později protokolem ssh, který využívá aplikace v druhé části práce.

Kapitola 3

Současné nástroje přenosu dat

3.1 Úvod do ssh

3.1.1 terminologie

Pro ssh jako protokol budu používat označení ssh, pro ssh jakožto konkrétní program (komerční implementaci) budu zapisovat jako SSH.

3.1.2 Základy ssh

Ssh je protokol, který umožňuje (mimo jiné) navázat zabezpečené spojení klienta a serveru přes internet. Je důležité zdůraznit, že umožňuje navazovat spojení, jeho název je tedy mírně matoucí, na shell není přímo vázán, ikdyž je k tomuto účelu využíván asi nejčastěji. K úspěšnému navázání spojení potřebujeme běžící server (sshd) na straně serveru, klienta ssh a klíč nebo heslo k připojení k serveru. Na ssh je zásadní vlastnost, že se povinně prokazuje jak server klientovi, tak klient serveru. Dnes je ssh v několika verzích, podstatné rozdíly reprezentují verze 1 a verze 2.

3.1.3 Autentifikace v ssh

Každý server disponuje dvojicí klíčů. Jeden je generován při instalaci sshd, druhý při startu a poté je v pravidelných časových intervalech obměňován (defaultně jednou za hodinu). Budeme je (po řadě) nazývat klíč stroje a klíč serveru. Pokud se chce klient připojit k serveru, dostane od serveru veřejnou část obou klíčů. Pokud klient klíč stroje zná, je server považován za známý.

Pokud ho nezná, je na uživateli, aby klíč zkontroloval a posoudil, zda se jedná o stroj, na který se chtěl připojit. Pokud klient zná klíč serveru a server se prokazuje jiným klíčem, je uživatel velmi sugestivní hláškou upozorněn. Tato skutečnost znamená buď man-in-the-middle-attack, nebo (a to mnohem častěji) reinstalaci sshd na serveru.

V dalším kroku vygeneruje klient session key, kterým se šifruje další komunikace (symetricky). Tento klíč zašifruje veřejným klíčem serveru a klíčem stroje, a pošle jej serveru. Tím je zajištěno, že server musí znát svůj tajný klíč (a nevydává se tedy za někoho jiného). Tím je dokončena autentizace serveru vůči klientovi.

Nyní je na serveru aby si ověřil identitu klienta. Způsobů je několik, buď přes heslo, nebo přes asymetrickou kryptografii. V druhém případě musí být již na serveru instalován veřejný klíč. Má obvyklé umístění, které se může mírně lišit podle jednotlivých verzí a implementací, ale to je vcelku nepodstatné.

Tímto okamžikem jsou obě strany autentizované sobě navzájem a mohou začít další komunikaci.

3.1.4 Slabiny mechanismů ssh

Ssh je dobrý nástroj, pokud se s ním zachází opatrně. Pokud se připojíme k neznámému stroji, jen málo lidí skutečně kontroluje, že stroj je ten pravý, než potvrdí jeho připojení mezi známé. Dalším nebezpečím jsou pozměněné servery nebo klienti... Pokud používáte k prokázání identity heslo, může se vám snadno stát, že ho pošlete i lidem, kterým jste ho rozhodně posla nechtěli, pokud použijete pozměněného klienta, nebo někdo vhodně „opatchoval" server. Tomu se nejde nikdy zcela vyhnout, a je proto lepší napoprvé dopravit klíč na server třeba osobně na disketě a poté se přihlašovat pomocí asymetrické kryptografie. Ssh dává jistý pocit bezpečí, kterému je třeba nepodlehnout. Dalším problémem spojeným s použitím ssh mohou být „atypické" filesystémy typu AFS, které vydají token až po přihlášení. Potom nastává etida na téma slepice a vejce. sshd se k veřejné části klíče nedostane, protože nejste přihlášen, a přihlášen nebudete, dokud nebude přístup ke klíči, nebo dokud nezadáte heslo. Z takových situací někdy východisko je (pomocí speciálních nastavení filesystému) a někdy není a heslu se vyhnout nejde.

3.1.5 Souvislost ssh, scp a sftp

Jak jsem se snažil zdůraznit v úvodu, ssh není vázán pouze na aplikace typu shell. Může zajišťovat i spojení výrazně jiných aplikací, platí však omezení, že aplikace ke své komunikace musí využívat výhradně protokol TCP. V době vzniku ssh verze 1 nebyl žádný standart na zabezpečený přenos dat. Proto bylo do ssh implementována obdoba programu cp pod názvem scp. Chtěl bych vyvrátit rozšířený omyl. Scp není sám o sobě nijak bezpečný. jeho bezpečnost je plně založena na využití služeb ssh. Protokol sftp, který je až ve verzi ssh 2, nemá nic společného s protokolem FTP (snad krom toho, že také slouží k přenosu souborů). Scp i sftp jsou služby, které by s trochou snahy mohly fungovat samostatně (a nezabezpečeně) a celá jejich bezpečnost je postavena na autentizaci a šifrování ssh.

3.1.6 sftp zblízka

Sftp je je jedno z vylepšení ssh verze 2, a zavádí operace běžné pro soubory v UNIXU, jako je read, write, seek, změna atributů a podobně. Je spouštěn jako tzv. subsystém, což přináší jisté výhody z hlediska administrace uživatel sshd (jde povolit jen někomu) a výhodu v tom, že je součástí jednotného schématu subsystémů. Nevýhodou je větší režie spojená s takovouto samostatností. Podrobněji se tím zabývám v 3.1.8.

3.1.7 Současné implementace SSH

Ssh se stalo za dobu svého působení standartem, který byl portována na takřka všechny rozumné platformy (včetně MS Windows). Existují jak komerční tak nekomerční implementace, jak klientů tak serverů. Za zmínku stojí OpenSSH, které je vhodné ke studiu problematiky, nebo komerční produkt ssh2 pro komerční služby, který je však vyvíjen s otevřeným zdrojovým kódem a je v současné době považován za jednu z nejkvalitnějších implementací.

