Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Transpoziční šifry

2021

Jan Bičan

Vedoucí práce: doc. RNDr. Miroslav Kolařík, Ph.D.

Studijní obor: Aplikovaná informatika, kombinovaná forma

Bibliografické údaje

Autor:	Jan Bičan
Název práce:	Transpoziční šifry
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2021
Studijní obor:	Aplikovaná informatika, kombinovaná forma
Vedoucí práce:	doc. RNDr. Miroslav Kolařík, Ph.D.
Počet stran:	57
Přílohy:	1 CD
Jazyk práce:	český

Bibliograhic info

Author:	Jan Bičan
Title:	Transposition ciphers
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2021
Study field:	Applied Computer Science, combined form
Supervisor:	doc. RNDr. Miroslav Kolařík, Ph.D.
Page count:	57
Supplements:	1 CD
Thesis language:	Czech

Anotace

Transpozice patří mezi základní operace šifrování. Princip transpozice spočívá v přeskupení písmen. V teoretické části práce jsou vysvětleny základní pojmy šifrování, stručně popsány dějiny kryptologie a podrobně popsány významné transpoziční šifry. Kromě šifrování a dešifrování je prozkoumáno také jejich prolamování. Praktická část popisuje vytvoření webové aplikace, která umožňuje s transpozičními šiframi pracovat.

Synopsis

Transposition is one of the basic encryption operations. The principle of transposition consists in rearranging the letters. The theoretical part of the thesis explains the basic concepts of encryption, briefly describes the history of cryptology and describes in detail the important transposition ciphers. In addition to encryption and decryption, their breaking is also examined. The practical part describes the creation of a web application that allows to use transposition ciphers.

Klíčová slova: transpoziční šifra; transpozice; šifra; kryptologie; kryptografie; kryptoanalýza

Keywords: transposition cipher; transposition; cipher; cryptology; cryptography; cryptanalysis

Děkuji doc. RNDr. Miroslavu Kolaříkovi, Ph.D. za vedení práce a za cenné rady.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Úvod

Šifry umožňují tajně komunikovat – zajišťují skrytí významu zpráv před nepovolanými osobami. Význam šifer je značný, o čemž vypovídá jejich role v mnoha klíčových událostech historie – podílely se na rozvoji rozmanitých oborů lidské činnosti a technologií. Kryptologie, věda, která se šiframi zabývá, prošla za několik tisíciletí velkým vývojem. V textu jsou její dějiny stručně pokryté. V současnosti se s aplikacemi poznatků kryptologie setkáváme denně (jedním příkladem z mnoha je použití platební karty v internetových obchodech).

Transpoziční a substituční šifry tvoří základní skupiny historických šifer. Rozdělení je založeno na charakteru použité operace. Transpozice spočívá ve změně pořadí písmen. Substituce znamená nahrazení písmena jiným písmenem. Vedle transpozice a substituce se ještě setkáváme se steganografií, jejíž úkolem je skrytí samotné existence zpráv.

Tato práce je zaměřena na transpoziční šifry. Teoretická část poskytuje ucelený pohled na významné zástupce transpozice. Kromě popisu použití jsou v textu zkoumány způsoby, jakými lze tyto šifry prolamovat. Nedílnou součástí kryptologie je totiž snaha o získáním utajeného textu bez znalosti klíče. Byť historické šifry nejsou považovány za bezpečné, jejich prolamování není úplně triviální. Transpoziční šifry až na výjimky (např. Rail Fence) poskytují možnost volby z rozsáhlé množiny klíčů, což zajišťuje odolnost proti jednoduchému útoku vyzkoušení všech klíčů. V práci je zkoumáno použití hrubé síly a tam, kde selhává, je prověřen důmyslnější způsob v podobě horolezeckého algoritmu. Úspěšnost prolamování je v práci shrnuta.

Výstupem praktické části je webová aplikace pro jednoduché použití popsaných transpozičních systémů. Aplikace umožňuje do určité míry i jejich prolamování. Uživatelská část pak vysvětluje, jak vytvořenou aplikaci používat.

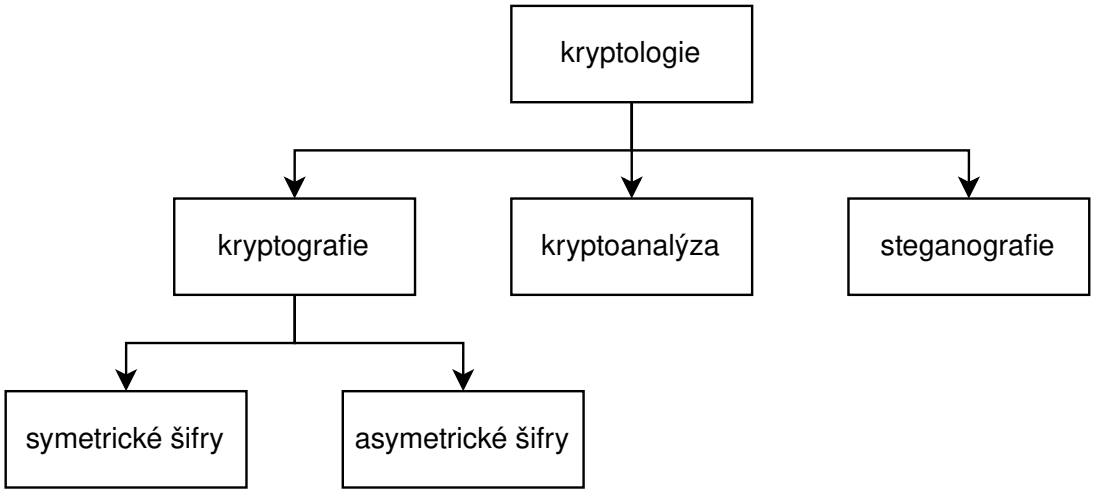
2 Teoretická část

2.1 Základní pojmy

Tato podkapitola je úvodem do problematiky šifrování. Pokládá teoretické základy kryptologie a seznamuje čtenáře s pojmy, které se v textu opakovaně vyskytují. Pro podkapitolu byly použity zdroje [1], [2], [3], [4].

2.1.1 Klasifikace kryptologie

Kryptologie je věda, která zkoumá způsoby utajování zpráv. Dělí se na kryptografii, kryptoanalýzu a steganografii. Kryptografie popisuje metody šifrování zpráv. Kryptoanalýza se zabývá slabými místy šifer za účelem jejich prolomení. Úkolem steganografie je skrytí samotné existence zpráv. Větvení kryptologie je znázorněno na obrázku č. 1.

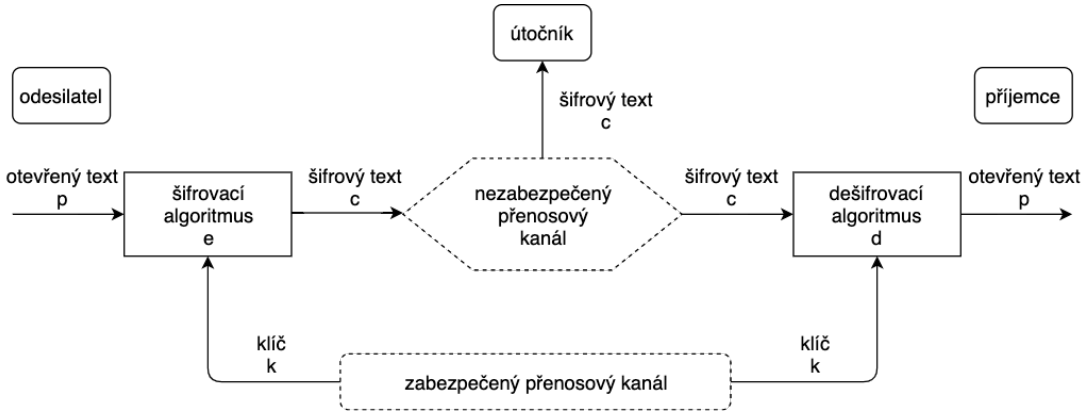


Obrázek 1: Klasifikace kryptologie

Kryptografie rozeznává symetrické a asymetrické šifry. Zatímco symetrické šifry používají pro šifrování a dešifrování totožný klíč, pro asymetrické toto neplatí. U asymetrických šifer definujeme dva typy klíčů – veřejný a soukromý. Utajen je pouze soukromý klíč, neboť je schopen dešifrovat každou zprávu zašifrovanou veřejným klíčem. Předmětem této práce jsou transpoziční šifry, které jsou podkategorií historických symetrických šifer.

2.1.2 Šifrovací systémy

Pro dosažení důvěrnosti komunikace je třeba přenášená data šifrovat. To obnáší transformaci dat do takové podoby, která bude srozumitelná pouze zamýšlenému příjemci. Požadovanou funkci poskytují šifrovací systémy (kryptosystémy)



Obrázek 2: Schéma symetrického šifrování

definující potřebné algoritmy – generování klíče, šifrování a dešifrování. Obecné schéma symetrického šifrovacího systému je znázorněno na obrázku č. 2.

Nyní uvedeme postup, jak spolu mohou dvě strany bezpečně komunikovat pomocí symetrického šifrovacího systému. V první řadě se musí vygenerovat klíč, což je tajný element, který je zásadní pro zajištění důvěrnosti. Klíč je třeba sdílet přes zabezpečený přenosový kanál, ke kterému nemá přístup nikdo nepovolaný[1]. Můžeme si to představit tak, že příjemce a odesilatel se jednou za čas osobně setkají za účelem výměny klíčů na následující časové období.

Jakmile klíčem disponují obě strany, komunikace může probíhat přes nezabezpečený přenosový kanál (např. Internet). Pokud odesilatel chce zaslat příjemci zprávu, použije šifrovací algoritmus e. Jeho vstupem je klíč k a otevřený text p (zpráva), výstupem je šifrový text c. Formálně lze operaci šifrování vyjádřit jako $e(k, p) = c$.

Šifrový text se v případě použití kvalitního kryptosystému jeví jako náhodná posloupnost. Šifrový text je v dalším kroku odeslán příjemci přes nezabezpečený kanál. Příjemce použije dešifrovací algoritmus d. Jeho vstupem je klíč k a doručený šifrový text c, výstupem je původní otevřený text p. Formálně lze operaci dešifrování vyjádřit jako $d(k, c) = p$.

Nezabezpečený kanál je z principu odposlouchávatelný nepovolanou osobou („útočníkem“), která tak může zachytit přenášený šifrový text. Bez znalosti tajného klíče ovšem není schopna z něj získat otevřený text. Zpráva je tak důvěrná.

V práci se budeme zabývat historickými šiframi, u kterých se šifrování zpravidla provádí nad znaky nebo písmeny. Otevřený a šifrový text tedy budeme uvažovat jako posloupnost písmen anglické abecedy (bez mezer)[2]. Podobné to bude v případě klíče, avšak u něj existují výjimky. U některých šifer může klíčem být číslo nebo fráze, v níž se vyskytují mezery.

