PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

DIPLOMOVÁ PRÁCE

Kryptoanalýza historických šifer

2010 Lukáš Havrlant

Místopříseżně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně.

25. květen Lukáš Havrlant

Anotace

Automatizované luštění šifer počítačem dnes může plně nahradit těžkou a zdlouhavou práci historických kryptoanalytiků. V práci se zabývám návrhem a realizací algoritmů, které dokáží v řádu sekund automaticky rozluštit zprávu zašifrovanou některou z vybraných historických šifer.

Děkuji RNDr. Miroslavu Kolaříkovi, Ph.D. za vedení této bakalářské práce a za rady při konzultacích.

Obsah

Seznam obrázků

Seznam tabulek

1. Slovníček pojmů

V tomto textu se zabývám výhradně historickými šiframi. Některé definice tak byly upraveny na úkor obecnosti, aby lépe popisovaly vztah k těmto šifrám.

1.1. Kryptologie

Kryptologie je věda zabývající se obecně šifrováním.

1.2. Kryptografie

Kryptografie je podobor kryptologie a zabývá se návrhem a realizací šifrovacích algoritmů.

1.3. Kryptoanalýza

Kryptoanalýza je podobor kryptologie a zabývá se prolamováním a luštěním šifrovacích algoritmů.

1.4. Otevřený text

Otevřený text je libovolný text, který chce uživatel zašifrovat. Obyčejně se jedná o nějakou zprávu v běžném jazyce, která by se neměla dostat do rukou každému, ale typicky pouze vybrané skupině jiných uživatelů. V této práci bude otevřený text vždy psán malými písmeny.

1.5. Šifrový text

Šifrový text je takový text, který v sobě ukrývá původní otevřený text, ale čtenář není schopný tento otevřený text z šifrového textu bez dalších dodatečných informací získat. Šifrový text může čtenáři připadat jako náhodná posloupnost znaků. V této práci bude šifrový text vždy psán velkými písmeny.

1.6. Šifrovací klíč

Šifrovací klíč je libovolná posloupnost písmen (často reálná slova), která spolu s použitou šifrou specifikuje, jakým způsobem se má otevřený text proměnit na šifrový text a naopak – jak z šifrového textu dostat zpět otevřený text.

Většina šifer obsahuje i klíče, které jsou sice platným vstupem do algoritmu, nicméně výsledný šifrový text je buď identický s otevřeným textem nebo je text velice podobný a na první pohled odhalitelný. Takové klíče se nazývají *slabé klíče*.

1.7. Šifra, šifrování

Šifra je algoritmus, který bere na vstupu otevřený text a šifrovací klíč a na výstupu vrátí šifrový text. Cílem šifry pak je vytvořit zašifrovanou zprávu, kterou bude schopný přečíst každý, komu je zpráva určena, ale nikdo nepovolaný. V praxi to znamená, že příjemce zašifrované zprávy musí znát jednak použitou šifru a dále použitý šifrovací klíč, aby byl schopen si přečíst původní otevřený text. Pro všechny ostatní by mělo být nemožné bez těchto informací otevřený text získat.

Přestože příjemce potřebuje znát jak použitou šifru, tak použitý klíč, obyčejně se nestaví zabezpečení zprávy na neznalosti použité šifry, ale hlavně na neznalosti klíče. I pokud zná útočník veškeré detaily o šifře, neměl by být schopný rozluštit ze šifrového textu původní otevřený text, pokud nezná klíč.

Poznámka: Šifrování má blízko ke steganografii, což je věda zabývající se skrytím komunikace tak, aby si ostatní pozorovatelé ani nevšimli, že vůbec nějaká komunikace probíhá. Zpozorování komunikace se rovná prolomení steganografie. Naopak zašifrovanou zprávu můžeme poslat i nezabezpečenými veřejnými cestami, bezpečnost dat je zajištěna utajením klíče.

Opačný proces k šifrování je dešifrování.

V praxi existují dvě základní techniky šifrování: substituční a transpoziční.

1.7.1. Substituční šifra

Substituční šifra provádí záměny částí textu za jiné části textu podle nějakého klíče. Časté jsou například záměny jednotlivých písmen za jiná písmena, místo „a“ můžeme v šifrovém textu napsat „b“. Při dešifrování pak obvykle provádíme opačný postup, místo „b“ píšeme „a“.

Dále můžeme rozlišovat, s kolika šifrovými abecedami šifra pracuje. Při použití jedné abecedy platí, že každé písmeno má pouze jednu substituci. Pokud použijeme více abeced, má každé písmeno více možných substitucí, přičemž kterou substituci zrovna použijeme rozhoduje šifrovací klíč. Šifra která používá jednu abecedu se nazývá *monoalfabetická*; šifra která používá více abeced se nazývá *polyalfabetická*.

1.7.2. Transpoziční šifra

Transpoziční šifra mění pořadí znaků v otevřeném textu. V šifrovém textu zůstávají všechna písmena, která jsou i v otevřeném textu, ale jsou zpřeházena podle nějakého klíče. Primitivní transpoziční šifrou pak může být například napsání textu pozpátku.

1.7.3. Kombinované šifry

Další typy šifer zpravidla vznikají kombinací těchto dvou typů šifer. I nejmodernější šifry jako DES či AES jsou založeny na složitých kombinací transpozic a substitucí.

1.8. Alice, Bob a Eva

Alice a Bob jsou dva lidé, uživatelé, kteří si chtějí navzájem poslat zprávu a typicky nechtějí, aby si obsah zprávy mohl přečíst někdo jiný. Alice posílá zprávu Bobovi. Eva je potenciální útočník, který se může dostat ke zprávě, přečíst si ji a případně ji rozluštit. Eva je pouze pasivní útočník, nemůže měnit obsah zprávy.

1.9. Obraz a vzor písmene

V substitučních šifrách je obraz písmene takové písmeno, na které se zobrazí písmeno z otevřeného textu. Pokud platí substituce a→b, písmeno „b“ je obrazem písmene „a“. Naopak písmeno „a“ je vzorem písmene „b“. Podobně můžeme definovat i obraz respektive vzor slova atp.

1.10. Abeceda

Pod pojmem abeceda budeme rozumět standardní anglickou abecedu a-zA-Z. Přestože je teoreticky i technicky možné, abychom při šifrování používali národní abecedy, obyčejně bývá zvykem používat právě anglickou abecedu. Algoritmy popsané v této práci jsou nicméně obecné a zvládly by bez dalších úprav přechod na národní abecedu.

2. Popis šifer

2.1. Základní definice

2.1.1. Kryptosystém

Kryptosystém je pětice $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, která splňuje následující podmínky:

1. $\mathcal{P}$ je konečná množina všech možných otevřených textů;
2. $\mathcal{C}$ je konečná množina všech možných šifrových textů;
3. $\mathcal{K}$ je konečná množina všech možných šifrovacích klíčů;
4. Pro každé $K \in \mathcal{K}$ existuje šifrovací pravidlo $e_K \in \mathcal{E}$ a odpovídající dešifrovací pravidlo $d_K \in \mathcal{D}$. Všechna $e_K : \mathcal{P} \to \mathcal{C}$ a $d_K : \mathcal{C} \to \mathcal{P}$ jsou zobrazení, pro která platí $d_K(e_K(x)) = x$ pro každý otevřený text $x \in \mathcal{P}$.

Nejdůležitější bod je čtvrtý – říká nám, že pokud zašifrujeme text pomocí nějakého klíče, pak při dešifrování šifrového textu se stejným klíčem získáme původní otevřený text.

2.1.2. Proces šifrování

Na začátku si musí Alice s Bobem nějakým bezpečným kanálem předat klíč $K \in \mathcal{K}$. Následně chce Alice poslat šifrovanou zprávu Bobovi. Tato zpráva už nemusí jít bezpečným kanálem. Předpokládáme, že zpráva je řetězec ve tvaru:

$$x = x_1 x_2 \dots x_n,$$

kde $n \in \mathbb{N} \wedge n \geq 1$ a $x_i \in \mathcal{P}, 1 \leq i \leq n$. Každé x_i je pak zašifrováno pomocí e_K. Alice spočítá

$$y_i = e_K(x_i), 1 \leq i \leq n.$$

Výsledný šifrový text y Alice získá zřetězením dílčích výsledků:

$$y = y_1 y_2 \dots y_n.$$

Tento řetězec je poslán nezabezpečeným kanálem. Když Bob přijme zprávu, dešifruje ji:

$$x_i = d_K(y_i), 1 \leq i \leq n$$

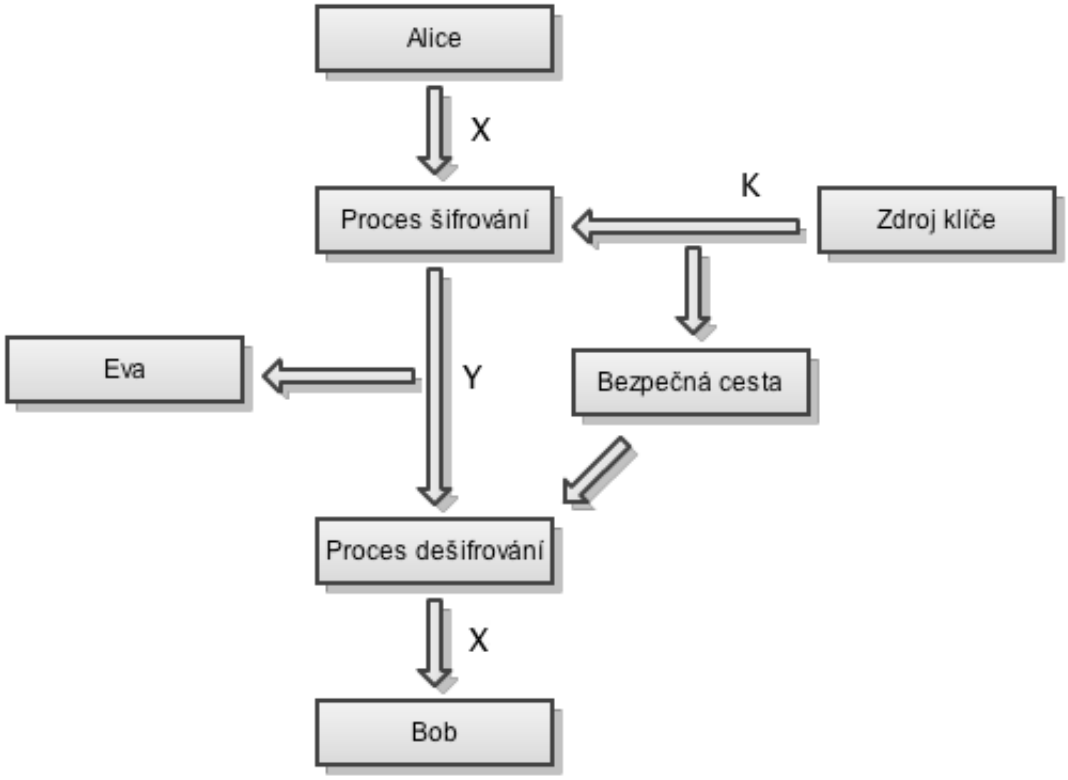
a původní otevřenou zprávu získá zřetězením:

$$x = x_1 x_2 \dots x_n.$$

Zobrazení e_K musí být zřejmě injektivní, má-li být šifrování jednoznačné. Pokud by totiž platilo

$$y = e_K(x_1) = e_K(x_2),$$

kde $x_1 \neq x_2$, nebylo by jasné, kterou z těchto dvou zpráv Alice Bobovi poslala. Celý proces šifrování znázorňuje diagram na obrázku 1.