3.1.8 Verze scp1, scp2, rychlosti při jednotlivých algoritmech

Scp1 a sc2 se li39 ve způsobu své implementace, zatím co sc1 je z hlediska komunikace součástí obslužných procesu sshd pro daného klienta, scp2 je

Tabulka 3.1: Srovnání rychlostí scp1 a scp2

Šifra	Doba scp1(s)	v scp1(KB/s)	Doba scp2(s)	v scp2(KB/s)
RC4	5	1024	22,5	227
Blowfish	6	853	24,5	208
Twofish	14	365	28,2	181
3DES	15	341	51,8	98

vlastně jen jakýsi alias pro protokol sftp. Z toho plyne, že pro přenos souboru přes scp2 je spouštěn takzvaný subsystém sftp. Aniž bych chtěl nějak zacházet do detailů, tak chci jen zdůraznit, že ačkoli se na uživatele obě verze „usmívají“ stejně, je v nich dosti značný rozdíl. V tabulce 3.1 je porovnání doby přenosu 5MB souboru a spočtené průchodnosti, převzaté z [2], která udává čas potřebný na přenos 5MB dat z linuxového počítače s 300MHz procesorem na počítač Sparc-20 se 100 MHz procesorem po jinak nezatížené 10BASE-T lince.

Další rozdíl, který stojí za zmínku (pro programátora určitě) je fakt, že scp1 je nedokumentované, zatímco k sftp poměrně rozumná dokumentace existuje.

3.2 SSL

Secure Socket Layer původně vznikl jako rozšíření webových klientů o autentizaci a šifrování. Jeho rozšíření je v jistém smyslu podobné „bezpečným soketů ve stylu Berkeley“. Nic z něj ovšem není vázáno přímo na http, je možné ho využít jako součást jiných protokolů a aplikací. Za pomoci SSL bylo „vylepšeno“ mnoho aplikací, počínaje telnetem a konče ftp, tak, že jejich komunikace přes protokol TCP je šifrovaná. Je ovšem třeba mít stále na paměti, že tyto aplikace nebyly od počátku psány s ohledem na bezpečnost vyvodit si z toho potřebné důsledky.

3.2.1 Autentifikace v SSL

Identita může být v SSL dokazována volitelně oboustraně nebo jednostraně na základě certifikátů. Certifikát si jde představit jako nějaká data, která udávají vazbu mezi autoritou (někdo třetí, kterému důvěřují obě strany) a klíčem, za pomocí kterého je identita prokazována.

3.2.2 Výhody a nevýhody SSL

Nespornou výhodou pro programátora je, že je v současné době dostupná dobrá implementace SSL na úrovni zdrojových kódů. Nevýhodou je, že SSL neimplementuje digitální podpis, který je často potřebný.

Výhodou i nevýhodou v jednom je systém autentizace. Tato metoda odstraňuje potřebu prvotní instalace klíč ne na server, o které jsem mluvil v části o ssh. Ovšem tato metoda přináší i jisté nevýhody, a to, že uživatel musí tušit, které certifikáty jsou důvěryhodné, a případně je důvěryhodnou cestou zajistit. Kontrolní otázka na závěr: Kolik certifikátů je přednastaveno ve Vašem prohlížeči? Znáte všechny, o kterých se Váš výrobce browseru rozhodl za Vás, že jsou důvěryhodní? Já jich mám v Mozzile něco okolo 80, a znám z nich asi 5…

3.3 Ostaní nástroje

Mezi další nástroje a příbuzné technologie patří např. PGP protokol IPSEC, Kerberos, SRP a další. Ty se však svým zaměřením nevztahují přímo na přenos datových souborů. Jiné standardní nástroje, které nejsou postavené na ssh nebo ssl se vyskytují jen málo, a většinou jde o komerční systémy, které nemají tendence stávat se veřejným standardem.

Kapitola 4

Moje aplikace

V této části se budeme pokoušet aplikovat teoretický podklad na aplikaci používající zabezpečený přenos dat a jejich uložení na serveru.

4.1 Platforma

Aplikace je psána (jak serverová tak klientská část) pro 32-bitový operační systém linux provozovaný na PC. (Little endian). Podstatou je aplikace přenositelná na libovolný UNIXový systém, kryptografické moduly však budou fungovat správně jen na architektuře little endian.

4.2 Terminologie

Pro označení klientské části aplikace používám označení klient, pro označení serverové části server. Název server někdy použit i pro stroj na kterém běží serverová část aplikace, doufám, že tím nedojde k žádným nejasnostem.

4.3 Motivace vzniku aplikace

Internet je dnes používán jako poměrně univerzální prostředek pro sdílení dat. Jeho nevýhodou však je, že se často vystavujeme situaci, kdy data oproti původnímu záměru sdílíme nevědomě či neúmyslně s lidmi, se kterými je naprosto sdílet nehodláme. Máme potom možnost každý datový soubor zvlášť šifrovat, pak vyvstává problém v značné nepohodlnosti a pokud dlouho používáme jeden klíč, vystavujeme se riziku statistických útoků.

Z tohoto důvodu jsem si jako projekt vybral aplikaci, by měla napomoct řešit tyto problémy lidem, kteří mají účet někde na na linuxovém stroji, ale nemají možnost si na něm spouštět svoje persistentní servery a jinak zasahovat do jeho konfigurace. Modelovou situací může být chvíle, kdy si z domu potřebujeme někde zazálohovat sadu klíčů k bankovním účtům. Přenášet je nešifrovaně je nebezpečné, šifrovat a dešifrovat je jednom je nepohodlné.

4.4 Zaměření aplikace, specifikace

4.4.1 Cílová skupina uživatelů

Aplikace je určena jedincům, nebo malým skupinám lidí stejného nebo podobného zájmu, které společně sdílejí nějaká data malého objemu. Aplikace by měla umožnit sdílení přes účet jednoho z uživatelů na veřejně přístupném serveru, kde není počítáno s podporou správce. Pro jedince může mít význam například pro zálohování z práce domů a podobně. Dodatek, že se nepočítá s podporou správce znamená, že na serveru není možné trvale provozovat persistentní sever, není možné konfigurovat firewall nebo měnit jiné systémové parametry. Jako šikovný model poslouží linuxové stroje v laboratoři na Malé Straně.

4.4.2 Schéma ochrany dat

Co je na aplikaci zajímavé, je to, že aplikace chrání data na serveru, před ostatními uživateli a před případnými chybami v administraci. To znamená, že ani superuživateli nejsou data, která jsou na serveru k ničemu, a pokud data naruší, tak to uživatel aplikace pozná.

