Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Automatická kryptoanalýza súčtu dvoch otvorených textov

(Bakalárska práca)

Tomáš Žubrietovský

Vedúci: doc. RNDr. Martin Stanek PhD.

Bratislava, 2008

Automatická kryptoanalýza súčtu dvoch otvorených textov

BAKALÁRSKA PRÁCA

Tomáš Žubrietovský

UNIVERZITA KOMENSKÉHO, BRATISLAVA
FAKULTA MATEMATIKY, FYZIKY a INFORMATIKY
KATEDRA INFORMATIKY

Študijný odbor: 9.2.1. Informatika

Vedúci záverečnej práce
doc. RNDr. Martin Stanek PhD.

BRATISLAVA, 2008

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Poďakovanie

Chcem sa poďakovať môjmu školiteľovi doc. RNDr. Martinovi Stanekovi PhD. za výber témy, cenné rady a trpezlivosť, svojim rodičom a spolubývajúcim za mnohé užitočné rady a Mgr. Michalovi Foríškovi za poskytnutie templatov pre sadzbu v programe TEX.

Abstrakt

Autor:	Tomáš Žubrietovský
Názov práce:	Automatická kryptoanalýza súčtu dvoch otvorených textov
Škola:	Univerzita Komenského
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katerda:	Katedra informatiky
Vedúci bakalárskej práce:	doc. RNDr. Martin Stanek PhD.
Rozsah práce:	33 strán
Bratislava, jún 2008	

V tejto bakalárskej práci sa venujem automatickej kryptoanalýze súčtu dvoch otvorených textov písaných v slovenskom jazyku. Problému ako zo XOR-u dvoch otvorených textov získať tieto dva otvorené texty sa hovorí aj two-time pad problém a je ho možné riešiť viacerými metódami. Ako východiskovú metódu pre riešenie tohto problému som použil metódu z článku [7], ktorú som implementoval aj v mojom programe. Na tomto programe som vykonal viaceré testy a ich úspešnosť porovnal s testami tejto metódy pre anglický jazyk.

Kľúčové slová: Vernamova šifra, One-time Pad, Two-time pad problém, XOR

Predhovor

Moja bakalárska práca je z oblasti kryptológie. Venujem sa v nej automatickej kryptoanalýze súčtu dvoch otvorených textov. Táto kryptoanalýza je možná viacerými metódami, ja som si vybral metódu použitú v článku [7], ktorá dosiahla skoro stopercentnú úspešnosť dešifrovania. Túto metódu som implementoval pre otvorené texty písané v slovenskom jazyku.

Obsah

Zoznam tabuliek

Zoznam obrázkov

Úvod

Snaha ľudí ochrániť informácie speje k stále lepším šifrovacím algoritmom. Medzi tie, ktoré môžeme nazvať absolútne bezpečné šifrovacie algoritmy patrí aj Vernamova šifra. Avšak touto absolútne bezpečnou šifrou sa stáva až pri splnení viacerých predpokladov. Pri snahe eliminovať v reálnych systémoch najväčší nedostatok Vernamovej šifry, ktorým je dôverný prenos kľúča, sa predpokladu, že pri šifrovaní ďalšieho otvoreného textu zvolíme nový kľúč, nevenovala adekvátna pozornosť. Toto spôsobilo veľký bezpečnostný problém Vernamovej šifry a stratu jej absolútnej bezpečnosti.

V mojej bakalárskej práci sa venujem metódam, ako z dvoch šifrových textov šifrovaných použitím rovnakého kľúča získať zodpovedajúce otvorené texty. V práci ukážem, že tento problém sa dá pretransformovať na problém ako z XOR-u dvoch otvorených textov získať tieto dva otvorené texty. Tento problém sa nazýva aj two-time pad problém. Za východiskovú metódu som si zobral metódu uverejnenú v článku [7], ktorú som implementoval pre texty písané v slovenskom jazyku.

Bakalárska práca je rozdelená na tri hlavné kapitoly.

V 1.kapitole opisujem ako vznikla Vernamova šifra, jej výhody, jej hlavné nedostatky, bezpečnostné problémy a venujem sa jej hlavnému bezpečnost-

nému problému, ktorým je znovupoužitie kľúča.

V 2.kapitole opisujem metódu na riešenie two-time pad problému, ktorú som potom použil aj v mojom programe. Opis programu je tiež súčasťou tejto kapitoly.

Testom, ktoré som vykonal na mojom programe, ich výsledkom a dôvodom dosiahnutej úspešnosti sa venujem v 3.kapitole.

Kapitola 1

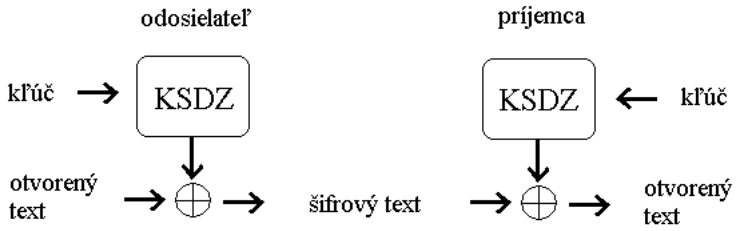
Vernamova šifra

1.1 Šifrovacie systémy

V poslednom čase sa stále väčšie úsilie venuje ochrane informácií. Vedný obor, ktorý sa zaoberá skúmaním bezpečnostných aspektov komunikácie sa nazýva kryptológia. Pôvodnou a dnes stále najrozšírenejšou úlohou kryptológie je ochrana dôvernosti prenášaných dát, teda šifrovanie.

Šifrovanie je proces transformácie vstupných dát do podoby, v ktorej sú pre potenciálneho útočníka nezrozumiteľné a nie je schopný ich previesť do pôvodnej podoby. Šifrovanie ako proces je realizované šifrovacím algoritmom (systémom). Vstupné dáta v ich pôvodnej podobe sa nazývajú otvorený text, výsledkom šifrovania bude šifrový text. Kľúčom bude parameter, ktorý je vstupom šifrovacieho algoritmu a nezávisí na otvorenom texte.
Šifrovacie systémy sa delia na dve veľké skupiny: symetrické a asymetrické.

U symetrických šifrovacích systémoch je šifrovací kľúč rovnaký ako dešifrovací, u asymetrických šifrovacích systémoch sú tieto kľúče rôzne. Symet-



Obr. 1.1: Prúdová šifra

rické šifrovacie algoritmy sa delia na blokové a prúdové.

Blokové šifry spracúvajú vstupný text po blokoch pevnej dĺžky, pričom šifrovacia funkcia je definovaná ako transformácia jedného bloku dát. Dešifrovanie je tiež definované nad jednotlivými blokmi.