[1] toto vytváří problém distribuce klíče, který byl vyřešen až vynalezením asymetrických šifer, které nevyžadují důvěrné sdílení klíčů

[2] pro jednoduchost opomíjíme, že obecně šifrování probíhá nejen nad písmeny (např. binární posloupnosti u moderních šifer)

2.1.3 Kryptoanalýza

Tato sekce uvádí čtenáře do kryptoanalýzy, větve kryptologie, která se ideálně doplňuje s kryptografií. Byly pro ni použity zdroje [3], [5].

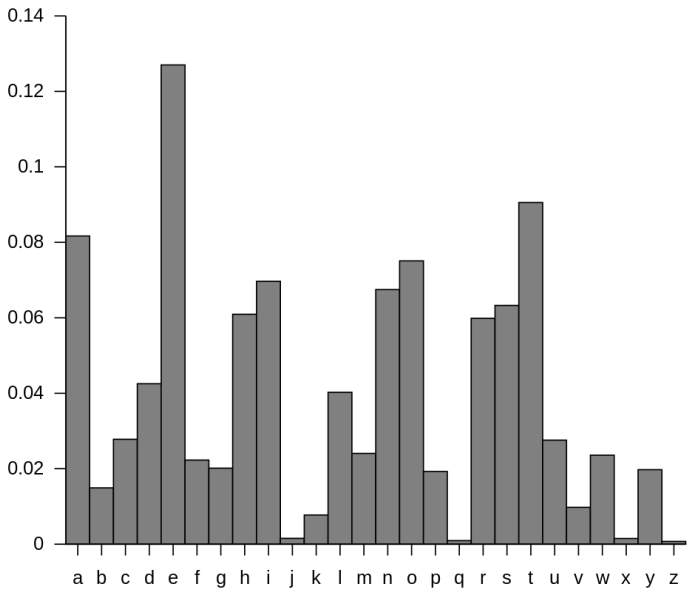
Zatímco kryptograf usiluje o vývoj šifrovacích systémů, které poskytují důvěrnost, kryptoanalytik se snaží nalézt jejich slabá místa. Kryptoanalytik má k dispozici šifrový text a další případné informace, avšak chybí mu tajný klíč. Jeho cílem je získat co nejvíce informací o otevřeném textu a ideálně danou šifru prolomit. Úkolem kryptoanalýzy je ukázat, že daný šifrovací systém není bezpečný. Na základě kryptoanalytických poznatků pak může být šifrovací systém vylepšen nebo nahrazen jiným.

Pokud se někomu podaří konkrétní šifrovací systém prolomit a tuto informaci nezveřejní, získává velkou výhodu. Nadále může číst zprávy, které jsou tímto systémem zašifrovány. V historii nalezneme celou řadu případů, kdy takové poznatky vyhrály mocnostem válku. Známé je například prolomení německé Enigmy britskými kryptoanalytiky v čele s Alanem Turingem v Bletchley Parku (více v sekci 2.1.4), o čemž se svět dozvěděl až 30 let po válce.

Vývoj kryptoanalýzy je v souladu s etapami kryptologie. Původně se šifry luštily na papíře. Pro jednoduché historické šifry je toto přijatelné, s větší komplexností však značně narůstá časová náročnost. S příchodem mechanických šifer došlo k rozvoji strojových zařízení (Turingovy bomby a Colossus), která procesy kryptoanalýzy zautomatizovala. S nástupem počítačů se toto ještě více umocnilo. Přestože jsou výpočetní možnosti dnešních počítačů obrovské, stále nedokážeme prolamovat všechny historické šifry za všech okolností. Pro některé šifry zatím nebyla nalezena efektivní metoda. Příkladem je dvojitá sloupcová transpozice, kterou lze luštit pouze omezeně. Mnoho historických šifer tak stále zůstává neprolomeno [6].

Původně byla kryptoanalýza doménou lingvistů, o čemž vypovídá první velký objev na poli luštění substitučních šifer. Pokud uvažujeme například angličtinu, můžeme vycházet z určité četnosti jednotlivých písmen. Písmeno „e“ je v běžném anglickém textu nejčastější a pokrývá 12,7 % textu (četnost písmen celé abecedy je na obrázku č. 3). Substituční šifru můžeme luštit tak, že v šifrovém textu spočítáme četnost jednotlivých písmen a snažíme se podle ní najít původní písmena otevřeného textu. Tato metoda se nazývá frekvenční analýza a stala se velmi efektivním nástrojem kryptoanalýzy. Důležitá je skutečnost, že frekvenční analýzu nelze aplikovat na transpoziční šifry. Transpozice zachovává výskyt písmen, a tak lze frekvenční analýzu použít pouze k detekci toho, zda je daná šifra transpozicí. Kryptoanalýze transpozičních šifer se věnujeme v sekci 2.3.

V závislosti na tom, jakými informacemi kryptoanalytik disponuje, rozlišujeme několik typů kryptoanalytických útoků. Základním je útok, kdy kryptoanalytik vychází pouze z šifrového textu (ciphertext-only attack). I v tomto případě však mohou vyplynout další informace – ze zdroje šifrového textu lze například předpokládat použitý jazyk. Kryptoanalytik také může mít dostupný otevřený text k určitém šifrovému textu (known-plaintext attack). Na základě toho se snaží odhalit klíč, který mohl být použit k šifrování dalších zpráv, které ještě



Obrázek 3: Četnost písmen v anglickém textu
převzato z https://en.wikipedia.org/wiki/Frequency_analysis

nejsou prolomeny. Je-li útočník schopen získat k libovolnému otevřenému textu jeho zašifrovanou variantu, jedná se o další typ útoku (chosen-plaintext attack). V tomto případě útočník zná použitý šifrovací algoritmus nebo má k dispozici zařízení, kterým se šifruje.

V kryptologii se ujalo pravidlo, které říká, že by bezpečnost kryptosystému neměla záviset na jeho utajení. Je tedy vhodné předpokládat, že útočník má k dispozici veškerou specifikaci použitého systému a jediné, co mu chybí, je tajný klíč. Jedná se o Kerckhoffsův princip, který formuloval holandský lingvista a kryptograf Auguste Kerckhoffs v 19. století [6].

Specifikace moderních šifer je známá, takže jsou matematicky velmi důkladně prověřeny. Z toho důvodu se vyvinul další kryptoanalytický útok, který spoléhá na fyzickou implementaci šifry – útok postranním kanálem. Zaměřuje se na informace o čase, energetické spotřebě a elektromagnetickém záření, z čehož odvozuje další skutečnosti. Používá se také sociální inženýrství – tajné informace jsou vylákány z lidí, kteří k nim mají přístup.

Principiálně jednoduchým útokem je tzv. brute-force attack. Ten spočívá ve vyzkoušení všech možných klíčů. Až na velmi jednoduché šifry s malým rozsahem použitelných klíčů je tato metoda pouze teoretická. Čím větší prostor klíče (key space), tím větší odolnost proti útoku hrubou silou. Za bezpečné se považují klíče o rozsahu alespoň 128 bitů. Moderní šifry jsou tímto způsobem teoreticky prolomitelné, ale ne v rozumném čase.

2.1.4 Stručné dějiny kryptologie

Pro tuto sekci byly použity zdroje [3], [7], [8].

Obrázek 4: Skytalé
převzato z https://en.wikipedia.org/wiki/Scytale

Kryptologie je disciplína, jejíž kořeny sahají několik tisíc let nazpět. Dějiny kryptologie jsou protkány soubojem tvůrců a luštitelů šifer, který vedl k celé řadě významných vědeckých objevů a rozvoji rozmanitých oborů a technologií. Nejvýznamnějším produktem tohoto zápolení jsou moderní počítače.

Vývoj kryptologie lze rozdělit do několika fází. V první se pro šifrování využívalo tužky a papíru a případně dalších fyzických pomůcek (např. mřížky). V 1. polovině 20. století však nastala éra mechanických a elektromechanických přístrojů, které šifrování urychlily a nabídly větší bezpečnost s použitím komplexnějších šifer. Poslední fáze začala v 70. letech minulého století, kdy došlo k objevu asymetrické šifry, která vyřešila problém s distribucí klíče.

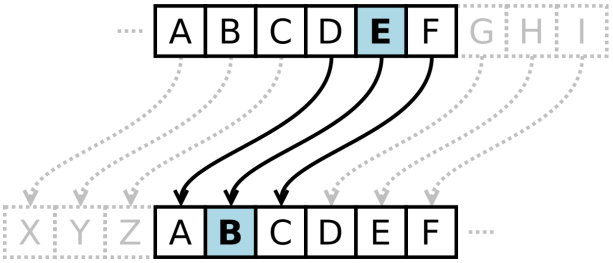
První náznaky kryptografie lze datovat do starověkého Egypta před čtyřmi tisíci lety. Pojí se s objevem nestandardních hieroglyfů. Spíše než o šifrování však mohlo jít o vytváření iluze tajemna.

O snaze utajit návod na výrobu keramiky vypovídají hliněné tabulky z Mezopotámie datované do roku 1500 př. n. l. Umění tajné komunikace bylo také zmíněno v Kámasútře, starověkém indickém textu pojednávajícím o lidské sexualitě.

Řekové a Peršané používali jednoduché kryptografické techniky ke sdělování vojenských plánů. Sparťané používali zařízení pojmenované skytalé (obrázek č. 4). Metoda spočívala v navinutí pásku papyru nebo pergamenu na dřevěný válec. Poté se na něj napsal text zprávy, pásek se svinul a mohl být poslán příjemci. Pro přečtení musel příjemce pásek navinout na válec stejných rozměrů. Šlo o jednoduchý způsob transpozice.

Do historie kryptologie se zapsal také Julius Caesar, který pro korespondenci používal jednoduchou substituční šifru. Ta je známá pod jeho jménem jako Caesarova šifra. Jedná se o posuvnou šifru, protože písmena otevřeného textu jsou nahrazována písmeny s určitým posunem v rámci abecedy. Posun o tři písmena znamená, že „A“ je nahrazeno písmenem „D“, „B“ písmenem „E“ a tak dále a např. písmeno „X“ je nahrazeno písmenem „A“ (dochází k přechodu na začátek abecedy). Obrázek č. 5 ukazuje vazby písmen pro dešifrování.

První významný průlom v oblasti kryptoanalýzy zaznamenali v 9. století



Obrázek 5: Caesarova šifra – dešifrování
převzato z https://en.wikipedia.org/wiki/Caesar_cipher

Arabové. Polyhistor Al-Kindi vynalezl techniku frekvenční analýzy, pomocí níž lze efektivně luštit substituční šifry. Více o frekvenční analýze je v sekci 2.1.3.