4.4.3 Synchronizace činnosti uživatelů

Navíc pokud bude zároveň pracovat v systému několik lidí, nebudou o sobě vědět. Tím myslím, že pokud si jeden začne obnovovat zálohu, dokončí operaci, a nebude ovlivněn například tím, že někdo v průběhu jeho práce nahrál na server nějaký soubor v jiné verzi. Motivace je například situace, kdy je na serveru soubor a jeho otisk v jiném souboru. Pak absence synchronizace může mít značně neblahé následky. Toto opatření nutí ke spuštění vlastního serverové aplikace, která bude koordinovat činnost uživatelů.

4.4.4 Další požadavky na aplikaci

Aplikace by měla umožňovat snadné doplnění nebo nahrazení algoritmů použitých pro šifrování a hashování. Uživateli by měla být aplikace blízká, a pokud možno intuitivní, měla by připomínat práci například v řádkovém klientu ftp. Do provedení operace *commit* by mělo být možné provedené změny odvolat (*rollback*).

4.5 Nástroje a knihovny potřebné k překladu aplikace

Potřebné nástroje jsou překladač jazyka C a C++ (gcc, g++), utilita replace, Bison, Flex. Potřebné jsou knihovny nsl, z, resolv, crypto(součást OpenSSL). Tyto knihovny jsou využity k překladu libssh.

Takřka celá aplikace je napsána v jazyce C, jen některé části využívají C++, z důvodu příjemného použití STL a generických algoritmů. Knihovna OpenSSL je použita pro překlad knihovny libssh. Tato knihovna, je integrována z pohledu uživatele přímo ve zdrojových kódech aplikace.

Bison a flex jsou nutné k vytvoření parseru indexového souboru (viz dále) a parseru konfiguračního souboru.

Utilita replace je potřebná k upravě zdrojových textů generovaných flexem a bisonem. V manuálu programu Bison je pasáž, která popisuje jak vytvořit program s více parsery bez kolizí jmen funkcí. Bohužel však praktické zkušenosti ukazují, že takto fungují jen některé verze (a některá sestavení) Bisonu. Z toho důvodu jsem zvolil „ruční“ přejmenování kolidujících jmen na jiná. (

```
replace yy yyy --neco.tab.c
```

). Tato metoda byla s úspěchem odzkoušena na Suse 9.3 a na Gentoo na Malé straně. „Oficiální“ metoda popsaná v manuálu fungovala pouze na Malé Straně.

4.6 Vybaveni požadované pro chod aplikace

Toto řešení bude jistě potřebovat jistou standardní koncovku, neboli nějakou službu, která běží na straně serverového počítače a zajistí nám prvotní

vstup na stroj, kde běží server. Takovou koncovku jsem zvolil službu ssh, koneckonců, tato má být aplikace zabývající se bezpečností.

4.6.1 Proč SSH

Ssh je standardní a běží dnes na mnoho počítačích unixového typu. Je portována už i na OS Windows a toto řešení je tedy vpodstatě poměrně dobře přenositelné, v případe potřeby. Krom toho, ssh má ji zabudovánu autentizaci obou stran, kterou tak můžeme využít, a nemusíme jí tak sami implementovat.

4.7 Principiální fungování aplikace

Aplikace funguje schematicky následovně:

1. Klient aplikace je spuštěn, jsou načteny knihovny pro šifrování, alespoň základní, tedy v našem případě Blowfish
2. Klient otevře spojení přes ssh na server a spustí serverovou část aplikace
3. Serverová část aplikace vypíšejako reakci (buď po startu, nebo po několikátém pokusu o nastartování) port na kterém běží a token (port na kterém běží není předem znám, nikdo nemůže vědět, které porty budou zabrány)
4. Klient se s tokenem připojí po komunikačním soketu k serveru a prokáže se tokenem. toto spojení je zabezpečeno pomocí tzv tunelování protokolem ssh. Tento soket bude sloužit prápráve pro koordinaci činnosti klientů. Konkrétně klient bude posílat serveru akce, které server provede, když to nebude narušovat činnost ostatních uživatelů)
5. Klient si stáhne a rozšifruje indexový soubor (popis adresářové struktury odzálohované na serveru)
6. Uživatel zálohuje, nebo stahuje zálohy.
7. Pokud klient něco nahrál na server, odešle před odpojením na server indexový soubor, nebo změny odvolá

8. Uživatel se odhlásí. V této chvíli server zváží (podle stavu ostatních klientů) jestli se může nebo nemůže smazat neaktuální soubory (ty, které dal uživatel smazat).

4.8 Aplikace z pohledu uživatele

4.8.1 Popis zprovoznění aplikace

Zprovoznění aplikace se skládá ze tří kroků:

1. Aplikaci je nejprve nutno zkompilovat.
2. Druhým krokem je nahrání na server. Na serveru vytvořte ve svém domovském adresáři podadresář, a nakopírujte do něj ze složky server aplikaci server a podadresáře var a data včetně jejich obsahu.
3. Změňte konfigurační soubor (viz 4.8.2).
4. Spusťte aplikaci klient umístěnou v adresáři client.

4.8.2 Konfigurační soubor

Konfigurační soubor pomáhá nastavit základní chování programu a zejména adresáře ve kterých program hledá data a moduly. Volby v souboru jsou buď číselné, nebo se jedná o řetězce, které jsou uzavřeny do uvozovek. Příklad:

```
INDEX_NAME = "index.xml"
KEEP_KEY  = 0
```

Konfigurační soubor se musí nacházet pod názvem *config* ve stejném adresáři jako klient. Možnosti konfigurace popisuje tabulka 4.1

Možná by stála za vysvětlení poslední volba. Data jsou na serveru ukládána pod náhodnými názvy. Může proto docházet ke kolizím (Ovšem pravděpodobnost kolize je vzhledem k délce jména velice nízká, a pokud ke kolizím

Tabulka 4.1: Volby konfiguračního souboru

Volba	Význam	Možné hodnoty
SERVER_DIR	Umístění aplikace serveru na serverové straně relativně k $HOME	string
MODULE_DIR	Relativní nebo absolutní cestě k adresáři s kryptovacímy moduly	string
DEFAULT_ENCODE_ALGO	Defaultně modul pro šifrování	string
DEFAULT_HASH_ALGO	Defaultně modul pro hašování	string
INDEX_ENCODE_ALGO	Modul pro šifrování indexového souboru	string
INDEX_HASH_ALGO	Modul pro hašování indexového souboru	string
INDEX_NAME	Jméno indexového souboru	string
KEEP_KEY	Určuje, zda má klient držet ve vnitřních strukturách zvolený klíč	{0,1}
VERBOSE	Určuje množství výpisů klienta	{0,1}
SERVER_START_WAIT_TIME	Doba čekání na odezvu serveru při startu	integer
NUMBER_SAVE_ATTEMPTS	Množství pokusů o uložení souboru na server	integer

přesto dochází, značí to spíš špatnou funkci generátoru náhodných jmen). Data jsou tedy ukládána s volbou O_EXCL, a pokud dojde ke kolizi, je proveden další pokus o uložení pod jiným jménem. Počet těchto pokusů je dán tímto parametrem. Rozumná hodnota je podle mého mínění asi 5. Pokud i přesto dochází k chybám, znamená to spíš špatnou funkci zařízení /dev/urandom pomocí kterého jsou generována náhodná jména.

