Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Kryptoanalýza klasických šifer

2016

Josef Podstata

Vedoucí práce:
RNDr. Eduard Bartl, Ph.D.

Studijní obor: Aplikovaná informatika,
prezenční forma

Bibliografické údaje

Autor:	Josef Podstata
Název práce:	Kryptoanalýza klasických šifer
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2016
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	RNDr. Eduard Bartl, Ph.D.
Počet stran:	45
Přílohy:	1 CD
Jazyk práce:	český

Bibliographic info

Author:	Josef Podstata
Title:	Cryptanalysis of classical ciphers
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2016
Study field:	Applied Computer Science, full-time form
Supervisor:	RNDr. Eduard Bartl, Ph.D.
Page count:	45
Supplements:	1 CD
Thesis language:	Czech

Anotace

Šifrování zpráv a luštění šifer jsou disciplíny s dlouhou historií. Práce rozebírá algoritmy prolamování vybraných historických šifer bez znalosti šifrovacího klíče a zkoumá jejich efektivitu vzhledem k počtu znaků zašifrované zprávy. Praktickou částí práce je mobilní aplikace na systém iOS, která implementuje vybrané šifry a jejich prolamování.

Synopsis

Encryption and breaking ciphers are fields with a long history. This thesis analyzes code breaking algorithms of specific historical ciphers without knowing the key and their effectivity considering the text length. Practical part of this thesis is a mobile application for iOS operating system that can encrypt, decrypt and break those specific ciphers.

Klíčová slova: šifra, kryptologie, kryptografie, kryptoanalýza, iOS

Keywords: cipher, cryptology, cryptography, cryptanalysis, iOS

Děkuji RNDr. Eduardu Bartlovi, Ph.D. za vedení práce a za cenné rady při konzultacích.

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

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Šifrování jako nástroj pro utajení informací je velmi starou vědní disciplínou. Nejstarší dochované záznamy o použití šifer se datují až do starověkého Egypta, kde se okolo roku 1900 př. n. l. používaly tajné hieroglify, které měly skrytý význam i pro jinak gramotné obyvatele. Archeologické nálezy dále naznačují použití substitučních šifer v Mezopotámii okolo 1500 př. n. l. a nebo Číně okolo roku 1000 př. n. l.

Kryptologie od té doby prošla velkým vývojem. Dnes jsou definované a veřejně známé desítky šifrovacích postupů, které vznikly napříč lidskou historií. Největším vývojem prošla kryptologie během válečných konfliktů, kdy armáda jedné strany potřebovala zaručit bezpečnou komunikaci mezi vlastními jednotkami, ale znepřístupnit tajné informace druhé straně konfliktu, která mohla komunikaci odchytit.

K většině historických šifer se po letech objevily postupy, jak zašifrovaný text prolomit i bez znalosti tajného klíče. Používání prolomené šifry se tedy stalo nebezpečným a jako reakce na to se zvýšil zájem o novější, bezpečnější způsoby šifrování.

Klasické šifrování, ke kterému stačí pouze tužka a papír, postupem času vymizelo a dnes se téměř každý den setkáváme s moderní kryptografií, ve které lze díky moderní elektronice dosáhnout daleko větší bezpečnosti.

Cílem práce je prozkoumat vybrané historické šifrovací algoritmy, seznámit s kryptoanalytickými útoky na zašifrované zprávy a změřit přibližnou úspěšnost těchto útoků.

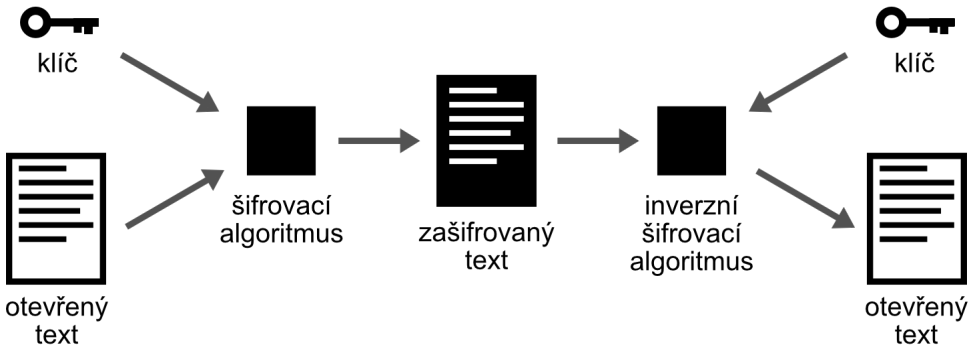
Zdroje k 1. kapitole: [2], [5], [6], [13]

1.1 Definice základních pojmů

- *Kryptografie* je věda zabývající se šifrovacími algoritmy.
- *Kryptoanalýza* je věda zabývající se prolamováním šifrovacích algoritmů.
- *Kryptologie* je obecně věda zabývající se šifrováním. Zahrnuje kryptografii i kryptoanalýzu.
- *Otevřený text* je text zprávy určené k zašifrování. V ukázkových příkladech této práce se píše malými písmeny.
- *Zašifrovaný text* je text zašifrovaný některým z šifrovacích algoritmů. Člověku může připadat jako nesmyslná posloupnost náhodných znaků. Zašifrovaný text má v sobě stále ukryt původní otevřený text. Zpravidla se píše velkými písmeny.
- *Šifrovací klíč* může být libovolný znak, posloupnost znaků, číslo, nebo i celá abeceda znaků. Záleží na zvolené šifře, každá definuje svou množinu možných klíčů. K šifrování i rozšifrování je klíč nezbytný a spolu se zvolenou

šifrou určuje přesný postup zašifrování otevřeného textu a opačně i pro rozšifrování.

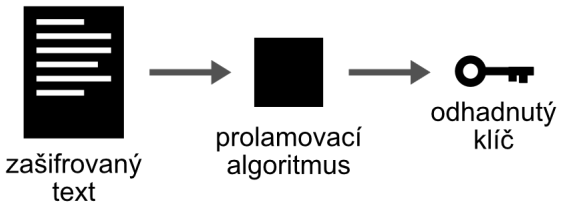
- *Šifra* Šifrou se většinou rozumí šifrovací algoritmus, který z otevřeného textu a šifrovacího klíče vytvoří zašifrovanou zprávu, která je na první pohled nepochopitelná. Taková zpráva se může dostat ke komukoli, komu není určena, ale nebude ji schopen přečíst. Pouze pověřená osoba, pro kterou je zpráva určena, zná klíč i použitou šifru a pomocí inverzního algoritmu může zprávu rozšifrovat na původní čitelný text. Celý šifrovací proces ukazuje následující obrázek:



Obrázek 1: Proces šifrování a dešifrování

Šifrovací algoritmus je tedy přesný popis, jak text přepsat na nečitelný šifrovaný za použití určitého klíče. K němu inverzní algoritmus popisuje přesně opačný proces. Oba algoritmy jsou většinou veřejně známé a pouze klíč je tajná informace.

- *Prolomení šifry* neboli *rozluštění šifry* znamená uhodnutí klíče ze samotného zašifrovaného textu. Slouží k tomu vybraný prolamovací algoritmus, který text analyzuje a vypočítá nejpravděpodobnější klíč. Proces ukazuje obrázek 2.



Obrázek 2: Proces prolomení šifry – získání klíče

- *Frekvenční analýza* je jedna ze základních kryptoanalytických metod. Jedná se o výpočet jednotlivých znaků v textu a vytvoření procentuální statistiky, která určuje, jak často se která písmena v textu vyskytují.

- *Bigramy* jsou dvojice písmen jdoucí po sobě. Nápodobně *trigramy* trojice písmen. Pokud známe procentuálně nejpoužívanější bigramy a trigramy jazyka, můžeme je využít k posílení přesnoti algoritmů pracujících s frekvenční analýzou.

- *Steganografie* je věda zabývající se ukrytím zpráv, aby k nim neměl přístup nikdo nepověřený. Slovo pochází z řeckých *steganos* (schovaný) a *graphein* (psát) [2]. Patří sem například psaní zpráv neviditelným inkoustem nebo zazdění kamenné tabulky se zprávou. Ukryté texty většinou bývají navíc zašifrované.

1.2 Typy šifer

1.2.1 Substituční šifry

U substituční šifry je každý znak otevřeného textu nahrazen (substituován) za jiný znak nebo skupinu znaků. Většinou se definuje přepisovací (šifrovací) abeceda pro monoalfabetickou nebo více takových abeced pro polyalfabetickou šifru. Množina šifrovacích abeced společně s původní abecedou zvoleného jazyka tvoří vše potřebné k šifrování.

Speciální případ je Vernamova šifra, jejíž klíč je stejně dlouhý jako otevřený text a každý znak otevřeného textu se šifruje jiným znakem z klíče.

1.2.2 Transpoziční šifry

Na rozdíl od substitučních šifer transpoziční nepřepisují znaky, ale pouze přehazují jejich pořadí. Šifrováním se nemění četnost znaků a k prolamování nelze použít klasické frekvenční analýzy.

1.2.3 Kombinované šifry

Existují a dodnes se používají také šifry, které kombinují postupy substitučních i transpozičních šifer dohromady. Zvýšenou složitostí algoritmů se zvyšuje i bezpečnost.

1.2.4 Asymetrické šifry

Všechny dosud probrané šifry jsou symetrické. Symetrické šifry používají k šifrování i rozšifrování stejný klíč. Asymetrické šifry používají dvojici klíčů, což markantně zvyšuje bezpečnost a výpočetní náročnost. První klíč, tzv. veřejný, zašifruje zprávu a druhý klíč, tzv. tajný, ji může zpátky rozšifrovat. Tajný klíč by měla mít k dispozici pouze pověřená osoba.

S asymetrickými šiframi se dnes setkáváme v řadě případů, např. u elektronického podpisu, ukládání citlivých dat do databáze nebo pro ověření integrity. Dále se práce zabývá pouze šiframi symetrickými.

2 Caesarova šifra

Caesarova šifra je jedna z nejjednodušších variant monoalfabetické substituční šifry. Využívá posunutí původní abecedy o předem určený počet míst, proto některé zdroje uvádí název „posuvná šifra“. Také se může vyskytnout označení „Caesarův kód“.

Zdroje ke 2. kapitole: [2], [3], [4], [7], [11]

2.1 Historie

Šifra je pojmenovaná podle římského politika a vojevůdce Julia Caesara (100 př. n. l. – 44 př. n. l.), který využíval šifru při psaní korespondence s důvěrnými informacemi (např. týkající se vojenských tažení). Bezpečnost používání šifry byla v té době vysoká, protože algoritmus nebyl veřejně známý a hodně obyvatel ani nebylo gramotných. Caesar ale údajně používal i jiné šifry.

Šifru můžeme také nalézt na některých židovských mezuzách k zašifrování božských jmen. V roce 2006 byl usvědčen člen sicilské mafie Cosa Nostra, protože vládní složky odchytily a lehce prolomily členskou komunikaci. Používali modifikovanou Caesarovu šifru (písmena přepisovali na po sobě jdoucí čísla). Podobně v roce 2011 byl Rajib Karim, bývalý zaměstnanec British Airways, usvědčen kvůli projednávání výbuchu letadel s islámskými aktivisty, přičemž údajně použili Caesarovu šifru.

2.2 Popis algoritmu

Algoritmus bere na vstupu text určený k zašifrování (otevřený text) a klíč ve tvaru jednoho písmene standardní abecedy. Z klíče se vypočítá délka posunu. Klíč ’A’ značí posun o nula pozic, ’B’ o jednu pozici atd.