Ve středověku kryptografie pronikla do diplomatických kruhů – stala se základním nástrojem utajení politických informací. Vzhledem k tomu, že se rozvíjela i kryptoanalýza, bylo třeba přicházet se silnějšími šiframi. Takovou byla například Vigenèrova šifra. Jednalo se o polyalfabetickou šifru, která otevřený text šifruje pomocí až 26 šifrových abeced. I tato na danou dobu složitá šifra nakonec neodolala náporu prolamování, o což se postaral anglický matematik a vynálezce Charles Babbage. Koncem 19. století tak kryptografie upadala. Zdálo se, že kryptoanalýza má neustále navrch. S vynálezem rádia navíc význam kryptografie ještě vzrostl, neboť odposlouchávání bylo velmi snadné.

1. světová válka byla přehlídkou kryptografických selhání – šifry se prolamovaly jedna za druhou. Přesto byla v roce 1917 navržena šifra, která je v principu nerozluštitelná. Jmenuje se Vernamova šifra (one-time pad). Přestože je její bezpečnost absolutní, z praktického hlediska ji lze těžko využít. Pro šifrování je třeba náhodně generovat klíč o délce otevřeného textu. Sestavení a distribuce takového klíče je problematická.

S příchodem 2. světové války se na scénu dostávají mechanické a elektromechanické šifrovací přístroje. Nejvýznamnějším příkladem je Enigma, kterou používali Němci s přesvědčením, že ji nelze prolomit. Enigma je přístroj podobný psacímu stroji (viz obrázek č. 6). Skládá se z několika částí – propojovací desky, pohyblivých rotorů a reflektoru. Klíčem je nastavení Enigmy – použité rotory, uspořádání rotorů, propojení písmen na desce a použitý reflektor. Před šifrováním se toto nastavení navolí, poté se na přístroji zpráva napíše. S každým úhozem na klávesnici se rozsvítí žárovka reprezentující písmeno šifrového textu. Dešifrování probíhá stejně. Fungování Enigmy je podrobně popsáno v [3].

Již před začátkem 2. světové války Enigmu úspěšně luštili Poláci. Matematik a kryptograf Marian Rejewski zkonstruoval adaptaci Enigmy, která byla schopna prověřovat všechna nastavení, dokud nenalezla to správné. Později ovšem Němci zvýšili počet možných nastavení, což další luštění dočasně znemožnilo.

Po invazi do Polska v luštění Enigmy pokračovali Britové v Bletchley Parku, kde za války došlo k řadě kryptoanalytických průlomů. Za klíčovou osobu se považuje Alan Turing, britský matematik a informatik, který objevil slabiny Enigmy a vylepšením polské metody dokázal Enigmu i nadále prolamovat. Kromě

Obrázek 6: Enigma
převzato z https://cs.wikipedia.org/wiki/Enigma

Enigmy v Bletchley Parku luštili i japonské a italské šifry. Práce kryptografů se ukázala jako klíčová – odhaduje se, že zkrátila válku o několik let.

Za 2. světové války začali kryptoanalytikové pro luštění využívat stroje, které na rozdíl od lidí dovedly pracovat rychle a efektivně. Vývojem těchto strojů byl položen základní kámen počítačovým technologiím. Kromě Turingových bomb prolamující Enigmu, byl vynalezen přístroj pojmenovaný Colossus, kterým se luštila ještě složitější šifra používaná Hitlerem a jeho generály. Colossus bylo možné přizpůsobit konkrétnímu problému, a tudíž šlo o první programovatelný počítač vůbec.

V poválečné éře kryptologie hrály počítače zásadní roli a přispěly k jejímu výraznému rozmachu. Tím vzrostla potřeba po standardizaci, a tak vznikla šifra Data Encryption Standard (DES). Problém distribuce klíče však stále přetrvával. Aby mohly dvě strany tajně komunikovat, musely si předat klíč, což mohl být logisticky obrovský problém. Kryptografie byla v takovém případě pro soukromý sektor těžkopádná a byla záležitostí pouze vládních a armádních kruhů. To se změnilo s vynálezem asymetrické šifry RSA v roce 1977. Ta umožňuje tajnou komunikaci i stranám, které si dopředu klíče tajně nevyměnily. Princip spočívá v existenci dvou klíčů – veřejného a soukromého. Pomocí sdíleného veřejného klíče lze zprávu zašifrovat. Vzniklý šifrový text pak dešifruje soukromý klíč, který

má k dispozici pouze příjemce. Bezpečnost metody je postavena na skutečnosti, že rozklad obrovských čísel na prvočísla je časově velmi složitá operace.

2.2 Transpoziční šifry

V této podkapitole jsou popsány významné transpoziční šifrovací systémy se zaměřením na jejich použití. Šifrování a dešifrování je kromě obecného popisu demonstrováno na příkladech. Pro podkapitolu byly použity zdroje [1], [7].

Transpozice a substituce jsou základní šifrovací operace. Pokud provádíme transpozici, znamená to, že přeskupujeme písmena[3] textu podle určitých pravidel. Znalost pravidel nám pak umožní z přeskupeného textu získat původní text. Příkladem transpozice jsou např. jednoduché přesmyčky jako je psaní textu pozpátku. Substituce znamená nahrazení všech výskytů písmena v textu jiným písmenem.

Transpoziční šifry jsou z pohledu historických šifer důležitou kategorií. Tato práce se zaměřuje na několik nejvýznamnějších transpoziční šifer. Transpozice je důležitá i pro moderní symetrické šifry. Ty používají kombinaci operací, které jsou opakovaně aplikovány. Jednou z těchto operací je permutace bitů, což je v principu transpozice [2].

2.2.1 Šifra Rail Fence

Jednoduše a rychle aplikovatelná šifra, která však poskytuje malou míru bezpečnosti. Při použití vzniká mřížka připomínající plot, z čehož vznikl její název. Alternativně se pojmenovává také jako zig-zag šifra.

Popularitu získala zejména v raných desetiletích vývoje kryptografie. Se vznikem komplexnějších šifrovacích systémů byla odsunuta do ústraní. Později se vyskytla za Americké občanské války, kdy se používala k utajování vojenských informací a zpráv zasílaných agenty Unie a Konfederace [7]. V současnosti se s ní lze setkat např. v geocachingu (mystery cache).

Popis šifrování

Prvním krokem je volba klíče, jímž může být jakékoli číslo větší než jedna. Klíč představuje počet řádků mřížky, se kterou se při šifrování pracuje. Otevřený text se do této mřížky vepisuje. První písmeno se vepíše do prvního řádku a sloupce. Dále se pokračuje druhým řádkem a druhým sloupcem. Po každém vepsaném písmenu se pozice posunuje o jeden sloupec doprava a o jeden řádek dolů nebo nahoru. Začíná se posunem směrem dolů, ale jakmile se dospěje k poslednímu řádku, směr se otáčí. To se stane i v případě horního řádku. Postupně tak otevřený text v mřížce vytváří véčka. Toto probíhá, dokud nejsou použita všechna písmena otevřeného textu. Poté se přepsáním jednotlivých řádků vygeneruje šifrový text. Postup si ukážeme na příkladu.

[3] obecně mohou šifrovací operace pracovat s jednotkami otevřeného textu, což mohou být např. i n-tice písmen

otevřený text: letsmeettomorrow
klíč: 3 (řádky)

Začneme vytvářet mřížku o třech řádcích a otevřený text do ní vepíšeme podle popsaného postupu. Můžeme zkonstruovat kompletní mřížku nebo stačí jednotlivé pozice přibližně dodržovat, což proces urychlí. Následuje mřížka se čtyřmi počátečními písmeny otevřeného textu a po ní zjednodušená verze složená pouze ze zaplněných políček obsahující celý otevřený text. Výsledný šifrový text získáme přepsáním jednotlivých řádků, které jsou pro názornost barevně rozlišeny.

l															
	e		s												
		t													

l				m				t				r			
	e		s		e		t		o		o		r		w
		t				e				m				o	

šifrový text: LMTR ESETOORW TEMO

Popis dešifrování

Prvně je nutné připravit mřížku s počtem řádků daným klíčem. To se provede stejným způsobem jako při šifrování, avšak místo doplňování písmen se nejprve vyznačí políčka, která budou použita. Jakmile máme počet políček odpovídající délce šifrového textu, vepíšeme do nich písmena šifrového textu, a to po řádcích. Otevřený text se pak ve mřížce ukáže zleva doprava. Následuje příklad.

šifrový text: LMTRESETOORWTEMO
klíč: 3 (řádky)

Sestavíme mřížku o třech řádcích a šedě vyznačíme políčka, která budou použita – bude jich šestnáct, což odpovídá délce šifrového textu. Poté do nich po řádcích vepíšeme šifrový text – do prvního řádku „LMTR“. Po zkompletování mřížky přečteme otevřený text zleva doprava. Následuje znázornění mřížky, ve které jsou pouze použitá políčka.

l				m				t				r			
	e		s		e		t		o		o		r		w
		t				e				m				o	

otevřený text: letsmeettomorrow

2.2.2 Sloupcová (jednoduchá) transpozice

Sloupcová transpozice je považována za nejpopulárnější transpoziční šifrovací systém. Používala se zejména ve 2. polovině 19. století a 1. polovině 20. století. Například ve 20. letech minulého století sloužila jako hlavní šifra Irské republikánské armády (IRA) [7]. Stala se základem komplexnějších šifer, např. dvojité transpoziční šifry. Princip spočívá v sestavení tabulky, do které se po řádcích vepisuje otevřený text. Šifrový text pak vzniká přepsáním sloupců. Rozměry tabulky a pořadí sloupců určuje klíč. V tabulce č. 1 je uvedeno značení použité při popisu šifry.

popis	**symbol**
délka klíče / počet sloupců	$\|K\|$
délka šifrového textu	$\|C\|$
počet řádků	r
počet úplných sloupců	f

Tabulka 1: Značení pro sloupcovou transpozici

Abecední (permutační) vyčíslení

Důležitou složkou sloupcové transpozice (i dalších šifer) je abecední vyčíslení. Jde o funkci, která písmenům posloupnosti přiřazuje relativní pořadí podle abecedy. Abecední vyčíslení slova „keyword“ je $(3, 2, 7, 6, 4, 5, 1)$. Písmenu „d“ odpovídá jednička, jelikož se nachází v abecedě nejdříve. Následuje písmeno „e“ s dvojkou a tak dále. Opakovaný výskyt písmen je řešen tak, že nižší index dostává to, které se nachází více vlevo. Příkladem je „tomato“, u kterého máme vyčíslení $(5, 3, 2, 1, 6, 4)$. Písmenu „o“ na druhé pozici byla přidělena trojka, zatímco na šesté pozici čtyřka.

Popis šifrování

Proces začíná výběrem klíče, kterým zpravidla bývá jedno slovo. Obecně jde o libovolnou posloupnost písmen anglické abecedy o délce alespoň dva. Z klíče se získá abecední vyčíslení. Poté se vytvoří tabulka s počtem sloupců daným délkou klíče $|K|$, v jejímž záhlaví se uvede abecední vyčíslení. Následně se otevřený text vepíše do tabulky po řádcích. Počátečních $|K|$ písmen bude tvořit první řádek, dalších $|K|$ písmen druhý řádek atd. Šifrový text se poté získá přepsáním sloupců, jejichž pořadí určuje abecední vyčíslení klíče. Šifrování si nyní ukážeme na příkladu.

otevřený text: thisiscolumnartransposition
klíč: secret

Délka klíče $|K|$ určuje tabulku o šesti sloupcích a abecední vyčíslení klíče je $(5, 2, 1, 4, 3, 6)$. Vytvoříme záhlaví tabulky a postupně zaplňujeme řádky. Do každého políčka zaneseme jedno písmeno otevřeného textu. Výsledná tabulka vypadá následovně (barevné rozlišení sloupců je pouze pro názornost).