4.8.3 Start aplikace

Aplikace očekává jako svůj parametr řetězec ve tvaru [*user*@]*host*, jehož význam je obdobný jako u běžných klientů ssh. Ostatní volby jsou předány programu v konfiguračním souboru. Osobně tuto cestu považuji vzhledem k povaze aplikace za příjemnější pro uživatele.

Po startu proběhne autorizace podobně jako u běžných klientů ssh. Pozor na to, že aplikace podporuje správně jen klíče RSA. Při použití dsa klíče dojde k chybě, a jste vyzváni k autorizaci heslem. Tento nedostatek pramení z knihovny libssh, a jejích příštích verzích by měl být napraven. Klíče jsou hledány na standardních místech, tedy v adresáři $HOME/.ssh/. Po autorizaci proběhhne startovní dialog klienta a serveru. Pokud proběhne v pořádku, jste vyzváni k zadání klíče k indexovému souboru. Pokud je tento rozšifrován správně, aplikace vypíše prompt a je připravena k další činnosti.

Při prvních spouštěních doporučuji použít volbu verbose. Pomůže odhalit možné konfigurační problémy, které by mohly po instalaci nastat.

4.8.4 Uživatelské rozhraní

Ovládání klienta pracuje je přibližně následující: Klient je buď ve stavu, kdy očekává příkaz, nebo stavu, kdy provádí příkaz. Pokud očekává příkaz je vypsán prompt popisující pozici v systému souborů uložených na serveru. Příkazy poněkud neuměle vystihuje obrázek 4.1. Příkazy se dají logicky rozdělit do 3 skupin.

příkazy lokální Všechny jsou uvozeny znakem '!'. Jsou to příkazy, které neprovádí přímo klient, ale jsou provedeny shellem, pomocí volání system(). Výjimkou je příkaz !cd, který mění pracovní adresář klienta.

příkazy ssh Jsou uvozeny řetězcem "ssh". Celý zbytek řádku je předán shellu na serveru ke zpracování.

příkazy pro klienta Jsou příkazy, pomocí kterých je ovládán klient a jsou pomocí nich měněny vnitřní struktury, popisující strukturu souboru na serveru. Příkazy jsou navrženy tak, aby uživateli v jistém smyslu připomínali příkazy ftp. Jejich význam je shrnut v obrázku 4.1. Podrobnější popis si snad zaslouží příkazy put a icheck.

put *l*name [:sname] [algo hash] Je příkaz na nadefinování uploadu souborů. Soubor *lname* je nahrán [pod jménem *sname*] [s použitím algoritmů šifrování *algo* a hashování *hash*]. Pokud nejsou nepovinné parametry zadány, použijí se defaultní algoritmy a jméno stelné jako je lokální. Algoritmy musí být dostupné v adresáři, který je uveden v konfiguračním souboru jako adresář s moduly.

icheck Je test konzistence indexového souboru a dat na serveru. Nekonzistence může vzniknout násilným ukončením serveru. Pokud se jeví data na serveru jako nekonzistentní, nemusí to být chyba, může to být dáno neaktuálností indexového souboru klienta, který icheck provádí vzhledem k práci dalších klientů.

4.8.5 Výpisy klienta

Množství výpisů klienta je možné ovlivnit parametrem verbose v konfiguračním souboru. Výpisy jsou uvozeny řetězcem, který uživateli říká, za jaké situace byl výpis pořízen

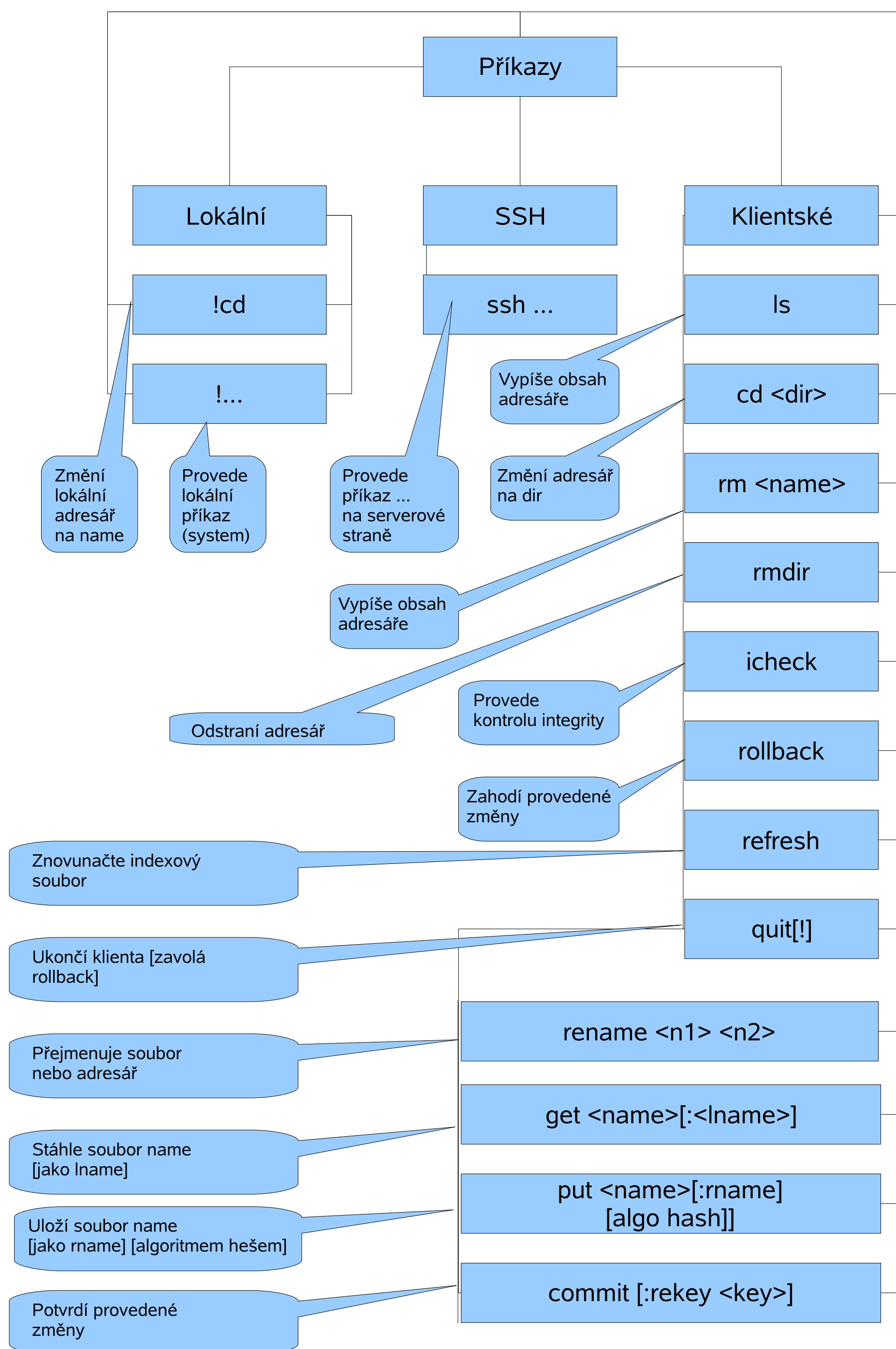
CLIENT-STD je standardní výpis klienta.