Další postup se nejlépe ukáže na vypsání dvou abeced pod sebe, které společně ukazují všechny potřebné substituce. První z nich je původní abeceda používaného jazyka, druhá se nazývá šifrovací abeceda. Šifrovací abeceda se vytvoří posunutím původní o počet pozic, který udává klíč. Písmena vysunutá za levý okraj abecedy rotují na konec. Příklad abeced pro klíč ’G’:

```
abcdefghijklmnopqrstuvwxyz
GHIJKLMNOPQRSTUVWXYZABCDEF
```

V posledním kroku se každé písmeno otevřeného textu přepíše na příslušnou substituci podle posunuté abecedy a takový text se vrátí. Příklad použití s klíčem ’G’, kde první řádek značí otevřený text a druhý řádek zašifrovaný text:

```
"caesar je vazne nemocen"
"IGKYGX PK BGFTK TKSUIKT"
```

Při ručním šifrování nám může pomoci jednoduchý šifrovací disk (též „kruhový dekodér“ z anglického „ring decoder“), což je nástroj složený ze dvou kruhů, kde na každém kruhu je vypsaná abeceda. Vnější abeceda je šifrovací, s kruhem lze otáčet a tím měnit klíč. Následující obrázek ukazuje moderní variantu [18].

Obrázek 3: Šifrovací disk k Caesarově šifře

2.3 Výhody

Caesarova šifra je jednoduchá na pochopení i použití. Jako pomůcku pro zrychlení můžeme využít šifrovací disk nebo prsten.

2.4 Nevýhody

Existuje pouze tolik možných klíčů, kolik je znaků v abecedě. Typicky pro anglickou abecedu pouze 26 znaků. Z toho znak 'A' je tzv. slabý klíč, který po zašifrování otevřený text ani nepozmění. Šifrovaný text lze prolomit bez dodatečných pomůcek v řádu minut tím, že jednoduše vyzkoušíme všechny možné klíče.

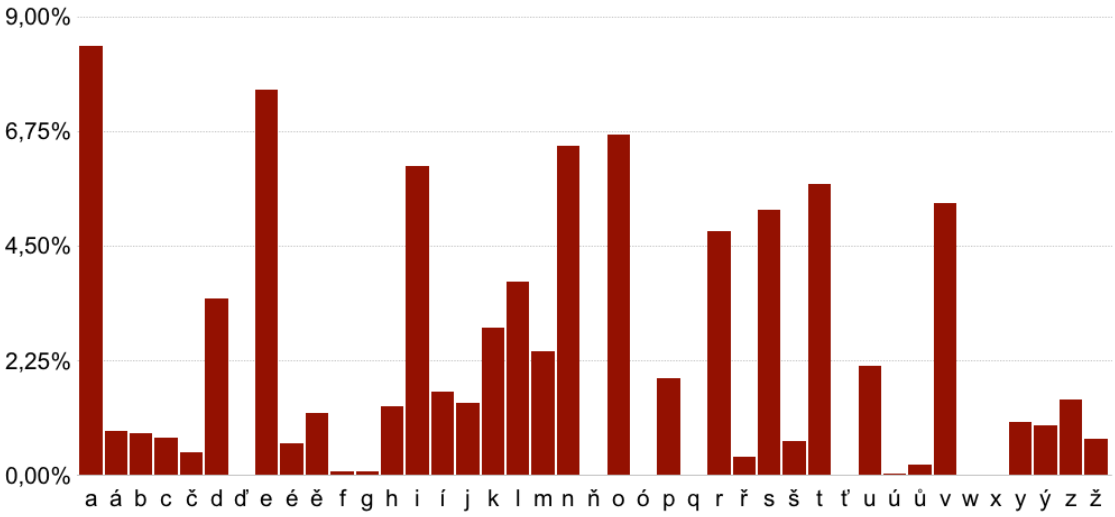
2.5 Prolamování

2.5.1 Útok hrubou silou pomocí frekvenční analýzy

Tento algoritmus využívá slabiny obecně všech monoalfabetických šifer, že ze zašifrovaného textu se dá vypočítat procentuální výskyt použitých znaků. Z každého jazyka se dá vytvořit statistika, udávající procentuální výskyt znaků, bigramů, trigramů atd.

První musíme znát předpokládaný výskyt všech písmen abecedy v daném jazyce, s kterým pracujeme. Pro češtinu ji znázorňuje graf na obrázku 4. Poté sestavíme obdobnou statistiku ze zašifrovaného textu. Inicializujeme pole s tolika prvky, kolik je znaků v abecedě. Všechny prvky jsou na počátku nuly. Projdeme celý zašifrovaný text a s každým výskytem písmene inkrementujeme o jedna příslušný prvek v poli. Nakonec každý prvek vydělíme počtem znaků v zašifrovaném

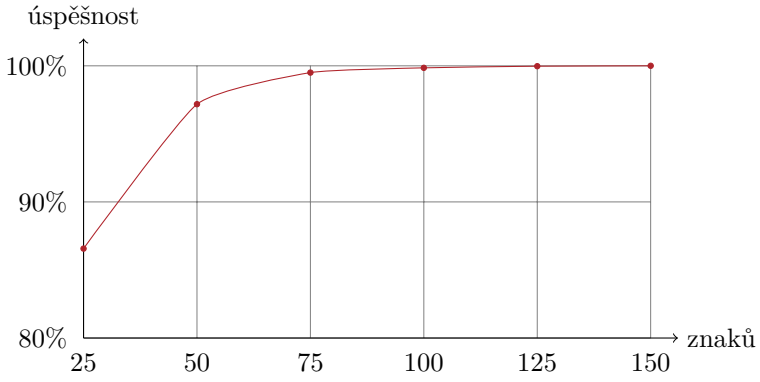
textu a máme procentuální četnost znaků. Následující graf ukazuje průměrnou četnost písmen v českém textu podle [7].



Obrázek 4: Přibližná četnost znaků v českém textu

Při luštění na papíře stačí porovnat vypočítanou statistiku se statistikou jazyka. Pro obě statistiky nakreslíme přibližný graf a položíme je pod sebe. Poté posouváme grafem nalevo nebo napravo a hledáme pozici, kde se budou grafy vertikálně nejvíc podobat.

Pro program je postup jiný. Zašifrovaný text postupně rozšifruje každým možným klíčem a výsledný rozšifrovaný text podrobí frekvenční analýze. Tuto statistiku porovná s předpokládanou statistikou daného jazyka a vypočítá odchylku. Odchylka nyní určuje podobnost zvoleného klíče s hledaným klíčem. Postup se opakuje pro všechny klíče a algoritmus vybere ten z nich, pro který vypočítal nejmenší odchylku. Takový klíč vrátí na výstupu. Úspěšnost algoritmu vzhledem k počtu znaků zašifrované zprávy ukazuje graf na obrázku 5.



Obrázek 5: Úspěšnost frekvenční analýzy na Caesarovu šifru

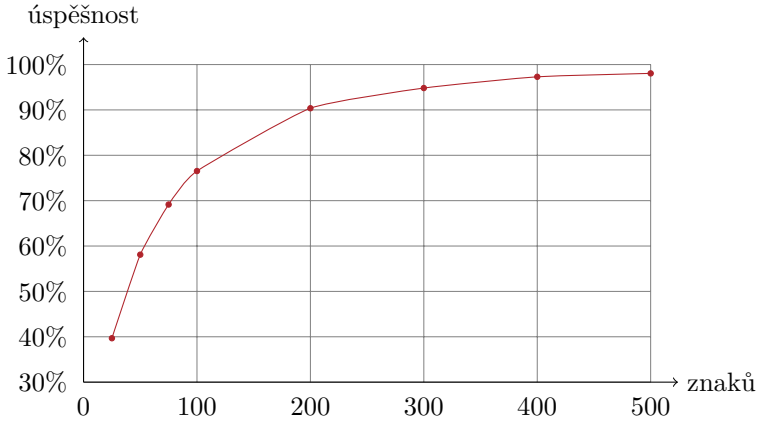
Algoritmus používá pouze základní verzi frekvenční analýzy, kde se pracuje

jen s jednotlivými znaky. Vylepšení algoritmu je možné například přidáním počítání bigramů a trigramů.

2.5.2 Útok hrubou silou hledáním reálných slov

Tento algoritmus také postupně zkouší všechny přijatelné klíče. Pro vypočítání úspěšnosti jednotlivých klíčů využívá seznam nejpoužívanějších slov daného jazyka.

Každým klíčem rozšifrujeme text a poté u něho spočítáme počet výskytů jednotlivých slov ze seznamu nejpoužívanějších slov. Výskyty slov sečteme. Součet udává úspěšnost aktuálně zkoušeného klíče. Postup zopakujeme pro každý klíč a nakonec vybereme ten nejúspěšnější. Úspěšnost s použitím seznamu o 100 nejpoužívanějších slovech ukazuje graf na obrázku 6.



Obrázek 6: Úspěšnost útoku hledáním reálných slov na Caesarovu šifru

2.5.3 Trojúhelníkový útok

Tento útok je založený na porovnávání nejčastěji (nebo nejméně často) vyskytujících se písmen textu a jejich vzájemné vzdálenosti v abecedě. Zase budeme potřebovat procentuální četnost znaků ve zvoleném jazyce.

Jako první si ze statistiky jazyka vybereme k písmen s nejčastějším výskytem a uložíme je do proměnné `langChars`. Útok je nejefektivnější, pokud zvolíme $k = 3$, proto pojmenování *trojúhelníkový*. Dále provedeme frekvenční analýzu na zašifrovaný text a uložíme si n nejčastěji vyskytujících se znaků do proměnné `textChars`. Za n si zvolíme $n = 8$. Kvůli následujícímu postupu útoku musí platit pravidlo $k \leq n$.

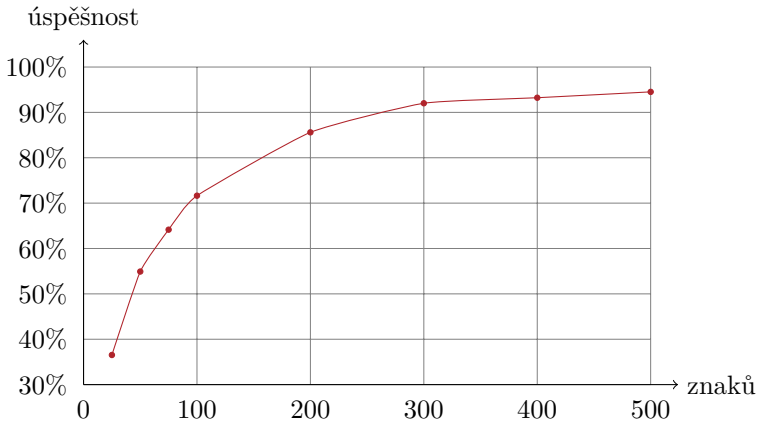
Snažíme se nalézt kombinaci k písmen z `textChars`, označme si ji K, pro kterou platí, že *minimální vzdálenost* v abecedě (viz 2.5.4) dvou libovolně vybraných písmen z ní je rovna minimální vzdálenosti stejně vybraných písmen z `langChars`. Takže MD($K[i], K[j]$) se rovná MD(`langChars`$[i]$, `langChars`$[j]$)

pro $0 \leq i, j < k$, kde operace $\mathrm{MD}(x, y)$ značí minimální vzdálenost písmen x a y v abecedě (2.5.4) a $X[y]$ je přístup do pole X na index y.

Takto nalezená kombinace K spolu s `langChars` už pravděpodobně prozrazuje klíč. Původní znak otevřeného textu `langChars[i]` se pravděpodobně zobrazil v šifrovaném textu na $K[i]$. Z tohoto předpokladu vypočítáme délku posunu a máme výsledný klíč.