U prúdovej šifry (pozri obr. 1.1) sa text šifruje postupne. Kľúč je použitý na inicializáciu konečnostavového deterministického zariadenia (KSDZ), ktoré produkuje prúd bitov. Kombináciou tohto prúdu bitov s bitmi otvoreného textu vzniká šifrový text. Príjemca šifrového textu použije kľúč a vygeneruje rovnaký prúd bitov, ktorého kombináciou so šifrovým textom vzniká otvorený text.

1.2 Bezpečnosť šifrovacích systémov

Bezpečnosť, teda odolnosť voči potenciálnemu útoku, je najdôležitejšou vlastnosťou šifrovacieho systému. Bezpečnosť šifrovacieho systému môže byť buď výpočtová alebo nepodmienená. Výpočtová bezpečnosť predpokladá, že útočník má obmedzenú výpočtovú silu, nepodmienená predpokladá, že výpočtová sila útočníka je neobmedzená. V ďalšej časti opíšem, čo musí spĺňať šifrovací systém, aby jeho bezpečnosť bola nepodmienená.

1.2.1 Pojem absolútne bezpečnej (nepodmienenej) šifry

Keďže bezpečnosť šifrovacieho systému závisí aj od typu útoku, bude aj pojem absolútne bezpečnej (nepodmienenej) šifry obmedzený niekoľkými predpokladmi. Pripomeniem, že poznáme 4 typy útoku:

- COA-ciphertext only attack, útok len so znalosťou šifrového textu
- KPA-known plaintext attack, útok so znalosťou otvoreného textu
- CPA-chosen plaintext attack, útok s možnosťou voľby otvoreného textu
- CCA-chosen ciphertext attack, útok s možnosťou voľby šifrového textu

Útoky sú zoradené podľa rastúcej sily. Útočník má najťažšiu situáciu pri type útoku COA, najľahšiu pri CCA.

Pojem absolútne bezpečnej šifry je definovaný v situácii útoku len so znalosťou šifrového textu (COA).

Šifrovacím systémom nazveme dvojicu funkcií $E : P \times K \rightarrow C$ a $D : C \times K \rightarrow P$ s vlastnosťou

$$\forall k \in K \ \forall p \in P : D_k(E_k(p)) = p,$$

kde P je konečná množina otvorených textov, C konečná množina šifrových textov a K konečná množina kľúčov. Pojem absolútne bezpečného šifrovacieho systému sa opiera o nasledujúce predpoklady.

1. Predpokladáme fixnú distribúciu pravdepodobností nad P a nad K. Označíme $p_P(x)$ pravdepodobnosť, že otvorený text je x ($x \in P$).
2. Pri šifrovaní ďalšieho otvoreného textu zvolíme nový kľúč z K.

3. Voľba kľúča je náhodná (nezávislá na voľbe otvoreného textu).

Definícia 1.2.1 *Šifrovací systém* (E, D) *je absolútne bezpečný, ak*

$$\forall x \in P \ \forall y \in C : p_P(x|y) = p_P(x).$$

Teda absolútne bezpečný šifrovací systém je taký, kde (apriórna) pravdepodobnosť otvoreného textu sa nezmení ani keď máme k dispozícii šifrový text. Čiže aj keby útočník poznal šifrový text a mal neobmedzenú výpočtovú silu, nepomôže mu to získať žiadnu novú informáciu o otvorenom texte.

1.3 Vernamova šifra

Jednou z absolútne bezpečných šifier je Vernamova šifra. Vernamova šifra patrí medzi symetrické prúdové šifrovacie algoritmy.

Vernamov šifrovací algoritmus bol vynájdený v roku 1917 americkým vedcom Gilbertom Sandford Vernamom a patentovaný v roku 1919. Pri šifrovaní Vernamovým systémom bolo možné použiť kľúč viackrát. O pár rokov neskôr Joseph Mauborgne reorganizoval Vernamovu šifru tým, že kľúč musí byť volený úplne náhodne a použitý len raz. Touto zmenou sa z Vernamovej šifry stala absolútne bezpečná šifra, nazývaná aj One-time Pad, avšak jej absolútnu bezpečnosť dokázal až Claude Elwood Shannon v roku 1949. V texte bakalárskej práce budem pomenovanie One-time Pad zamieňať s názvom Vernamova šifra.

Definícia 1.3.1 (Vernamova šifra) *Nech $A = \{0,1\}$ je abeceda otvoreného textu, nech P je konečná množina otvorených textov , C je konečná množina šifrových textov a K je konečná množina kľúčov, pre ktoré platí $P = C = K = \{0,1\}^n$, kde $n \in N \wedge n \geq 1$. Potom šifrovanie spočíva v pripočítaní kľúča $k = (k_1, k_2, ..., k_n)$ po bitoch k otvorenému textu $p = (p_1, p_2, ...p_n)$ modulo 2.*

$$E(p,k) = p \oplus k = p_1 \oplus k_1, ..., p_m \oplus k_m.$$

Keďže pre všetky $x \in \{0,1\}$ platí: $x \oplus x = 0$, potom dešifrovanie spočíva v pripočítaní kľúča k k šifrovému textu, čiže:

$$D(c,k) = c \oplus k = (p \oplus k) \oplus k = p.$$

Operácia $\oplus$ sa nazýva XOR. Vernamovu šifru z definície je možné upraviť pre písmená abecedy. Pre anglickú abecedu nech kľúč $k = k_1, k_2, ..., k_n$ a otvorený text $p = p_1, p_2, ..., p_n$, kde $\forall p_i, k_i \in \{A, B, C, ..., Z\}$. Pre potreby šifrovania sa každému písmenu abecedy priradí číslo prislúchajúce jeho poradiu so začiatkom od 0, čiže A=0, B=1, ..., Z=25. Potom pre šifrovanie platí: $c_i = (p_i + k_i) \mod 26$ a pre dešifrovanie platí: $p_i = (c_i - k_i) \mod 26$.

Aby sa z Vernamovej šifry stala absolútne bezpečná šifra (čiže One-time Pad), musia byť ešte splnené body 1, 2 a 3 z predpokladov pojmu absolútne bezpečného šifrovacieho systému, čiže:

1. Kľúče sú volené (vyberané) z množiny K náhodne, nezávisle a s rovnakou pravdepodobnosťou. (1. a 3. bod)

2. Pri šifrovaní ďalšieho otvoreného textu zvolíme vždy nový kľúč z K. (2. bod)

Pri takto volenom kľúči nie je možné zo šifrového textu zistiť žiadne informácie o pôvodnej správe, ani vzťahy medzi skupinami znakov a pod., pretože z každého písmena vznikne úplne náhodne volené písmeno. Útok hrubou silou, voči ktorému nie je odolná prakticky žiadna šifra, nám v tomto prípade veľmi nepomôže, lebo aj keby mal útočník neobmedzený výpočtový výkon a vyskúšal by všetky možné kľúče dĺžky n, dostane 2^n resp. $|\{A, ...Z\}|^n$ rovnako pravdepodobných otvorených textov, medzi ktorými nájsť ten správny je prakticky nemožné. Preto pravdepodobnosť, že otvorený text je x ($x \in P$) sa nezmení ani pri znalosti šifrového textu. Teda Vernamova šifra je absolútne bezpečná šifra.