CLIENT-ERR je chybový výpis klienta.

SSH-STD-MESS je neočekávaný výpis nějakého programu na stdout terminálu na servrové straně ssh spojení

SSH-ERR-MESS je neočekávaný výpis nějakého programu na stderr terminálu na servrové straně ssh spojení

Client může vypsat ještě CLIENT=SOC což by znamenalo neočekávaný výpis do řídícího soketu mezi klientem a jeho vláknem serveru, ale k této události by nikdy nemělo dojít (je to pouze ošetření chyby komunikace).



Obrázek 4.1: Ovládání klienta

4.8.6 Příklad session

Pro bližší představu přidávám příklad jedné jednoduché session (se zapnutou volbou verbose = 1 a keep_key = 1).

```
tom@SNEK:~/sfsprojekt/client> ./client localhost
CLIENT-STD> Encoder function will be loaded from : /home/tom/sfsprojekt/client/
CLIENT-STD> Basic crypto functions loaded OK.
CLIENT-STD> Waiting up to 10 seconnds for server message...
CLIENT-STD> Server start OK .
SERVER-STD> Get message server is running on port number: 1026
CLIENT-STD> Trying to open channel to: localhost:1026
CLIENT-STD> Channel opened OK.
SERVER-STD> Get message server has pid: 7648
SERVER-STD> Get message server's token is: VlkB3eiaENHSDXBk7BHPoqtEETRmul9ep
CLIENT-STD> Server-subproces start (client id is: 1)... OK
CLIENT-STD> Checking for server protokol version. Suported(by client) SFS-1.0.0
CLIENT-STD> Protokol verison OK...
CLIENT-STD> Server-subproces want token...
CLIENT-STD> Authentification with token OK
CLIENT-STD> Going to download index file...
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Index file key:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CLIENT-STD> index file decode status... OK
CLIENT-STD> index file parse status... OK.
=============================================
NameOfParentDirectory/>ls -l
d xxx
d some_addr
NameOfParentDirectory/>put Makefile:makefile
CLIENT-STD> file Makefile uploaded succesfully (as makefile)
NameOfParentDirectory/>ls -l
d xxx
d some_addr
f makefile
NameOfParentDirectory/>cd xxx
NameOfParentDirectory/xxx/>ls -l
f xxxxx
```

```
NameOfParentDirectory/xxx/>rm xxxxx
CLIENT-STD> File removed.
NameOfParentDirectory/xxx/>icheck
NameOfParentDirectory/xxx/>cd ..
NameOfParentDirectory/>ls -l
d xxx
d some_addr
f makefile
NameOfParentDirectory/>!cd tmp
NameOfParentDirectory/>commit
CLIENT-STD> Index file written. OK
NameOfParentDirectory/>get makefile:Makefile_lokal
CLIENT-STD>> file makefile succesfuly downloaded as Makefile_lokal.
NameOfParentDirectory/>!ls
index.xml  Makefile_lokal  xxxx
NameOfParentDirectory/>ls -l
d xxx
d some_addr
f makefile
NameOfParentDirectory/>rm makefile
CLIENT-STD> File removed.
NameOfParentDirectory/>rollback
CLIENT-STD> Rollbacked succesfully.
NameOfParentDirectory/>ls -l
d xxx
d some_addr
f makefile
NameOfParentDirectory/>quit
CLIENT-STD> xentering cleanup function...
```

4.8.7 Licence

V projektu byly použity části pod licencí GPL (např. libssh) a tedy celá aplikace spadá pod licenci GPL.

4.9 Technická řešení

Pro celou tuto sekci platí, že je spíše zdůvodněním, proč jsem se kde rozhodl tak a ne jinak. Kompletní dokumentace toto není, pokud někoho zajímá konkrétní řešení dopodrobna, odkazuji na komentáře ve zdrojových kódech aplikace, zejména v hlavičkových souborech a dokumentaci generovanou programem Doxygen.

4.9.1 Spuštění serveru a schéma propojení s ssh

Vše podstatné je zachyceno na obrázku 4.2. Klient se připojí přes ssh na serverový stroj. Spustí si shell a příkazem spustí server. Ten vypíše port na které běží a token, kterým se klient bude prokazovat podřízenému vláknu. Je zajištěno, že server vždy poběží jen v jedné instanci. Klient se poté připojí přes tunel budovaný pomocí ssh k serveru, pomocí tokenu ověří svou identitu. Tím je zaregistrován v systému a může pracovat. Pro každého klient sshd vytváří zvláštní proces sftp-serveru.