Kombinace K ale nemusí být nalezena správně. Pro větší přesnost opakujeme stejný postup pro nejméně často vyskytující se znaky (jak ve statistice jazyka, tak v zašifrovaném textu). Čísla n a k zůstávají stejná, pouze na začátku algoritmu vybereme ze statistiky jazyka 6 nejméně používaných písmen místo k, protože obecně platí, že u nejméně vyskytujících se písmen je rozdíl četnosti minimální.

Nakonec porovnáme první a druhý uhodnutý klíč. Pokud jsou stejné, velmi pravděpodobně jsme našli správný klíč a algoritmus ho vrátí. Jestli ne, rozšifrujeme text oběma klíči a vybereme ten z nich, který má menší odchylku frekvenční analýzy vzhledem k předpokládané statistice jazyka. Úspěšnost útoku ukazuje graf na obrázku 7.



Obrázek 7: Úspěšnost trojúhelníkového utoku na Caesarovu šifru

2.5.4 Minimální vzdálenost písmen v abecedě

Jedná se o pomocný algoritmus využívaný při trojúhelníkovém útoku.

Vezmeme-li například písmena 'a' a 'g', pak jejich vzdálenost v abecedě je 6. Takové počítání vzdálenosti je jednoduché a intuitivní. *Minimální vzdáleností* ale rozumíme výběr menší ze dvou vzdáleností. Druhá se počítá přechodem přes konec abecedy zpět na začátek, stejně jako by abeceda byla zapsána v kruhu. Například písmena 'a' a 'q' mají klasickou vzdálenost 16, ale jejich minimální vzdálenost je pouze 10, protože přechodem přes konec abecedy je cesta kratší.

3 Vigenèrova šifra

Vigenèrova šifra je polyalfabetická (využívající více šifrovacích abeced) substituční šifra. Písmena se sice šifrují stejným způsobem jako u Caesarovy šifry, ale po každém substituovaném písmenu se změní šifrovací abeceda. Celý postup se tedy skládá z postupné aplikace několika různých Caesarových šifer.

Zdroje ke 3. kapitole: [2], [3], [10], [17]

3.1 Historie

Šifra je nesprávně pojmenovaná po francouzském diplomatovi a kryptografovi jménem Blaise de Vigenère, který roku 1586 publikoval popis podobné, ale silnější šifry. Název Vigenèrova šifra se začal mylně používat až v 19. století.

Nejstarší dochovaný popis šifry je z roku 1467, kdy italský matematik Leon Battista Alberti popsal velmi podobnou polyalfabetickou šifru. Šifrovací abecedu ale nemění po každém znaku, nýbrž až po celých slovech nebo i větách.

V roce 1508 Němec Johannes Trithemius vynalezl Vigenèrův čtverec (známý také jako Vigenèrova tabulka), tedy pomůcku k šifrování znázorňující všech 26 použitelných šifrovacích abeced. Původní latinský název je „tabula recta“.

	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 8: Vigenèrův čtverec

Šifru, kterou dnes chápeme pod pojmenováním Vigenèrova šifra, přesně popsal Giovan Battista Bellaso ve svém díle *La cifra del. Sig. Giovan Battista Bellaso* z roku 1553. Využil už existující Vigenèrův čtverec a definoval klíč o více

znacích (například slovo nebo krátká fráze), díky kterému se jasně střídají šifrovací abecedy po každém přepsaném znaku.

Šifra se prakticky začala používat až daleko později, protože na svou dobu byla moc složitá. Velmi dlouho byla považována za nerozluštitelnou. V roce 1863 publikoval německý kryptoanalytik Friedrich Wilhelm Kasiski práci, ve které popisuje prolomení „neprolomitelné“ Vigenèrovy šifry. Byl to sice první publikovaný popis prolomení, ale britský matematik Charles Babbage prolomil tuto šifru už daleko dříve v roce 1854, pouze svou práci nezveřejnil.

Vigenèrova šifra byla použita jako vojenská polní šifra v Americké občanské válce (1861–1865) stranou států Konfederace. Státy Unie jako druhá strana konfliktu pravidelně prolamovaly odchycené zašifrované zprávy.

3.2 Popis algoritmu

Algoritmus bere na vstupu otevřený text a klíč ve tvaru libovolně dlouhé posloupnosti znaků. Klíčem je většinou nějaké slovo používaného jazyka, aby si ho jednoduše zapamatovaly obě strany komunikace.

Podle prvního písmene z klíče vytvoříme šifrovací abecedu stejně jako u Caesarovy šifry. Takovou šifrovací abecedou zašifrujeme první písmeno otevřeného textu. Následuje stejný postup pro druhé písmeno klíče a druhé písmeno otevřeného textu. To se opakuje pro všechny znaky klíče. Po posledním znaku klíče se zase začíná od prvního a postup se opakuje dokud nezašifrujeme všechny znaky otevřeného textu.

3.2.1 Příklad

Zprávu „lidska povaha je plna rozporu“ zašifrujeme klíčem „capek“. Aktuální šifrovací abecedu vždy vidíme ve Vigenerově čtverci.

První zvolená šifrovací abeceda je podle znaku „c“, tedy 3. řádek čtverce. V ní se první písmeno otevřeného textu „l“ zašifruje na „N“. Další šifrovací abeceda je „a“, tedy stejná jako původní. Znak „i“ se v ní zobrazí na „I“. Třetí abeceda je podle znaku „p“, tedy 16. řádek čtverce. Znak „d“ se v ní zobrazí na „S“.

Po zašifrování celého textu stejným způsobem dostaneme výsledný zašifrovaný text „NISWUC PDZKJA YI ZNNP VYBPDVE“.

3.3 Výhody

Šifra je daleko bezpečnější než většina monoalfabetických šifer nebo její polyalfabetičtí předchůdci. Přijímá tak velké množství možných klíčů, že útok hrubou silou je i pro dnešní počítače neřešitelný v rozumném čase. I pro poměrně krátký klíč o 7 znacích by to znamenalo 26^7 možností, tedy 8 miliard možných klíčů.

Další výhodou je odolnost proti frekvenční analýze. Samotná četnost znaků nebo bigramů zašifrovaného textu nenese žádnou důležitou informaci, které by se při prolamování dalo využít.

3.4 Nevýhody

Složitost šifry odrazuje od jejího použití. Bez pomocného Vigenèrova čtverce nebo alespoň šifrovacího disku je proces šifrování na papíře velmi zdlouhavý.

Použití klíče o jednom znaku znamená použití obyčejné Caesarovy šifry. Také znak „a“ v klíči šifru oslabuje, protože stejně jako u Caesarovy šifry nevytváří žádný posun.

Asi o tři století později od vynálezu Vigenèrovy šifry se objevily slabiny využitelné při kryptoanalýze, například hledání skupin stejných po sobě jdoucích znaků k určení délky klíče. Také můžeme využít předpokladu, že většina klíčů tvoří reálná slova a ne pouze náhodné znaky.

3.5 Prolamování

3.5.1 Odhadnutí délky klíče

Délku klíče odhadneme pomocí výskytu stejných shluků znaků v zašifrovaném textu. V otevřeném textu se může vyskytovat stejné slovo vícekrát. Předpokládáme, že se některá z takových slov zašifrovala stejným způsobem. Například poměrně časté slovo „nebo“ se v českém textu pravděpodobně objeví více jak jednou. Při zašifrování klíčem o délce 6 je pouze 6 možností, jak se takové slovo zašifrovalo. Jeho 7. výskyt už určitě prozradí možnou délku klíče.

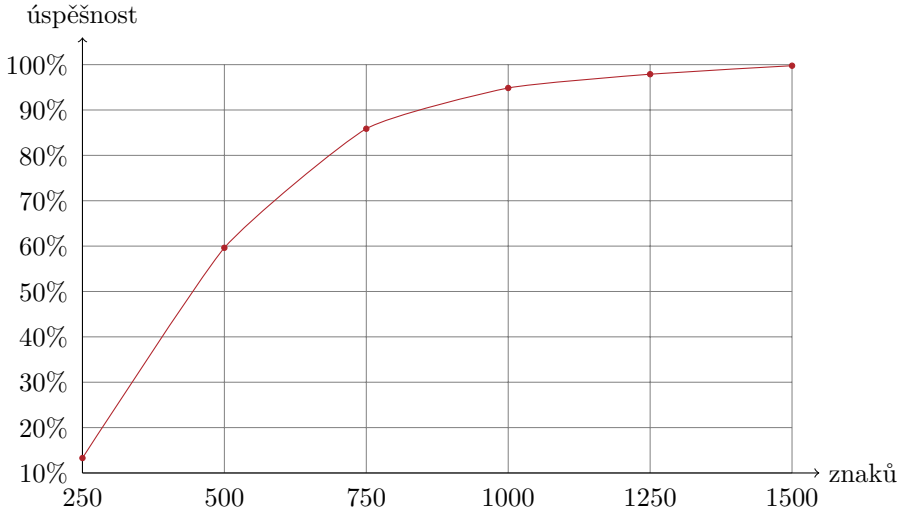
V prvním kroce si sestavíme seznam možných délek klíče. Při luštění na papíře bychom čísla vypsali pod sebe. Program si inicializuje pole `divisors` a všechny jeho prvky nastaví na číslo 0. Předpokládáme, že délka klíče se pohybuje mezi 4 až 10 znaky. Pole `divisors` tedy bude mít 7 prvků.

V zašifrovaném textu postupně nalezneme všechny shluky znaků o délce n, kde $n \in \{4, 5, 6, 7\}$, které se v textu vyskytují více jak jednou. Pro $n < 4$ můžeme nalézt rozdílné sekvence otevřeného textu zašifrované jinou částí klíče, které náhodou dávají stejný zašifrovaný výsledek (z knihy [2]). Z jejich vzdálenosti vypočítáme všechny dělitele. Každý takový dělitel je možná délka klíče. Množinu dělitelů omezíme minimální a maximální předpokládanou délkou klíče. Poté pro každý dělitel inkrementujeme příslušný prvek v poli `divisors` o 1.

Po zpracování všech stejných shluků se z pole `divisors` vybere dělitel, který má nejvyšší hodnotu a vyskytl se tedy nejvícekrát. Pokud je v poli více prvků s nejvyšší hodnotou, můžeme si vybrat, jestli předpokládáme výskyt krátkého nebo dlouhého klíče. Úspěšnost uhodnutí délky klíče pro generované klíče v rozmezí 4 až 10 znaků ukazuje graf na obrázku 9.

3.5.2 Útok pomocí frekvenční analýzy

Pomocí algoritmu 3.5.1 zjistíme nejpravděpodobnější délku klíče a označíme ji k. Kdybychom délku klíče nemohli přesně určit, můžeme využít hrubé síly. Následující postup by se tak zopakoval pro každé k z množiny definované minimálním a maximálním počtem znaků klíče.



Obrázek 9: Úspěšnost odhadnutí délky klíče u Vigenèrovy šifry

Zašifrovaný text T o délce n rozdělíme na k posloupností písmen. Pro každou posloupnost P_i platí:

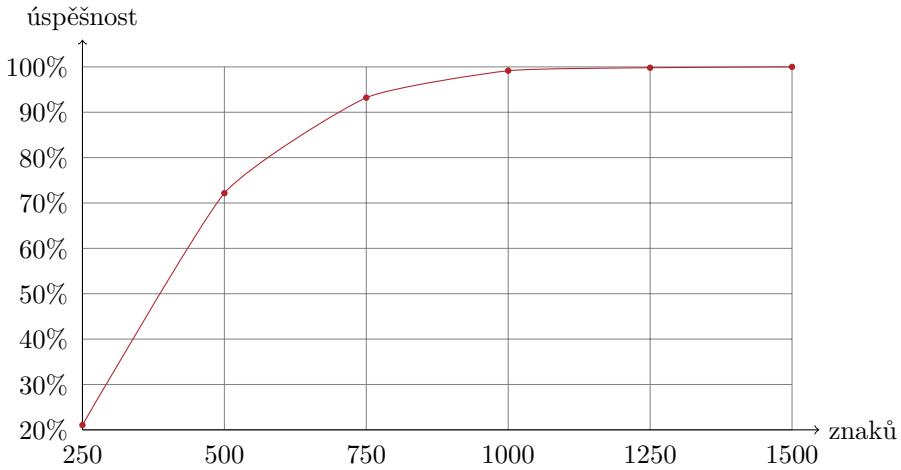
$$P_i[j] = S[i + j \cdot k] \ .$$

Pro $0 \leq i < k$ a zároveň $0 \leq (i + j \cdot k) < n$, kde $X[y]$ vrací y-tý znak řetězce X.