5	2	1	4	3	6
t	h	i	s	i	s
c	o	l	u	m	n
a	r	t	r	a	n
s	p	o	s	i	t
i	o	n			

Šifrový text získáme přepsáním sloupců v pořadí daném abecedním vyčíslením. Začneme tedy sloupcem s číslem 1, který je na třetí pozici a pokračujeme sloupcem s číslem 2, který je na druhé pozici atd. Před vygenerováním šifrového textu je možné tabulku modifikovat tak, že se permutují její sloupce, aby se nemusely zpracovávat na přeskáčku.

1	2	3	4	5	6
i	h	i	s	t	s
l	o	m	u	c	n
t	r	a	r	a	n
o	p	i	s	s	t
n	o			i	

šifrový text: ILTON HORPO IMAI SURS TCASI SNNT

Popis dešifrování

V prvním kroku se zjistí abecední vyčíslení klíče a vytvoří se záhlaví tabulky stejně jako při šifrování. Počet sloupců je dán délkou klíče $|K|$. Předem se určuje také počet řádků r, jelikož je nezbytné znát délku sloupců. Situaci komplikuje případná neúplnost tabulky, která znamená, že délka jednotlivých sloupců se může lišit. Některé budou mít délku r, zatímco jiné zanechají poslední řádek prázdný (délka $r-1$). Pro správné dešifrování je třeba identifikovat, které sloupce jsou úplné a které nikoli.

Počet řádků r se vypočítá tak, že se podíl délky šifrového textu $|C|$ a počtu sloupců $|K|$ zaokrouhlí směrem nahoru na celé číslo. Vzorec vypadá následovně:

$$r = \left\lceil \frac{|C|}{|K|} \right\rceil.$$

Počet úplných sloupců f se získá tak, že se určí počet políček v tabulce vyjma posledního řádku a výsledek se odečte od délky šifrového text $|C|$. Jedná se o počet písmen, které se budou nacházet na posledním řádku. Pro výpočet se použije následující vzorec:

$$f = |C| - [|K| \cdot (r - 1)].$$

Dále se připraví tabulka se zjištěným počtem řádků r. Šifrovým textem se postupně zaplňují sloupce. Pořadí je určeno abecedním vyčíslením. Přitom se dbá na (ne)úplnost jednotlivých sloupců. Dešifrování si nyní ukážeme na příkladu.

šifrový text: ILTONHORPOIMAISURSTCASISNNT
klíč: secret

Podle vzorců určíme počet řádků r a počet úplných sloupců f:

$$r = \left\lceil \frac{|C|}{|K|} \right\rceil = \left\lceil \frac{27}{6} \right\rceil = 5,$$

$$f = |C| - [|K| \cdot (r - 1)] = 27 - [6 \cdot (5 - 1)] = 3.$$

Sestavíme prázdnou tabulku o rozměrech $r \times |K|$. Na posledním řádku napočítáme úplné sloupce a zbývající označíme šedě (neúplné sloupce). Šifrový text postupně zapisujeme do tabulky po sloupcích. Začneme třetím sloupcem v tabulce, který má v abecedním vyčíslení číslo 1. Jelikož počet úplných sloupců f je 3, je tento sloupec úplný – je složen z pěti písmen. Následuje tabulka po doplnění tohoto sloupce.

5	2	1	4	3	6
		i			
		l			
		t			
		o			
		n			

Po vepsání celého šifrového textu
ILTON HORPO IMAI SURS TCASI SNNT (barevné rozlišení je pouze pro názornost) dostaneme tabulku, která odpovídá té, která vznikla při šifrování. Otevřený text získáme přepsáním jednotlivých řádků od horního po dolní.

5	2	1	4	3	6
t	h	i	s	i	s
c	o	l	u	m	n
a	r	t	r	a	n
s	p	o	s	i	t
i	o	n			

otevřený text: thisiscolumnartransposition

2.2.3 Myszkowskiho transpozice

Jedná se o upravenou sloupcovou transpozici. Rozdíl spočívá v odlišném přístupu k opakujícím se písmenům klíče. Tuto šifru navrhl Émile Victor Théodore Myszkowski v roce 1902. Šlo o francouzského plukovníka ve výslužbě, který napsal knihu Cryptographie indéchiffrable [9].

V případě, že jsou všechna písmena v klíči unikátní, Myszkowskiho transpozice pro daný otevřený text vytváří identický šifrový text jako sloupcová transpozice. Pro využití odlišného postupu je tedy nezbytné zvolit takový klíč, v němž budou alespoň dvě stejná písmena. Abecední vyčíslení klíče bude mít pro totožná písmena opakující se hodnoty. Rozdíl je zřejmý z tabulky č. 2, v níž jsou abecední vyčíslení pro klíč „secret“. Zatímco u sloupcové transpozice máme $(5, 2, 1, 4, 3, 6)$, pro Myszkowskiho transpozici je to $(4, 2, 1, 3, 2, 5)$ – výskyty písmena „e“ sdílí hodnotu 2.

Sloupcová transpozice je blíže rozvedena v sekci 2.2.2. V popisu šifrování a dešifrování se podíváme pouze na odlišnosti v použití Myszkowskiho transpozice.

	s	e	c	r	e	t
Sloupcová	5	2	1	4	3	6
Myszkowskiho	4	2	1	3	2	5

Tabulka 2: Abecední vyčíslení sloupcové a Myszkowskiho transpozice

Popis šifrování

Sloupce se stejnou hodnotou abecedního vyčíslení se zpracovávají průběžně zleva doprava. Pokud jsou takto dva sloupce (označme je levým a pravým podle jejich relativní pozice), do šifrového textu se nejdříve zapíše první písmeno levého sloupce a za něj první písmeno pravého sloupce. Pokračuje se druhým písmenem levého sloupce a druhým písmenem pravého a tak dále. Sloupce se stejnou hodnotou se tak přepisují do šifrového textu na přeskáčku – slévají se. Sloupců se stejnou hodnotou může být libovolný počet. Proces si ukážeme na příkladu.

otevřený text: thisismyszkowskitransposition
klíč: secret

Hodnotu 2 v abecedním vyčíslení mají dva sloupce (druhý a pátý) a tak je slejeme – vznikne posloupnost H I Y K S T N O T N.

4	2	1	3	2	5
t	h	i	s	i	s
m	y	s	z	k	o
w	s	k	i	t	r
a	n	s	p	o	s
i	t	i	o	n	

šifrový text: ISKSI H I Y K S T N O T N SZIPO TMWAI SORS

Popis dešifrování

Sloupce se stejnou hodnotou abecedního vyčíslení se do prázdných sloupců doplňují postupně na přeskáčku zleva doprava. Ukážeme si příklad.

šifrový text: ISKSIHIYKSTNOTNSZIPOTMWAISORS
klíč: secret

Při zaplňování tabulky budeme druhý a pátý sloupec zpracovávat průběžně, protože mají stejnou hodnotu. Přijde do nich následující část šifrového textu – H I Y K S T N O T N. Do druhého sloupce přepíšeme písmeno „H“ a do pátého písmeno „I“. Následně do druhého sloupce písmeno „Y“ a do pátého písmeno „K“. A tak dále. Oba sloupce budou úplné. Dostaneme následující tabulku.

4	2	1	3	2	5
	h	i		i	
	y	s		k	
	s	k		t	
	n	s		o	
	t	i		n	

2.2.4 Šifra ÜBCHI

Jedná se o dvojitou sloupcovou transpozici. Během šifrování je sloupcová transpozice (viz 2.2.2) aplikována dvakrát po sobě. Pro obě transpozice se používá

stejný klíč – fráze, která se skládá z určitého počtu slov. Před druhou transpozicí se podle tohoto počtu přidávají do textu klamače. Dvojitá sloupcová transpozice staví na skutečnosti, že transpozici je možné provést vícekrát po sobě, čímž se zvyšuje bezpečnost šifry. Existují i další varianty dvojité sloupcové transpozice, které pro každou transpozici vyžadují vlastní klíč.

Šifru ÜBCHI používali Němci za 1. světové války. Nebyli však příliš důslední ve výměně klíčů a posílali opakující se zprávy (např. „všude je klid“), což Francouzům usnadnilo luštění [10].

Popis šifrování

Vybere se klíč, který má představovat frázi obsahující slova. Z klíče se získá abecední vyčíslení a otevřený text se pomocí sloupcové transpozice zašifruje, čímž vznikne „částečný“ šifrový text. Poté se na konec tohoto textu přidají klamače (např. písmeno „w“), jejichž počet odpovídá počtu slov v klíči. Dále se podruhé provede sloupcová transpozice se stejným klíčem a výsledkem je šifrový text. Postup si ukážeme na příkladu.

otevřený text: doublecolumnartransposition
klíč: thekey (dvě slova)

Provedením první sloupcové transpozice vytvoříme následující tabulku.

5	3	1	4	2	6
d	o	u	b	l	e
c	o	l	u	m	n
a	r	t	r	a	n
s	p	o	s	i	t
i	o	n			

Z tabulky získáme „částečný“ šifrový text ulton lmai oorpo burs dcasi ennt. Vzhledem k tomu, že v klíči máme dvě slova, přidáme na konec textu dva klamače (zvolíme písmeno „w“). Na takto vzniklý text aplikujeme sloupcovou transpozici podruhé, přičemž použijeme stejný klíč jako v prvním případě. Postup zjednodušíme tak, že budeme z tabulky výše přepisovat sloupce v pořadí abecedního vyčíslení do identické tabulky jako řádky (viz barevné rozlišení sloupců). Nakonec přidáme zmíněné klamače, což může v některých případech vyžadovat rozšíření tabulky o další řádek. Vzniklá tabulka je níže. Z ní vygenerujeme výsledný šifrový text tak, že přepíšeme sloupce v pořadí daném abecedním vyčíslením.

5	3	1	4	2	6
u	l	t	o	n	l
m	a	i	o	o	r
p	o	b	u	r	s
d	c	a	s	i	e
n	n	t	w	w	

šifrový text: TIBATNORIWLAOCNOOUSWUMPDNLRSE

Popis dešifrování

Při dešifrování postupujeme obráceně vůči operaci šifrování. Invertujeme druhou transpozici, odebereme klamače a následně invertujeme první transpozici. Tím získáme původní otevřený text. Postup si ukážeme na příkladu.

šifrový text: TIBATNORIWLAOCNOOUSWUMPDNLRSE
klíč: thekey (dvě slova)

Provedeme dešifrování jako v případě sloupcové transpozice. Vznikne následující tabulka.