Obrázek 1. Komunikace šifrovým kanálem

2.1.3. Modulární aritmetika

Nechť $a, b \in \mathbb{N}_0$ a $m \in \mathbb{N}$. Pak píšeme

$$a \equiv b \,(\mathrm{mod}\, m),$$

pokud m dělí $b - a$. Čteme „a je kongruentní s b modulo m“. Notaci bez závorek $a \bmod m$ budeme používat k označení zbytku po celočíselném dělení, tj. pokud v tomto případě platí $a = x \cdot m + r \wedge r < m; x, r \in \mathbb{N}_0$, pak $a \bmod b = r$.

Definujme okruh $\mathbb{Z}_m$: $\mathbb{Z}_m$ je množina $\{0, 1, \ldots, m-1\}$ s definovanými operacemi sčítání $+$ a násobení $\times$. Tyto operace fungují stejně jako běžné operace sčítání a násobení, pouze s tím rozdílem, že výsledek je redukován modulo m. Máme-li čísla $a, b \in \mathbb{N}_0$ a $a_m, b_m \in \mathbb{Z}_m$ taková, že $a = a_m \wedge b = b_m$, pak platí:

$$a_m + b_m = (a + b) \bmod m$$

a stejně tak:

$$a_m \times b_m = (a \cdot b) \bmod m.$$

Příklad: Pokud bychom měli okruh $\mathbb{Z}_{10}$ a chtěli vypočítat 7×9, vypočítáme nejprve $7 \cdot 9 = 63$ a následně provedeme modulo: $63 \bmod 10 = 3$. Výsledek je tři.

Protože $\mathbb{Z}_m$ je okruh, musí pro každé číslo $x_m \in \mathbb{Z}_m$ existovat i inverzní číslo $x'_m \in \mathbb{Z}_m$, pro které platí $x_m + x'_m = 0$. Protože se nepohybujeme v záporných číslech, nemůžeme k číslu x přičíst jeho záporný ekvivalent, ale můžeme využít modularity a přičíst k x_m takové číslo x'_m, které dá v součtu číslo m, protože platí $m \bmod m = 0$. Inverzní prvek $x'_m \in \mathbb{Z}_m$ k prvku $x_m \in \mathbb{Z}_m$ můžeme definovat jako

$$x' = m - x,$$

kde $x, x' \in \mathbb{N}_0$ a zároveň $x = x_m \wedge x' = x'_m$. Tímto způsobem můžeme realizovat odečítání v $\mathbb{Z}_m$:

$$(a_m - b_m) \bmod m = (a_m + b'_m) \bmod m.$$

Přestože následující šifry pracují výhradně s písmeny anglické abecedy, budeme v definicích používat pouze čísla. Převod čísel na písmena a naopak realizujeme jednoduchou substitucí $a \leftrightarrow 0, b \leftrightarrow 1, c \leftrightarrow 2, \ldots, z \leftrightarrow 25$.

Definice zavedené v kapitole 2.1. pochází z knihy [5].

2.2. Caesarova šifra

2.2.1. Definice

Kryptosystém 1. Nechť $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$. Pro $0 \leq K \leq 25$ a $x, y \in \mathbb{Z}_{26}$ definujeme

$$e_K(x) = (x + K) \bmod 26$$

a

$$d_K(y) = (y - K) \bmod 26.$$

— Citováno z [5].

2.2.2. Historie

Caesarova šifra je jednoduchá substituční šifra pojmenována po římském vojevůdci Juliu Caesarovi, který tuto šifru používal. Jedná se o vůbec první dokumentovaný záznam použití substituční šifry pro vojenské účely. Zmínka o ní se nachází v publikaci *Zápisky o válce galské*, ve které Caesar popisuje, jak poslal zprávu Ciceronovi, který byl obklíčen nepřítelem:

> „*Posel dostal rozkaz, at' vhodí kopí s připevněnou zprávou přes hradby tábora, pokud by se nemohl dostat dovnitř. Tak se i stalo. Gal, vystrašený možným nebezpečím, mrštil kopí. Nešt'astnou náhodou se stalo, že se kopí zabodlo do věže. Teprve třetího dne si ho povšiml jeden z vojáků, který kopí sejmul a zanesl Ciceronovi. Ten si přečetl zprávu a poté ji oznámil svým vojákům, což všem přineslo velikou radost.*“

— Citováno z [4].

2.2.3. Popis šifry

Caesarova šifra je speciálním případem monoalfabetické substituční šifry, viz kapitola 2.3. Na začátku šifrování si zvolíme klíč, který nám bude říkat, o kolik písmen posuneme každé písmeno v otevřeném textu. Tento klíč bude reprezentován jedním písmenem abecedy, přičemž platí, že písmeno „a“ značí posun o nula písmen, písmeno „b“ o jedno písmeno atd. Výsledný šifrový text získáme tak, že každé písmeno v otevřeném textu posuneme o tolik písmen dál, kolik určuje klíč. Máme-li posunovat o tři písmena (klíčem je písmeno „d“), tak pokud máme v otevřeném textu písmeno „a“, v šifrovém textu bude písmeno „D“, dále pak b→E, c→F, ... Přehledně to zobrazuje následující tabulka, v prvním řádku je otevřená abeceda, ve druhém šifrová:

```
abcdefghijklmnopqrstuvwxyz
DEFGHIJKLMNOPQRSTUVWXYZABC
```

Jak je vidět, klíč zároveň udává, na které písmeno se v šifrovém textu zobrazí písmeno „a“.

Příklad: text „prsi a venku se setmelo“ by s klíčem „k“ dal při použití Caesarovy šifry výsledek „ZBCS K FOXUE CO CODWOVY“.

2.2.4. Výhody

- Šifra je především jednoduchá a rychlá. Zašifrovat text Caesarovou šifrou zvládne každý i bez jakékoliv dodatečné pomůcky.

2.2.5. Nevýhody

- Šifra obsahuje pouze velmi omezené množství klíčů. Anglická abeceda obsahuje 26 písmen, které můžeme použít jako klíč. Nicméně pokud použijeme jako klíč písmeno „a“, šifrový text se nijak nezmění, jedná se o slabý klíč. V úvahu připadá pouze 25 písmen, které můžeme použít jako šifrovací klíč. Takové množství klíčů je velmi malé a potenciální útočník je schopen ručně vyzkoušet všechny klíče v řádu minut.
- Výsledný šifrový text si uchovává informace o četnosti písmen v otevřeném textu. Pokud se například v otevřeném textu vyskytlo písmeno „e“ celkem stokrát, v šifrovém textu musí být nějaké písmeno, které se tam také vyskytuje stokrát.
- Šifrový text zachovává vzdálenosti mezi písmeny. Pokud jsou v otevřené abecedě písmena „a“ a „e“ od sebe vzdálena o čtyři písmena, budou jejich obrazy v šifrové abecedě také vzdáleny o čtyři písmena.

Zdroje ke kapitole 2.2.: [7] a [4].

2.3. Monoalfabetická substituční šifra

2.3.1. Definice

Kryptosystém 2. Nechť $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}$. Množina $\mathcal{K}$ obsahuje všechny permutace skupiny 26 symbolů $0, 1, 2 \ldots, 25$. Pro každou permutaci $\pi \in \mathcal{K}$ definujme

$$e_\pi(x) = \pi(x),$$

a

$$d_\pi(y) = \pi^{-1}(y),$$

kde π^{-1} je inverzní permutace k π.

— Citováno z [5].

2.3.2. Popis šifry

Tato šifra funguje podobně jako Caesarova šifra, pouze nemá nijak omezené možné substituce. Na začátku si musíme vytvořit tabulku substitucí, která nám bude určovat, na co se mají jednotlivá písmena z otevřeného textu zobrazit. Zobrazení musí být jednoznačné, aby bylo následně jednoznačné i dešifrování. Tabulka substitucí může vypadat například takto:

```
abcdefghijklmnopqrstuvwxyz
MUNBQPEKVZAJTXRCWSYLGIFDOH
```

V první řadě je opět otevřená abeceda, ve druhé řadě šifrová abeceda. Ta je poskládána náhodně (respektive jak zrovna chceme), není třeba dodržovat pravidlo posunutí jako u Caesarovy šifry. Výsledný šifrový text vznikne záměnou písmen z otevřené abecedy za jejich obrazy v šifrové abecedě. V tomto případě bychom místo písmene „a“ napsali písmeno „M“, dále b→U, c→N, ... Dešifrování by probíhalo stejně, pouze s obrácenou tabulkou substitucí, tj. N→c, ...

Příklad: text „tato noc nebude kratka“ by s klíčem „munbqpekvzajtxrcwsylgifdoh“ dal při použití monoalfabetické šifry výsledek „LMLR XRN XQUGBQ ASMLAM“.

Aby si uživatel nemusel pamatovat celou přeházenou abecedu, je možné zvolit si heslo, pomocí něhož se vygeneruje požadovaný klíč. Heslem může být libovolné slovo nebo věta, klíč se poté vygeneruje tak, že se z hesla odstraní všechny vícenásobné výskyty písmen – nechá se pouze první výskyt písmene – a doplní se chybějící písmena v abecedním pořadí. Pokud máme heslo „informatika“, po odstranění zdvojených písmen dostáváme „informatk“ a po doplnění písmen získáme klíč „informatkbcdeghjlpqsuwxyz“.

2.3.3. Výhody

- Šifra je jednoduchá, její použití není o moc složitější než u Caesarovy šifry. Po vytvoření tabulky substitucí je použití prakticky stejné.

- Velký počet klíčů. Největší slabinou Caesarovy šifry byl nízký počet klíčů (26), monoalfabetická substituční šifra používá jako klíč všechny permutace anglické abecedy, existuje tedy celkem 26! možných klíčů, což je přibližně $4 \cdot 10^{26}$ klíčů. Pokud by hypotetický počítač byl schopen ověřit miliardu klíčů za sekundu, trvalo by celkem dvanáct miliard let, než by prověřil všechny možné klíče. Jestli by tento počítač testoval pro každou zašifrovanou zprávu všech 26! klíčů, tak začal by-li s testováním okamžitě při vzniku vesmíru, luštil by dnes teprve druhou zašifrovanou zprávu. Některé z těchto klíčů jsou ovšem slabé klíče – to jsou takové klíče, které mají mnoho substitucí ve tvaru x→x. I po odstranění těchto klíčů zůstává množina možných klíčů dostatečně velká.

- Na rozdíl od Caesarovy šifry nemusí být šifrová abeceda jen posunutá otevřená abeceda, takže šifra netrpí nevýhodou v podobě stejné vzdálenosti písmen.

2.3.4. Nevýhody

- Stejně jako u Caesarovy šifry si šifrový text s sebou nese informace o frekvenci jednotlivých písmen. Při útoku můžeme využít klasickou frekvenční analýzu, ve které budeme zkoumat procentuální výskyt jednotlivých znaků, bigramů, trigramů apod. Pokud text obsahuje mezery a jsou tak zřejmá jednotlivá zašifrovaná slova, můžeme též využít frekvence písmen na začátku a na konci slov. Můžeme využít i další vztahy daného jazyka, například v češtině se často za souhláskami vyskytuje méně různých písmen než za samohláskami.

- Obsahuje-li šifrový text mezery, můžeme zkusit najít taková slova, která mají i po zašifrování stejný *kanonický tvar*, viz kapitola 3.5.2. I po zašifrování si totiž slovo uchovává informace o vztazích mezi obsaženými písmeny. Slovo v otevřeném textu, které začíná i končí na stejné písmeno, bude začínat a končit na stejné písmeno i v zašifrovaném textu. A podobně pro všechny dvojice písmen ve slově. Pokud máme k dispozici slovník daného jazyka, můžeme nalézt taková *unikátní slova*, která mají v rámci slovníku zcela unikátní kanonický tvar. Pokud pak tento tvar nalezneme i v šifrovém textu, bude to pravděpodobně ono *unikátní slovo*. Tento postup lze v omezené míře aplikovat i pokud neznáme hranice mezi slovy.