Nyní máme v každé posloupnosti P_i takové znaky zašifrovaného textu, které se zašifrovaly stejnou šifrovací abecedou. Pro každou posloupnost zavoláme algoritmus pracující s frekvenční analýzou (viz. 2.5.1 na straně 12) a zjistíme klíč, kterým byly znaky posloupnosti pravděpodobně zašifrovány.

Výsledný klíč zašifrovaného textu se postupně složí z výsledků frekvenční analýzy na každé posloupnosti P_i. Úspěšnost útoku ukazuje graf na obrázku 10.

Při měření program generoval klíče o maximální délce 7 znaků. Pro menší maximální délku klíče jsou výsledky příznivější, pro větší naopak nepříznivější.

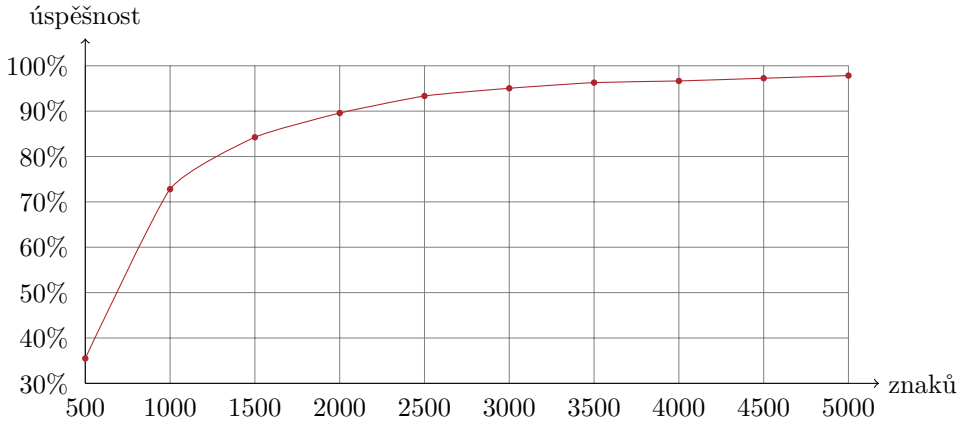


Obrázek 10: Úspěšnost frekvenční analýzy na Vigenèrovu šifru

3.5.3 Trojúhelníkový útok

Pomocí algoritmu 3.5.1 zjistíme nejpravděpodobnější délku klíče a označíme ji k. Následující postup je stejný jako u předchozího algoritmu 3.5.2, tedy že text rozdělíme na k posloupností písmen podle zmíněných pravidel.

Dále pro každou takovou posloupnost zjistíme znak klíče, kterým byla zašifrována, pomocí algoritmu 2.5.3 na straně 14. Výsledné znaky postupně skládáme za sebe a tím vytvoříme konečný klíč. Úspěšnost ukazuje graf na obrázku 11.



Obrázek 11: Úspěšnost trojúhelníkového útoku na Vigenèrovu šifru

4 Obecná monoalfabetická šifra

Základní monoalfabetická substituční šifra používá jako klíč libovolnou kompletní šifrovací abecedu. Proto je zobecněním pro několik dalších substitučních šifer, například pro Caesarovu nebo afinní šifru, které mají daleko jednodušší a snadněji zapamatovatelné klíče, podle kterých se následně celá šifrovací abeceda dopočítá.

Zdroje ke 4. kapitole: [1], [2], [14], [15]

4.1 Historie

Do historie monoalfabetických šifer tedy patří i historie Caearovy šifry (2.1). Ještě starší záznamy ukazují použití hebrejské šifry Atbaš. Ta je speciální podobu monoalfabetické šifry, která jako šifrovací abecedu používá převrácenou původní. Znak 'a' se tedy zašifruje na 'Z', 'b' na 'Y' atd. Šifra tedy neuvažuje žádný klíč, protože vždy používá jen jeden. Byla použita ve starých biblických textech z *Knihy Jeremjášovy* z 6. století př. n. l.

Šifrování bylo hojně používáno Araby v době zlatého věku islámské civilizace. Kniha *Adab al-Kuttáb* (Příručka úředníkova) z 10. století obsahuje pasáže o šifrování monoalfabetickou šifrou, kde kromě náhodně uspořádaných písmen používali v šifrovací abecedě i jiné symboly než písmena [2]. Arabové také položili základy kryptografie.

Francie v 16. století odchytávala španělské zprávy šifrované monoalfabetickou šifrou. Španělská kryptoanalýza byla v té době na velmi nízké úrovni. Francouzi už dávno věděli jak efektivně prolomit monoalfabetickou šifru, zatímco Španělé si mysleli, že jsou zprávy bezpečně zašifrované.

4.2 Popis algoritmu

Šifrovací algoritmus přepíše každé písmeno otevřeného textu na jeho obraz v šifrovací abecedě. Každý znak má tedy právě jeden obraz a šifrovací abeceda se během šifrování nijak nemění. Stejně jako u Caesarovy šifry (2.2) si pod sebe vypíšeme původní a šifrovací abecedu jako pomůcku k přepisování.

4.2.1 Příklad

Chceme zašifrovat zprávu „Největším vítězstvím je nepotřebovat žádné vítězství“ klíčem „gcijfqmxdnakstplyhbeouvzwr“.

```
abcdefghijklmnopqrstuvwxyz
GCIJFQMXDNAKSTPLYHBEOUVZWR
```

První písmeno otevřeného textu 'n' se přepíše na 'T', druhé 'e' na 'F' a stejně tak pro každé další písmeno otevřeného textu. Výsledný zašifrovaný text bude „TFNUFEBDS UDEFREUDS NF TFLPEHFCPUGERGJTF UDEFREUD“.

4.2.2 Doplnění písmen do šifrovací abecedy

Kvůli nevýhodě moc dlouhého a těžko zapamatovatelného klíče se dá využít postup, který takový klíč vytvoří ze slova nebo krátké fráze. Šifrovací abecedu sestavíme následovně:

Z fráze odstraníme všechny nepovolené znaky (mezery, tečky atd) a písmena s diakritikou převedeme na klasické znaky anglické abecedy ('š' → 's' atd.). Pro všechna písmena, která se vyskytují více jak jednou, odstraníme jejich duplicity a necháme pouze první výskyt. Nakonec doplníme všechna písmena abecedy, která se ve slově nevyskytují. Například pro frázi „luštit šifry" vytvoříme abecedu:

```
LUSTIFRYABCDEGHJKMNOPQVWXZ
```

4.3 Výhody

Šifra používá jako klíč libovolnou permutaci abecedy. Anglická abeceda má 26 znaků, to znamená 26! (přibližně $4,0329 \times 10^{26}$) možných klíčů. Útok vyzkoušením všech permutací by i dnešním nejmodernějším počítačům trval daleko déle než je kdokoliv ochotný čekat. Pravděpodobně i déle, než jak dlouho lidstvo na Zemi vůbec existuje.

Množství možných klíčů je hlavní výhodou oproti Caesarově šifře. Obě šifry jsou téměř stejně složité na pochopení i stejně náročné na použití, zato obecná monoalfabetická šifra poskytuje daleko vyšší bezpečnost.

K vylepšení šifry můžeme použít i znaky mimo klasickou abecedu, například jednoduché obrázky. Kdybychom takových znaků měli více než 26, můžeme vybrat právě 26 do šifrovací abecedy a zbytek využít jako tzv. klamače – znaky bez významu. Ty náhodně rozmístíme do textu za účelem zmatení nepřítele.

4.4 Nevýhody

Zašifrovaný text stále obsahuje nezměněnou četnost znaků otevřeného textu. Je tedy možné využít frekvenční analýzu, i když základní analýza na úrovni jednotlivých znaků nebude tolik efektivní. Lepších výsledků dosáhneme, zaměříme-li se i na bigramy a trigramy.

Ke zrychlení procesu šifrování nemůžeme využít známých jednoduchých pomůcek jako šifrovací disk nebo prsten.

U substitučních šifer se opakovaným šifrováním nijak nezvýší bezpečnost šifry. Kvůli tranzitivitě postupné substituce platí, že pokud v prvním kroku přepíšeme znak 'a' na 'g' a v druhém kroku 'g' na 'w', jedná se dohromady o stejnou substituci jako když přepíšeme 'a' rovnou na 'w'.

4.5 Prolamování

4.5.1 Útok hledáním unikátních slov

K využití této metody musíme mít v zašifrovaném textu zachovány mezery vyznačující začátky a konce slov. Dále musíme mít k dispozici dostatečně obsáhlý seznam nejčastěji používaných slov daného jazyka.

Jako první si ze slovníku nejčasteji používaných slov vybereme pouze unikátní slova. To jsou taková slova, která mají v daném seznamu unikátní kanonický tvar (4.5.2). Uložíme si je do pole `uniqueWords`.

Dále si zašifrovaný text rozdělíme podle mezer na jednotlivá slova a uložíme je do pole `textWords`. Inicializujeme asociační pole `pairs`. Postupně projdeme všechna slova a snažíme se najít dvojice slov (w_0, w_1), kde $w_0 \in$ `textWords` a $w_1 \in$ `uniqueWords`, aby měli navzájem stejný kanonický tvar. Všechny takto nalezené dvojice slov uložíme do pole `pairs` ve tvaru `pairs[`w_0`]` $= w_1$. Pokud jsme nenalezli žádnou dvojici, útok už dále nepokračuje a končí neúspěchem.

Nyní některé z dvojic slov představují vzory otevřeného textu. Velmi pravděpodobně jsou mezi nimi dvojice, které nejsou správné. Musíme rozhodnout, které dvojice jsou správné a dále pracovat pouze s nimi. K tomu využijeme faktu, že všechny správně nalezené dvojice definují stejné přespisovací pravidla. Nebo alespoň taková pravidla, která si navzájem neprotiřečí. Najdeme největší možnou množinu dvojic ve které platí, že všechny dvojice slov tvoří ekvivalentní substituční tabulky (viz. 4.5.3) a následně všechny substituce sjednotíme do jedné tabulky. Tabulka sice pravděpodobně není úplná, ale můžeme z ní vytvořit částečnou šifrovací abecedu, která může vypadat například takto:

```
abcdefghijklmnopqrstuvwxyz
G**JF**X*NA*S*PL**B*****WR
```

Nyní budeme zase potřebovat seznam nejčastěji používaných slov daného jazyka. V cyklu budeme opakovat následující kroky a postupně doplňovat jednotlivá substituční pravidla do tabulky:

1. Z neúplné substituční tabulky vytvoříme klíč a rozšifrujeme text, ten poté rozdělíme na jednotlivá slova podle vyznačených mezer.

