KRY
Vigenerova šifra

Peter Grofčík

Marec 2021

1 Popis implementovaného balíčka

Balíček pozostáva z troch a hlavných zdrojových súborov:

- friedman.cpp - implementácia Friedmanovho testu,
- kasisk.cpp - implementácia Kasiského testu,
- main.cpp - main a pomocné funkcie pre získanie reálnej dĺžky hesla a hesla samotného.

2 Friedmanov test

$$\mathrm{IC} = \frac{1}{N(N-1)} \sum_{i=1}^{n} F_i(F_i - 1)$$

Metóda je založená na indexe koincidencie vypočítaného pomocou zobrazeného vzorca, kde:

- N predstavuje dĺžku skúmaného vstupu,
- F_i predstavuje počet výskytov jednotlivých písmen z použitej/vstupnej abecedy.

Pre výpočet indexu vrámci implementácie slúži funkcia *getIndexOfCoin*, ktorá je oddelená od samotnej funkcie pre odhad dĺžky hesla pomocou friedmana hlavne z dôvodu znovupoužitia pri výpočte reálnej dĺžky hesla. Pre samotný odhad dĺžky hesla slúži nasledujúci vzorec, ktorý využíva pravdepodobnosť, že ddve ľubovoľne zvolené písmená sú rovnaké a pravdepodobnosť koincidencie z uniformej náhodnej selekcie zo vstupnej abecedy jazyka.

$$\frac{\kappa_p - \kappa_r}{\kappa_o - \kappa_r}$$

Kde (pre anglický jazyk):

- K_p predstavuje hodnotu 0.0655,
- K_r predstavuje hodnotu 1/26,
- K_o predstavuje výstup z funkcie *getIndexOfCoin*, a teda vypočítaný index koincidencie.

3 Kasiského test

Metóda je založená na výpočte najväčšieho spoločného deliteľa vzdialeností výskytu indentických úsekov zašifrovaného textu. V implementácii tak vo funkcii *kasis* zaradom dohľadávam úseky textu o dĺžke 3 a viac, ktoré sa v zašifrovanom texte nachádzajú viac ako jeden krát. Z definície zo skrípt nie je úplne jasné koľko takýchto identických častí je potrebné nájsť, či podľa čoho si tie správne programovo vybrať, preto metódu aplikujem na dohľadanie všetkých možných. Z takto dohľadaných výskytov pre jednotlivé úseky vypočítam vzdialenosti medzi nimi a pomocou funkcie *findDivider* vypočítam najvačší spoločný deliťel pre poskytnuté vzdialenosti.

Spomenutý výpočet sa opakuje pre každý úsek textu, ktorý sa vo vstupe nachádza viac ako jeden krát pričom si ukladám počet jednotlivých najvačších deliteľov a to za účelom výberu najfrekventovanejšieho znich, keďže prejdením všetkých možných identických úsekov nie je možné nájsť jeden spoločný najväčší deliteľ a to hlavne z dôvodu náhodného výskytu identity v zašifrovanom texte. Zo zistených počtov tak jednoducho vyberiem najfrekventovanejší najväčší spoločný deliteľ, ktorý predstavuje odhadnutú dĺžku hesla pre kasiskiho metódu.

Pri výbere všetkých možných nájdených úsekov však stále dochádza k veľkému počtu náhodných výskytov pri väčšom vstupe (testované na 10k znakoch), tak to v značnej miere už pri menších heslách zkreslí výsledok na výslednú veľkosť hesla v rozmedzí $< 1, 4 >$, čo sa podľa informácii v skriptách nedá považovať za relevantnú dĺžku hesla a pre dané veľkosti by bol efektívnejší brute-force attack. Preto som sa rozhodol pri výbere ignorovať tieto dĺžky a implementovaná metóda je tak schopná odhadovať heslá od dĺžky 5 a vyššie. Vďaka tomu je metóda schopná vo väčšine prípadov odhadnúť správnu dĺžku hesla do veľkosti +-150 za cenu ignorácie veľmi malého odhadu pre 3 najmenšie dĺžky hesla. Nie som si istý či sa jedná o úpravu metódy alebo len mierne prispôsobenie výsledku, keďže v príklade zo skrípt bola tiež vyradená dĺžka 3 ako "unlikely to have", avšak pre testovanie do dĺžok +-100 hesla mi prišla logická.