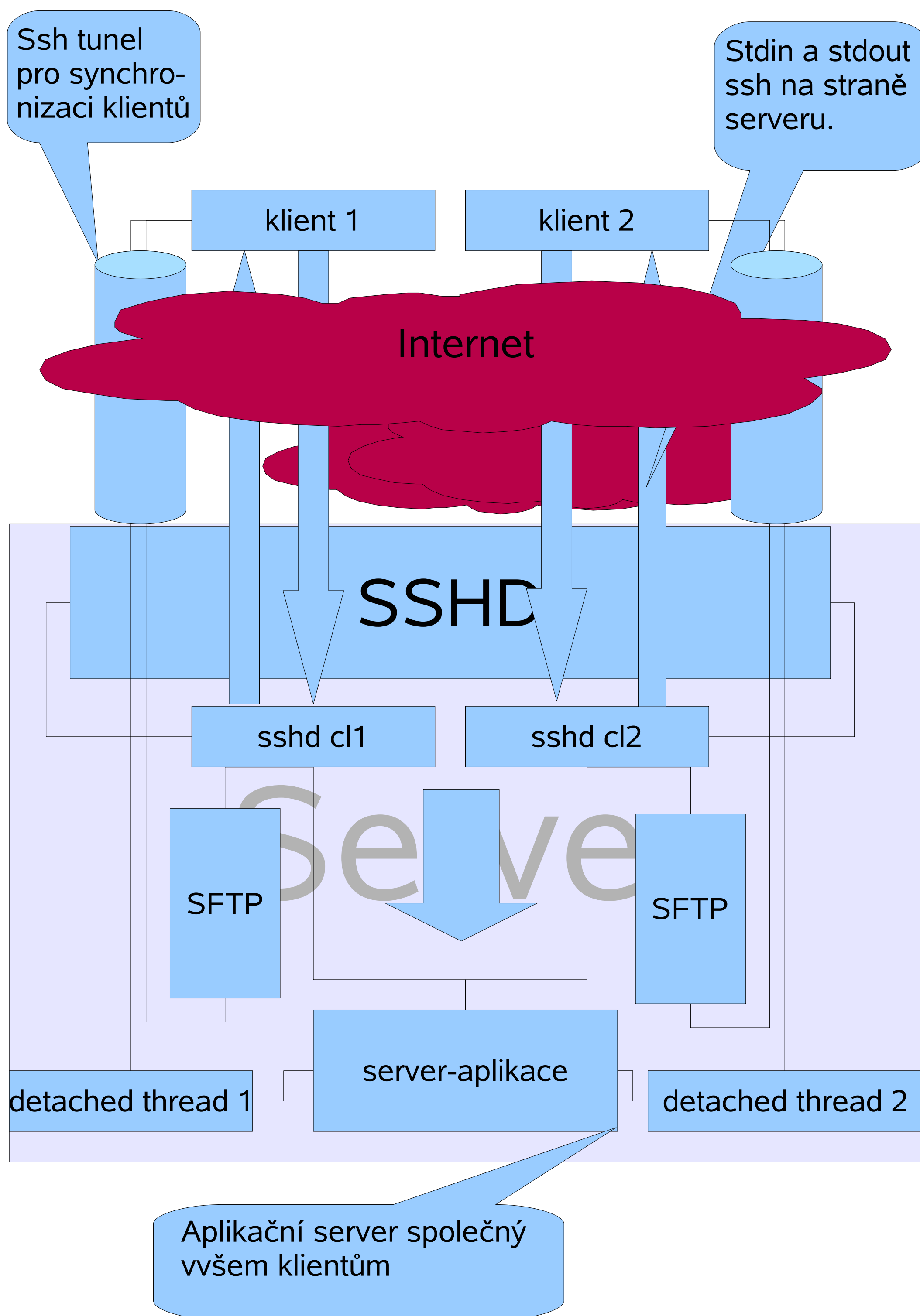
4.9.2 Rozdělení práce na projektu

Projekt během doby své realizace přerostl velikost, kterou jsme původně plánoval. Zdrojové kódy jsou proto rozděleny do adresářů podle logické struktury projektu. Každý adresář obsahuje Makefile, a zpravidla je zakončen jednoduchou aplikací která testuje funkčnost konkrétního dílu celku.

4.9.3 Protokol komunikace mezi klientem a serverem

Klient a server si mezi sebou vyměňují zprávy ve tvaru *číslo (jako řetězec) číslo (jako řetězec) řetězec*. Jejich význam je: *příkaz, hodnota, komentář* , ikdyž komentář je často využíván i k přenosu názvů a jako komentář neslouží. Pokud někoho zajímají detaily, bude pro něj zajímavý soubor /inlcude/client/commands.h, kde jsou definovány jednotlivé příkazy protokolu.

Pomocí zpráv se klient nejprve zeptá na verzi serveru. Pokud odpovídá, prokáže se tokenem. V průběhu komunikace slouží protokol k synchronizaci klientů serverem. Klient se ptá, zda může zapisovat nebo mazat, server referuje aktuální stav (zda někdo jiný zapisuje, zda má klient aktuální index...) Protokol a jeho příkazy jde podrobně pochopit ze souboru /include/commands.h, kde jsou definovány pomocí maker jejich hodnoty.



Obrázek 4.2: Propojení aplikace a ssh

4.9.4 Popis adresářové struktury a schéma uložení dat na serveru

Pro pohodlnou práci uživatele se aplikace jeví klientovi, jako by procházel adresářová strukturou. Na serveru jsou ovšem všechna data ukládána do jednoho adresáře (Není žádoucí ukazovat někomu neoprávněnému jak jsou data rozložena v adresářích nebo snad dokonce jaka mají jména), je proto třeba utvořit nějakou vazbu. Touto vazbou je indexový soubor. Jelikož adresářový prostor je rekurzivní struktura, je popsán rekursivním nástrojem – jistou podmnožinou xml tagů, které mohou být libovolně vnořeny. Použitá gramatika byla popsána a je zpracovávána pomocí standardního nástroje Bison, lexikální analýza probíhá pomocí nástroje Flex. Toto řešení dává, jak jen je to možné, možnost snadno a rychle a modifikovat schéma uložení adresářové struktury. V paměti aplikace je souborový systém uložen pomocí struktur fsdir a fsfile. V každé struktuře fsdir je spojový seznam struktur fsfile a fsdir. Pokud by se aplikace měla jednou rozrůst a měla by sloužit k uložení většího množství dat, bylo by třeba implementovat tyto množiny ne seznamem, ale třeba pomocí b-stromů.