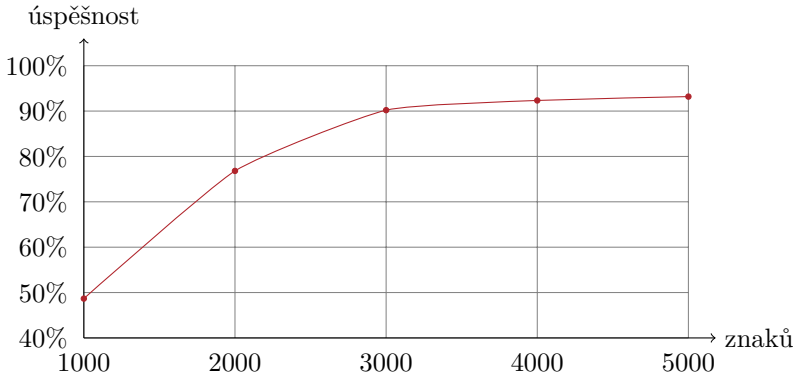
2. Mezi slovy hledáme taková slova, která se nerozšifrovala kompletně kvůli jednomu znaku (obsahují jednu hvězdičku). Pokud žádná taková slova nenalezneme, podařilo se pravděpodobně rozšifrovat celý text a cyklus zde úspěšně končí. Jinak si uložíme všechna slova s jednou hvězdičkou.

3. Nalezneme takové slovo s jednou hvězdičkou, ve kterém se za hvězdičku dá doplnit libovolné písmeno tak, aby výlsedné slovo bylo ekvivalentní s některým reálným slovem ze slovníku. Obě nalezená slova si uložíme.

4. Podle nekompletně rozšifrovaného slova a reálného slova ze slovníku doplníme chybějící substituci jako dvojici písmen do substituční tabulky. Jestli tabulka obsahuje všechny možné substituce, cyklus zde úspěšně končí.

Pokud se cyklem nepodařilo tabulku dostatečně doplnit, můžeme postup opakovat s hledáním dvou hvězdiček ve slově, popřípadě počet hvězdiček navyšovat.

Substituční tabulka nakonec obsahuje potřebné substituční pravidla, z kterých se vytvoří výsledný klíč. Kompletní tabulku se všemi substitucemi často nelze správně doplnit, protože některá písmena se nemusí vyskytovat v textu ani v použitém slovníku. Proto je můžeme doplnit podle předpokládané frekvence, náhodně nebo nedoplnit vůbec a nechat klíč nekompletní. Úspěšnost útoku ukazuje obrázek 12.



Obrázek 12: Úspěšnost útoku unikátními slovy na monoalfabetickou šifru

4.5.2 Kanonický tvar

První pomocný algoritmus slouží k vytvoření kanonického tvaru řetězce.

Každé slovo nebo obecně posloupnost libovolných znaků má svůj příslušný kanonický tvar. Slovo si označíme jako w. Kanonický tvar slova w je poslouponost čísel, kterou vytvoříme následujícím algorimem:

1. Inicializujeme asociativní pole `substitutions`, proměnnou `result` jako prázdný řetězec a číselnou proměnnou `counter`, kterou nastavíme na 0.
2. Procházíme slovo w po znacích. Pro každý znak $w[i]$ provedeme:
   (a) Pokud pole `substitutions` obsahuje cokoliv na pozici $w[i]$, pak přeskočíme cyklus rovnou na další písmeno $w[i+1]$, jinak nastavíme `substitutions[`$w[i]$`]` = counter.
   (b) Proměnnou `counter` inkrementujeme o jedna.
   (c) Číslo `substitutions[`$w[i]$`]` si uložíme do proměnné `c`. Na konec řetězce `result` připojíme číslo `c` ve dvoumístném formátu. Tedy pokud je číslo `c` menší než 10, připojíme k řetězci první znak '0' a až poté číslo `c`, jinak připojíme rovnou `c`.
3. Výsledný kanonický tvar je uložený v proměnné `result`.

Například slovo „lampa“ má kanonický tvar „0001020301“. Číslo z proměnné `c` se k výsledku přidává v dvoumístném tvaru proto, aby každé písmeno mělo stejně dlouhou reprezentaci. Jinak by u dlouhých slov mohlo dojít k tomu, že nevíme jestli „10“ ve výsledku znamená dvě písmena nebo jedno. Naše abeceda má 26 znaků, a proto nám stačí dvoumístný formát.

4.5.3 Ekvivalence substitučních tabulek

Druhý pomocný algoritmus porovnává námi definovanou ekvivalenci dvou substitučních tabulek.

Substituční neboli přepisovací tabulka je množina dvojic znaků, která definuje přepisovací pravidla u monoalfabetických substitučních šifer. Nemusí z ní být jasně odvoditelný klíč, protože nemusí obsahovat pravidla pro všechny znaky použité abecedy.

Substituční tabulka může být reprezentována například jako asociativní pole s pravidly zapsanými ve tvaru $c_i \rightarrow c_j$.

Dvě substituční tabulky T_0 a T_1 nazveme ekvivalentní, právě když pro každé pravidlo $c_i \rightarrow c_j \in T_0$ platí, že se buď stejné pravidlo se nachází i v T_1, nebo že $c_i \rightarrow c_k \notin T_1$ pro $k \neq j$ a zároveň $c_k \rightarrow c_j \notin T_1$ pro $k \neq i$.

5 Sloupcová transpozice

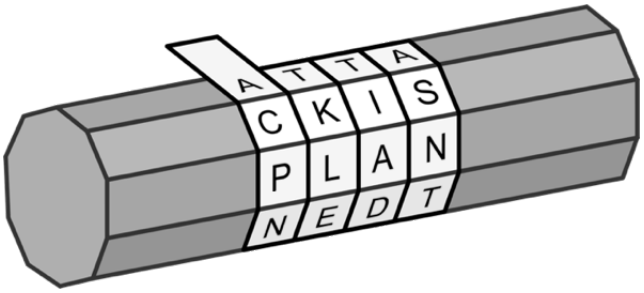
Ve sloupcové transpozici se nevyužívá žádné substituce písmen jako u předchozích šifer. Využívá se pouze přehazování pořadí jednotlivých písmen.

Zdroje k 5. kapitole: [2], [9], [12], [16]

5.1 Historie

Záznamy o použití transpozičních šifer sahají až do 7. století před naším letopočtem. Tehdy se v Řecku využíval válec zvaný Skytalé. Na válec se namotal pásek kůže nebo pergamenu, na který se poté napsala zpráva. Po odmotání má výsledný zašifrovaný text přeházené pořadí znaků. Zmínky o Skytalé sahají sice daleko, ale první dochovaný popis fungování je až z 1. století našeho letopočtu [8].

Skytalé je primitivní předchůdce jednoduché sloupcové transpozice. Dá se simulovat jednoduchou sloupcovou transpozicí tak, že použijeme klíč s abecedně seřazenými písmeny.



Obrázek 13: Skytalé

5.2 Popis algoritmu

První si písmena otevřeného textu poskládáme do obdélníku (tabulky). Pro klíč o délce n bude mít každý řádek právě n znaků otevřeného textu. První řádek prvních n znaků textu, druhý řádek dalších n znaků atd. Pokud na poslední řádek nevyjde n znaků a řádek tedy nebude úplně zaplněn, doplní se do něj náhodně vygenerovaná písmena.

Poté seřadíme podle abecedy jednotlivé znaky klíče a způsob zpřeházení si zapamatujeme. Například klíč „sen“ se podle abecedy seřadí na „ens“ a způsob zpřeházení si uložíme jako uspořádanou n-tici $(1, 2, 0)$. Stejným způsobem pak uspořádáme každý řádek. Tabulku přečteme po sloupcích a to je náš výsledný zašifrovaný text.

Klíče skládající se z abecedně seřazených písmen nebo klíče o jednom znaku neposkytují žádné zpřeházení. Pro klíče délky 5 a více s právě jedním přehozením písmen (např „abdce“) se zašifrovaný text také dá bez větší námahy prolomit.

5.2.1 Příklad

Chceme zašifrovat zprávu „Smích chytré lidi léčí, blbce uráží“ klíčem „werich“. Nejdříve z textu odstraníme vše kromě písmen (mezery, tečky, čárky a diakritiku). Poté text zarovnáme do obdélníku, a jelikož do úplného zaplnění chybí na posledním řádku jeden znak, doplní se náhodné písmeno:

w	e	r	i	c	h
s	m	i	c	h	c
h	y	t	r	e	l
i	d	i	l	e	c
i	b	l	b	c	e
u	r	a	z	i	f

Písmena v klíči seřadíme podle abecedy, permutace zpřeházení je $(4, 1, 5, 3, 2, 0)$. Stejným způsobem seřadíme sloupce.

c	e	h	i	r	w
h	m	c	c	i	s
e	y	l	r	t	h
e	d	c	l	i	i
c	b	e	b	l	i
i	r	f	z	a	u

Tabulku přečteme zprava po sloupcích. Výsledný zašifrovaný text je „HEECI MYDBR CLCEF CRLBZ ITILA SHIIU“.

5.3 Výhody

Zašifrovaný text má stejnou četnost znaků jako text otevřený. Frekvenční analýza nám proto při luštění nijak nepomůže.

Pokud bychom chtěli použít klíč o n znacích, existuje $n!$ různých klíčů, které text zašifrují navzájem různým způsobem. Pro klíč o 16 znacích to znamená 20922789888000 možných klíčů, takže útok hrubou silou by i velmi silnému počítači trval nesmyslně dlouho.

5.4 Nevýhody

Proces šifrování ručně je zdlouhavý a nemáme žádnou zrychlovací pomůcku. Pouze pro základní transpozici lze použít Skytalé (popřípadě hranatou tužku nebo jiný válec).

Z délky textu lze jednoduše vypočítat možné délky klíče. Délka klíče musí totiž vždy být dělitel délky textu. Tuto slabost šifry řeší modifikace šifrovacího algoritmu s neúplnou tabulkou (5.6.1).

Mnoho různých klíčů šifruje stejným způsobem. Znaky klíče se totiž můžou abecedně uspořádat stejným způsobem. Například klíče „argo“, „wzxy“ a „adbc“ zašifrují text stejně. Při luštění pak stačí vyzkoušet pouze jeden z nich.

Pro krátké klíče je šifra lehce prolomitelná hrubou silou. Pokud bychom předpokládali, že klíč má minimálně 2 a maximálně 5 znaků, znamenalo by to vyzkoušet $2! + 3! + 4! + 5!$, tedy 152 možných klíčů (včetně slabých), což je prolomitelné i pro ruční luštění s tužkou a papírem.

5.5 Prolamování

5.5.1 Útok nalezením slova v řádku

Zvolíme si minimální a maximální délku hledaného slova (klíče). Z délky zašifrovaného textu vypočítáme všechny možné dělitele a z výsledné množiny vezmeme pouze ty dělitele, kteří splňují minimální a maximální délku. Tato čísla uložíme do pole `keyLengths`. Nyní známe všechny možné délky klíče, protože při procesu šifrování se otevřený text zarovnává do tabulky.

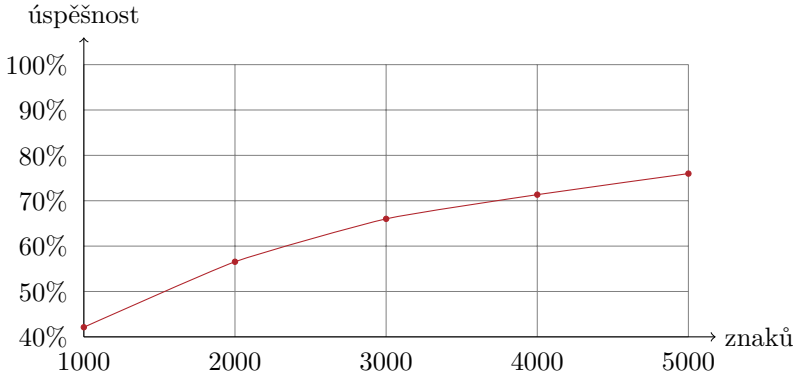
Pro každé n z `keyLengths` rozdělíme zašifrovaný text na n částí, kde každá část má počet znaků rovný $(length/n)$, kde $length$ značí délku zašifrovaného textu. Tyto části zarovnáme jako sloupce svisle vedle sebe a budeme je postupně číst po řádcích. Každý řádek má přesně n znaků.