1.3.1 Nedostatky Vernamovej šifry

Napriek bezpečnostným kvalitám One-time Pad šifry, má táto šifra aj dôležitý nedostatok. Ak by sme chceli preniesť n bitov dlhú správu, je potrebné vygenerovať a bezpečne doručiť adresátovi n bitov dlhý kľúč. Týmto sa z problému dôverného prenosu správy stáva problém dôverného prenosu kľúča. V situácii, keď sú náhodné bity vygenerované v dostatočnom množstve dopredu a neskôr postupne používané, nám tento problém nemusí prekážať.

1.3.2 Bezpečnostné problémy Vernamovej šifry

Nedostatok Vernamovej šifry, generovať a udržiavať náhodný kľúč rovnako dlhý ako samotný text sa v prúdových šifrách rieši pomocou generovania pseudonáhodných čísel. O generovanie pseudonáhodných čísel sa stará generátor pseudonáhodných čísel, ktorý na základe nejakého IV (inicializačného vektora) a tajného kľúča generuje pseudonáhodný kľúč dĺžky otvoreného textu.

Tu ale nastáva problém, že veľakrát je dĺžka IV nedostatočná. Ak je poslaných veľa správ s rovnakým tajným kľúčom, je veľká pravdepodobnosť, že niektorý IV je použitý dvakrát, čím sa vygeneruje rovnaký pseudonáhodný kľúč, a teda sa dá použiť útok pre systémy s rovnakým kľúčom.

Asi najväčším bezpečnostným problémom prúdových šifier, a teda aj Vernamovej šifry je práve znovupoužitie kľúča. Aj napriek tomu, že to v One-time Pad šifre spôsobuje stratu jej absolútnej bezpečnosti a tento problém je už dlho známy, bol v reálnych systémoch podceňovaný [3, 6, 10].

V čom spočíva nebezpečenstvo znovupoužitia kľúča. Nech k je kľúč a p a q sú otvorené texty rovnakej dĺžky. Nech c_1 a c_2 sú šifrové texty, kde $c_1 = p \oplus k$ a $c_2 = q \oplus k$.
Potom

$$c_1 \oplus c_2 = (p \oplus k) \oplus (q \oplus k) = (p \oplus q).$$

Teda XOR dvoch šifrových textov s rovnakým kľúčom je XOR pôvodných otvorených textov. Problém, ako z $p \oplus q$ určiť p a q, sa nazýva two-time pad problém. Two-time pad problém je možné s vysokou pravdepodobnosťou efektívne riešiť použitím viacerých techník.

Kapitola 2

Riešenie two-time pad

Ako už bolo spomenuté v predchádzajúcej kapitole, znovupoužitie kľúča vo Vernamovej šifre je bezpečnostný problém, ktorý sa však v reálnych systémoch dosť podceňuje. Ani dobré výsledky metód obnovy otvorených textov nezastavili systémových dizajnérov pred znovupoužitím kľúča vo svojich systémoch. Systémy, ktorých sa to týka sú napr. Microsoft Office, 802.11 WEP, WinZip, PPTP a z histórie napr. sovietska diplomacia, armáda a tajné služby [7].

2.1 Rôzne prístupy obnovy otvorených textov

Asi najzaujímavejšie metódy obnovy otvorených textov, ktoré boli šifrované tým istým kľúčom, sú známe ako National Security Agency VENONA projekt.

Army‘s Signal Intelligence Service, predchodca dnešnej NSA, oznámil, že

v niektorej šifrovanej sovietskej telegrafnej komunikácii objavili znovupoužitie kľúča. Program obnovy otvorených textov začal v roku 1943 a v roku 1980 ešte nebol skončený. Za toto obdobie bolo dešifrovaných viac ako 3000 správ.

Teoretický útok na dva šifrové texty vytvorené s tým istým kľúčom bol navrhnutý Frankom Rubinom v roku 1978 [7].

V roku 1996 Dawsom a Nielsen vytvorili program, ktorý používal sériu heuristických pravidiel pre automatický prístup dešifrovania [5].

2.2 Automatický prístup

V tejto časti opíšem metódu dešifrovania použitú v článku [7], ktorá dosiahla oveľa lepšie výsledky ako metóda Dawsona a Nielsena, a ktorú som použil aj v mojom programe. Dešifrovanie v tejto metóde je závislé na type dokumentov, z ktorých dané otvorené texty pochádzajú. Preto vytvorím zbierky dát, ktoré budú reprezentovať typy dokumentov v slovenskom jazyku. Potom pri dešifrovaní daná metóda zisťuje pravdepodobnosti výskytu všetkých možných otvorených textov prislúchajúcich k danému šifrovému textu v danej zbierke dát a tie najpravdepodobnejšie otvorené texty budú s najväčšou pravdepodobnosťou pôvodné otvorené texty.

Nech $x = c_1 \oplus c_2$. Výsledkom danej metódy má byť dvojica (p, q) taká, že $x = p \oplus q$ a p je s veľkou pravdepodobnosťou otvorený text k jednému zo šifrových textov c_1, c_2 a q k druhému. Nech Pr_1 a Pr_2 udávajú pravdepodobnosti výskytu otvorených textov p a q na rôznych typoch dokumentov a $\mathrm{Pr}(p, q)$ udáva pravdepodobnosť, že p a q sú otvorené texty prislúchajúce k

šifrovým textom c_1 a c_2. Keďže predpokladám, že tieto otvorené texty boli písané nezávisle, potom platí:

$$\Pr(p, q) = \Pr_1(p) \cdot \Pr_2(q).$$

2.2.1 n-gramový model jazyka

Nech $p = (p_1, p_2, ..., p_l)$ je reťazec dĺžky l a $\Pr_1(p)$ je pravdepodobnosť výskytu reťazca p v danej zbierke údajov. Nech n je dĺžka n-gramu v jazykovom modely. Nech platí:

$$Pr_1(p) = \prod_{i=1}^{l} Pr_1(p_i | p_{i \dot{-} n+1}, ...p_{i-2}, p_{i-1}), \tag{2.1}$$

kde $i \dot{-} n + 1$ označuje $\max(i - n, 0)$.

Rovnosť (2.1) sa nazýva n-gramový model, pretože numerické faktory v súčine sú odvodené zo štatistík podreťazcov dĺžky n. Tieto štatistiky som získal z rôznych zbierok údajov, ktoré popíšem v kapitole 3.