4 Odhad reálnej dĺžky hesla

Pre určenie reálnej dĺžky som použil vylepšenú verziu friedmanovej metódy (funkcia *find_real*), kedy k výpočtu indexu koincidencie nie je použitý celý vstupný text ako celok ale dôjde k rozdeleniu vstupného zašifrovaného textu na bloky, ktorých počet odpovedá dĺžke kľúča, ktorá mohla byť použitá pre zašifrovanie textu. Každý blok vždy obsahuje postupnosť vždy $k*i$-teho písmena, kde **i** predstavuje index bloku a **k** patrí intervalu:

$$< 0, \text{dĺžka vstupu / počet blokov} >.$$

K rozdeleniu dochádza niekolko krát a to v indervale:

$$< 2, 10 * \text{odhad z friedmana} >^{1},$$

čím skúmame niekoľko dĺžok hesla, ktoré mohli byť použité pri šifrovaní.

Pre každý takto vytvorený blok dôjde k výpočtu indexu koincidencie z friedmanovho testu a následnému ortátaniu hodnoty kappa (0.0655) z čoho následnou absolútnou hodnotou dostaneme rozdiel vypočítanej hodnoty indexu oproti hodnote koincidencie anglického jazyka na danom bloku. Z tejto hodnoty zo všetkých blokov následne vypočítame priemernú hodnotu a odhad dĺžky hesla tak predstavuje počet blokov, na ktorý text delíme a pri ktorom výsledná priemerná hodnota indexu výde najmenšia, čo predstavuje najviac pravdepodobnú dĺžku hesla, ktorá bolá použitá pre zašifrovanie vstupného textu.

5 Určenie hesla

K určeniu hesla používam podobný princíp ako pri odhade reálnej dĺžky hesla. Opäť tak dôjde k rozdeleniu vstupného textu na bloky, avšak teraz už len jeden krát a to pre bloky o počte odhadnutej reálnej dĺžky hesla. Za predpokladu že odhadnutá dĺžka hesla je správna tak toto rozdelenie vygeneruje bloky, z ktorých každý je zašifrovaný Caesarovou šifrou o premenlivej dĺžke kľúča, čo znamená že každý blok je zašifrovaný jedným písmenom z kľúča použitého pre zašifrovanie celého textu.

Pre každý blok je tak nutné zistiť počet posunov vrámci anglickej abecedy (resp. písmeno ktoré určuje počet posunov o koľko bolo každé písmeno v riadku posunuté doprava lexikologicky). Pre následné zistenie správneho posunu som použil metódu *Chi-squared test*.

$$X^2 = \sum_{i=1}^{k} \frac{(x_i - m_i)^2}{m_i}$$

Kde:

- x_i predstavuje frequenciu výskytu jednotlivých písmen vo vstupnom texte,

[1] Zvolený interval je rozsiahli z dôvodu, že v extrémnych prípadoch pri testovaní výstup z friedmanovho testu nadobúdal extrémne nízku ale aj extrémne vysokú hodnotu oproti reálnej dĺžke použitého kľúča

- m_i predstavuje frequenciu výskytu jednotlivých písmen v anglickom jazyku (v kóde tabulka *EnglishFrequency*),
- k predstavuje počet písmen v abecede (26 pre angličtinu).

Z princípu tak v kóde na jednotlivých blokoch cyklicky lexikologicky rotujem každé písmeno pomocou pomocnej funkcie *leftShift* a pre každú verziu bloku zistím hodnotu z chi-squared testu, pričom počet posunov vykonaných pri šifrovaní daného bloku je určený posunutým blokom s najmenšou hodnotou na výstupe chi-squared testu. Takto zistený počet posunov slúži ako index písmena z abecedy, ktorým bol daný blok zašifrovaný a pre vyskladanie kľúča použitého pre zašifrovanie celého vstupu ich stačí poskladať zasebou podľa indexu jednotlivých blokov.