Nyní se snažíme nalézt takový řádek, který se skládá ze stejných písmen jako některé slovo ze seznamu slov zvoleného jazyka, který máme k dispozici. Z takového seznamu získáme pouze ty slova, které mají délku n. Program porovná všechna slova tak, že každé slovo rozloží na setřízené pole znaků a až tyto pole porovná. Nalezené dvojice slov skládajících se ze stejných znaků si uložíme do slovníku `foundPairs` ve tvaru (slovo ze seznamu slov -> řádek šifrovaného textu). To se v cyklu zopakuje pro každé n z `keyLengths`. Nyní máme v `foundPairs` všechny nalezené dvojice.

Pro každou dvojici slov z `foundPairs` zjistíme, jaké přeházení znaků bylo potřeba k transformaci slova ze slovníku na slovo šifrovaného textu. Způsob přeházení (permutaci), např. $[0, 3, 1, 2]$, přidáme do pole `letterOrders`. Ze všech zjištených permutací v `letterOrders` vymažeme duplicity, které se mohly objevit. Např. klíče „adbc“ a „jzkl“ při šifrování přehází pořadí znaků stejným způsobem. Permutace v `letterOrders` nyní reprezentují možné klíče. Převedeme je na řetězce znaků následovně:

Každé číslo z permutace nahradíme vzorem z množiny znaků písmen tak, že číslo 0 se zobrazí na znak 'a', číslo 1 na znak 'b' atd. Např. pro permutaci $[0, 3, 1, 2]$ vytvoříme klíč „adbc“. Výsledný klíč přidáme do pole `possibleKeys` a totéž provedeme pro každou permutaci. Nakonec máme v proměnné `possibleKeys` uloženy všechny pravděpodobné klíče.

Z klíčů v `possiblePairs` vybereme jako nejpravděpodobnější ten klíč, po jehož rozšifrování textu nalezneme v textu nejvíce reálných slov ze seznamu nejpoužívanějších slov jazyka, který máme k dispozici. Algoritmus vrací právě takový nejpravděpodobnější klíč. Úspěšnost algoritmu ukazuje graf na obrázku 14.



Obrázek 14: Úspěšnost útoku nalezením slova v řádku na transpoziční šifru

5.5.2 Útok hrubou silou hledáním reálných slov

Pokud byl text zašifrován krátkým klíčem, můžeme využít útok hrubou silou a spočítat počet reálných slov. Algoritmus v prvním kroku zjistí všechny možné délky klíče a uloží je do proměnné `keyLengths`. Pro každé n z `keyLengths` vytvoříme množinu čísel $\{0, 1, ..., n-1\}$ a pro každou permutaci takové množiny vytvoříme klíč pomocí známé funkce ($0 \rightarrow$ 'a', $1 \rightarrow$ 'b', ... , $25 \rightarrow$ 'z'). Všechny takto vytvořené klíče uložíme do proměnné `possibleKeys`.

V cyklu projdeme každý klíč z `possibleKeys` a zkusíme jím rozšifrovat zašifrovaný text. Rozšifrovaný text následně projdeme a hledáme výskyt slov ze seznamu nejčastěji používaných slov jazyka, který máme k dispozici. Úspěšnost aktuálně testovaného klíče určíme jako počet takových nalezených slov. Ze všech klíčů vybereme pouze ten nejúspěšnější.

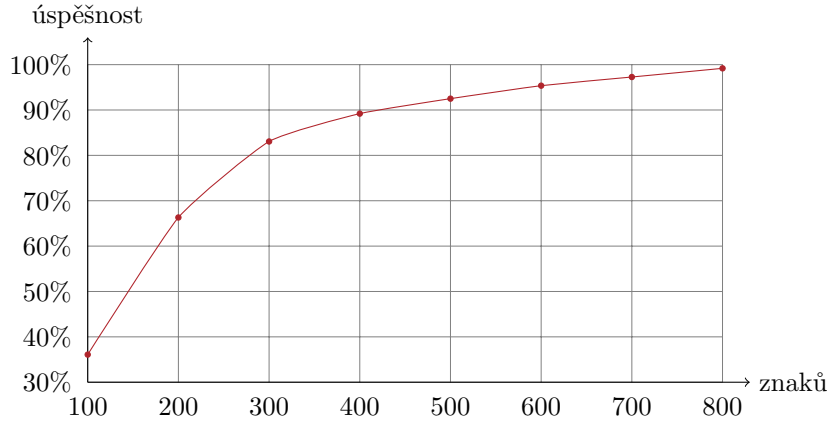
Proces opakujeme pro každou možnou délku klíče n z proměnné `keyLengths`. Jako odhadnutý použitý klíč vrátíme ten nejúspěšnější z nich.

Algoritmus testuje všechny permutace, a proto je velmi neefektivní pro dlouhé klíče. Testováním jsem zjistil, že v rozumném čase lze hledat klíče do délky 7 znaků. Úspěšnost algoritmu ukazuje graf na obrázku 15 na straně 30. Při měření jsem počítal s tím, že náhodně generované klíče mají délku nanejvýš 7 znaků.

5.6 Variace sloupcové transpozice

5.6.1 Jednoduchá sloupcová transpozice s neúplnou tabulkou

Algoritmus šifrování je stejný jako u klasické jednoduché sloupcové transpozice, až na doplnění náhodných písmen na posledním řádku sestavené tabulky. Ty se nechají prázdné a tedy ne každý sloupec má stejnou výšku. Šifra se takto stává bezpečnější, protože odstraňuje slabinu jednoduše zjistitelné délky klíče.



Obrázek 15: Úspěšnost útoku hledáním reálných slov na transpoziční šifru

5.6.2 Dvojitá sloupcová transpozice

Pravidla šifrování se oproti jednoduché sloupcové transpozici nemění. Jediný rozdíl oproti obyčejnému postupu je, že se text zprávy zašifruje jednoduchou sloupcovou transpozicí dvakrát po sobě. Na druhý krok může být použit stejný nebo rozdílný klíč. Dvojitá transpozice zvyšuje bezpečnost a množství možných klíčů, obzvlášť při volbě rozdílných klíčů. Rozšifrování se provede obdobně jako u jednoduché transpozice, ale použité klíče se musí pochopitelně zadat v opačném pořadí. Množství použitých klíčů teoreticky můžeme ještě navýšit.

5.6.3 Myszkowskiho transpozice

Tuto modifikaci sloupcové transpozice navrhl francouzský plukovník Émile Victor Théodore Myszkowski v knize *Cryptographie indéchiffrable* z roku 1902. Řeší problém opakujících se písmen v klíči. Z těch sloupců tabulky, které spadají pod stejná písmena klíče, se znaky čtou postupně po sobě. Například pro klíč „lampa“ zašifrujeme text „věci na stole“ následovně:

l	a	m	p	a
v	e	c	i	n
a	s	t	o	l
e	x	g	r	y

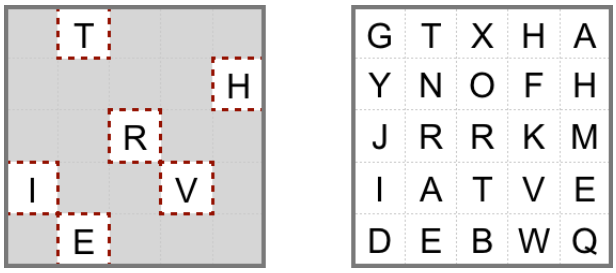
a	a	l	m	p
e	n	v	c	i
s	l	a	t	o
x	y	e	g	r

První dva sloupce uspořádané tabulky (vpravo) pak přečteme jako jeden. Zašifrovaný text bude „EN SL XY VAE CTG IOR“.

5.6.4 Transpoziční mřížka

Mřížka je předmět, který má libovolné množství nepravidelně rozmístěných děr. Šifrování probíhá tak, že se znaky otevřeného textu zapisují do otvorů v mřížce a jakmile jsou všechny vyplněny, mřížka se posune a proces se opakuje.

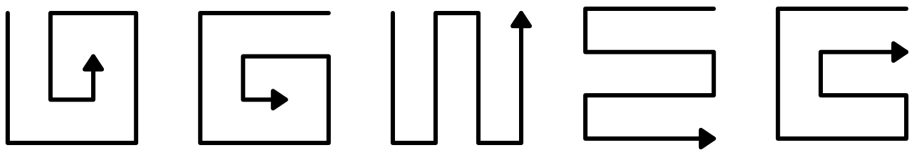
Zbylá prázdná místa (kde mřížka neměla otvor) se poté vyplní náhodně zvolenými znaky. Čím méně děr v mřížce je, tím je sice šifra bezpečnější, ale výsledná zpráva delší. Další varianta šifry mřížku neposouvá, ale otáčí o určitý úhel (čtverec o 90°, šestiúhelník o 60°, atd). Obě strany šifrované komunikace musí sdílet tutéž mřížku.



Obrázek 16: Zašifrování slova transpoziční mřížkou

5.6.5 Primitivní transpozice

Jeden z nejjednodušších způsobů transpozice je obyčejné zapsání otevřeného textu do obdélníku a přečtení po sloupcích. Klíč takového šifrovaní je předem dohodnutý způsob zapsání do obdélníku. Kromě klasického po řádcích můžeme použít například spirálu a další.



Obrázek 17: Trasy pro zapsání textu

Opačný způsob využívá trasová transpoziční šifra (route šifra). Text se do obdélníku zapíše vždy stejně a poté se přečte specifickým způsobem (trasou). Trasy na obrázku 17 ukazují jen základní možnosti, jejich kombinacemi se dají vymyslet i daleko složitější.

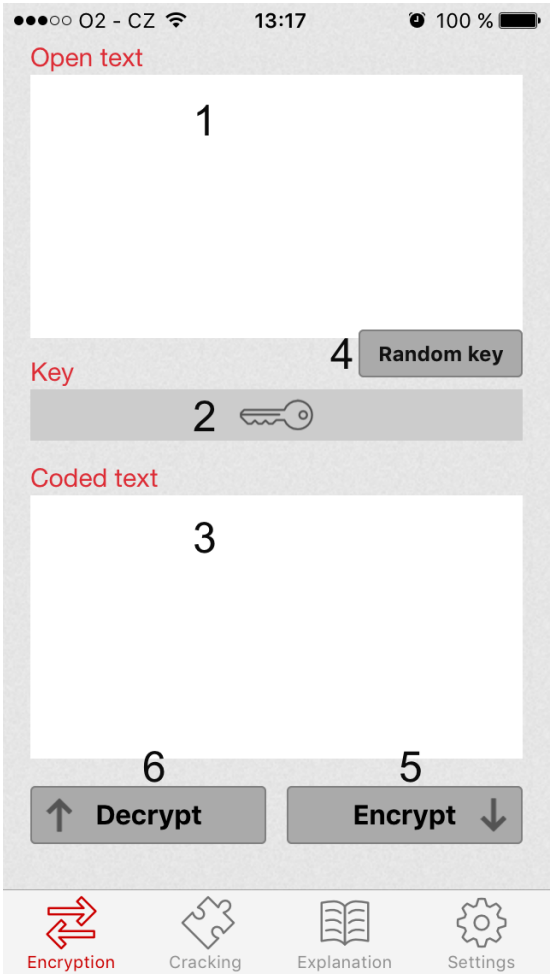
6 Uživatelská příručka

Praktickou část bakalářské práce představuje mobilní aplikace na operační systém iOS od firmy Apple. Aplikace je napsána v prostředí Xcode 7.2 v jazycích Objective-C a Swift 2 a k běhu vyžaduje systém iOS ve verzi minimálně 8.1.