2.2.2 Dešifrovanie

Keďže predpokladám, že otvorené texty sú písané (vyberané) nezávisle, platí $\Pr(p, q) = \Pr_1(p).\Pr_2(q)$. Dosadením rovnosti (2.1) za $\Pr_1(p)$ a $\Pr_2(q)$ dostanem

$$\Pr(p, q) = \prod_{i=1}^{l} \Pr(p_i, q_i | p_{i \dot{-} n+1}, ...p_{i-2}, p_{i-1}, q_{i \dot{-} n+1}, ...q_{i-2}, q_{i-1}), \tag{2.2}$$

kde

$$\Pr(p_i, q_i | p_{i-n+1}^{\cdot}, ...p_{i-1}, q_{i-n+1}^{\cdot}, ...q_{i-1}) = \\ \Pr_1(p_i | p_{i-n+1}^{\cdot}, ...p_{i-1}) \cdot \Pr_2(q_i | q_{i-n+1}^{\cdot}, ...q_{i-1}) \quad (2.3)$$

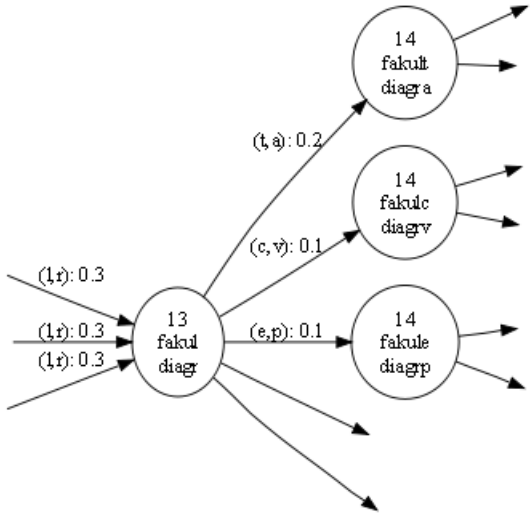
Na vstupe mám XOR dvoch otvorených textov dĺžky l, ktorý budem v ďalšom texte nazývať šifrovým textom. Algoritmus postupne číta vstup po znakoch. Prvý znak mohol vzniknúť XOR-om niekoľkých dvojíc znakov, ale u každej dvojice s inou pravdepodobnosťou.

Program konštruuje graf, ktorý má hrany ohodnotené dvojicou znakov, z ktorých mohol daný šifrový znak vzniknúť s príslušnou pravdepodobnosťou.

Vrcholmi grafu je posledných osem znakov aktuálne dešifrovaného textu (v prípade, že aktuálne dešifrovaný text je kratší ako osem znakov, tak je vrcholom celý dešifrovaný text) pre prvý a druhý otvorený text. Pravdepodobnosť Pr je počítaná podľa vzťahu (2.3). Výstupom algoritmu bude graf $G = (V_1 \times V_2, E)$, ktorý je skonštruovaný z grafov $G_1 = (V_1, E_1)$ a $G_2 = (V_2, E_2)$, kde V_1 je množina vrcholov, ktorými je posledných osem znakov momentálne dešifrovaného textu pre prvý otvorený text a V_2 množina vrcholov, ktorými je posledných osem znakov momentálne dešifrovaného textu pre druhý otvorený text.

Teraz nech $(u_1, v_1), (u_2, v_2) \in V_1 \times V_2$, potom E obsahuje ohodnotené hrany $(u_1, u_2) \xrightarrow{(znak_1, znak_2):pravd_1.pravd_2} (v_1, v_2)$, ak E_1 obsahuje hrany $u_1 \xrightarrow{znak_1:pravd_1} v_1$ a E_2 obsahuje $u_2 \xrightarrow{znak_2:pravd_2} v_2$, kde $znak_i$ je znak, na ktorý sa dostanem zo stavu u_i do stavu v_i s pravdepodobnosťou $pravd_i$ počítanou podľa vzťahu (2.1) a $i \in \{1, 2\}$.

Z tohto grafu najpravdepodobnejšia cesta definuje najpravdepodobnej-



Obr. 2.1: Na obrázku je fragment grafu G, ktorý vzniká pri dešifrovaní metódou z článku [7]. Z vrcholu 13 vytvorí hrany pre každú dvojicu, ktorou mohol vzniknúť 14-ty šifrový znak. V mojom príklade 14-ty šifrový znak vznikol z dvojice (t,a) s pravdepodobnosťou 0.2, z dvojíc (c,v) a (e,p) s rovnakou pravdepodobnosťou 0.1.

šiu podobu otvorených textov pre daný šifrový text. Fragment grafu G je znázornený na obrázku 2.1.

Vzniká tu však problém veľkosti grafu G. Keďže znaky textu sú reprezentované ako bajty, z každého vrcholu môže teoreticky vychádzať 256 hrán, čo pri šifrovom texte dĺžky n je 256^n hrán.

Avšak z pravdepodobnostného hľadiska niektoré stavy nemôžu nastať alebo môžu nastať len s veľmi malou pravdepodobnosťou. Preto riešením je nepamätať si všetky možnosti, ale len tie najpravdepodobnejšie.

Pri dešifrovaní prvého šifrového znaku zo všetkých potenciálnych možností sa vyberie 100 najpravdepodobnejších. Pri dešifrovaní druhého znaku pôjde teoreticky z každého zo 100 vrcholov 256 hrán, vznikne 25600 nových hrán, z ktorých sa znovu vyberie len 100 najpravdepodobnejších. Po dešif-

rovaní celého šifrového textu vznikne na výstupe oklieštený pôvodný graf G len na najpravdepodobnejšie stavy. V tomto grafe sa už pomocou Viterbiho algoritmu ľahko nájde najpravdepodobnejšia cesta a vrcholy, cez ktoré prechádza, budú s najväčšou pravdepodobnosťou určovať dvojicu otvorených textov, ktorých XOR-om vznikol daný šifrový text.

2.3 Implementácia

V tejto časti popíšem môj program, ktorý je implementovaním postupu z článku [7] a umožňuje dešifrovanie šifrových textov, ktorých otvorené texty boli písané v slovenskom jazyku. Program som písal v Jave 1.5 vo vývojom prostredí Eclipse 3.3.1.1.

2.3.1 Modelovanie jazyka

Na vytvorenie jazykového modelu som použil open source sadu javovských knižníc pre lingvistickú analýzu prirodzených jazykov pod názvom LingPipe [4][1].