5	3	1	4	2	6
u	l	t	o	n	l
m	a	i	o	o	r
p	o	b	u	r	s
d	c	a	s	i	e
n	n	t	w	w	

Podle počtu slov v klíči určíme počet klamačů. V tomto případě jsou to dva klamače, které budeme ignorovat. Získáme „částečný“ šifrový text „ultonlmaioorpobursdcasiennt“. Dešifrování poté aplikujeme podruhé, čímž dostaneme tabulku níže. Výsledný otevřený text vznikne přepsáním řádků tabulky.

5	3	1	4	2	6
d	o	u	b	l	e
c	o	l	u	m	n
a	r	t	r	a	n
s	p	o	s	i	t
i	o	n			

otevřený text: doublecolumnartransposition

2.2.5 Fleissnerova otočná mřížka

Jedná se o transpoziční šifrovací metodu, která nese jméno rakouského plukovníka Edouarda Fleissnera von Wostrowitze, který v roce 1881 napsal příručku o vojenské kryptografii [11]. Metoda získala velkou popularitu. Jules Verne ji zakomponoval do svého románu Matyáš Sandorf. Fleissnerovu mřížku používala také německá armáda za 1. světové války – nejčastěji v rozměrech 5x5 a 10x10 [10].

Mřížkou se rozumí fyzická pomůcka většinou čtvercových rozměrů vyrobená například z papíru nebo lepenky. Vybraná políčka mřížky jsou vystřižena, takže se skrz ně dá psát. Už v 16. století mřížky používal italský matematik a filosof Gerolam Cardano [12]. Mřížka se přiložila k papíru, do otvorů se vepsala zpráva a mřížka se odebrala. Zbývající části papíru byly doplněny písmeny a slovy, které zprávu skryly – šlo o steganografickou metodu. Fleissnerova mřížka má tvar čtverce a funguje na stejném principu s tím, že se v průběhu šifrování otáčí. Celkem je použita ve čtyřech orientacích (postupně se rotuje o 90° ve směru hodinových ručiček). Výsledkem je šifrový text, který není třeba doplňovat. Jde tedy o transpozici.

Návrh mřížky

Před šifrováním i dešifrováním je nezbytné zhotovit mřížku nebo využít již připravenou mřížku. Mřížka a její počáteční orientace totiž představují klíč, který musí být distribuován příjemci zprávy (stačí informace, jak ji zhotovit). Výroba začíná volbou rozměru n, který určuje počet sloupců a řádků mřížky. Vytvoří se tedy mřížka o rozměrech $n \times n$. V dalším kroku se vyberou políčka, která budou vystřižena. Přitom se musí dbát na to, aby každé místo pod vystřiženým políčkem bylo pokryto právě jednou při provedení všech rotací. Pro usnadnění procesu výběru políček je možné si mřížku rozdělit na kvadranty. V nich se jednotlivá políčka očíslují tak, jako by každý kvadrant měl jinou orientaci. Ukážeme si to na mřížce 4×4.

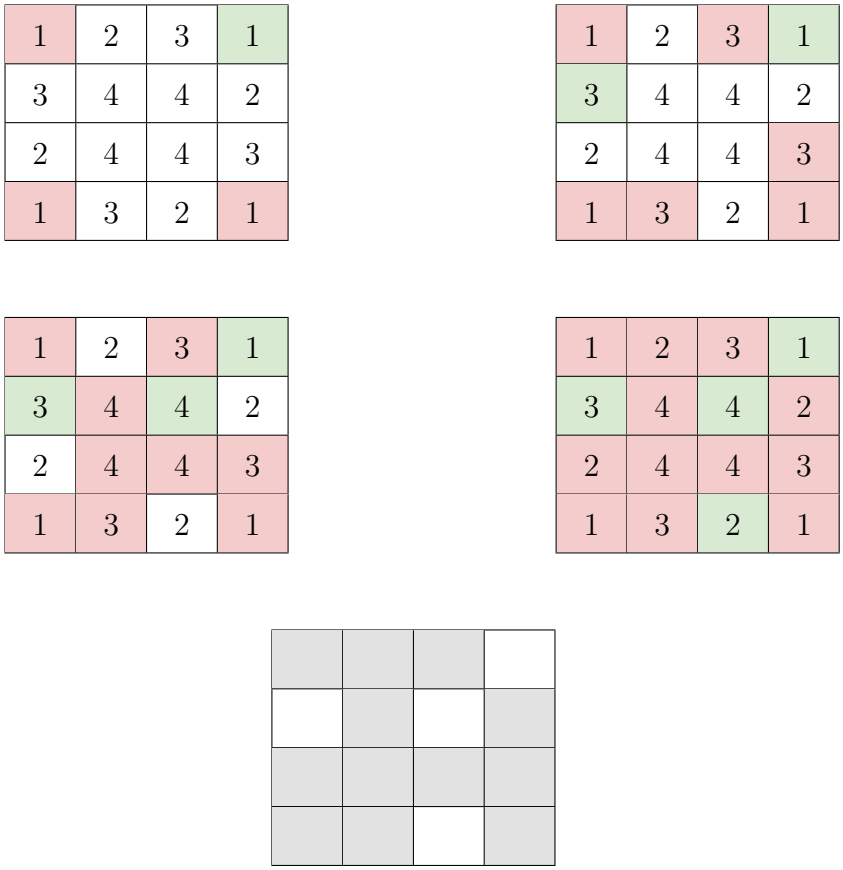
1	2	3	1
3	4	4	2
2	4	4	3
1	3	2	1

Pokud je pro vystřižení zvoleno políčko s číslem 1, ostatní políčka s číslem 1 budou pro další výběr vyloučena. Zvolíme-li například 1 ve druhém kvadrantu

(pravý horní roh), žádné další rohové políčko již vybrat nelze. Toto platí, protože při postupné rotaci pokryje zvolené rohové políčko všechna ostatní rohová políčka.

Políčka jsou vybírána, dokud je to možné. Pro rozměry $n \times n$ bude vystřiženo $\lfloor \frac{n \cdot n}{4} \rfloor$ políček. Pro lichou délku n zůstává prostřední políčko nevyužito.

Na obrázku č. 7 je příklad vytvoření Fleissnerovy mřížky. Postupně jsou vybírána políčka k vystřižení (zbarvena zeleně). Políčka, která již nelze vybrat, jsou zbarvena červeně. Dále je znázorněna výsledná mřížka, která má vystřižená políčka v bílé barvě.



Obrázek 7: Příklad Fleissnerovy mřížky

Popis šifrování

Mřížka se přiloží k papíru a otevřený text se začne vepisovat do prázdných políček ve směru zleva doprava a shora dolů. Jakmile se všechna políčka zaplní, mřížka se rotuje ve směru hodinových ručiček o 90 stupňů a postup se zopakuje. Mřížka se v průběhu šifrování posune celkem třikrát. Po vepsání otevřeného textu ve všech orientacích se mřížka odejme a šifrový text se získá přepsáním řádků.

Rozměry mřížky omezují délku otevřeného textu. Jsou-li rozměry mřížky $n \times n$, otevřený text může mít pro sudé n délku $n \cdot n$ a pro liché n délku $n \cdot n - 1$. Delší otevřený text je třeba rozdělit a šifrovat zvlášť. Pokud je otevřený text

příliš krátký, doplní se libovolnými písmeny (např. „x“). Šifrování si ukážeme na příkladu.

otevřený text: fleissnergrille
mřížka: viz obrázek č. 7

Postup šifrování je znázorněn na obrázku č. 8. Otevřený text vepisujeme do vystřižených políček mřížky (znázorněny bíle). Po prvních čtyřech písmenech se nám naskytne pohled na mřížku vlevo nahoře. Mřížku dále rotujeme a postup opakujeme, dokud nepokryjeme všechny orientace. Při poslední orientaci zůstává jedno políčko prázdné, a tak jej doplníme písmenem „x“.

l	r	s	f
l	l	e	e
s	g	n	r
i	x	i	e

Obrázek 8: Šifrování Fleissnerovou mřížkou

šifrový text: LRSFLLEESGNRIXIE

Popis dešifrování

Pro dešifrování je třeba stejná mřížka, jaká byla použita pro šifrování. Šifrový text se zapíše do tabulky korespondující s rozměry mřížky. Následně se mřížka přiloží k tabulce v počáteční orientaci a odkrytá písmena se začnou přepisovat

ve směru zleva doprava a shora dolů. To se provede postupně ve všech orientacích mřížky, čímž se získá otevřený text.

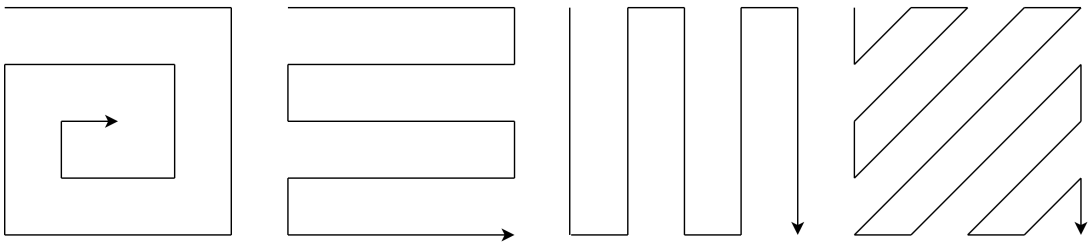
Příklad viz obrázek č. 8. Ze spodní tabulky dostaneme otevřený text přiložením mřížky a následnou rotací.

2.2.6 Šifra Route

Šifra Route je jednoduchá geometrická transpozice, při níž se používá tabulka čtvercového nebo obdélníkového tvaru. Otevřený text se do této tabulky vepíše a šifrový text se vygeneruje průchodem tabulkou po zvolené cestě (odtud název, route = cesta). Příkladem cesty může být „spirála ve směru hodinových ručiček začínající v levém horním rohu“. Zvolená cesta spolu s rozměry tabulky představují klíč šifry.

Cest, které lze vybrat, je velké množství. Je však vhodné brát ohled na jejich použitelnost a bezpečnost. Příjemce musí být schopen cestu replikovat pro dešifrování zprávy. Případná složitost mu to nemusí umožnit. Některé cesty jsou navíc méně vhodné, jelikož jejich použitím šifrový text odhaluje úseky otevřeného textu (ve správném nebo opačném směru).

Mezi základní typy cest patří spirální, horizontální, vertikální a diagonální cesty (obrázek č. 9). Kromě typu je dále nutné specifikovat počáteční bod a v některých případech směr. Například u spirální se může začínat v libovolném rohovém políčku nebo uprostřed a pokračovat ve směru nebo proti směru hodinových ručiček.



Obrázek 9: Příklady cest pro šifru Route

Někdy se pro šifrování používají dvě cesty. První určuje způsob, jakým je otevřený text vepsán do tabulky. Druhá cesta pak specifikuje, jak je text z této tabulky transformován do šifrového textu. Prakticky jde o dvojnásobné použití šifry s různými klíči.