2.3.5. Jak zvýšit bezpečnost šifry

Šifru lze některými úpravami otevřeného textu vylepšit bez zásadních úprav šifrového algoritmu:

- Obě popsané nevýhody staví na tom, že text bude vykazovat známky běžného textu daného jazyka. Pokud upravíme otevřený text tak, že se nebude tvářit jako běžný text, ztížíme použití kryptoanalytických metod. Úpravy ovšem musí být takového charakteru, aby příjemce zprávy byl stále schopen text přečíst. V úvahu tak připadají různé „gramatické chibi a přkleepi“, které útočníkovi ztíží práci, ale příjemce text pochopí.

- Účinné vylepšení je používání *nomenklátorů*, což jsou zástupné symboly pro určitá slova. Bob s Alicí se mohou domluvit na tom, že určitá specifická a důležitá slova nahradí speciálním symbolem nebo obrázkem. Například „kostel“ bude obrázek kytary a „továrna“ obrázek nosorožce. Otevřený text „Mrtvolu zakopeme za továrnou.“ by se šifroval tak, že část „Mrtvolu zakopeme za“ by se zašifrovala monoalfabetickou šifrou podle zvoleného klíče a na místo továrny by se nakreslil obrázek nosorožce. I pokud by Eva zjistila šifrový klíč, stejně by nevěděla, kam chtějí Bob s Alicí mrtvolu zakopat.

- Při šifrování může Alice vložit do textu klamače neboli nuly. To jsou takové symboly, které nemají žádný význam, Bob ví, že nemají žádný význam, ale

útočníka zmatou, protože se jeví jako reálné písmeno. Bob při dešifrování zprávy klamače automaticky zahazuje, ale Eva musí nejprve nějakým způsobem zjistit, které symboly jsou klamače a které symboly jsou reálná (zašifrovaná) písmena. Toto vylepšení je velmi efektivní, ale čím více klamačů zpráva obsahuje, tím více narůstá její velikost.

Zdroje ke kapitole 2.3.: [3], [4] a [8].

2.4. Vigenèrova šifra

2.4.1. Definice

Kryptosystém 3. Nechť $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_{26})^m$. Pro klíč $K = (k_1, k_2, \ldots, k_m) \in \mathcal{K}$ definujme

$$e_K(x_1, x_2, \ldots, x_m) = (x_1 + k_1, x_2 + k_2, \ldots, x_m + k_m)$$

a

$$d_K(y_1, y_2, \ldots, y_m) = (y_1 - k_1, y_2 - k_2, \ldots, y_m - k_m),$$

kde jsou všechny operace prováděny v $\mathbb{Z}_{26}$.

— Citováno z [5].

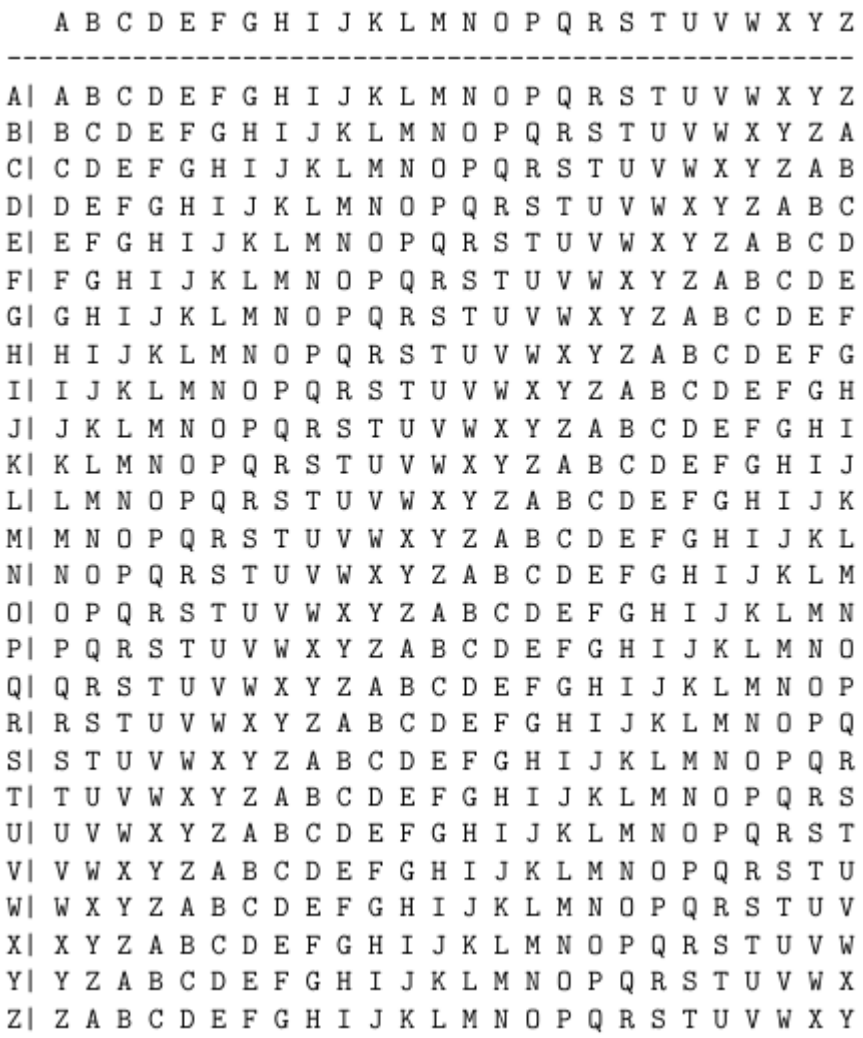
2.4.2. Historie

Roku 1587 nechala královna Alžběta pro údajnou podporu katolické opozice popravit Marii Stuartovnu. K tomuto činu přispělo i to, že Marie Stuartovna používala pro svou korespondenci mírně vylepšenou monoalfabetickou šifru, kterou bohužel pro Marii královnini luštiči prolomili a rozluštěná šifra sloužila jako jeden z usvědčujících důkazů. Bylo jasné, že kryptografové musí vymyslet novou, silnější šifru, protože monoalfabetická substituční již nestačila.

Základní koncept Vigenèrovy šifry vymyslel v 60. letech 15. století florentský polyhistor Leon Battista Alberti, který jako první objevil a sepsal myšlenku polyalfabetické šifry – šifry, která využívá více než jednu šifrovou abecedu. Konečnou podobu ale vymyslel francouzský diplomat Blaise de Vigenère. Ve své době se šifře přezdívalo *Le chiffre indèchiffrable* (česky: nerozluštitelná šifra).

2.4.3. Popis šifry

Vigenèrova šifra je polyalfabetická substituční šifra, která používá až 26 šifrových abeced. Klíč tvoří libovolně dlouhé slovo, nicméně klíč by neměl obsahovat pouze jedno písmeno, protože by se pak šifra degradovala na Caesarovu



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Obrázek 2. Vigenèrův čtverec

šifru. Při šifrování se dále používá Vigenèrův čtverec, což je složitější tabulka substitucí, viz obrázek 2.

Na každém řádku se nachází jedna šifrová abeceda. Kterou šifrovou abecedu zrovna použijeme určuje klíč. V prvním řádku budeme hledat písmena otevřeného textu a v prvním sloupci písmena klíče. Výsledné písmeno šifrové abecedy nalezneme v průsečíku tohoto řádku a sloupce. V otevřeném textu postupujeme jednoduše stále kupředu po jednom písmenku, stejně tak v klíči. Pokud se dostaneme na konec klíče, začneme brát písmena z klíče opět od začátku.

Zašifrujeme text „ahoj“ s klíčem „iq“, budeme postupovat takto: první písmeno v textu je „a“, první písmeno v klíči je „i“. Na průsečíku se nachází písmeno „I“. Dále máme dvojici „h“ a „q“. Na průsečíku je „X“. Nyní jsme se

dostali na konec klíče – budeme brát písmena opět od začátku. To nám dá dvojici „o“ a (první písmeno z klíče) „i“. Ve výsledku „W“. Poslední dvojice je „j“ a „q“, výsledek „Z“. Zašifrovaný text „ahoj“ by byl „IXWZ“. Delší příklad: text „beranka vlku se zachtelo“ by s klíčem „kryl“ dal text „LVPLXBY GVBS DO QYNRKCWY“.

Slabých klíčů má Vigenèrova šifra celou řadu. Pokud je v klíči obsaženo písmeno „a“, bude se text šifrovaný tímto písmenem zobrazovat sám na sebe. Pokud tak bude klíč tvořen pouze písmeny „a“, šifrový text bude identický s otevřeným textem. Pokud bude klíčové slovo složeno jen z jednoho písmene (např. „rrrrrrr“), degradujeme šifru na Caesarovu šifru.

2.4.4. Výhody

- Hlavní vylepšení Vigenèrovy šifry spočívá v použití více abeced, díky čemuž nemůže útočník použít klasickou frekvenční analýzu. Protože se jedno písmeno otevřeného textu může zobrazit na několik různých písmen v šifrovém textu a zároveň jedno písmeno v šifrovém textu představuje několik různých písmen v otevřeném textu, útočníkovi nijak nepomůže informace, kolikrát se v daném šifrovém textu vyskytuje nějaké písmeno. Obdobně pro bigramy (dvojice písmen, např. „ov“) a trigramy (trojice písmen, např. „ova“).

- Šifra je odolná proti útoku hrubou silou, protože délka klíče je teoreticky neomezená. Počet variací klíče délky n je pak 26^n. Při délce 19 dosáhneme na stejný počet klíčů jako v případě monoalfabetické šifry. Nicméně i při délce 12 máme na výběr z 10^{17} klíčů, což by počítač testující miliardu klíčů za sekundu luštil řádově roky.

2.4.5. Nevýhody

- Šifra už je složitá, její použití vyžaduje dodatečné pomůcky nebo překreslení čtverce. Hledání ve čtverci je také náročnější než hledání v jednom řádku, jako tomu bylo u předchozích metod.

- Při dostatečně dlouhém textu lze vysledovat délku klíče. V běžném textu se totiž některá slova opakují, typicky například „nebo“, „ale“ a podobně. Přitom platí, že každé slovo může být zašifrováno pouze tolika způsoby, jak je dlouhý klíč. Pokud je klíč dlouhý například pět písmen a text obsahuje šestkrát slovo „nebo“, minimálně jedenkrát se celé slovo zašifruje úplně stejně. Pokud nalezneme v šifrovém textu co nejvíce takových dvojic, můžeme vypočítat délku klíče.

- Vigenèrova šifra se skládá z několika Caesarových šifer. Lze říci, že pokud šifrujeme Vigenèrovou šifrou s klíčem o délce pět, je to jako bychom zprávu rozdělili na pět částí a každou tuto část bychom zašifrovali Caesarovou šifrou

s patřičným klíčem. Vigenèrova šifra tak dědí některé nevýhody Caesarovy šifry, například stejnou vzdálenost písmen v těch jednotlivých částech textu.

Zdroje ke kapitole 2.4.: [4], [5] a [10].

2.5. Jednoduchá sloupcová transpozice

2.5.1. Definice

Kryptosystém 4. Nechť $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$ a nechť množina $\mathcal{K}$ obsahuje všechny permutace $\{1, \ldots, m\}$. Pro každý klíč π definujeme:

$$e_\pi(x_1, \ldots, x_m) = \left(x_{\pi(1)}, \ldots, x_{\pi(m)}\right)$$

a

$$d_\pi(y_1, \ldots, y_m) = \left(y_{\pi^{-1}(1)}, \ldots, y_{\pi^{-1}(m)}\right),$$

kde π^{-1} je inverzní permutace k π.

— Citováno z [5].

2.5.2. Popis šifry

Sloupcová transpozice je transpoziční šifra – zachovává písmena, pouze je přehazuje. Klíčem může být slovo libovolné délky. Slabými klíči jsou všechny klíče délky jedna a všechny klíče, které se skládají z abecedně seřazených písmen, například „abcdegp“.