Na serveru jsou data uložena pod „náhodným názvem“, všechny soubory v jednom adresáři. Náhodný název je získáván pomocí zařízení /dev/urandom.

4.9.5 Model serveru

Server má poměrně klasický model, tedy jeden proces je po celou dobu běhu serveru ve smyčce a očekává nové klienty. Pro každého klienta je vytvořeno obslužné vlákno (typu detached, neví se kdy skončí). Synchronizace vláken je realizována pomocí mutexů. Server si po dobu svého běhu udržuje počet klientů. Pokud je server startován poprvé (žádná jiná instance spuštěná ze stejného souboru neexistuje), vypíše server na stdout token, a port který mu přidělil systém a začne přijímat klienty. Pokud je startován po několikáté, server pouze vypíše port a token a tato (v systému druhá) instance serveru se ukončí.

4.9.6 Získávání dat ze serveru

Samo se nabízí řešení udělat si vlastní protokol, protože na serverové straně nám stejně běží serverová část aplikace. Druhou možností je získávat data pomocí sftp, tedy součásti integrované přímo v protokolu ssh. Zvolil jsem druhou možnost a to zejména ze tří důvodů. Jednak bych dělal něco, co je

už hotové, druhak pomocí sftp se vyhnu problémům s firewally na serveru (nemám podporu administrátora!). Třetí důvod je, že se o případném útoku na ssh přenos dozvím, už v dobe nahrávání nebo stahování, a ne až při jejich dešifrování. Nevyhnutelnou cenou za tyto výhody je, že data budou šifrována dvakrát. Je ovšem třeba poznamenat, že aplikace je navržena tak, aby přenos dat nebyl pevně vázán na sftp, a pokud by někdo považoval tuto skutečnost za zcela fatální chybu, bylo by s přiměřenou námahou aplikaci upravit tak, aby používala k přenosu dat např. protokol ftp nebo http.

4.9.7 Kryptovcí jádro aplikace

Jádro aplikace je tvořeno souborem funkcí v souboru encoder.c. Objekt je navržen tak, aby byla co nejobecnější a aby pomocí rozhraní k modulům s algoritmy bylo možné využít co nejširší třídu algoritmů. Funguje asi následujícím způsobem

1. Nejprve jsou načteny šifrovací funkce z knihoven realizovaných v modulech, které mohou být dynamicky loadovány. Knihovny jsou načteny do struktur, které obsahují vždy sadu pointerů na funkce potřebné k šifrování a hashování.

2. Potom je otevřen soubor. Soubor je za běhu šifrován, tak dlouho, dokud nedojdeme na konec souboru, tj, dokud nezbývá v souboru méně, než délka jednoho bloku dat.

3. Tento blok je doplněn vycpávkou a zašifrován

4. Na konec souboru je umístěn poslední blok, ve kterém je hash souboru, délka posledního bloku a vycpávka. (Délka souboru je tedy vždy alespoň dva bloky)

5. Dešifrování probíhá obdobně.

Každý modul pro šifrování musí poskytovat alespoň čtyři funkce:

- inicializační void * enc_init(unsigned const char *key)
- šifrovací int enc_transform_enc(void * ctx, unsigned char * data, unsigned int length)
- dešifrovací int enc_transform_dec(void *ctx, unsigned char * data, unsigned int length)

- úklid pomocných struktur void enc_free(void *ctx)

Obdobně každý modul pro hašovánímusí poskytovat alespoň čtyři funkce:

- inicializační void * hash_init()
- update hash_update(void * ctx, unsigned char *data, unsigned int length)
- finalizační int hash_final(void *ctx, unsigned char * dig, int *hl)
- uklid pomocných struktur void hash_free(void * ctx)

Třída algoritmů implementovatelných s tímto rozhraním

Tento návrh modulů umožňuje jejich pomocí implementovat libovolnou šifru, která nemění objem dat během transformace. Z hashovacích algoritmů jistě umožňuje funkce MD5 a celou rodinu funkcí SHA.

Objekt je završen komfortním rozhraním, tedy funkcemi, které podle daného klíče přímo šifrují daný stream (čtou z jednoho souboru nebo soketu a do druhého zapisují). Pro účely naší aplikace je vytvořeno ještě jedno rozhraní, a to sice funkce, které dostanou pointer na funkci, pomocí které si mají získávat, nebo odevzdávat data. (tedy by funkce sftp_put šla snadno nahradit např funkcí ftp_put). Interface této funkce je navržen také velmi obecně. V příkladu je použita funkce xread, v aplikace je využito funkcí sftp sftp_read, a sftp_write.

4.9.8 Synchronizace práce uživatel

Aplikace je navržena tak, aby umožnila součastnou práci několika uživatel. O správu přihlášených klientů se stará objekt, jehož kód je jako jeden z mála v této aplikaci napsán v C++, jeho zdrojový kód můžete naleznout v adresáři todolist pod názvem todol.cpp. Synchronizace vpodstatě zajišťuje následující

- v každém okamžiku je nejvýše jeden uživatel v systému, který zapisuje(tj. buď odstraňuje soubory, nebo je přidává).
- každý uživatel, který započne svoji práci s indexovým souborem, má zajištěno, že všechny soubory budou dostupné až do konce jeho práce, bez ohledu na to, že už nějaký jiný uživatel dal pokyn k jejich odstranění.

- každý, kdo chce začít zapisovat, musí mít aktuální verzi indexového souboru.

- soubor je odstraněn neprodleně po skončení činnosti uživatelů, kteří měli verzi indexu, který se na daný soubor odkazoval.

Realizace synchronizace

K realizaci jsem využil kontejnerů STL. Pokud klient chce provést například vložení souboru, pošle dotaz serveru. Ten se pokusí vložit akci, a pokud to jde, tj. pokud jsou splněny všechny požadavky uvedené výše, odešle klientovi kladnou odpověď. V opačném případě dostane klient zprávu, že buď nemá aktuální index (od té doby, co se přihlásil do systému někdo zapisoval), nebo že zapisuje někdo jiný. Při přihlášení k příslušnému vláknu serveru je klientovi přiřazeno číslo. Klient se pod tímto číslem zaregistruje do map-u. Během jeho činnosti jsou zaznamenávány jeho akce. Akce jsou realizovány jako struktury ACTION , které popisují co se má provést, s čím se to má provést a kdy se to má provést. Jejich definice je následující:

```
typedef struct ACTION_ {
int when;
int what;
char from[FNAME_LENGTH+1];
} ACTION;
```

- při přihlášení je vložena akce ON_INCOME READ

- pokud uživatel smaže nějaký soubor, je vložena akce ON_COMMIT REMOVE.