Aplikace je rozdělena na čtyři části. První část se zabývá kryptografií, slouží k šifrování a dešifrování vloženého textu. V druhé části lze vyzkoušet kryptoanalytické útoky na vložený zašifrovaný text. Třetí část je výuková a snaží se popsat historii a chování šifry, ukázat šifrování na příkladech a vysvětlit možnou kryptoanalýzu. Poslední čtvrtá část je nastavení aplikace, slouží k výběru šifry, kryptoanalytického útoku na ní a výběru jazyka, s kterým algoritmy budou pracovat.

6.1 Kryptografická část

Záložka „Encryption“ slouží k provádění šifrování a dešifrování textu.



Obrázek 18: Aplikace – kryptografická část

Jednotlivé ovládací prvky jsou:

1. Textové pole pro otevřený text. Uživatel zde může vkládat text určený k zašifrování.

2. Textové pole pro klíč. Uživatel do něj může klíč napsat nebo vygenerovat tlačítkem `Random key` (4). Pokud je pole prázdné nebo klíč není validní k momentálně zvolené šifře a uživatel se snaží šifrovat, vyskočí dialogové okno s chybovou hláškou.

3. Textové pole pro zašifrovaný text. Uživatel zde může vkládat text určený k rozšifrování.

4. Tlačítko pro vložení náhodného klíče. Po stisknutí se podle momentálně zvolené šifry vygeneruje náhodný klíč a vloží se do textového pole `Key` (2).

5. Tlačítko pro zašifrování zprávy. Vezme se otevřený text z textového pole `Open text` (1), zašifruje se momentálně zvolenou šifrou a klíčem z pole `Key` (2) a výsledek se vloží do textového pole `Coded text` (3).

6. Tlačítko pro rozšifrování zprávy. Vezme se zašifrovaný text z textového pole `Coded text` (3), rozšifruje se momentálně zvolenou šifrou a klíčem z pole `Key` (2) a výsledek se vloží do textového pole `Open text` (1).

6.1.1 Příklad použití

V této záložce může uživatel šifrovat nebo dešifrovat text zvoleným klíčem. Pro šifrování vloží otevřený text do pole `Open text`, napíše klíč do pole `Key` nebo si ho nechá náhodně vygenerovat tlačítkem `Random key`. Tlačítko `Random key` je neviditelné, viditelným se stává až po zaměření textového pole `Key` a do neviditelného stavu se vrací se zaměřením na jiný ovládací prvek. Poté uživatel stiskne tlačítko `Encrypt` a výsledný zašifrovaný text se zobrazí v textovém poli `Coded text`.

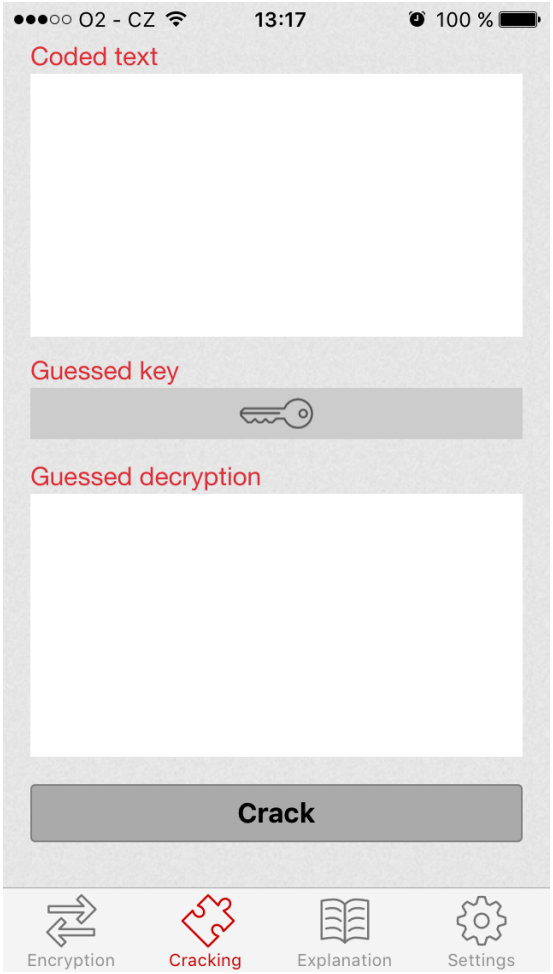
Opačný proces pro dešifrování je podobný. Uživatel první vloží zašifrovaný text do pole `Coded text`, zvolí klíč již popsaným způsobem a stiskne tlačítko `Decrypt`, které zašifrovaný text z pole `Coded text` rozšifruje inverzním šifrovacím algoritmem momentálně zvolené šifry a vloží jej do pole `Coded text`.

6.2 Kryptoanalytická část

Záložka „Cracking“ slouží ke kryptoanalýze. Ovládací prvky jsou stejného typu a jejich rozložení velmi podobné jako u předchozí části 6.1.

Textové pole „Coded text“ slouží pro vložení zašifrované zprávy, pod ním pole „Guessed key“ zobrazuje uhodnutý klíč a pole „Guessed decryption“ zobrazuje text dešifrovaný takovým klíčem. Celý proces kryptoanalýzy spustí tlačítko „Crack“ (český překlad: prasknout, rozluštit, rozbít, vyřešit).

Textová pole „Guessed key“ a „Guessed decryption“ zůstávají nepřístupná (disabled) dokud neskončí kryptoanalýza. Po dokončení útoku se odemknou, aby si uživatel mohl popřípadě zkopírovat výsledek.



Obrázek 19: Aplikace – kryptoanalytická část

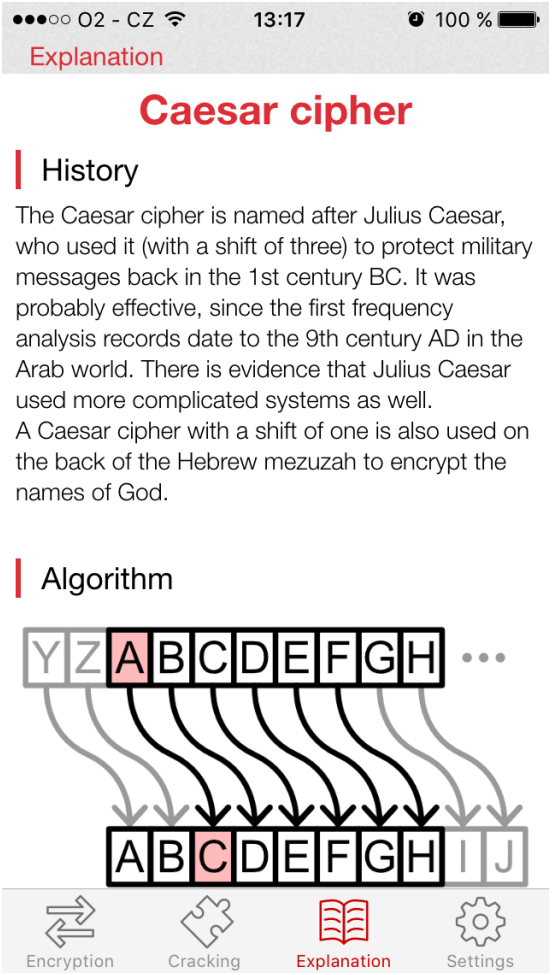
6.2.1 Příklad použití

Uživatel chce rozšifrovat zašifrovanou zprávu a nezná klíč. Zprávu vloží do textového pole `Coded text` a stiskne tlačítko `Crack`. Spustí se zvolený prolamovací algoritmus nad vloženým textem. Podle typu šifry a útoku může výpočet trvat různou dobu. Po dokončení se do textového pole `Guessed key` vloží uhodnutý klíč a do pole `Guessed decryption` zpráva rozšifrovaná tímto klíčem.

6.3 Učební část

Tato část slouží k vysvětlení zvolené šifry. Jedná se o stručný popis historie a fungování šifry, včetně pomocných obrázků, příkladů a popisu kryptoanalýzy.

Informací je většinou tolik, že se na jednu obrazovku zařízení nevejdou. Standardním posunem dolů (drag and drop) se uživatel dostane ke všem sekcím.



Obrázek 20: Aplikace – učební část

6.4 Nastavení

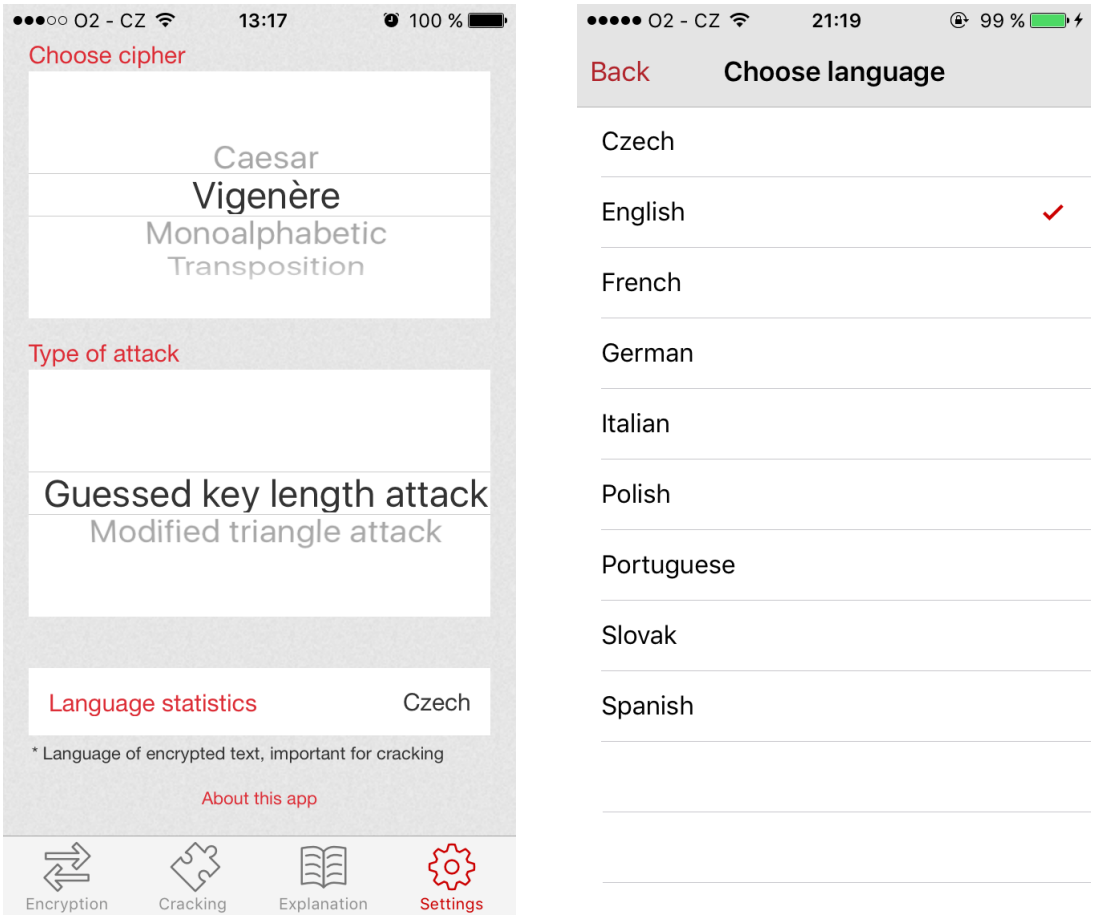
V záložce nastavení si uživatel volí šifru, útok na ni a dodatečné informace o jazyku. Šifru, s kterou chce pracovat, zvolí ze seznamu v ovládacím prvku s nadpisem `Choose cipher`.