LingPipe

LingPipe poskytuje API rozhranie pre veľa NLP (natural language processing-spracovanie prirodzených jazykov počítačom) úloh ako napr. klastrovanie, úprava pravopisu alebo označovanie slovných druhov. Použil som ho na vytvorenie n-gramového jazykového modelu zo zbierky slov charakteristických pre daný typ dokumentu. Vnútorne, LingPipe uchováva model ako prefixový

[1] Prístupné na `http://www.alias-i.com/lingpipe`

strom s väčšou dĺžkou n-gramov blízko listov. Prefixový strom je dátová štruktúra, ktorá sa používa pre uchovanie asociatívneho poľa. Na rozdiel od binárneho vyhľadávacieho stromu, kde sa podľa hodnoty uzla rozhoduje, po ktorej vetve bude pokračovať, prefixový strom obsahuje v každom uzle všetky podreťazce, ktorými môže pokračovať reťazec v doteraz prehľadanej ceste. Všetky deti daného uzla majú spoločný prefix, ktorým je práve reťazec priradený k otcovi. Koreňu je priradený prázdny reťazec.

Na generovanie znakového jazykového modulu som použil triedu *NGramProcessLM*, ktorej som v konštruktore nastavil maximálny n-gram na 8 a maximálny počet znakov v zbierke na 256 a jej metódu *train(CharSequence cSeq)*, kde *cSeq* sú vstupné dáta, pomocou ktorých sa vytvára prefixový strom.

Trieda *NGramProcessLM* poskytuje aj metódy na počítanie pravdepodobnosti výskytu reťazca v tejto dátovej štruktúre. O prácu s týmito metódami sa stará trieda *Prepocet*.

Trieda Prepocet

Pri počítaní pravdepodobnosti výskytu reťazca v danej zbierke dát sa zavolá metóda *prepocitaj* triedy *Prepocet*.

```
public double prepocitaj(char[] text, int dlzka)
```

Parameter *text* je reťazec a *dlzka* je dĺžka podreťazca z tohto reťazca, ktorého pravdepodobnosť sa má počítať.

V metóde *prepocitaj* sa volá metóda *log2ConditionalEstimate* triedy *NGramProcessLM*.

```
log2ConditionalEstimate"(char[] cs, int start, int end,
                         int maxNGram, double lambdaFactor)
```

Táto metóda vracia pravdepodobnosť výskytu podreťazca začínajúceho na pozícii *start* a končiaceho na pozícii *end* v reťazci *cs* prevedenú na dvojkový logaritmus. Premenná *maxNGram* udáva maximálnu dĺžku n-gramu, ktorá sa má použiť pri počítaní pravdepodobnosti výskytu daného podreťazca a premenná *lambdaFactor* nastavuje parameter K zo vzťahu (2.5). Nech X označuje reťazec a c, d znaky. Potom pravdepodobnosť reťazca dXc sa počíta podľa vzťahu:

$$\begin{aligned} \hat{P}(c|dX) &= \lambda(dX)P_{ML}(c|dX) + (1 - \lambda(dX))\hat{P}(c|X) \\ \hat{P}(c) &= \lambda()P_{ML}(c) + (1 - \lambda())(1/|Char|), \end{aligned} \tag{2.4}$$

kde

$$\lambda(X) = \frac{extCount(X)}{extCount(X) + K.numExts(X)} \tag{2.5}$$

$P_{ML}(c|X) = \frac{count(Xc)}{extCount(X)}$, kde $count(X)$ je počet výskytov reťazca X v zbierke dát a $extCount(X) = \sum_{c \in Char} count(Xc)$, čiže počet podreťazcov vyskytujúcich sa v danej zbierke, za ktorými nasleduje nejaký znak. Počet rôznych znakov, ktoré sa nachádzajú za reťazcom X v zbierke dát je $numExts(X) = |\{c|count(Xc) > 0\}|$.

Vzťah (2.5) je výpočet interpolačného parametra $\lambda(X)$ podľa Wittenovej-Bellovej odhadovej funkcie distribúcie pravdepodobnosti, v ktorom je parameter K nastavený na 1. Ak by bol parameter K nastavený na 0, tak by vzťah (2.4) počítal pravdepodobnosť, že za reťazcom dX nasleduje znak c iba ako pomer počtu všetkých reťazcov dXc a reťazcov dX. Uvediem príklad prečo je takéto počítanie pravdepodobnosti nevyhovujúce.

Nech parameter $K = 0$ a nech jeden z otvorených textov je text: *"Bratislava je mesto"*. V zbierke dát sa toto slovné spojenie nenachádza, ale nachádzajú sa tam slovné spojenia: *"Bratislava je hlavné mesto"* a *"Nech je mesto"*.

Keďže v mojom programe pracujem s 8-gramami, tak pravdepodobnosť $\hat{P}(\text{"}m\text{"}|\text{"}ava\ je\ \text{"})$ bude 0 aj napriek tomu, že *"m"* je naozaj znak, ktorý má nasledovať. Preto je teraz potrebné zisťovať pravdepodobnosť pre 7-grami, 6-grami, 5-grami atď., až kým bude pravdepodobnosť nenulová. V tomto prípade to bude nenulová pravdepodobnosť pre 5-gram *" je m"*.

Ale čo ak pre nejaký znak x dostanem $\hat{P}(x|\text{"}ava\ je\ \text{"})$ nenulovú, aj keď x nie je znak, ktorý má nasledovať. V akom vzťahu má potom byť pravdepodobnosť 8-gramu s nasledujúcim znakom x a pravdepodobnosť 5-gramu s nasledujúcim znakom m? Tento problém rieši práve parameter K. Čím je parameter K nastavený bližšie k hodnote 1, tým viac sa do pravdepodobnosti výskytu reťazca v danej zbierke dát započítava aj pravdepodobnosť výskytu jeho podreťazcov v tejto zbierke dát. V mojom programe som parameter K nastavil na hodnotu 1. Preto pre môj príklad, ak $K = 1$, tak $\hat{P}(\text{"}m\text{"}|\text{"}ava\ je\ \text{"})$ už nebude nulová.

2.3.2 Riešenie two-time pad

V tejto časti opíšem samotný proces získavania otvorených textov z daného šifrového textu. O tento proces sa starajú metódy triedy *Desifruj*, ktorá šifrový text dostane na vstupe ako pole integerov, ktorými sú kódy znakov v kódovaní ISO 8859-2.

metóda Desifrujem

Táto metóda sa stará o dešifrovanie prvého znaku šifrového textu. V metóde sa na začiatku vytvorí pomocná premenná *maxrad* typu ArrayList, do ktorej sú ukladané všetky možné dvojice spolu s ich pravdepodobnosťami výskytu v zbierke údajov, XOR-om ktorých mohol vzniknúť prvý šifrový znak. Tieto dvojice metóda získava z hashtabuľky, ktorú si trieda *Desifruj* vytvorila v konštruktore a ich pravdepodobnosť výskytu v zbierke údajov sa vypočíta pomocou metódy *prepocitaj* triedy *Prepocet*. Prvých 100 najpravdepodobnejších dvojíc ukladá metóda do ArrayListu *rad*.

metóda krok

Táto metóda sa stará o dešifrovanie jedného konkrétneho znaku šifrového textu na pozícii i. Z ArrayListu *rad*, v ktorom je uložených 100 najpravdepodobnejších dvojíc otvorených textov dĺžky $i-1$, sa pre každú dvojicu vytvorí 224 (predpokladám, že otvorené texty neobsahujú prvých 32 znakov ASCII tabuľky) dvojíc otvorených textov dĺžky i, ktoré vzniknú z pôvodnej dvojice a dvojíc znakov, XOR-om ktorých mohol vzniknúť i -ty šifrový znak.