Máme-li klíč o délce n, rozdělíme otevřený text do n sloupců. V prvním řádku bude prvních n písmen otevřeného textu, v druhém řádku druhých n písmen a tak dále. Pokud nebude poslední sloupec zcela zaplněn, doplníme náhodnými písmeny. Následně seřadíme písmena v klíči podle abecedy a stejné záměny písmen, které jsme takto provedli v klíči, provedeme na sloupcích. Výsledný šifrový text čteme po sloupcích. Příklad: zašifrujeme text „bratricku, zavirej vratka“ s klíčem „karel“. Nejprve rozdělíme text do pěti sloupců, do prvního řádku vepíšeme klíč:

```
karel
-----
bratr
ickuz
avire
jvrat
kalan
```

Poslední tři písmena jsou náhodně doplněná písmena, aby byla tabulka úplná. Nyní seřadíme písmena v klíči podle abecedy a stejné posuny provedeme u sloupců:

```
aeklr
-----
rtbra
cuizk
vraei
vajtr
aaknl
```

Výsledný šifrový text přečteme po sloupcích: „rcvva turaa biajk rzetn akirl“.

2.5.3. Výhody

- Velký počet klíčů. Pro klíč o délce n platí, že existuje maximálně $n!$ výsledných permutací sloupců, přičemž celkový počet klíčů o délce n je 26^n. Některé klíče totiž dají ve výsledku stejnou permutaci, například klíče „abc“ a „def“. Při délce klíče 20 máme celkem 20! možných permutací, což je zhruba 10^{18} permutací a to by počítač s miliardou otestovaných klíčů za sekundu řešil řádově desítky let.

- Šifra téměř znemožňuje použití klasické frekvenční analýzy. Už z principu známe všechny reálné frekvence písmen. Nicméně můžeme díky frekvenci jednotlivých písmen uhádnout například použitý jazyk.

- Už v prvním bodu zaznělo, že šifra obsahuje takové dva různé klíče, které se ve výsledku chovají stejně, vytvoří stejnou permutaci. Těchto klíčů existuje více. Z toho plyne, že pokud si zvolíme klíč z množiny klíčů, které mají tuto vlastnost, potenciální útočník nikdy nerozluští naše pravé heslo, může se pouze dopátrat obecného vzoru – může zjistit všechny klíče v této množině, ale ne konkrétní použitý klíč.

2.5.4. Nevýhody

- Je jednoduché z délky šifrového textu zúžit množinu možných klíčů na několik jednotek. Protože se text vždy doplňuje do obdélníku, pak platí, že délka klíče je jedním z dělitelů délky šifrového textu.

- Protože šifra nijak nezaměňuje použitá písmena, můžeme snadno vyfiltrovat, která slova se mohou v dané části textu vyskytnout a která ne – buď v daném úseku je nebo není dostatek písmen pro dané slovo.

- Pokud je použit krátký klíč, je snadno prolomitelná. Při délce klíče osm existuje 40 320 permutací sloupců, což i průměrný počítač zvládne za okamžik. Ručně je v řádu hodin prolomitelný text zašifrovaný klíčem o délce 5 (120 možností).

Zdroje ke kapitole 2.5.: [9], [5].

3. Kryptoanalytické algoritmy

3.1. Prolomení Caesarovy šifry

3.1.1. Útok hrubou silou

Algoritmus bere na vstupu šifrový text a dodatečné statistické informace ξ (viz kapitola 6.1.); výstupem je šifrovací klíč.

Jednou z největších nevýhod Caesarovy šifry je velmi nízký počet klíčů. I pokud vezmeme v potaz slabý klíč „a“, obsahuje celkem 26 možných klíčů. Algoritmus hrubé síly dešifruje předaný šifrový text všemi 26 klíči, spočte Index podobnosti (viz algoritmus 3.5.1.) a vybere ten dešifrovaný text, který bude mít Index podobnosti nejvyšší. Klíč, kterým jsme získali tento text, vrátíme jako výstup. Algoritmus končí.

Tabulka 1. ukazuje úspěšnost algoritmu v závislosti na délce vstupního řetězce.

Počet znaků	Úspěšnost
500	100 %
250	100 %
100	97 %
50	81 %

Tabulka 1. Úspěšnost útoku hrubou silou na Caesarovu šifru

Metodika měření je popsána v kapitole 6.2.

3.1.2. Trojúhelníkový útok

Algoritmus bere na vstupu šifrový text a dodatečné statistické informace ξ; výstupem je šifrovací klíč.

Druhá podstatná chyba Caesarovy šifry je, že zachovává vzdálenosti mezi písmeny. Pokud jsou v otevřeném textu písmena „a“ a „c“ vzdálena o dvě písmena, budou jejich obrazy v šifrovém textu vzdáleny taky o dvě písmena. Problém činí „přechod“ přes písmeno „z“. Písmena „x“ a „z“ jsou od sebe vzdálena dvě písmena, ale při posunutí o jedno písmeno se z nich stanou písmena „y“ a „a“, a ta jsou od sebe vzdálena 24 písmen.

Tento nedostatek vyřešíme dvojím počítáním vzdáleností – jednou budeme počítat vzdálenost klasicky a podruhé spočítáme vzdálenost přes přechod přes písmeno „z“. Z těchto dvou vzdáleností následně vybereme tu menší a budeme počítat pouze s ní. Tuto vzdálenost nazveme *minimální vzdálenost písmen*. V předchozím příkladu je klasická vzdálenost písmen „y“ a „a“ 24, ale vzdálenost přes

„z“ je pouze dva. Tato vzdálenost už odpovídá naměřené vzdálenosti v otevřeném textu. Minimální vzdálenost označíme md.

Dále musíme získat k nejčastějších a k nejméně častých písmen v daném jazyku. V prvním kroku spočítáme procentuální výskyt písmen v šifrovém textu a vybereme z nich n nejčastějších a n nejméně častých písmen. Přitom musí platit, že $k \leq n$. Označme k nejčastějších písmen jazyka L a n nejčastějších písmen šifrového textu C. Nyní hledáme takovou k-tici, která se skládá z písmen z C, pro kterou platí:

$$md(L_i, L_j) = md(C_i, C_j)$$

pro všechny $1 \leq i, j \leq k$.

Pokud takovou k-tici nalezneme, jedná se pravděpodobně o obrazy písmen v šifrovém textu. Pokud známe byt' jenom jednu substituci, můžeme získat šifrovací klíč. Stejný postup můžeme aplikovat na nejméně častá písmena. Pokud se výsledné klíče shodují, jedná se téměř jistě o správný klíč. Pokud se neshodují, můžeme dále postupovat jedním z těchto kroků:

1. Můžeme se pokusit najít další k-tice, které mohou vést ke stejnému klíči.
2. Můžeme text dešifrovat oběma výslednými klíči a pomocí Indexu podobnosti (viz algoritmus 3.5.1.) vybrat ten pravděpodobnější.
3. Můžeme změřit, která k-tice dává obvykle správnější výsledky (jestli nejméně častá písmena nebo nejvíce častá písmena) a následně vrátit klíč určený touto k-ticí.
4. Můžeme si z nalezených klíčů náhodně vybrat.

Pokud nenalezneme ani jednu k-tici, algoritmus může buď skončit neúspěchem, nebo pokračovat znova s jinými hodnotami k nebo n.

Testováním různých variací čísel k a n jsem ověřil, že nejlepší výsledky při zachování vysoké rychlosti dával algoritmus pro hodnoty $k = 3$ (proto název *trojúhelníkový*) a $n = 8$.

Algoritmus vrátí nalezený klíč a skončí. Úspěšnost algoritmu ukazuje tabulka 2.

3.2. Prolomení monoalfabetické šifry

3.2.1. Algoritmus pracující s unikátními slovy

Algoritmus bere na vstupu šifrový text s vyznačenými slovy pomocí mezer a dodatečné statistické informace ξ; výstupem je šifrovací klíč.

Počet znaků	Úspěšnost
500	99 %
250	98 %
100	94 %
50	74 %

Tabulka 2. Úspěšnost Trojúhelníkového útoku na Caesarovu šifru

Budeme potřebovat seznam všech *unikátních slov*, což jsou slova, která mají v rámci daného slovníku unikátní kanonický tvar (viz algoritmus 3.5.2.). Tato slova uložíme do proměnné `unique`.

V prvním kroku nalezneme v textu všechna slova a uložíme si je do pole `words`. Dále projdeme všechna tato slova a zjistíme, jestli některé z těchto slov nemá stejný kanonický tvar jako některé unikátní slovo z `unique`. Tato nalezená slova si uložíme včetně odpovídajícího unikátního slova do pole `matches`. Pokud žádnou dvojici nenalezneme, algoritmus neúspěšně končí. V opačném případě máme seznam slov šifrového textu a jejich pravděpodobné vzory v otevřeném textu.

Hlavní problém je, že nalezené dvojici nemusí být správné – slovník, ze kterého jsme vygenerovali unikátní slova, nemohl být úplný, navíc v textu se mohou vyskytovat překlepy nebo cizí slova, která mohou seznam unikátních slov narušit. Proto musíme ze seznamu vyfiltrovat jen ta slova, která jsou nejpravděpodobnější. Všechna slova, která jsme identifikovali správně, budou mít jistě ekvivalentní tabulky substitucí (viz algoritmus 3.5.3.). Naopak slova, která jsme identifikovali špatně, nebudou mít pravděpodobně ekvivalentní tabulky substitucí. Musíme najít takovou největší množinu dvojic slov, která mají navzájem ekvivalentní tabulky substitucí.

Tento úkol je ekvivalentní s hledáním největší kliky grafu. Vrcholy grafu představují jednotlivé dvojice šifrového a unikátního slova, hrana mezi dvěma vrcholy představuje ekvivalenci tabulek substitucí mezi oběma dvojicemi. Úloha nalezení největší kliky grafu je NP obtížná úloha. Úlohu ale můžeme vyřešit některým známým heuristickým algoritmem nalezení maximální kliky grafu, viz [6]. Nalezenou množinu odpovídající maximální klice označíme `clique`.

Protože všechny tabulky substitucí v `clique` jsou ekvivalentní, můžeme je sjednotit do jedné. Označíme ji `S`. Dešifrujeme šifrový text pomocí této tabulky substitucí a zjistíme, jestli text obsahuje nějaká písmena, pro která není v tabulce `S` záznam. Pokud se záznam vyskytuje pro všechna písmena, známe substituce pro každé písmeno šifrového textu a můžeme tabulku proměnit v šifrovací klíč. Algoritmus zde úspěšně končí.

V opačném případě máme pouze částečný klíč a pro některá písmena šifrového textu neznáme jejich vzory. Zbylá písmena získáme slovníkovým útokem.

1. Nyní budeme postupně doplňovat tabulku substitucí:

    1. 1. Dešifrujeme text pomocí aktuální substituční tabulky `S`.
    1. 2. Pokud se podařilo dešifrovat kompletní zprávu, cyklus končí.
    1. 3. Nalezneme všechna slova, u kterých se nepodařilo dešifrovat pouze jedno písmeno, označíme ho `c`. Pokud žádné takové slovo není, cyklus končí. Tato slova uložíme do pole `temp`.
    1. 4. Nyní musíme zjistit, jestli lze nějaké slovo z `temp` doplnit libovolným písmenem tak, aby vzniklo reálné slovo obsažené ve slovníku daného jazyka. Vyfiltrujeme pouze ta slova z `temp`, pro která existuje pouze jedno reálné slovo ze slovníku. V tuto chvíli víme, na které slovo otevřeného textu se šifrové slovo zobrazuje, čímž získáme i tu jednu původně chybějící substituci písmene `c` v šifrovém slově. Nově vzniklou dvojici písmen přidáme do tabulky substitucí `S`.

2. Předchozí cyklus pravděpodobně doplnil tabulku substitucí `S` tak, aby bylo možné dešifrovat celý šifrový text. Pokud se tak nestalo, můžeme cyklus opakovat s tím, že budeme hledat slova, kterým chybí dvě písmena nebo ještě více. Druhou možností je, že se spokojíme se současným výsledkem a pokračujeme dále.