- pokud uživatel nahraje nějaký soubor je vložena akce ON_ROLLBACK REMOVE

- pokud uživatel přejmenuje nějaký soubor v indexu je vložena akce ON_COMMIT RENAME

Pokud klient provede commit nebo rollback, je z mapu vybrána jeho fronta akcí, tyto akce jsou provedeny (má smysl z nich provést pouze REMOVE, ostatní jsou jen pro účely synchronizace). Klient je vyjmut a je vložen zpět pod novým číslem. Tato synchronizace zajišťuje, že pokud nedojde k násilnému ukončení činnosti klienta nebo k chybě v síťové komunikaci, bude

zajištěna konzistence indexového souboru a dat na serveru. Případná nekonzistence může nastat při násilném ukončení činnosti serveru. V tom případě ovšem budou dostupné všechny soubory z indexového souboru. Pro ilustraci je zde tabulka 4.2, která popisuje (zestručněně) souběžnou práci dvou klientů a odpovídající části ladících výpisů serveru:

4.10 Zhodnocení aplikace

Aplikace je v současné době plně použitelná podle specifikace. Projekt ověřil možnost realizace podobného systému. Zdrojové kódy, které jsou přílohou této práce by měly být návodem a inspirací pro případné zájemce o vytvoření podobných aplikací na zabezpečené zálohování a sdílení dat.

Aplikace by v příštích verzích šla rozšířit o grafické uživatelské rozhraní, bylo by dobré implementovat další kryptografické algoritmy, nejlépe za použití standardních knihoven jako jsou OpenSSL nebo libcrypto. V současné verzi se token jednou vygeneruje při startu serveru a potom se již nemění. Toto neumožňuje nikoho „vyloučit z přepravy“. Pokud aplikace bude sloužit jen malým skupinám, lidí, kteří se znají, tak toto není překážkou. při masovějším nasazení by se tento nedostatek musel odstranit. (Například generováním nového tokenu pro každého nového klienta).

4.10.1 Možné útoky na aplikaci

Aplikace je založená na autentizaci pomocí ssh, a v poslední době se s útoky na konkrétní implementace ssh doslova a do písmene roztrhl pytel. Další možností útoku pokus o dešifrování indexového souboru, který by útočníkovi dal prakticky naprostou vládu nad daty. Taková možnost by znamenala prolomení šifrovacího algoritmu. V této souvislosti je nepříjemné, že indexový soubor má pevnou strukturu (začíná jistou množinou pevně daných tagů), což by za jistých okolností mohlo útočníkovi zjednodušit situaci (viz 1.6). Další nepříjemností je, že knihovna libssh ukládá data na serveru přes ssh s právy

```
-rw-r--r--
```

Tabulka 4.2: Součinnost klientů

Server	Klient 1	Klient 2
1: action :[ON_INCOME READ] Added new user id = 1	RootDir/>icheck	
2: action :[ON_INCOME READ] Added new userid = 2		RootDir/>icheck
1: action :[ON_ROLLBACK REMOVE 2VtKX5uQD7Wjrdo5lKD5AchfIfUxblsW]	RootDir/>put Makefile CLIENT-STD> file Makefile uploaded succesfully (as Makefile)	
2: action :[ON_ROLLBACK REMOVE 8saoLQtQBPhZEAXtyYxGNSlYe8hDB2WQ] this action can not be added. Adder id is 2 writer_id is 1		RootDir/>put Makefile CLIENT-ERR> You can not write file, refused by server(somebody else is writing.)
Commit actions done	RootDir>commit ~~~~~~~~~~~~~~~~~~~~~~~~~ Enter key for index file: Enter the confiramtion: ~~~~~~~~~~~~~~~~~~~~~~~~~ Index file written. OK	
client: 2 adding action given action :[ON_ROLLBACK REMOVE OmqMAakOmQxvgeG4PnBiM8etZEJHnBfp] this action can not be added. User has not actual index file Logout: Starting logout for client nr: 2 Logout: Checking for zombie... Logout: Succesfully after logout. client: 4 adding action given action :[ON_INCOME READ]		RootDir/>put Makefile:makefile >Your index is not actual. Try "refresh" command. NameOfParentDirectory/>refresh Going to download index file... ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ >Enter key for index file: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ > index file decode status... OK > index file parse status... OK. ===============================
client: 4 adding action given action :[ON_ROLLBACK REMOVE NpQZ3sNtBSr2PsYr3ZMHAx9vs7mknsp3]		RootDir/> put Makefile:xxx > file Makefile uploaded succesfully (as xxx)

. a funkce pro práci s atributy sftp souborů ještě nejsou připravené. (Protol ssh-sftp atributy podporuje).

Kapitola 5

Závěr

Tato práce jistě není (a ani neměla být) úplnou dostačující pomůckou programátora, který se zabývá zabezpečeným přenosem dat, ale podle mého názoru se podařilo vytvořit pomůcku, která může pomoci začít studovat tuto oblast programování. Aplikace, kterou jsem předložil a která měla původně sloužit jen jako jakási ukázka se během realizace rozrostla a je v současné době připravena k použití, které v dnešní době, kdy má v soukromých datech snahu něco zajímavého najít doslova kde kdo, možná velmi brzy přijde.

Kapitola 6

Přílohy

6.1 Zdrojové kódy aplikace

Zdrojové kódy aplikace jsou na přiložené na CD.

Literatura

[1] Derbes D.: *Applied Cryptography*, John Wiley & Sons, Inc. , 1996

[2] Daniel J. Barret & Richard E Silverman: *SSH: Kompletní průvodce*, O'Reilly, 2003.

[3] Libor Dostálek a kolektiv: *Velký průvodce protokoly TCP/IP: Bezpečnost*, Computer Press, a. s. 2003

[4] Vylém vychodil: *Algoritmus RSA*, Elektronický studijní materiál.