Pravdepodobnosť dvojíc otvorených textov dĺžky i sa počíta podľa vzťahu:

$$\text{pravdepodobnost} = \text{pravdepodobnost} \cdot \Pr(p, q),$$

kde p a q sú otvorené texty dĺžky i a $\Pr(p, q)$ je zo vzťahu (2.2). Novými 100 najpravdepodobnejšími dvojicami otvorených textov pre i znakov šifrového textu sa vyplní dátová štruktúra *rad*.

metóda run

V tejto metóde sa dešifruje šifrový text od druhého znaku až do konca. Dešifrovanie je rovnaké ako v metóde *krok*.

V metódach *krok* a *decrypted* je proces dešifrovania rozdelený do dvoch častí.

1. Dešifrovanie druhého až siedmeho znaku.
2. Dešifrovanie ďalších znakov.

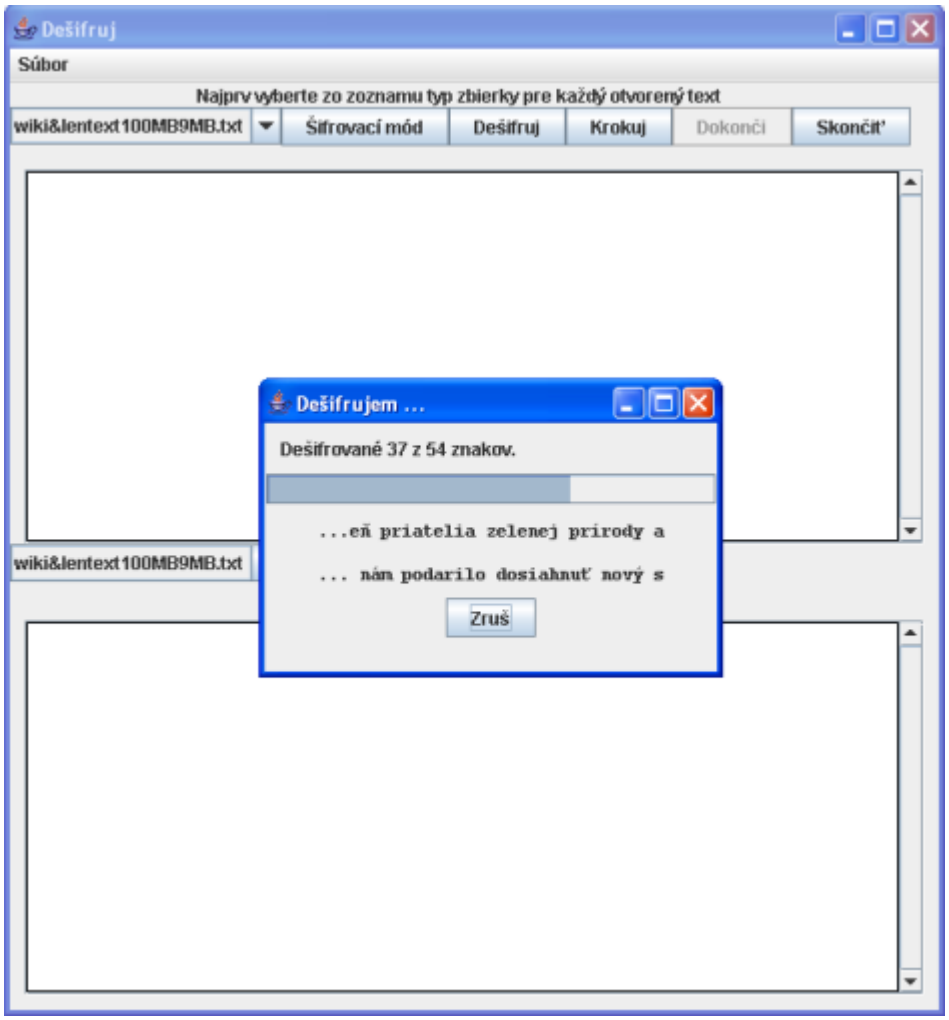
Dôvodom tohto rozdelenia je, že počítanie pravdepodobnosti reťazca je dovolené pre reťazce dĺžky najviac osem. Preto pre dlhšie reťazce sa posledných osem znakov uloží do pomocnej premennej a počíta sa pravdepodobnosť výskytu len týchto posledných osem znakov v zbierke údajov.

2.3.3 GUI

Triedou, ktorá celý môj program obaľuje, je trieda *GUI*. V nej sa vytvoria inštancie dvoch ďalších tried *GUI_sifruj* a *GUI_desifruj*. Obe tieto triedy vytvárajú užívateľské prostredie pre šifrovanie a dešifrovanie textu.

Trieda GUI_sifruj

Táto trieda sa stará o užívateľské prostredie pre šifrovanie dvoch otvorených textov. Po stlačení tlačítka Šifruj sa otvorí dialógové okno, kde užívateľ zadá pod akým názvom má byť daná šifra uložená. Následne sa pomocou metódy *sifrujem* triedy *Sifruj* , ktorej parametrami sú dva texty napísané užívateľom do dvoch textových polí, vykoná XOR týchto otvorených textov po znakoch a takto vytvorená šifra sa uloží pod názvom ako zadal užívateľ.



Obr. 2.2: Prostredie pre dešifrovanie

Trieda GUI_desifruj

Táto trieda poskytuje užívateľské rozhranie pre dešifrovanie šifrového textu. Užívateľ má možnosť vybrať zo zoznamu typ dokumentu pre obidva otvorené texty. Pre daný typ dokumentu sa použije zbierka dát uložená v prefixovom strome pre tento typ dokumentu. Užívateľ si môže vybrať, či chce šifrový text dešifrovať naraz alebo po krokoch. Pri dešifrovaní naraz sa dešifruje celý vstupný šifrový reťazec pomocou metód *Desifrujem* a *run* triedy *Desifruj* a dešifrované otvorené texty sú vypísané do textových polí a do súboru. Pri

možnosti po krokoch je dešifrovaný $(i + 1)$-vý znak šifrového textu, kde i je počet už dešifrovaných znakov. Dešifrovanie je vykonávané pomocou metód *Desifrujem krok* triedy *Desifruj*. Výstup sa tiež vypisuje do dvoch textových polí. Počas procesu dešifrovania sa striedajú dve vlákna, kde jedno vlákno slúži na dešifrovanie pomocou triedy *Desifruj* a druhé sa stará o progressbar triedy *ProgressBar*, ktorý vizualizuje priebeh dešifrovania daného šifrového textu.