Tabulka se uvažuje úplná. V případě nedostatečné délky otevřeného textu se do ní doplňují náhodná písmena.

Popis šifrování

V první řadě se zvolí rozměry tabulky tak, aby se do ní vešel otevřený text (jinak se rozděluje na více částí). Tabulka se sestrojí a vepíše se do ní otevřený text.

Následně se vybere cesta, po které se písmena v tabulce přepíšou do šifrového textu. Šifrování si ukážeme na příkladu.

otevřený text: thisisspiralroutecipher
počet sloupců: 5
cesta: spirála z pravého horního rohu ve směru hodinových ručiček

Začneme sestrojením tabulky o pěti sloupcích, do které po řádcích přepíšeme otevřený text. Jelikož dvě políčka zůstanou prázdná, zaplníme je náhodnými písmeny – zvolíme „x“. Dostaneme následující tabulku.

t	h	i	s	i
s	s	p	i	r
a	l	r	o	u
t	e	c	i	p
h	e	r	x	x

Poté z tabulky vygenerujeme šifrový text. Postupujeme od pravého horního rohu s písmenem „i“ a dodržíme vybranou cestu. Směr průchodu je zřejmý z následující tabulky – písmena přepíšeme v uvedeném číselném pořadí.

13	14	15	16	1
12	23	24	17	2
11	22	25	18	3
10	21	20	19	4
9	8	7	6	5

šifrový text: IRUPXXREHTASTHISIOICELSPR

Popis dešifrování

Prvním krokem je sestrojení tabulky daných rozměrů. Počet sloupců je určen klíčem a počet řádků se vypočítá tak, že se délka šifrového textu vydělí počtem sloupců. Poté se do tabulky vepíše šifrový text ve směru příslušné cesty. Otevřený text se získá přepsáním tabulky po řádcích od horního po dolní. Dešifrování si ukážeme na příkladu.

šifrový text: IRUPXXREHTASTHISIOICELSPR
počet sloupců: 5

cesta: spirála z pravého horního rohu ve směru hodinových ručiček

Jelikož má šifrový text délku 25, vytvoříme tabulku o rozměrech 5×5. Do této tabulky vepíšeme šifrový text, přičemž začneme v pravém horním rohovém políčku a dodržíme specifikovanou cestu. Následuje znázornění postupného zaplňování tabulky – počátečních 9, 15, 20 a 25 písmen šifrového textu. Otevřený text poté vygenerujeme z řádků tabulky od horního po dolní.

				i
				r
				u
				p
h	e	r	x	x

t	h	i		i
s				r
a				u
t				p
h	e	r	x	x

t	h	i	s	i
s			i	r
a			o	u
t		c	i	p
h	e	r	x	x

t	h	i	s	i
s	s	p	i	r
a	l	r	o	u
t	e	c	i	p
h	e	r	x	x

šifrový text: thisisspiralroutecipherxx

2.3 Kryptoanalýza transpozičních šifer

V této sekci je prozkoumáno prolomení transpozičních šifer. Různé varianty transpozičních šifer se v obtížnosti prolomení liší. Konkrétní metody prolomení mohou být často aplikovány pouze při splnění určitých podmínek. Prověřen je útok hrubou silou a ve složitějších případech metoda lokálního prohledávání – horolezecký algoritmus. Úspěšnost prolamování je testována a výsledky shrnuty.

Prováděn je základní typ útoku, při kterém je znám pouze šifrový text (ciphertext only attack). Vycházíme také z Kerckhoffsova principu, který předpokládá, že útočník zná specifikaci použitého šifrovací systému [6]. Při každém útoku tak máme k dispozici šifrový text, který vznikl zašifrováním náhodného otevřeného textu v anglickém jazyce[4], a informaci o použitém kryptosystému (např. sloupcová transpozice). Prolamování je testováno na zařízení Mac mini (M1, 2020)

[4]popsané principy lze aplikovat i na texty v jiném jazyce, stačí tomu uzpůsobit ohodnocovací funkci (viz dále)

s procesorem Apple M1 (8 jádrové CPU) a 8 GB operační paměti. Kód je napsaný v JavaScriptu (Node.js).

Pro tuto sekci jsou použity zdroje [6], [2], [13], [5], [14].

2.3.1 Ohodnocovací funkce

Pro každý potenciální otevřený text je potřeba ověřit jeho kvalitu. Pokud textů není mnoho, je možné je všechny manuálně zkontrolovat a určit případ, kdy je výsledek smysluplný. Pokud je prověřovaných textů velké množství (tisíce a více), je vhodné použít ohodnocovací funkci (tzv. fitness funkci). Ta každému textu přiděluje číselnou hodnotu, která vyjadřuje jeho smysluplnost. U textu s nejvyšší hodnotou ohodnocení je nejpravděpodobnější, že se jedná o hledaný otevřený text. Na základě ohodnocení tak lze strojově vyfiltrovat nejlepší kandidáty na otevřený text.

Ohodnocovací funkce je implementovaná na základě poznatků a statistiky ze zdroje [14]. Používá statistiku výskytu 4-gramů (n-gram je posloupnost n písmen) v běžném anglickém textu. Každý 4-gram má ohodnocení podle četnosti jeho výskytu. Např. „tion“ se v anglickém textu vyskytuje nejčastěji, zatímco „qkpc“ se v něm nevyskytuje vůbec, takže pokud se „tion“ v daném textu často vyskytuje, je to příznak, že se jedná o text v anglickém jazyce.

Máme ohodnocovací funkci S a chceme získat ohodnocení textu T. Pro každý 4-gram G máme poměr jeho výskytů $count(G)$ vůči počtu všech 4-gramům N, ten se zlogaritmuje a pro všechny 4-gramy (počet c) v textu T se výsledky sečtou. Výpočet se provede podle následujícího vzorce:

$$S(T) = \sum_{i=1}^{c} log\left(\frac{count(G_i)}{N}\right).$$

Text s vyšším ohodnocením je více podobný anglickému textu než text s nižším ohodnocením.

2.3.2 Útok hrubou silou

Princip útoku hrubou silou (brute-force attack) spočívá ve vyzkoušení všech možných klíčů [13]. Pro daný šifrový text je výstupem otevřený text, který je nejlepším kandidátem na původní otevřený text. Pro určení kvality otevřeného textu se použije ohodnocovací funkce. Teoreticky lze tento útok použít na téměř jakoukoli šifru, problémem je jeho časová složitost. Počet potenciálních klíčů roste exponenciálně s délkou klíče. Zdrojový kód č. 1 ukazuje pseudokód útoku hrubou silou.

Šifra Rail Fence je prolomitelná hrubou silou. Klíčem šifry je počet řádků, které byly pro šifrování použity. Pro prolomení je nutné otestovat počet řádků v rozmezí od 2 do $n - 1$, kde n je délka šifrového textu. Větší počet řádků není třeba testovat, protože by vedl na šifrový text identický otevřenému textu. Rychlost algoritmu je zřejmá z tabulky č. 3. I pro šifrové texty o značné délce

```
function prolomitHrubouSilou(sifrovyText) {
  let nejlepsiText = desifrovat(nahodnyKlic(), sifrovyText)
  for (const klic of vsechnyKlice()) {
    const text = desifrovat(klic, sifrovyText)
    if (ohodnoceni(text) > ohodnoceni(nejlepsiText))
      nejlepsiText = text
  }
  return nejlepsiText
}
```

Zdrojový kód 1: Pseudokód prolomení hrubou silou

vrací správný výsledek do několika sekund. V případě textu do délky 2000 písmen trvá vyzkoušení všech klíčů méně než jednu sekundu.

délka textu	**čas prolomení**
100	357,177ms
200	351,773ms
500	383,637ms
1000	503,616ms
2000	936,112ms
5000	4,049s
10000	17,239s

Tabulka 3: Rychlost prolomení šifry Rail Fence hrubou silou

Šifru Route lze za určitých podmínek také prolomit hrubou silou. Pokud definujeme základní množinu přijatelných klíčů (cest), můžeme všechny otestovat. Teoreticky existuje velké množství cest, ale základních je pouze omezené množství. Složitější cesty se také složitěji aplikují a jejich použití může vést k chybám. Pro pokrytí více možností je možné dodefinovat další přijatelné cesty. Při prolomení jsou všechny definované cesty vyzkoušeny pro všechny potenciální rozměry tabulky. Uvažujeme úplnou tabulku (poslední řádek nemá prázdná políčka), a tak počet sloupců musí být dělitelem délky šifrového textu. Výsledky prolomení jsou zaznamenány v tabulce č. 4. Prolomení je velmi rychlé. Text o délce deseti tisíc písmen je prolomen za 1,295 sekund, zatímco kratší texty pod jednu sekundu.

Další transpoziční šifry lze za určitých podmínek také prolomit hrubou silou. Šifrový klíč však musí být krátký. Ukážeme si to na sloupcové transpozici[5]. Počet potenciálních klíčů[6] pro zvolenou délku klíče n je dán funkcí $n!$, která roste velmi

[5]pro ostatní šifry je to podobné, jen u Fleissnerovy mřížky místo délky klíče uvažujeme velikost mřížky

[6]klíč a abecední vyčíslení bereme jako totéž

délka textu	čas prolomení
100	340,03ms
200	358,982ms
500	378,518ms
1000	428,161ms
2000	580,337ms
5000	778,677ms
10000	1295ms

Tabulka 4: Rychlost prolomení šifry Route hrubou silou

rychle. Na počítači se specifikací uvedenou výše trvá prolomení textu délky 100 pro klíč délky 10 přibližně jednu minutu. Přímou úměrou dospějeme k tomu, že pro klíč délky 11 prolomení trvá 11x déle. Pro délku 15 je ovšem potenciálních klíčů už 360360x více než pro délku klíče 10. To znamená, že prolomení bude trvat zhruba 250 dní. Je tedy nezbytné přijít s „chytřejší“ metodou prolomení.

2.3.3 Horolezecký algoritmus

Na kryptoanalýzu historických šifer lze často pohlížet jako na optimalizační problém, což blíže popisuje [6]. Pro takový problém existuje příliš mnoho možných řešení, takže nelze všechny prověřit v rozumném čase. To je i případ sloupcové transpozice, jak jsme ukázali výše. Proto se používají heuristické algoritmy, jejichž cílem je nalezení kvalitního řešení v rozumném čase. Nevýhodou je, že tento způsob nezaručuje nalezení nejlepšího řešení.

Pro kryptoanalýzu historických šifer se používá horolezecký algoritmus (hill climbing). Jedná se o příklad obecné heuristické strategie (metaheuristiky) – lokálního prohledávání. Princip spočívá v iterativním vylepšování náhodně vygenerovaného řešení. Základem je ohodnocovací funkce, podle které se pozná lepší řešení, a způsob získávání „sousedů“ aktuálního řešení – ti se vytvářejí jeho transformací. Jak název napovídá, způsob fungování si můžeme představit jako šplhání na horu. Začne se na náhodném místě a postupuje se výše. V ideálním případě se dospěje až na vrchol, tedy k nejlepšímu řešení. Lokální prohledávání se uplatňuje také při řešení problému obchodního cestujícího[7], což je známý optimalizační problém.