Tabulka substitucí `S` obsahuje substituce potřebné pro dešifrování textu. Převedeme tabulku na šifrovací klíč a vrátíme jako výstup algoritmu. V případě, že chceme, aby algoritmus vracel klíč jen v případě, kdy je téměř jisté, že klíč je správný, je možné ještě dodefinovat pravidla na „téměř jistý klíč“ (například minimální počet nalezených substitucí nebo spočítat Index podobnosti, viz 3.5.1.) a pokud nalezený klíč tato pravidla nesplňuje, algoritmus neúspěšně končí.

Úspěšnost algoritmu ukazuje tabulka 3.

Počet znaků	Úspěšnost
3000	96 %
1500	84 %
750	54 %
250	12 %

Tabulka 3. Úspěšnost útoku pracující s unikátními slovy na monoalfabetickou šifru

3.3. Prolomení Vigenèrovy šifry

3.3.1. Modifikovaný trojúhelníkový útok

Tento algoritmus bere na vstupu šifrový text S, délku klíče k a dodatečné statistické informace ξ; na výstupu vrací použitý klíč.

Vigenèrova šifra je složena z několika Caesarových šifer dohromady, což je její největší slabina. Má-li šifrovací klíč délku k, pak každé k-té písmeno otevřeného textu je šifrováno stejným písmenem, neboli tato posloupnost písmen se chová identicky jako Caesarova šifra. Proto tuto posloupnost písmen můžeme zkusit prolomit pomocí známých algoritmů na prolomení Caesarovy šifry. Vhodná je metoda Trojúhelníkového útoku (viz 3.1.2.), protože nepracuje s Indexem podobnosti, který by měl v této posloupnosti písmen horší výsledky.

Pokud máme šifrový text S o délce n a klíč o délce k, vznikne celkem k různých posloupností písmen P_0 až P_{k-1}. Posloupnost písmen P_i bude definována:

$$P_{i_j} = S\left[i + j \cdot k\right]$$

Pro $0 \leq j \leq \mathrm{up}(\frac{n}{k})$ a $0 \leq i < k$, kde $\mathrm{up}(x)$ vrací nejmenší celé číslo větší nebo rovno číslu x.

Posloupnost písmen K reprezentující hledaný šifrový klíč získáme aplikací Trojúhelníkového útoku na každou získanou posloupnost písmen:

$$K_i = \mathrm{triangle}(P_i, \xi),$$

kde triangle značí Trojúhelníkový útok, jak je popsán v kapitole 3.1.2. Z posloupnosti K vytvoříme řetězec, čímž získáme šifrovací klíč. Tento klíč je výstupem algoritmu.

3.3.2. Útok hrubou silou

Vstupem algoritmu je šifrový text S a dodatečné statistické informace ξ. Výstupem je šifrovací klíč.

Zvolíme si konstanty MIN a MAX ($MIN \leq MAX$), které budou určovat minimální délku klíče a maximální délku klíče. Zjistíme možné klíče při délce klíče i a tyto nalezené klíče si uložíme do proměnné P.

$$P_i = \mathrm{mtu}(S, i, \xi)$$

pro $MIN \leq i \leq MAX$, kde mtu značí Modifikovaný trojúhelníkový útok jak byl popsán v kapitole 3.3.1.

V tuto chvíli máme v P uložený seznam možných klíčů, mezi kterými se pravděpodobně nachází i původní šifrovací klíč. Ten nalezneme pomocí Indexu podobnosti (viz algoritmus 3.5.1.). Zkusíme dešifrovat text S pomocí všech klíčů

z P a uložíme si do proměnné *key* ten klíč, se kterým dával dešifrovaný text největší Index podobnosti.

Na konci vrátíme klíč uložený v proměnné *key*. Algoritmus končí. Úspěšnost algoritmu ukazuje tabulka 4.

Počet znaků	Úspěšnost
5000	98 %
2500	93 %
1500	71 %
1000	49 %

Tabulka 4. Úspěšnost modifikovaného Trojúhelníkového útoku na Vigenèrovu šifru

3.3.3. Útok odhadnutím délky klíče

Další slabinou Vigenèrovy šifry je možnost zjistit z analýzy šifrového textu délku klíče. Víme, že pokud má klíč délku k a otevřený text obsahuje jedno slovo v krát, pak pokud je $v > k$, minimálně dvě slova mají stejný obraz. Platí-li $v \geq 2$, pak se taková dvojice slov vyskytnout může, ale nemusí. Pokud takové stejné obrazy najdeme, můžeme z jejich vzdálenosti mezi sebou vypočíst délku klíče. Pokud je vzdálenost mezi dvěma stejnými obrazy d, pak může mít klíč délku jednoho z dělitelů vzdálenosti d.

Vstupem algoritmu je šifrový text S a statistické informace ξ; výstupem je šifrovací klíč.

1. Na začátku si uložíme všechna slova, která můžeme z šifrového textu složit, o délce i, kde $7 \leq i \leq 20$, včetně indexu, na kterém se slovo nachází – index prvního písmene. (Poznámka: tato čísla jsem takto nastavil na základě testování různých variací čísel, v případě potřeby je možné je změnit, algoritmus bude stále fungovat. Čísla by měla odrážet průměrnou délku slov, které se mohou v textu opakovat, nicméně v textu se často opakuje i delší kus textu. Navíc v šifrovém textu se mohou vyskytnout stejné kusy krátkého textu, ačkoliv v otevřeném textu odpovídající vzory stejné nejsou. Čím delší slovo v šifrovém textu, tím je tato pravděpodobnost menší.)

2. Z těchto nalezených slov vyfiltrujeme jen ta, která jsme našli vícekrát.

3. Zjistíme vzdálenosti mezi těmito vyfiltrovanými slovy a spočteme všechny dělitele. Ty si následně uložíme.

4. Seřadíme dělitele podle četnosti výskytu od nejčastěji se vyskytujícího po nejméně často se vyskytujícího, případně nějakým vhodným způsobem

zvýhodníme delší klíče před kratšími, pokud předpokládáme, že krátké klíče uživatel spíše nepoužije – nebo naopak.

V tuto chvíli provedeme stejný postup jako v případě Útoku hrubou silou (viz algoritmus 3.3.2.), pouze s tím rozdílem, že budeme testovat námi nalezené délky klíče, nikoliv všechny od MIN do MAX. Množinu testovaných klíčů můžeme ale omezit, buď minimální či maximální délkou klíče nebo maximálním počtem testovaných klíčů. Úspěšnost algoritmu ukazuje tabulka 5.

Počet znaků	Úspěšnost
5000	81 %
2500	70 %
1500	60 %
1000	41 %

Tabulka 5. Úspěšnost útoku odhadnutím délky klíče na Vigenèrovu šifru

3.4. Prolomení Jednoduché sloupcové transpozice

3.4.1. Zjištění délky klíče

Jednoduchá sloupcová transpozice má tu nevýhodu, že z délky šifrového textu lze vyčíst délku klíče. Protože musíme být schopni zarovnat text „do obdélníku“, bude délka klíče rovna jednomu z dělitelů délky šifrového textu. Blíže délku klíče určit nemůžeme, ale můžeme předpokládat omezenou délku klíče, čímž množinu klíčů opět zredukujeme.

3.4.2. Útok nalezením slova v řádku

Vstupem algoritmu je šifrový text S o délce n, délka klíče k a dodatečné statistické informace ξ. Výstupem je šifrovací klíč.

Vytvoříme z šifrového textu obdélník o k sloupcích a $\frac{n}{k}$ řádcích tak, aby v prvním řádku bylo prvních k písmen z otevřeného textu, akorát ve zpřeházeném pořadí. Ve druhém řádku druhých k znaků apod.

Budeme dále předpokládat, že otevřený text obsahuje alespoň jedno slovo o délce k. Pro krátké klíče ($k \leq 15$) je to velmi pravděpodobné. Další předpoklad bude, že se alespoň jedno z těchto slov začíná v otevřeném textu na indexu $m \cdot k, m \in \mathbb{N}_0$, což znamená, že při použití klíče o délce k nedošlo k rozlomení slova na dva řádky, ale celé slovo se zpřeházené nachází na jednom řádku. Pravděpodobnost, že se slovo o délce k nachází na indexu $m \cdot k$ je $\frac{1}{k}$.

Ze slovníku získáme všechna slova o délce k a uložíme je do proměnné `words`. Nyní budeme hledat, jestli se některé z těchto slov nedá složit z písmen některého z řádků. Každý řádek obsahuje k (ne nutně různých) písmen, takže můžeme jednoduše ověřit, zda lze z těchto písmen složit nějaké slovo z pole `words`. Uložíme si jak slovo `words`, tak příslušný řádek písmen. Pokud žádné takové slovo nenalezneme, algoritmus neúspěšně končí.

V tuto chvíli máme seznam slov, která se možná vyskytují v otevřeném textu i s příslušnými řádky, kde by se mohla vyskytovat – máme dvojice ve tvaru slovo v otevřeném textu (označme w_o) $\Rightarrow$ jeho obraz v šifrovém textu (w_c). Pro každou dvojici pomocí algoritmu 3.5.4. zjistíme, jaké záměny jsou nutné k tomu, aby se slovo w_o transformovalo do slova w_c. Z těchto získaných záměn vytvoříme klíče pomocí následujícího algoritmu:

Klíč získáme nahrazením čísla za písmeno podle klíče: 1→a, 2→b, 3→c, 4→d, ... Permutace $[2, 5, 4, 1, 3]$ by představovala klíč „bedac“. Je zřejmé, že takto je možné provést pouze 26 náhrad, nicméně tento algoritmus stejně není použitelný pro délku klíče $k > 26$, protože je velmi nepravděpodobné, že by otevřený text obsahoval tolik slov o délce více než 26, aby byl tento algoritmus účinný.

V tuto chvíli máme dvě možnosti:

1. Nalezenými klíči následně zkusíme dešifrovat text a ze všech klíčů vybereme ten, jehož dešifrovaný text bude mít největší Index podobnosti. Ten si uložíme do proměnné `key`.

2. Zjistíme, který klíč se vyskytuje nejvícekrát. Tento klíč si uložíme do proměnné `key`. Pokud se všechny klíče vyskytují stejně často, je tento postup neúčinný. Můžeme zkusit buď předchozí postup, vybrat klíč náhodně nebo můžeme algoritmus ukončit.

Vrátíme na výstupu klíč `key`. Algoritmus končí.

3.4.3. Útok testem všech klíčů

Vstupem algoritmu je šifrový text S a dodatečné statistické informace ξ. Výstupem je šifrovací klíč.

Pomocí algoritmu 3.4.1. zjistíme všechny přípustné délky klíče textu S. Následně pro každou možnou délku zjistíme nejvhodnější klíč pomocí algoritmu 3.4.2. A nakonec vybereme ten klíč, pomocí kterého vznikne dešifrováním text s největším Indexem podobnosti, viz 3.5.1. Algoritmus končí.

Úspěšnost algoritmu ukazuje tabulka 6.

Počet znaků	Úspěšnost
5000	80 %
2500	62 %
1500	45 %
1000	25 %

Tabulka 6. Úspěšnost útoku testem všech klíčů na Jednoduchou sloupcovou transpozici

3.5. Pomocné algoritmy

3.5.1. Index podobnosti

Index podobnosti určuje, jak moc je daný text podobný s běžným textem daného jazyka. Vstupem algoritmu je libovolný text a dodatečné statistické informace (viz konkrétní podoby algoritmu) a výstupem je číselný Index podobnosti, pro který platí, čím větší index je, tím je text podobnější.