Užívateľ má taktiež možnosť importovať svoju zbierku dát, z ktorej sa vytvorí model pomocou metódy *train* triedy *NGramProcessLM* Tento model je ako dátová štruktúra prefixový strom uložený do súboru pod rovnakým názvom ako sa volala daná zbierka dát. Ako vyzerá prostredie pre dešifrovanie zobrazuje obrázok 2.2.

Kapitola 3

Štatistické výsledky

V tejto časti opíšem, aké zbierky údajov som použil, aké výsledky dešifrovania som na týchto zbierkach údajov dosiahol, s ktorými šifrovými textami mal program problémy a prečo a s akými zlepšeniami sa dajú dosiahnuť ešte lepšie výsledky.

3.1 Zbierky dát

Na testovanie som používal štyri zbierky dát. Na výrobu prvých troch zbierok som použil súbor veľkosti 1GB, ktorý bol vytvorený z html súborov slovenskej wikipédie väčších ako 20KB, ktoré som stiahol zo stránky
`http://static.wikipedia.org/downloads/April_2007/sk/`

1. Najväčšiu zbierku som z tohto 1GB súboru vytvoril vyhodením všetkých HTML tagov, viacnásobné medzery som prepísal na jednu a konce riadkov zamenil za medzery.

2. Druhou zbierkou bolo prvých 100MB z daného 1GB súboru. Keďže

Zbierka	Veľkosť zbierky	Veľkosť dátovej štruktúry
SK wikipédia - celá	159MB	366MB
SK wikipédia 100MB bez úpravy	93.7MB	83MB
SK wikipédia 100MB s úpravou	8.67MB	33.3MB
Biblia+klasici.sk	9,86MB	47.6MB

Tabuľka 3.1: Veľkosti použitých zbierok

daný 1GB súbor bol vytváraný z textov zodpovedajúcim pojmom wikipédie zoradených podľa abecedy, cieľom tejto zbierky bolo zistiť, aký vplyv má na úspešnosť dešifrovania menšia zbierka ochudobnená o niektoré slová začínajúce na písmená z určitej časti abecedy.

3. Tretiu zbierku som vyrobil z druhej tým, že som odstránil všetky HTML tagy, z duplikovaných medzier spravil jednu a konce riadkov nahradil medzerami.

4. Štvrtou zbierkou bola zbierka vytvorená zo slovenskej verzie Biblie a 29. literárnych diel slovenských autorov stiahnutých so stránky `http://www.klasici.sk`. Vytvorený súbor som nijako neupravoval.

Všetky zbierky údajov boli uložené v kódovaní ISO-8859-2. Veľkosti týchto zbierok aj s veľkosťami súborov, ktoré obsahovali prefixové stromy vytvorené z týchto zbierok, udávam v tabuľke 3.1.

3.2 Testovanie úspešnosti dešifrovania

V tejto časti opíšem dešifrovacie testy, ktoré som robil na daných zbierkach údajov a úspešnosť dešifrovania, ktorú som dosiahol.

	Korektne dešifrované	Nekorektne dešifrované	Nedešifrované
[7]	100%	0%	0%
[5]	62.7%	17.8%	19.5%
Moja práca	99,8%	0,2%	0%

Tabuľka 3.2: Úspešnosť dešifrovania

1. test

Pri prvom testovaní som použil podobný postup ako autori článkov [5] a [7]. Ako zbierku dát použili prvých 600000 znakov anglickej Biblie, ktorú špeciálne naformátovali: všetky špeciálne znaky okrem medzier boli odstránené a všetky znaky prevedené na veľké. Ako otvorené texty použili tri náhodne vybrané podreťazce z tejto zbierky s veľkosťou 12000 znakov [5, 7].

Ja som ako zbierku dát použil 600000 znakov textu zo slovenskej wikipédie bez HTML tagov, viacnásobných medzier a konce riadkov som nahradil medzerami. Zostávajúci text som nechal bez zmeny. Z tejto zbierky dát som náhodne vybral dva texty po 1000 znakov, z ktorých som vytvoril jeden šifrový text. Porovnania výsledkov uvádzam v tabuľke 3.2.

Dôvodom nie úplne stopercentne dešifrovaného textu v mojej práci je, že problémové otvorené texty končili len časťou slova a dešifrovanie neúplného slova nie je vždy stopercentné.

2. test

Na druhé testovanie som použil 60 otvorených textov dĺžky 37 znakov. Texty som povyberal zo serverov slovenských denníkov SME a Pravda a niektoré trochu pozmenil, aby mali dĺžku 37 znakov. Z týchto 60 otvorených textov

Zbierka	Priemerná úspešnosť v %	Počet stopercentne dešifrovaných textov (max. 30)	medián v %
SK wikipédia - celá	**91 (94.5)**	**14** (15)	**97** (98.5)
SK wikipédia 100MB bez úpravy	86.8 (93.6)	11 (13)	92 (97)
SK wikipédia 100MB s úpravou	87.97 (94)	12(**16**)	**97 (100)**
Biblia+klasici.sk	80.8 (89.4)	9 (13)	83.5 (97)

Tabuľka 3.3: Úspešnosť dešifrovania

som vyrobil 30 šifrových textov XOR-om i-teho a $i+1$-vého otvoreného textu pre $i = (2k+1), k \in \{0, 1, ..., 29\}$. Takto vytvorené šifrové texty som dešifroval pomocou všetkých štyroch zbierok. Výsledky som počítal ako percentuálnu úspešnosť správne dešifrovaného textu spriemerovanú pre všetkých 30 šifrových textov. Pri veľa textoch sa stalo, že po dešifrovaní mali otvorené texty časť textu vymenenú medzi sebou. Výsledky sú znázornené v tabuľke 3.3. Výsledok pred zátvorkou je taký, kde som ako správnu bral len tú časť textu, ktorá sa presne zhodovala s pôvodným otvoreným textom, v zátvorke udávam výsledok, kde som takúto výmenu nebral do úvahy a považoval šifrový text za korektne dešifrovaný. V treťom stĺpci je medián postupnosti, ktorá vznikla zoradením dešifrovaných textov podľa úspešnosti dešifrovania. Najlepšie výsledky sú v tabuľke zvýraznené hrubo.