Po zvolení šifry se v seznamu ovládacího prvku s nadpisem `Type of attack` vypíšou možné útoky na zvolenou šifru.

Po kliknutí na tlačítko `Language statistics` vyjede seznam přístupných jazyků, z kterých si uživatel jeden zvolí. Zvolená statistika jazyka slouží k prolamování, využije se tedy jen na záložce 6.2.

Obrázky 21 ukazují prostředí záložky nastavení a také seznam zvolitelných statistik jazyka. K dispozici je angličtina, čeština, slovenština, polština, němčina, francouzština, italština, španělština a portugalština.

Na spodu záložky je malé tlačítko `About this app`, po jejímž stisknutí vyjede lišta se stručnými informacemi o aplikaci a jejím účelu.



Obrázek 21: Aplikace – nastavení (vlevo), volba statistiky jazyka (vpravo)

7 Dokumentace zdrojového kódu

Aplikace je napsána ve dvou programovacích jazycích a ukazuje jejich jednoduchou integraci. Ve starším z jazyků, Objective-C, jsou napsány všechny kryptografické a kryptoanalytické algoritmy pracující pouze s textem. Ve druhém jazyku, Swift 2, je napsáno grafické uživatelské rozhraní (GUI).

7.1 Ciphers

Obsahuje všechny algoritmy pro šifrování a dešifrování jednotlivých šifer. Každá šifra je samostatná třída implementující protokol `Cipher`, který udává povinné metody. Mezi ně patří šifrování, rozpoznání validního klíče a generování náhodného klíče.

Jednotlivé třídy šifer se jmenují `Caesar`, `Vigenere`, `Monoalphabetic` a `Transposition`. Každá z nich obsahuje vlastní pomocné algoritmy potřebné k šifrování a dešifrování.

7.2 Cracking

Obsahuje algoritmy kryptoanalytických útoků a k nim pomocné algoritmy analyzující zašifrovaný text. Podobně jako u `Ciphers` se tu využívá protokol `Crack` a všechny třídy jej implementují.

Jednotlivé třídy na prolamování jsou `CaesarCrack`, `VigenereCrack`, `MonoalphabeticCrack` a `TranspositionCrack`.

7.3 Utilities

Obsahuje všechny dodatečné třídy a metody, které pomáhají zpracovat text, analyzovat text a provádět složitější matematické operace.

Třída `Combinatorics` implementuje kombinatorické algoritmy pro permutace a variace nad textovými řetězci a znaky v nich.

Třída `FileReader` implementuje pomocnou strukturu pro lepší práci s textovými soubory jako například jednoduché čtení po řádcích. Stejně tak třída `LanguageParser` implementuje parser pro XML soubory z `Languages`, které obsahují statistické informace o jazycích.

Poslední je třída `Utils`, ve které se nachází metody pro práci s textem (normalizace textu, smazání bílých mezer na konci řetězců atp.), metody pro spočítání četnosti znaků a četnosti reálných slov. Dále metody pro hledání dělitelů a pro inicializaci často používaných struktur.

7.4 Storage

Obsahuje třídy, díky nímž se aplikace dostává k datům potřebným při prolamování nebo při výpisu dostupných šifer. Dále obsahuje textové soubory slovníků

všech slov a slovníků unikátních slov. Ve složce `Languages` jsou navíc XML soubory se statistikami jazyka.

Třída `Storage` implementuje metody například pro získání všech dostupných šifer, pro získání filereaderů jednotlivých slovníků všech slov nebo filereaderů souborů se seznamem unikátních slov.

Třída `Language` obsahuje metody pro získání předpokládané frekvence písmen jazyka, nejčastějších bigramů, trigramů a celých slov. Dále umožňuje nastavit zvolený jazyk a načíst potřebné struktury.

7.5 Explanations

Tato složka obsahuje HTML soubory s informacemi o jednotlivých šifrách. Ke každé šifře je jeden takový soubor, který dodržuje jednoduchou strukturu nadpisů a odstavců. Přiložen je také CSS soubor `styles.css` pro nastylování vzhledu elementů a použité obrázky ve složce `img/`.

7.6 ViewControllers

Obsahuje všechny ViewControllery pro GUI aplikace. Každé okno má svůj ViewController obsluhující události, které mohou nastat.

Třída `EncryptionViewController` obsluhuje kryptografickou část (první zleva ze spodních záložek, viz. 6.1), `CryptanalysisViewController` kryptoanalytickou část (druhá záložka, 6.2), `ExplanationViewController` učební část (třetí záložka, 6.3) a poslední `SettingsViewController` nastavení (6.4).

`LanguagesTableViewController` a `AboutAppViewController` jsou třídy obsluhující vyjíždějící okno na zvolení jazyka a okno s informacemi o aplikaci.

7.7 Supporting Files

Zde se nachází soubor `Bridging-Header.h`, který je nezbytný pro souběh jazyku Objective-C v aplikaci postavené na jazyku Swift. Jsou v něm sepsané importy všech tříd, které aplikace může použít.

Soubory `dictionary_filter.awk` a `unique_filter.awk` jsou krátké AWK programy, které slouží k filtrování textových souborů se slovníky.

Závěr

Cílem této práce bylo naprogramovat mobilní aplikaci pro operační systém iOS, která bude umožňovat šifrovat a prolamovat vybrané klasické šifry bez znalosti původního šifrového klíče, popsat použité kryptoanalytické algoritmy a zhodnotit jejich efektivitu.

Z hlediska teorie jsem vysvětlil základní pojmy a typy šifer, a dále se snažil vybrané šifry co nejvíce detailně popsat. Ke každé z nich jsem popsal historii a vývoj šifry, šifrovací algoritmus s ukázkami na konkrétních příkladech, výhody, nevýhody a nakonec samotné prolamovací a k nim pomocné algoritmy.

V praktické části se mi podařilo naprogramovat mobilní aplikaci umožňující šifrovat, dešifrovat a prolamovat vložené texty. Navíc aplikace disponuje učební částí, která se snaží uživatele seznámit s šifrovacími algoritmy a problematikou jejich kryptoanalýzy.

Aplikace zvládá prolamovat texty v devíti jazycích (angličtina, čeština, slovenština, polština, němčina, francouzština, italština, španělština a portugalština) a je jednoduše rozšířitelná o další jazyky stejně tak jako o další šifry.

Conclusion

Aim of this bachelor thesis was to program a mobile application for the iOS operation system, which will enable encrypting, decrypting and breaking selected historical ciphers without previous knowledge of the used key, describe used cryptoanalytic algorithms and evaluate their effectiveness.

In the theoretical part, I explained basic terms and types of cyphers and continued to describe selected ciphers in more detail. Each description contained history and development of the cipher, the encryption algorithm shown on specific examples, their advantages and disadvantages and lastly the cipher breaking algorithms.

In the practical part, I managed to program a mobil application which can encrypt, decrypt and break inserted texts. This application also contains a didactic part, which attempts to introduce the user with the ciphers and with their possible cryptoanalysis.

This application has the ability to break texts in nine languages (english, czech, slovak, polish, german, french, italian, spanish and portuguese) and can have other languages easily added as well as other ciphers.

A Naměřená data

počet znaků	úspěšnost
25	86.57%
50	97.18%
75	99.50%
100	99.85%
125	99.97%
150	100.00%

Tabulka 1: Caesarova šifra – útok hrubou silou pomocí frekvenční analýzy

počet znaků	úspěšnost
25	39.68%
50	58.10%
75	69.15%
100	76.53%
200	90.37%
300	94.82%
400	97.30%
500	98.05%

Tabulka 2: Caesarova šifra – útok hrubou silou hledáním reálných slov

počet znaků	úspěšnost
25	36.53%
50	54.92%
75	64.15%
100	71.64%
200	85.60%
300	92.01%
400	93.23%
500	94.52%

Tabulka 3: Caesarova šifra – trojúhelníkový útok

počet znaků	úspěšnost
250	21.06%
500	72.16%
750	93.21%
1 000	99.15%
1 250	99.81%
1 500	100.00%

Tabulka 4: Vigenèrova šifra – útok pomocí frekvenční analýzy

počet znaků	úspěšnost
500	35.49%
1 000	72.81%
1 500	84.24%
2 000	89.57%
2 500	93.34%
3 000	95.04%
3 500	96.29%
4 000	96.65%
4 500	97.25%
5 000	97.84%

Tabulka 5: Vigenèrova šifra – trojúhelníkový útok

počet znaků	úspěšnost
1 000	48.67%
2 000	76.81%
3 000	90.20%
4 000	92.34%
5 000	93.19%

Tabulka 6: Monoalfabetická šifra – útok hledáním unikátních slov

počet znaků	úspěšnost
1 000	42.12%
2 000	56.55%
3 000	66.01%
4 000	71.34%
5 000	75.98%

Tabulka 7: Sloupcová transpozice – útok nalezením slova v řádku

počet znaků	úspěšnost
100	36.09%
200	66.31%
300	83.06%
400	89.20%
500	92.50%
600	95.36%
700	97.27%
800	99.18%

Tabulka 8: Sloupcová transpozice – útok hrubou silou hledáním reálných slov

B Obsah přiloženého CD

`Application/`
: Složka obsahující všechny soubory se zdrojovými kódy mobilní aplikace vytvářené ve vývojovém prostředí Xcode 7.2.

`Images/`
: Složka s veškerou grafikou vytvořenou pro použití jak v aplikaci, tak v této textové práci. Obsahuje vektorové obrázky včetně rastrových exportů.

`Text/`
: Složka s textem bakalářské práce vypracovaném v sázecím systému LaTeX, včetně zdrojových souborů, použitých obrázků, dat použitých v grafech i finálního exportu ve formátu PDF.

Literatura

[1] Adams, S.: *Šifry a kódy: od hieroglyfů po hackery*. Slovart, 2003, ISBN 80-7209-503-X.

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

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

[4] Wikipedia: Caesar cipher. https://en.wikipedia.org/wiki/Caesar_cipher, 2015.

[5] Wikipedia: Cryptography. https://en.wikipedia.org/wiki/Cryptography, 2015.

[6] Wikipedia: Frequency analysis. https://en.wikipedia.org/wiki/Frequency_analysis, 2015.

[7] Wikipedia: Letter frequency. https://en.wikipedia.org/wiki/Letter_frequency, 2015.

[8] Wikipedia: Scytale. https://en.wikipedia.org/wiki/Scytale, 2015.

[9] Wikipedia: Transposition cipher. https://en.wikipedia.org/wiki/Transposition_cipher, 2015.

[10] Wikipedia: Vigenère cipher. https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher, 2015.

[11] Wikipedie: Caesarova šifra. https://cs.wikipedia.org/wiki/Caesarova_%C5%A1ifra, 2015.

[12] Wikipedie: Jednoduchá sloupcová transpozice. https://cs.wikipedia.org/wiki/Jednoduchá_sloupcová_transpozice, 2015.

[13] Wikipedie: Kryptografie. https://cs.wikipedia.org/wiki/Kryptografie, 2015.

[14] Wikipedie: Monoalfabetická šifra. https://cs.wikipedia.org/wiki/Monoalfabetic%6B%C3%A1_%C5%A1ifra, 2015.

[15] Wikipedie: Substituční šifra. https://cs.wikipedia.org/wiki/Substitučn%C3%AD_šifra, 2015.

[16] Wikipedie: Transpoziční šifra. https://cs.wikipedia.org/wiki/Transpozičn%C3%AD_šifra, 2015.

[17] Wikipedie: Vigenèrova šifra. https://cs.wikipedia.org/wiki/Vigen%C3%A8ro%76a_%C5%A1ifra, 2015.

[18] Šifrovací disk od firmy Retroworks. http://ecx.images-amazon.com/images/I/812GVgp-DSL._SL1500_.jpg, (Staženo dne: 12. 12. 2015).