Horolezecký algoritmus lze na historické šifry aplikovat z důvodu limitovaného rozptýlení. To znamená nedostatečnou schopnost skrývat statistické vzorce otevřeného textu. Naproti tomu moderní šifry jsou navrženy s velkým rozptýlením [2]. To se projevuje tak, že malá změna v klíči způsobí vygenerování úplně rozdílného šifrového textu. U historických šifer je naopak šifrový text podobný,

[7] https://en.wikipedia.org/wiki/Travelling_salesman_problem

pokud v klíči uděláme pouze malou změnu.

Horolezecký algoritmus nezkouší všechny potenciální klíče. Z toho plyne možnost, že uvázne na lokálním vrcholu před dosažením globálního vrcholu. Přeneseně to znamená, že sice dojde k vyšplhání hory, ovšem nikoli hory nejvyšší. Z toho důvodu je výhodné nechat algoritmus běžet vícekrát, aby byl započat z různých výchozích bodů.

Při aplikaci horolezeckého algoritmu na prolomení šifry je vstupem šifrový text. Prvním krokem je vygenerování náhodného klíče. Následně je mezi jeho „sousedy“, což jsou lehké modifikace původního klíče, hledán lepší kandidát pro dešifrování šifrového textu. Původní klíč je lepším klíčem nahrazen. To se děje iterativně a je tak neustále hledán lepší a lepší klíč. Proces pokračuje, dokud jsou lepší sousední klíče nacházeny. Pro určení lepšího klíče se použije ohodnocovací funkce. Generování sousedních klíčů je pro každou šifru jiné. Zdrojový kód č. 2 ukazuje pseudokód horolezeckého algoritmu.

```
function prolomitHorolezeckymAlgoritmem(sifrovyText) {
  let nejlepsiKlic = nahodnyKlic()
  let nejlepsiText = desifrovat(nejlepsiKlic, sifrovyText)
  let lokalniMaximum = false
  while (!lokalniMaximum) {
    lokalniMaximum = true
    for (const klic of sousedniKlice(nejlepsiKlic)) {
      const text = desifrovat(klic, sifrovyText)
      if (ohodnoceni(text) > ohodnoceni(nejlepsiText)) {
        nejlepsiKlic = klic
        nejlepsiText = text
        lokalniMaximum = false
      }
    }
  }
  return nejlepsiText
}
```

Zdrojový kód 2: Pseudokód prolomení horolezeckým algoritmem

Sloupcová transpozice

Pomocí horolezeckého algoritmu lze prolamovat sloupcovou transpozici i pro délku klíče, která je hrubou silou neprolomitelná. Pro kratší klíče je navíc výrazně rychlejší. Algoritmus hledá klíč, který dešifrováním daného šifrového textu vygeneruje nejlepší otevřený text. K určení kvality klíče se použije na vzniklý otevřený text ohodnocovací funkce.

Pro získání sousedních klíčů se aplikuje operace segmentového posunu podle [6]. Při segmentovém posunu se část klíče začínající na pozici p o délce l posune o s míst vpravo. Operace je cyklická – pokud se při posunu narazí na konec klíče, pokračuje se na jeho začátku. V algoritmu pro prolomení sloupcové transpozice

generujeme pro klíč všechny segmentové posuny jeho částí o délce l na pozici p s posunem s míst. Vzniká velká šance, že tímto bude nalezeno lepší než dosavadní řešení. Segmentový posun si ukážeme na příkladu. Máme klíč $(0, 1, 2, 3, 4, 5, 6, 7)$. Zvolíme segment $(3, 4)$, který má délku $l = 2$ a začíná na pozici $p = 3$. Posuneme jej o $s = 3$ pozice. Výsledkem je klíč $(0, 1, 2, 5, 6, 7, 3, 4)$.

Při použití horolezeckého algoritmu předpokládáme znalost délky klíče. Pokud by délka klíče nebyla známa, bylo by třeba algoritmus postupně vyzkoušet pro všechny možné délky klíče (běžně se používalo rozmezí 5–25). Prvně otestujeme jednodušší variantu sloupcové transpozice s úplnou tabulkou. Zvolíme náhodný text v angličtině o délce 200 písmen. Algoritmus necháme pro každou délku klíče běžet stokrát (každý běh je dále v textu označován jako jedna iterace), aby se zamezilo uváznutí v lokálních maximech. Výsledky jsou v tabulce č. 5. Až do délky 25 včetně je algoritmus úspěšný a k získání správného klíče potřebuje maximálně několik iterací. Čas pozvolně narůstá, ale je maximálně v řádu několika sekund na iteraci. Pro délku klíče 30 už však správného klíče není dosaženo a i časová náročnost se prodloužila na průměrných 8 sekund na iteraci. Vyzkoušíme dále šifrový text o délce 1000 pro klíče o délce 30. Správný klíč je tentokrát nalezen, ovšem průměrný čas jedné iterace se blíží k jedné minutě.

délka klíče	**úspěšná iterace**	**průměrný čas iterace**
5	4	4,93ms
10	2	82,77ms
15	1	580,89ms
20	2	1339,2ms
25	2	3666,2ms
30	nezjištěno	7711,6ms

Tabulka 5: Prolomení sloupcové transpozice s úplnou tabulkou horolezeckým algoritmem

Otestujeme i variantu sloupcové transpozice s neúplnou tabulkou, což znamená, že poslední řádek není úplný. Výsledky jsou zaznamenány v tabulce č. 6. Je z ní zřejmé, že u této varianty je algoritmus méně úspěšný. Už pro délku klíče 20 potřebuje 27 iterací a pro délku klíče 25 není správného výsledku po 100 iteracích dosaženo. Neúspěch pro délku klíče 25 je částečně kompenzován tím, že jsou odhalovány alespoň některé pasáže původního otevřeného textu.

Algoritmus může mít potíže s prolomením zprávy, která je příliš krátká. Toto prověřujeme pro délku klíče 15. Výsledky jsou v tabulce č. 7. Délka textu 75 písmen stačí pro prolomení, ale délka 50 už ke správnému výsledku nevede. Přesto lze z výsledného textu určité úseky vyčíst.

délka klíče	délka textu	úspěšná iterace
5	198	2
10	194	1
15	203	2
20	210	27
25	214	nezjištěno

Tabulka 6: Prolomení sloupcové transpozice s neúplnou tabulkou horolezeckým algoritmem

délka klíče	délka textu	úspěšná iterace
15	150	1
15	100	3
15	75	1
15	50	nezjištěno

Tabulka 7: Prolomení sloupcové transpozice s krátkým textem horolezeckým algoritmem

Šifra ÜBCHI

Jedná se o variantu dvojité sloupcové transpozice, která je používána v méně bezpečné podobě. Pro obě transpozice se používá stejný klíč. I přesto jde o bezpečnější šifru v porovnání s jednoduchou sloupcovou transpozicí. Důvodem je dosažení většího rozptýlení, protože transpozice je aplikována dvakrát.

Horolezecký algoritmus není při prolamování šifry ÜBCHI tak úspěšný. Poradí si s případy, kdy je použit klíč do délky 15 písmen. Zvolíme náhodný text v angličtině o délce 200 písmen a z důvodu větší náročnosti necháme algoritmus běžet tisíckrát. Výsledky jsou v tabulce č. 8. Do délky klíče 13 je šifra prolomena, s klíčem o 13 písmenech už je problém. Pro 200 písmen šifrového textu se totiž jedná o neúplnou tabulku, takže vyzkoušíme prolomení textu o délce 208 písmen a toto již je úspěšné. Pro délku klíče 15 však správný klíč nalezen není. Při prolamování vycházíme ze znalosti délky klíče a počtu slov v něm. V případě neznalosti je třeba různé varianty zkoušet.

Fleissnerova mřížka

Horolezeckým algoritmem lze prolamovat i Fleissnerovu mřížku. Narozdíl od ostatních popisovaných šifer není klíčem posloupnost písmen, ale mřížka o rozměrech $n \times n$ se zvolenými políčky. Pro velikost mřížky 15×15 máme srovnatelný počet možností jako v případě klíče o délce 30 písmen.

Algoritmus v každé iteraci u mřížky uvažuje všechny možné změny jednoho

délka klíče	úspěšná iterace	průměrný čas iterace
5	2	6,38ms
10	1	102,09ms
12	38	135,38ms
13	nezjištěno	214,39ms
14	375	503,04ms
15	nezjištěno	352,05ms

Tabulka 8: Prolomení šifry ÜBCHI horolezeckým algoritmem

políčka a vybírá tu nejlepší z nich. V případě, že lepší změny již nelze dosáhnout, jsou provedeny dvě náhodné změny, aby došlo k opuštění lokálního maxima. Iterace jsou výrazně rychlejší než v případě ostatních šifer, a proto algoritmus při testování necháme běžet 10000 krát. Opět předpokládáme znalost velikosti klíče, v tomto případě mřížky.

Algoritmus je až do velikosti mřížky 15×15 úspěšný a rychlý. Pro velikost 15×15 a 16×16 je již dosaženo limitu metody. I přesto, že se pro tyto rozměry mřížky objevují ve výsledném textu určité pasáže původního otevřeného textu, správného výsledku dosaženo není. Pro velikost mřížky 15×15 vyzkoušíme při dosažení lokálního maxima tři náhodné změny místo dvou, což nakonec vede k původnímu otevřenému textu. Vysvětlujeme to tak, že to usnadnilo opouštění lokálního maxima. Kompletní výsledky jsou zaznamenány v tabulce č. 9.

velikost mřížky	délka textu	počet změn	úspěšná iterace
4	16	2	28
8	64	2	118
10	100	2	1848
12	144	2	276
14	196	2	143
15	224	2	nezjištěno
15	224	3	1462
16	256	2	nezjištěno
16	256	3	nezjištěno

Tabulka 9: Prolomení Fleissnerovy mřížky horolezeckým algoritmem

Myszkowskiho transpozice

Horolezecký algoritmus lze aplikovat také na Myszkowskiho transpozici. Jde o šifru, která vychází ze sloupcové transpozice s tím rozdílem, že se v abecedním vyčíslení mohou vyskytovat stejná čísla a s těmito sloupci je pak zacházeno jinak. Pokud je opakujících se čísel malé množství, lze šifru prolomit stejným způsobem jako sloupcovou transpozici, protože větší část otevřeného textu bude odkryta. Algoritmus však upravíme tak, aby náhodně vygenerované abecední vyčíslení mohlo obsahovat stejné hodnoty. Takové řešení si dokáže poradit s klíči do délky 10, pro delší je třeba vymyslet sofistikovanější metodu generování sousedních klíčů.