Najlepšiu priemernú úspešnosť dešifrovania som dosiahol pri zbierke jedna, pretože táto zbierka bola najväčšia. Ale najviac stopercentne dešifrovaných textov, ak som výmenu medzi otvorenými textami nebral do úvahy, som dosiahol pri zbierke tri. Ak som túto výmenu bral do úvahy, tak najlepšie

dopadla zbierka jedna.

Zhodnotenie testov

Pri druhom teste, kde otvorenými textami neboli priamo texty z daných zbierok, robili najväčšie problémy texty ktoré obsahovali skratky politických strán, priezviská alebo cudzie slová. Ak sa skratka nevyskytuje v danej zbierke je jej dešifrovanie ťažšie ako dešifrovanie normálneho slova, lebo pri skratke je výskyt jej podreťazca v zbierke údajov menej pravdepodobný ako u normálneho slova. Podobne je to aj u priezvisk a cudzích slov. V tabuľke 3.4 uvádzam príklady problémových slov spolu so zbierkami, pomocou ktorých boli dešifrované a pozície v otvorenom texte, na ktorých sa tieto problémové reťazce nachádzajú.

Problémy s dešifrovaním mal program aj so začiatkom dešifrovaných textov, lebo dešifrovanie konkrétneho znaku závisí aj od predchádzajúcich znakov a čím je menej predchádzajúcich znakov, tým je dešifrovanie zložitejšie. Riešenie tohto problému som naznačil v časti 3.4.

3.3 Rozdiel v dešifrovaní slovenských a anglických textov

Slovenský jazyk na rozdiel od anglického obsahuje znaky s mäkčeňmi a znaky s diakritikou ako ô, ä, ž, ľ, ť, atď. Týmto je dešifrovanie textu zložitejšie, lebo daný šifrový znak mohol potenciálne vzniknúť z viacerých dvojíc znakov ako v anglickom jazyku.

Ďalším rozdielom je, že v anglickom jazyku má jedno slovo veľakrát pod-

Pôvodný reťazec	Dešifrovaný reťazec	Číslo zbierky	Pozícia reťazca v texte
”SNS” ”sú ”	”. S” ”.” ”	4.zbierka	Začiatok
”70 percent Barmčanov” ”vládnuca vojenská ju”	”5. der a modelubínov” ”trápnu eno Lapkám ju”	4.zbierka	Začiatok
”Trištvrte” ” Slotovci”	” Niž jasá” ”Tolk sedí”	4.zbierka	Začiatok
”pokutám za **nafúkanie**” ”podľa návrh**u zákona,**”	”pokutám za **baliszof,**” ”podľa návrh**y prstone**”	3.zbierka	Stred
”šéfa” ”tom ”	”„él ” ”joga”	3.zbierka	Stred
”fascinujú” ”z Oregonu”	”4 Reedový” ”(antimurr”	3.zbierka	Koniec
”**KDH** si chce” ”**Opo**zícia, ”	”**sku** si chce” ”**W_R**zícia, ”	2.zbierka	Začiatok
”Falšovanie” ”**Vládni** politici”	”dbe?üranie” ”**točným** politici”	2.zbierka	Začiatok

Tabuľka 3.4: Príklady problémových reťazcov

statne viac významov ako v slovenčine, čiže na normálnu komunikáciu je v anglickom jazyku potrebná menšia slovná zásoba ako v slovenskom. Preto zbierka dát v slovenskom jazyku vytvorená pre dešifrovanie rovnakého typu textov ako v anglickom jazyku musela obsahovať väčší počet slov.

3.4 Zlepšenia

Najčastejšie odchýlky dešifrovaných textov od originálu boli na začiatku dešifrovaných textov. Tento problém by sa dal riešiť spätným dešifrovaním.

Najprv v danom dešifrovanom texte nájdem prvé, správne dešifrované slovo, napr. porovnávaním dešifrovaných slov so slovami nejakého slovníka. Potom budem dešifrovať od tohto prvého, správne dešifrovaného slova a budem postupovať smerom na začiatok. Keďže koniec bol dešifrovaný správne a každá časť textu závisí od predchádzajúcej, potom týmto spôsobom môže dôjsť k opraveniu nesprávne dešifrovaného začiatku textu. Pre tento postup dešifrovania je potrebný aj nový model, ktorý je vytvorený z reverzov slov daného jazyka.

Kapitola 4

Záver

V tejto práci som sa venoval riešeniu two-time pad problému, opísal som metódu z článku [7], ktorá rieši tento problém na základe pravdepodobnosti výskytu reťazcov v prirodzenom jazyku a implementoval túto metódu pre slovenský jazyk. Tento prístup kryptoanalýzy som otestoval a výsledky som zhrnul do viacerých tabuliek. V poslednej časti som navrhol možné zlepšenia tejto metódy.

Prínos tejto bakalárskej práce vidím v poukázaní na tento veľký bezpečnostný problém Vernamovej šifry, ktorý sa v reálnych systémoch stále podceňuje. Kryptoanalýza Vernamovej šifry pomocou tejto metódy nie je zložitá, čím stráca použitie šifrovania v systémoch so znovupoužitím kľúča svoj hlavný význam, ktorým je ochrana informácií.

Literatúra

[1] Gilbert vernam. Website. `http://en.wikipedia.org/wiki/Gilbert_Vernam`.

[2] One-time pad. Website. `http://en.wikipedia.org/wiki/One-time_pad`.

[3] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: The insecurity of 802.11. `http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html`, 2001.

[4] Bob Carpenter. Scaling High-Order Character Language Models to Gigabytes. In *Association for Computational Linguistics Workshop on Software*. Ann Arbor, MI, 2005.

[5] E. Dawson and L. Nielsen. Automated cryptanalysis of xor plaintext strings. In *Cryptologia*, April 1996.

[6] T. Kohno. Attacking and repairing the winyip encryption scheme. In *ACM Conference on Computer and Communications Security*, pages 72–81, Oct. 2004.

[7] Joshua Mason, Kathryn Watkins, Jason Eisner, and Adam Stubblefield. A natural language approach to automated cryptanalysis of two-time pads. In *ACM Conference on Computer and Communications Security*, pages 235–244, 2006.

[8] Martin Stanek. Absolútne bezpečná šifra. `http://www.dcs.fmph.uniba.sk/~stanek/crypto/shann.pdf`, 2003.

[9] Martin Stanek. Základy kryptológie. `http://www.dcs.fmph.uniba.sk/~stanek/crypto/main2.pdf`, 2004.

[10] Hongjun Wu. *The Misuse of RC4 in Microsoft Word and Excel*. Cryptology ePrint Archive, Report 2005/007, 2005. `http://eprint.iacr.org/`.

Dodatok A

Príloha

K bakalárskej práci prikladám CD, na ktorom je napálený program opisovaný v 2. kapitole. Súčasťou CD sú aj zdrojové kódy tohto programu.