V zásadě máme dva možné způsoby určení Indexu podobnosti:

1. Frekvenční analýza. Zjistíme, jak moc se liší hodnoty písmen, bigramů a trigramů v textu od statisticky definovaných hodnot. Přesný výpočet proběhne odečtením procentuálního výskytu písmene v textu od definovaného procentuálního výskytu, vypočítání absolutní hodnoty a přičtení k dočasné proměnné. Totéž pro bigramy a trigramy. Po provedení těchto operací máme v dočasné proměnné uložený Index podobnosti. Aby platilo, že čím vyšší Index podobnosti, tím je text podobnější, provedeme ještě výpočet $1/I$, kde I je současný Index podobnosti. Pro případ, že by index byl roven nule, vrátíme nekonečno nebo maximální hodnotu daného typu.

2. Počet reálných slov. Máme-li vygenerovaná nejčastěji používaná slova v daném jazyku, můžeme v textu tato slova hledat. Na výstupu vrátíme počet nalezených slov.

3.5.2. Kanonický tvar slova

Kanonický tvar slova w označíme $can(w)$. Kanonický tvar slova je takový řetězec znaků, pro který platí:

1. Každé slovo má právě jeden kanonický tvar.

2. Máme-li slovo w_o a slovo w_e, pro které platí, že w_e je monoalfabetickou šifrou zašifrované slovo w_o (s libovolným klíčem), pak $can(w_o) = can(w_e)$.

Kanonický tvar slova zůstává stejný i po zašifrování slova monoalfabetickou šifrou. Pokud nalezneme dvě slova, která mají stejný kanonický tvar, může být jedno slovo obrazem druhého.

Kanonický tvar získáme následujícím algoritmem. Vstupem je slovo `w`. Výstupem je kanonický tvar slova `w`.

1. Na začátku nastavíme proměnnou `i` na 10 a inicializujeme asociativní pole `subs`, do kterého budeme ukládat položky typu char ⇒ int.
2. Projdeme slovo `w` po písmenech. Aktuální písmeno označíme `c`.
   2. 1. Pokud pole `subs` obsahuje už všechna písmena anglické abecedy, cyklus ukončíme.
   2. 2. Pokud pole `subs` obsahuje písmeno `c`, přeskočíme k dalšímu písmenu.
   2. 3. Uložíme do pole `subs` pod klíč `c` hodnotu `i`: `subs[c]=i`.
   2. 4. Zvýšíme proměnnou `i` o jedna.
3. Vytvoříme nový řetězec `r`, který bude namísto písmen slova `w` obsahovat ekvivalentní číselné hodnoty (převedené na datový typ string) z asociativního pole `subs`. Namísto písmene `c` bude řetězec obsahovat číslo `subs[c]`.

Poznámka: Proměnnou `i` nastavujeme na 10 proto, aby každé možné písmeno abecedy mělo stejně dlouhou číselnou reprezentaci. Pokud bychom začali na `i=1`, nebylo by jednoznačné, zda číslo 11 představuje dvě písmena (s identifikátory 1 a 1) nebo jedno písmeno (s identifikátorem 11). Máme celkem 90 dvouciferných čísel, což stačí pro abecedu o délce 26 písmen. Pokud bychom pracovali s delší abecedou, jen posuneme `i` o požadovaný počet řádů dále.

Příklad: mějme slovo „tatar“. Na začátku nastavíme proměnnou `i` na 10 a inicializujeme `subs`. Projdeme všechna písmena ve slově. Písmeno „t“ v poli není, takže ho tam vložíme: `subs['t']=10`. Zvýšíme `i` na 11. Písmeno „a“ v poli není, vložíme ho: `subs['a']=11`. Zvýšíme `i` na 12. Písmeno „t“ už v poli je, přejdeme na další písmo. Písmeno „a“ už v poli je, přejdeme na další písmeno. Písmeno „r“ v poli není, přidáme ho: `subs['r']=12`. Už nemáme žádné další písmeno, cyklus končí. Přepíšeme původní slovo „tatar“ podle nově vzniklé tabulky na: „1011101112“. Tím jsme získali kanonický tvar slova „tatar“.

3.5.3. Ekvivalence tabulek substitucí

Tabulka substitucí představuje množinu dvojic, které reprezentují substituci v monoalfabetické šifře – jaké písmeno otevřené abecedy se má zašifrovat na které písmeno šifrové abecedy; nemusí obsahovat celou abecedu.

Tabulka substitucí je množina dvojic písmen ve tvaru $o_i \rightarrow c_j$, kde písmena o_i patří otevřené abecedě a písmena c_j šifrové abecedě.

Dvě tabulky substitucí S_1 a S_2 jsou *ekvivalentní*, pokud platí: Obsahuje-li S_1 substituci $o_i \rightarrow c_j$, pak tabulka S_2:

- musí obsahovat stejnou substituci, nebo
- nesmí obsahovat substituci $o_i \rightarrow c_k$ ani substituci $o_k \rightarrow c_i$.

3.5.4. Nalezení všech permutací slova

Vstupem algoritmu je slovo w_o, které představuje slovo v otevřeném textu a slovo w_c, které představuje slovo v šifrovém textu; výstupem je množina všech permutací pozic písmen ve slovu w_c tak, aby výsledná permutace po přečtení dala slovo w_o.

Příkladem vstupu může být w_o=„vata“ a w_c=„avta“. Jednotlivá písmena ve slově w_c jsou na pozicích: $v = \{2\}, a = \{1, 4\}, t = \{3\}$. Algoritmus má za úkol najít všechny takové permutace písmen slova „avta“, které dají slovo „vata“. Z těchto písmen můžeme slovo „vata“ složit dvěma způsoby: v(2)-a(1)-t(3)-a(4) a v(2)-a(4)-t(3)-a(1). Čísla v závorkách odpovídají pozici konkrétního písmene ve slově w_c. Získáváme dvě možné permutace: $[2, 1, 3, 4]$ a $[2, 4, 3, 1]$. Tyto dvě permutace by byly návratovou hodnotou algoritmu.

Samotný algoritmus, který bude tyto permutace hledat bude rekurzivní. Bude potřebovat celkem pět parametrů:

1. `word` – slovo, které máme z písmen poskládat, tedy slovo w_0.
2. `currPos` – aktuální pozici písmene, které se snažíme najít.
3. `visited` – již navštívená písmena. V této proměnné se bude uchovávat konečná permutace.
4. `positions` – pozice písmen ve slově w_c.
5. `results` – všechny výsledné uložené permutace.

Algoritmus pojmenujeme `getPerms`. V prvním kroku ho zavoláme s argumenty:

```
getPerms(wo, 0, [], pos, []);
```

Argument `pos` je seznam pozic písmen ve slově w_c. Algoritmus bude dále probíhat takto:

1. Pokud je `currPos` rovno délce slova `word`, přidáme do pole `results` pole `visited`. Aktuální větev algoritmu zde končí.
2. Do proměnné `c` si uložíme písmeno ze slova `word` na pozici `currPos`: `c=word[currPos]`.
3. Z `positions` vytáhneme seznam pozic pro písmeno `c` a uložíme je do proměnné `tempPos`: `tempPos=positions[c]`.

4. Z `tempPos` odstraníme ty pozice, které se nachází v seznamu `visited`. (Provedeme množinový rozdíl `tempPos` minus `visited`.) Výsledek uložíme do proměnné `freePos`: `freePos = tempPos.difference(visited)`.

5. Pro každý prvek v poli `freePos` znovu rekurzivně zavoláme `getPerms`:

```
foreach(p in freePos)
  getPerms(wo, currPos+1, visited.add(p), positions, results);
```

V proměnné `results` se nachází všechny možné permutace. Tuto proměnnou vrátíme na výstupu. Algoritmus zde končí.

4. Uživatelská příručka

Samotná aplikace je rozdělena do dvou samostatných částí – v jedné části se provádí kryptoanalýza a slouží k luštění šifer, druhá část je pomocná a jejím účelem je umožnit uživateli zašifrovat a dešifrovat text pomocí podporovaných šifer, aby si například mohl vyzkoušet funkčnost ostatních algoritmů.

Poznámka: Název aplikace „Bletchley Park“ odkazuje na důležité místo světové kryptoanalýzy – jedná se o část města Bletchley nacházející se v Anglii. Během 2. světové války se zde nacházelo hlavní sídlo spojeneckých kryptoanalytiků, kteří se snažili zlomit německé šifry, především slavnou Enigmu. Nejvýznamnější osobou, která se podílela na luštění, byl Alan Turing, který měl na prolomení Enigmy nejspíše největší zásluhy. Později po něm byla pojmenována Turingova cena. Zdroje: [1], [2] a [4].

4.1. Systémové požadavky

Aplikace vyžaduje operační systém Microsoft Windows (XP, Vista, 7) s nainstalovaným .NET 3.5 frameworkem. Pro hladký běh aplikace je potřeba alespoň 256 MB RAM.

4.2. Kryptoanalytická část

4.2.1. Popis jednotlivých ovládacích prvků

K prolomení šifry použijeme záložku „Kryptoanalýza“, kterou vidíme na obrázku 3.

Popis všech ovládacích prvků:

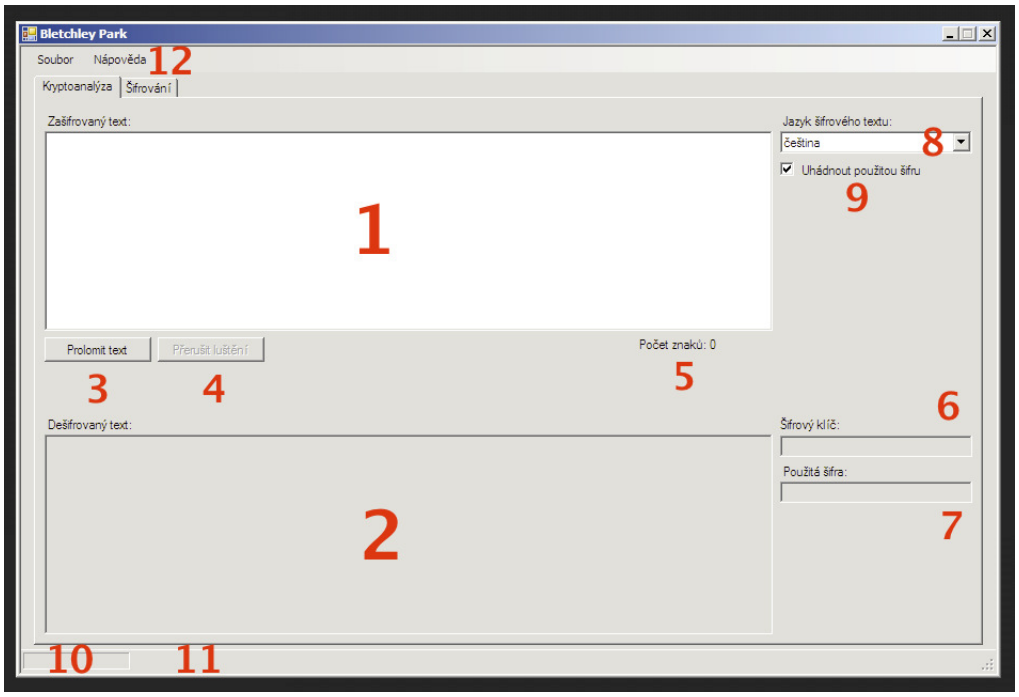
1. Primární textové pole, do kterého uživatel vkládá text, který chce rozluštit. Textové pole podporuje kontextové menu, které umožňuje jednoduše smazat celý text a vložit na místo něj text ze schránky a případně i nakopírovat celý text do schránky.

2. Sekundární textové pole, které zobrazuje výsledek luštění. V případě úspěšného prolomení šifry se zde objeví dešifrovaný text. Nepodaří-li se text prolomit, zobrazí se zde upozornění. Pole podporuje kontextové menu, které umožňuje jednoduše nakopírovat celý text do schránky.

3. Tlačítko, které spouští luštění. Během samotného luštění je neaktivní.

4. Tlačítko, kterým může uživatel v případě potřeby přerušit luštění. Je aktivní pouze během aktivního luštění.

5. Ukazatel počtu napsaných znaků v primárním textovém poli.



Obrázek 3. Kryptoanalytická část aplikace

6. Textové pole, které na konci úspěšného luštění zobrazí uhádnutý šifrovací klíč.

7. Textové pole, které na konci úspěšného luštění zobrazí použitou šifru.

8. Rozbalovací menu, které obsahuje několik jazyků a umožňuje uživateli nastavit, v jakém jazyce je vložený šifrový text (podporovány jsou tyto jazyky: čeština, angličtina, němčina a francouzština). Bez správného určení jazyka se algoritmy nebudou chovat správně.

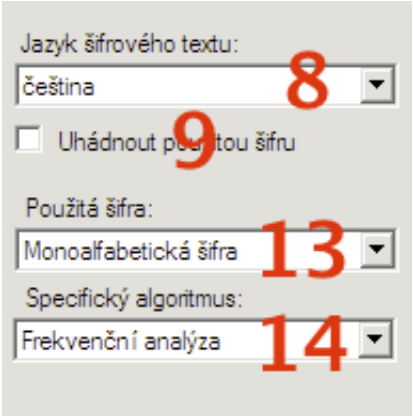
9. Checkbox, který indikuje, zda má aplikace sama určit použitou šifru (při zapnutém checkboxu) nebo zda uživatel sám zvolí, jakou šifrou byl text zašifrován.

10. Progress bar, který při startu aplikace indikuje průběh načítání slovníků, později zobrazuje průběh kryptoanalýzy.

11. Status bar – aplikace zde po najetí kurzorem na nějaký ovládací prvek zobrazuje pomocné informace.

12. Hlavní menu aplikace, obsahuje základní funkce programu a nápovědu.

V případě, že pomocí checkboxu zvolíme režim, kdy si uživatel může sám vybrat použitou šifru (viz ovládací prvek 9 na obrázku 3.), získáme další možnosti nastavení, viz obrázek 4.



Obrázek 4. Možnosti voleb uživatele, pokud vypne automatické zjišťování použité šifry

Popis nových voleb:

13. Výběr šifry, kterou je zašifrován šifrový text, tj. text vložený v primárním textovém okně.

14. Výběr algoritmu, který bude použit při luštění. Některé šifry nabízí více různých algoritmů, které mohou pro stejný text vrátit jiné výsledky. V případě, že jeden algoritmus nedokáže šifru vyluštit, může uživatel zkusit jiný algoritmus.

4.2.2. Typický příklad užití kryptoanalytické záložky

Záložka a všechny její ovládací prvky jsou uzpůsobeny k co možná nejjednodušší práci s algoritmy luštící šifry.

Na začátku máme nějakou zašifrovanou zprávu a chceme získat původní otevřený text. Spustíme aplikaci a nakopírujeme tento text do primárního textového pole. Pokud byl původní text jiného jazyka než českého, zvolíme ještě v comboboxu požadovaný jazyk (ovládací prvek 8). Dále klikneme na tlačítko „Prolomit text“ (ovládací prvek 3). Aplikace bude chvíli počítat a po několika sekundách se objeví výsledek – buď správně či nesprávně vyluštěný text nebo upozornění, že text nebylo možné rozluštit.

Známe-li šifru, kterou byl náš šifrový text zašifrován, můžeme odškrtnout „Uhádnout použitou šifru“ (ovládací prvek 9) a specifikovat, kterou šifrou byl text zašifrován. Poté opět stiskneme tlačítko „Prolomit text“ (ovládací prvek 3). Doba luštění bude kratší, aplikace nebude muset zkoušet všechny šifry.

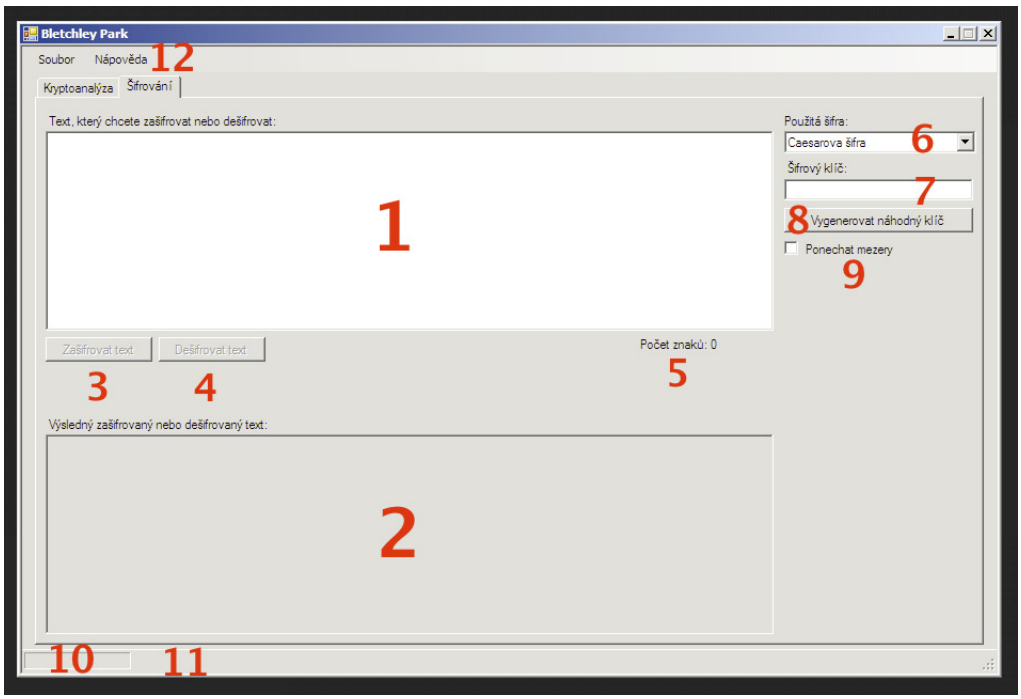
Obecně platí, že čím delší text, tím větší je pravděpodobnost, že bude správně rozluštěn. U Caesarovy šifry stačí text dlouhý řádově stovky znaků, ostatní šifry už jsou složitější a vyžadují řádově tisíce znaků.

Některé algoritmy mají kvůli efektivitě a rychlosti nastavenou maximální délku klíče, kterou jsou schopny vyluštit. Algoritmy u Caesarovy šifry a monoalfabetické substituční šifry pracují bez omezení, algoritmus u Jednoduché sloupcové transpozice (3.4.3.) má maximální délku klíče nastavenou na 12, algoritmus Hrubé síly (3.3.2.) u Vigenèrovy šifry má maximální délku klíče 20 a algoritmus Odhadnutím délky klíče (3.3.3.) má maximální délku 40. Aplikace neobsahuje algoritmus pro prolomení monoalfabetické šifry bez ponechaných mezer, takový šifrový text aplikace nedokáže prolomit.

4.3. Šifrovací část

4.3.1. Popis jednotlivých ovládacích prvků

Šifrovací část se nachází na záložce „Šifrování“ a slouží k šifrování a dešifrování zpráv, pokud předem známe šifrovací klíč. Šifrovací záložka je vidět na obrázku 5.



Obrázek 5. Šifrovací část aplikace

Popis ovládacích prvků:

1. Primární textové pole. Zde uživatel vkládá text, který chce šifrovat nebo dešifrovat. Textové pole podporuje kontextové menu, které umožňuje jednoduše smazat celý text a vložit na místo něj text ze schránky a případně i nakopírovat celý text do schránky.

2. Sekundární textové pole. V tomto textovém poli se zobrazí výsledný šifrový nebo otevřený text. I toto pole podporuje kontextové menu pro snadné nakopírování obsahu do schránky.

3. Stisknutím tohoto tlačítka aplikace zašifruje text z primárního textového pole a výsledek zobrazí v sekundárním poli.

4. Stisknutím tohoto tlačítka aplikace dešifruje text z primárního textového pole a výsledek zobrazí v sekundárním poli.

5. Zobrazuje počet znaků napsaných v primárním poli.

6. Combobox, který nabízí různé druhy šifrových algoritmů. Touto volbou uživatel specifikuje, jakou šifrou si přeje text šifrovat nebo dešifrovat.

7. Textové pole, kam uživatel vepíše použitý šifrovací klíč.

8. Tlačítko, kterým může uživatel vygenerovat náhodný klíč, nechce-li použít žádný vlastní. Vždy vygeneruje validní klíč pro zvolenou šifru. Pokud je textové pole podbarvené zeleně, je vepsaný klíč validní. Pokud je podbarvené červeně, není validní. Informace o tom, jaká pravidla musí klíč splňovat se nachází ve status baru.

9. Checkbox indikuje, jestli má aplikace při šifrování ponechat v šifrovém textu mezery, které budou reprezentovat hranice mezi slovy. Aplikace ponechá vždy jen jednu mezeru, všechny ostatní bílé znaky budou ignorovány. Jakékoliv další formátování (tabulátory, odřádkování apod.) bude ignorováno.

10. Progress bar, který zobrazuje průběh šifrování a dešifrování. Obvykle jen problikne, protože šifrování i dešifrování je rychlé.

11. Status bar – aplikace zde po najetí kurzorem na nějaký ovládací prvek zobrazuje pomocné informace, po aktivaci textového pole pro zadání klíče (ovládací prvek 7) se zde například zobrazují podrobnosti o tom, jak má vypadat šifrovací klíč.

12. Hlavní menu aplikace, obsahuje základní funkce programu a nápovědu.

4.3.2. Typický příklad použití šifrovací záložky

Uživatel může v této záložce provádět dvě opačné úlohy: může šifrovat text a dešifrovat text.

Pokud si přeje šifrovat text, vloží svou zprávu, kterou chce zašifrovat, do primárního textového pole. Následně si vybere šifru, kterou chce text zašifrovat (ovládací prvek 6) a do textového pole (ovládací prvek 7) vepíše šifrovací klíč, který chce použít. Pokud nechce použít konkrétní šifrovací klíč, může si nechat vygenerovat náhodný klíč (ovládací prvek 8). Nakonec stiskne tlačítko „zašifrovat text“ (ovládací prvek 3). V sekundárním textovém poli se zobrazí výsledný šifrový text. Vyvoláním kontextového menu lze šifrový text jednoduše nakopírovat do schránky.

Přeje-li si uživatel naopak dešifrovat text, vloží svůj zašifrovaný text do primárního textového pole, vybere si šifru, kterou byl text zašifrován (ovládací prvek 6) a do textového pole (ovládací prvek 7) vepíše šifrovací klíč. Nakonec stiskne tlačítko „dešifrovat text“ (ovládací prvek 4) a v sekundárním textovém poli se zobrazí původní otevřený text.

5. Dokumentace zdrojového kódu

Aplikace je napsána v programovacím jazyku C# a běží na .NET frameworku. Je rozdělena do tří oddělených částí: GUI aplikace, DLL knihovna `CryptanalysisCore` obstarávající všechny šifrovací algoritmy a DLL knihovna `ExtensionMethods`.

5.1. CryptanalysisCore

Obsahuje všechny algoritmy vztahující se k šifrování, dešifrování a luštění šifer. Knihovna je rozdělena do několika dalších částí:

5.1.1. Ciphers

Tato část obsahuje třídy, které přímo reprezentují jednotlivé šifry. Hlavní třídou je třída `Cipher`, ze které ostatní třídy dědí. Tato třída obsahuje základní metody pro šifrování, dešifrování, luštění šifer, generování náhodného klíče a pro kontrolu, zda je klíč validní. Potomci třídy si tyto metody přepisují po svém. Dále obsahuje pole delegátů `CrackMethods`, ve kterém jsou uloženy reference na implementované kryptoanalytické algoritmy. Toto pole naplňují až konkrétní šifry metodou `SetCrackMethods` podle toho, jaké algoritmy implementují.

Ostatní třídy jsou: `Caesar`, `Monoalphabetic`, `Transposition` a `Vigenere` a reprezentují konkrétní šifru. Každá z těchto tříd obsahuje vlastní kryptoanalytické algoritmy, které často využívají struktury a třídy z dalších částí knihovny.

Nejdůležitější metodou je zde metoda `Crack`, která bere na vstupu šifrový text, číselný identifikátor konkrétního algoritmu a identifikátor jazyka. Na výstupu vrací vyluštěný klíč, v případě neúspěchu vyvolá výjimku `MatchNotFound` (viz 5.1.2.).

5.1.2. Exceptions

Obsahuje výjimky, se kterými aplikace pracuje.

`InvalidCipherKey` oznamuje neplatný šifrový klíč pro danou šifru.

`InvalidOpentext` oznamuje neplatný otevřený text.

`InvalidCiphertext` oznamuje neplatný šifrový text.

`MatchNotFound` kryptoanalytický algoritmus nenalezl žádný shodný klíč.

5.1.3. Filters

Tato část obsahuje pomocné algoritmy ve formě tříd, které umožňují lepší rozdělení odpovědnosti. Jsou zde funkce využívané obvykle při kryptoanalýze, například různé filtrování slov na základě slovníkového útoku, algoritmus pro zjištění možných délek klíče u Vigenèrovy šifry apod.

5.1.4. HelpMethods

Obsahuje obecné pomocné metody pro práci s textem, dodatečné matematické metody jako například zjištění všech dělitelů čísla, kombinatorické metody pro zjištění permutací, variací apod. Velká část metod je natolik obecná, že jdou použít i mimo tento program.

5.1.5. Structures

Obsahuje pomocné struktury pro některé algoritmy. Obsahuje ale i dvě důležité struktury:

`LangCharacteristic` Struktura obsahující statistické informace o daném jazyku. Uchovává informace o frekvenci písmen, bigramů, trigramů, slovník všech a slovník nejčastějších slov a další potřebné informace.

`Storage` Nejdůležitější struktura, obsahuje pole struktur `LangCharacteristic` pro každý podporovaný jazyk, dále uchovává seznam šifer, které můžeme použít a seznam jazyků. Stará se také o samotné načítání těchto informací pomocí metody `LoadFiles`. Aplikace po zavolání této metody načte potřebné slovníky a další informace ze složky, jejíž jméno je uloženo ve statické proměnné `StatsFolderName`. Proměnná je `public`, takže ji lze měnit hodnotu ještě před zavoláním metody `LoadFiles`. V proměnné `StatsFolderPath` je pak uložena celá cesta ke slovníkům. Pomocí metody `GetLangChar` lze získat statistiky konkrétního jazyka.

5.2. GUI aplikace

Tato část obsahuje metody pro vykreslení a ovládání GUI celé aplikace. Zároveň se stará o správné volání kryptoanalytických metod, které volá z knihovny popsané v kapitole 5.1. I tato část je rozdělena do dalších podčástí:

5.2.1. Window

Obsahuje samotné metody pro obsluhu událostí. Tyto metody typicky nevykonávají funkční kód přímo, ale volají nějakou další obslužnou metodu.

5.2.2. Window_Properties

Obsahuje vlastnosti definované celému oknu. Jsou zde abstrakční bariéry pro nastavování hodnot do textových polí nebo nastavování stavu `Enabled` checkboxům a položkám v menu apod. Vlastnosti slouží především k větší přehlednosti. Pro každou šifru uchovává objekt vytvořený z požadované třídy. Při šifrování, dešifrování nebo luštění se pouze vybere aktuální šifra a zavolá se jedna z obslužných metod.

5.2.3. Window_Functions

Obsahuje funkční metody, které jsou volány z `Window`. Pokud uživatel klikne na nějaké tlačítko, v části `Window` se nachází kód k odchycení této události a tento kód následně zavolá nějakou funkci z `Window_Functions`, která obslouží zbytek požadavku. Výhoda spočívá v tom, že pokud dva ovládací prvky mají udělat totéž (například tlačítko a položka v menu), nemusí se žádný kód opakovat, jen se zavolá stejná funkce z `Window_Functions`.

5.2.4. Window_Statutes

Obsahuje metody, které mají za úkol měnit text ve status baru podle aktuálního stavu v aplikaci.

5.3. ExtensionMethods

Další DLL knihovna, která obsahuje obecné Extension Methods, které rozšiřují schopnosti vestavěných tříd. Je to knihovna zcela nezávislá na zbytku a lze ji připojit k jakémukoliv jinému projektu. Rozšiřuje například `int` o metodu `times`, která bere jako parametr metodu, která se provede tolikrát, kolik je hodnota `int`u, se kterým metodu voláme. Dalším příkladem je metoda `RemoveDiacritics` připojená ke `string`u, která odstraňuje diakritiku z řetězce.

6. Dodatky

6.1. Statistické informace ξ

Kryptoanalytické algoritmy popsané v kapitole 3. potřebují znát specifické statistické údaje o jazyku, nad kterým zrovna pracují. Tato sada informací se skládá z těchto částí:

- procentuální vyjádření výskytu jednotlivých znaků, bigramů a trigramů v daném jazyku,
- procentuální vyjádření výskytu jednotlivých písmen na začátku a na konci slov,
- průměrný počet různých písmen vyskytující se za daným písmenem v textu o přiměřeně dlouhé délce (zvolil jsem tisíc znaků),
- seznam „všech“ slov daného jazyka,
- seznam nejčastějších slov jazyka.

Tyto informace jsem pro každý jazyk zvlášť získal analýzou dlouhého textu v daném jazyce. Do tohoto textu jsem vždy zahrnul několik tisíc článků stažených z internetu a případně i několik knih v elektronické podobě, které jsou volně k dostání. Dále jsem použil slovníky pro kontrolu překlepů pro textový editor PSPad, ke stažení na adrese `<http://www.pspad.com/cz/download.php>`.

6.2. Metodika měření úspěšnosti algoritmů

Pro testování jsem použil text (označme t), ze kterého jsem generoval statistické informace ξ, viz 6.1. Postup byl pro každý algoritmus následující:

1. Vygenerujeme náhodný validní klíč K_o pro danou šifru.
2. Z textu t vybereme řetězec O dlouhý n znaků – podle toho, jak dlouhý text chceme zkoušet vyluštit (v tabulkách v kapitole 3. to odpovídá hodnotám ve sloupci „počet znaků“).
3. Zašifrujeme text O pomocí klíče K_o. Vznikne šifrový text C.
4. pomocí daného algoritmu zkusíme vyluštit šifrový text C. Algoritmus buď selže nebo vrátí nějaký klíč K_c.

V tuto chvíli porovnáme klíče K_o a K_c. Každá šifra má svou specifickou funkci (viz dále), která určí, zda bychom s klíčem K_c úspěšně dešifrovali otevřený text O. Nakonec spočítáme počet úspěšně vyluštěných textů děleno počtem všech testovaných textů a máme poměr vyjadřující úspěšnost algoritmu. Všechny algoritmy jsem nechal vyluštit tisíc textů. Výsledek jsem zaokrouhloval na jednotky procent.

Každá šifra má jinou funkci, která rozhoduje, zda bychom s klíčem správně dešifrovali šifrový text. Klíče K_o a K_c nemusí vždy shodovat, aby byl výsledný dešifrovaný text čitelný. Definujeme tato pravidla:

- Pokud pracujeme s Caesarovou šifrou, musí být klíče K_o a K_c shodné.
- Pokud pracujeme s monoalfabetickou šifrou, může klíč K_c obsahovat celkem šest různých chybných substitucí. Buď zcela nevyluštěných (typicky pro písmena „x“, „w“ nebo „q“, které se v českém textu vyskytují sporadicky) nebo chybných substitucí. I při šesti chybách zůstává text obvykle bez větších problémů čitelný.
- Pokud pracujeme s Vigenèrovou šifrou, může klíč K_c obsahovat jedno chybné písmeno. Pokud máme například klíč o délce šest, pak na jedno špatně dešifrované písmeno připadá pět správně dešifrovaných písmen. Text bude i tak bez větších problémů čitelný. Čím delší klíč, tím čitelnější text bude.
- Pokud pracujeme s transpoziční šifrou, musí být klíče K_o a K_c shodné.

Úspěšnost jsem měřil pouze pro české texty. Výsledky jsou pouze orientační, slouží především k bližší představě o tom, jak dlouhý by měl být vstupní šifrový text.

6.3. Obsah přiloženého CD

Popis jednotlivých složek na přiloženém CD:

`bin/` Složka obsahující zkompilované soubory určené k běhu aplikace bez nutnosti instalace. Spouštěcí soubor se jmenuje `BletchleyPark.exe`. V dalších podsložkách se nachází nápověda a pomocné statistické informace o jazycích.

`src/` Složka obsahující zdrojové soubory potřebné k sestavení celé aplikace. Aplikaci jsem vytvářel ve vývojovém prostředí *Microsoft Visual Studio 2008*.

`doc/` Složka obsahující text bakalářské práce ve formátu PDF včetně zdrojových souborů v LaTeXu a obrázků ve formátu EPS i PNG.

`readme.txt` Textový soubor obsahující základní pokyny a popis struktury CD.

Závěr

Podařilo se mi naprogramovat aplikaci, která zvládá na běžném počítači luštit v řádu sekund celkem čtyři různé druhy šifer. Úspěšnost algoritmů je uspokojující pro delší texty, výsledky pro kratší texty jsou horší a existuje zde prostor pro vylepšení. Aplikace umí luštit šifrový text v celkem čtyřech různých jazycích – v češtině, angličtině, němčině a francouzštině. Pro každý jazyk se používá stejný kryptoanalytický algoritmus, pouze s jinými statistickými informacemi o daném jazyku – proto je možné přidat do aplikace podporu pro další jazyk.

Všechny použité algoritmy jsou důsledně sepsány a specifikovány, včetně několika nápadů na jejich vylepšení či změnu. U každé použité šifry jsem zmínil i její nevýhody, na kterých jsem následně stavěl kryptoanalýzu. Ne všechny nevýhody ale byly využity, takže v případě dalšího vývoje aplikace může cesta pokračovat buď vylepšením stávajících algoritmů nebo se mohou využít právě tyto nepoužité nevýhody.

Reference

[1] Bletchley Park. [Online].
URL <http://www.bletchleypark.org.uk/>

[2] Hodges, A.: Alan Turing – Home page. [Online].
URL <http://www.turing.org.uk/turing/>

[3] Klíma, V.: Utajené komunikace - 2.díl : Tajné kódy, řeč a substituce. *časopis Chip*, červen 1994: s. 184–188.

[4] Singh, S.: *Kniha kódů a šifer*. Nakladatelství Dokořán, 2009, ISBN 978-80-7363-268-7.

[5] Stinson, D. R.: *Cryptography: Theory and Practice (Third edition)*. CRC Press, Inc, 2006, ISBN 978-1-58488-508-5.

[6] Večerka, A.: *Grafy a grafové algoritmy*. UPOL, 2007.
URL <http://phoenix.inf.upol.cz/esf/ucebni/Grafy_a_grafove_algoritmy.pdf>

[7] Vondruška, P.: *Kryptologie, šifrování a tajná písma*. Albatros, 2006, ISBN 978-80-00-01888-8.

[8] Wikipedie: Monoalfabetická šifra — Wikipedie, otevřená encyklopedie. 2010, [Online].
URL <http://cs.wikipedia.org/w/index.php?title=Monoalfabetic%6B%C3%A1_%C5%A1ifra&oldid=4959232>

[9] Wikipedie: Transposition cipher — Wikipedia, The Free Encyclopedia. 2010, [Online].
URL <http://en.wikipedia.org/w/index.php?title=Transposition_cipher&oldid=356226371>

[10] Wikipedie: Vigenèrova šifra — Wikipedie, otevřená encyklopedie. 2010, [Online].
URL <http://cs.wikipedia.org/w/index.php?title=Vigen%C3%A8ro%76a_%C5%A1ifra&oldid=5253539>