3 Praktická část

Výstupem práce je webová aplikace, která umožňuje s popsanými transpozičními šiframi pracovat. Jedná se o aplikaci postavenou na JavaScriptovém frameworku Vue.js. Spojuje jednoduchost použití desktopových aplikací a dostupnost webových stránek. V této kapitole je popsána její implementace – jsou vyjmenovány zvolené technologie a na příkladech zdrojového kódu ukázáno jejich použití.

Aplikace je napsána pomocí základních klientských (front-end) webových technologií – HTML, CSS a JavaScriptu. Základní rámec poskytuje Vue CLI, což je standardní balíček nástrojů pro vývoj Vue.js aplikací. Pro psaní kódu byl použit textový editor Visual Code Studio[8] od společnosti Microsoft. Aplikace je publikována prostřednictvím platformy Netlify a je přístupná na doméně transpozicnisifry.cz. V podkapitolách jsou použité technologie blíže představeny.

Pro kapitolu jsou použity zdroje [15], [16], [17] pro Vue.js a jeho knihovny. Pro HTML, CSS a JavaScript [18]. Pro comlink-loader [19].

3.1 Webové technologie

Mezi základní klientské (front-end) webové technologie patří HTML, CSS a JavaScript. Každá z nich je zodpovědná za jiný aspekt webové stránky. Následuje stručná charakteristika těchto technologií.

3.1.1 HTML

HTML (HyperText Markup Language) je značkovací jazyk. Jde o základní stavební kámen webových stránek – definuje strukturu a význam jejího obsahu. Pomocí hypertextových odkazů umožňuje propojovat webové stránky. Zdrojový kód napsaný v HTML je textový soubor zpravidla s příponou .html, který je tvořen značkami jazyka HTML a obsahem webové stránky.

[8] https://code.visualstudio.com/

3.1.2 CSS

Pro vizualizaci webových stránek se používá Cascading Style Sheets (CSS). CSS specifikuje, jak se mají elementy stránky vykreslit. Pomocí CSS se vybírají HTML elementy, kterým se definují vlastnosti. Zdrojovým souborem stylů webové stránky je textový soubor s příponou .css. Soubor se skládá z CSS pravidel. Selektorem se určí element na stránce a specifikují se hodnoty vlastností elementu. Použitím HTML a CSS je oddělen obsah webové stránky od její prezentace.

3.1.3 JavaScript

JavaScript je multiparadigmový dynamicky typovaný programovací jazyk, ve kterém jsou funkce elementy prvního řádu[9]. Je znám především jako skriptovací jazyk, který zajišťuje interaktivitu webových stránek[10]. Syntaxe JavaScriptu je velmi podobná jazykům Java, C++ a dalším. Poskytuje běžné konstrukty jako jsou podmíněné příkazy, cykly, switch a další. JavaScript podporuje objektově-orientovaný, imperativní a funkcionální styl programování. Běží ve webovém prohlížeči a bývá použit pro specifikaci toho, jak webová stránka reaguje na výskyt určitých událostí (např. kliknutí myší na tlačítko).

Webová stránka je dokument, který zobrazuje webový prohlížeč. Aby bylo možné programově měnit její strukturu, vzhled a chování, reprezentuje se pomocí Document Object Model (DOM). Jedná se o rozhraní se stromovou hierarchií, v níž každý uzel představuje element na stránce. Pomocí JavaScriptu lze k DOM přistupovat a manipulovat jej.

V základní podobě je JavaScript jedno-vláknový (single-threaded) programovací jazyk. V určitý okamžik je vykonávána pouze jedna operace a ostatní jsou blokovány. Protože uživatelské rozhraní (UI) běží na stejném vlákně, déletrvající operace způsobí jeho zamrznutí. Toto je nežádoucí, a proto existují nástroje, kterými lze zajistit, aby vlákno blokováno nebylo. Základem je asynchronní vykonávání kódu. Operace, jejíž provedení zabere delší dobu (např. načtení obrázku ze serveru), může být vykonána v pozadí. Pro takovou operaci specifikujeme kód, který se vykoná, jakmile dojde k jejímu dokončení. Vlákno s UI mezitím nebude blokováno a může běžet jiný kód. Asynchronní operace jsou vloženy do tzv. event queue a dostanou se na řadu, jakmile „hlavní program“ již nemá co vykonávat.

V minulosti se asynchronní kód řešil pomocí callbacků, což jsou funkce, které se předávají jako parametry. Moderním řešením jsou objekty *Promise*. Objekt *Promise* je proxy (prostředník) pro hodnotu, která nemusí být v době jeho vytvoření známa. Umožňuje definovat obsluhu (ne)získání hodnoty (metodou *then*) nebo chyby (metodou *catch*). Použití objektu *Promise* vede k přehlednějšímu kódu.

[9] element prvního řádu můžeme pojmenovat, předat jako parametr funkci a vracet jako výsledek aplikace funkce

[10] v této práci je použit za tímto účelem, jeho uplatnění je však mnohem širší

Pro výpočetně náročné operace existuje nástroj Web Workers API, který umožňuje spouštět skripty v dalších vláknech. Může se tak provádět více operací souběžně, aniž by došlo k zablokování UI vlákna. *Worker* je objekt vytvořený konstruktorem, kterému je předáno jméno JavaScriptového souboru. Kód tohoto souboru bude vykonán dalším vláknem. Toto řešení má i své limitace – *Worker* nemůže přímo manipulovat DOM a používat některé metody a vlastnosti objektu *window*. Komunikace mezi vlákny probíhá pomocí metody *postMessage* a obsluhy události *onmessage*.

Nedílnou součástí moderního front-end webového vývoje jsou JavaScriptové frameworky. Frameworky za programátora řeší složité, zdlouhavé a opakující se úkony – zjednodušují vývoj softwaru. Nástup JavaScriptových frameworků výrazně usnadnil vývoj dynamických a interaktivních aplikací. Nepřinášejí nové funkce, nýbrž poskytují snadnější přístup ke schopnostem JavaScriptu. Existuje celá řada JavaScriptových frameworků. Mezi nejpopulárnější patří React, Angular a Vue.js.

3.2 Vue.js

Podrobný popis najdete v oficiální dokumentaci[11].

Vue.js je JavaScriptový framework pro vývoj uživatelských rozhraní. Je navržen tak, aby mohl být integrován do již existujícího projektu. Za použití moderních nástrojů a podpůrných knihoven na něm ale může být postavena i sofistikovaná jednostránková aplikace (Single-page application)[12]. V jádru Vue.js je systém, který zajišťuje, že data a DOM jsou propojeny a všechno je tak reaktivní.

Důležitým konceptem Vue.js je systém, který umožňuje vytvářet komplexní aplikace složené z malých nezávislých komponentů. Jde o znovupoužitelné jednotky se jménem. Každý komponent lze definovat v jednom souboru s příponou .vue (single-file component). V tomto souboru se specifikuje jeho HTML šablona, CSS styly a JavaScriptový kód. Všechny složky komponentu jsou tedy pohromadě, což zajišťuje vysokou soudržnost a snadnou udržovatelnost. Komponenty jsou spojovány do výsledné aplikace. Zdrojový kód č. 3 ukazuje příklad jednoduchého jednosouborového komponentu.

3.2.1 Vue Router

Pro většinu jednostránkových aplikací je doporučeno použití oficiálně podporované knihovny Vue Router. Pomocí této knihovny se Vue.js komponenty mapují na URL. Místo běžné HTML značky `<a>` se používá značka `<router-link>` s atributem `:to`. Tímto Vue Router zajišťuje změnu URL bez nutnosti aktualizace stránky. Vue Router také řeší vygenerování URL. Pomocí značky `<router-view>` se specifikuje, kde se komponent spjatý s URL na stránce vyrenderuje.

[11] https://v3.vuejs.org/guide/introduction.html

[12] webová aplikace, která je dynamicky přepisována namísto toho, aby docházelo k načítání nových stránek

```
<template>
  <h1>{{ greeting }} World!</h1>
</template>

<script>
export default {
  data() {
    return {
      greeting: 'Hello'
    };
  }
};
</script>

<style scoped>
p { text-align: center; }
</style>
```

Zdrojový kód 3: Ukázka Vue.js komponentu

3.2.2 Vue CLI

Vue CLI si klade za cíl být standardním balíčkem nástrojů pro vývoj Vue.js aplikací. Zajišťuje, aby různé nástroje fungovaly hladce bez nutnosti zdlouhavé konfigurace. I tak nabízí flexibilitu pro vyladění konfigurace každého nástroje. Příkazem `vue create <nazev>` lze rychle vygenerovat nový projekt.

3.3 Implementace

Tato podkapitola popisuje proces vývoje aplikace za použití představených technologií. Postup je demonstrován na příkladech zdrojového kódu.

3.3.1 Struktura projektu

Aplikace byla vyvinuta pomocí Vue CLI, což je standardní balíček nástrojů pro vývoj Vue.js aplikací. Umožňuje rychlé vytvoření projektu se základní strukturou. Tato struktura byla dodržena a dále rozšířena o části, které byly potřebné pro realizaci aplikace. Struktura projektu je následující:

- **public**
  Obsahuje statické prostředky, které nejsou zpracovány při sestavování aplikace. Součástí je soubor index.html, favicon ikona a soubor _redirects.

- **src**
  Obsahuje zdrojový kód aplikace. Pro logickou organizaci kódu je složka rozdělena na podsložky a soubory:

    * **assets**
      Obsahuje obrázky použité na stránce.
    * **ciphers**
      Obsahuje kód realizující šifrování a dešifrování, který je rozdělen do samostatných modulů pro každý šifrový systém.
    * **components**
      Obsahuje komponenty rozdělené do několika podsložek.
    * **composables**
      Obsahuje kód, který používají některé komponenty.
    * **router**
      Obsahuje soubor index.js, ve kterém jsou specifikovány cesty (URL) stránky a komponenty, které jsou s nimi spjaty.
    * **solve**
      Obsahuje kód, který zajišťuje prolamování transpozičních šifer.
    * **views**
      Obsahuje komponenty, které představují sekce aplikace, tzv. view.
    * **App.vue**
      Soubor, který definuje základní šablonu a styly každého view.
    * **main.js**
      Soubor, který specifikuje vytvoření aplikace a přidává k ní Vue Router.

- **tests**
  Obsahuje testy.

- **konfigurační soubory**
  Součástí je několik konfiguračních souborů. Například package.json specifikuje balíčky, na kterých je projekt závislý.

3.3.2 Kód šifrování a dešifrování

Pro každý transpoziční systém je ve složce *ciphers* jeden modul (soubor), který obsahuje kód pro šifrování a dešifrování textu a řádně jej dokumentuje. Vše je napsáno v čistém JavaScriptu. Složka *ciphers* je nezávislá na ostatních částech projektu a mohla by z ní být vytvořena knihovna (balíček) pro samostatné použití. Pro jednoduchost je kód ponechán v projektu. Při programování bylo vycházeno z potřeb aplikace, ale pro úplnost byly dodefinovány i další funkce, aby rozhraní bylo úplné.

Například kód pro šifrování a dešifrování sloupcovou transpozicí je uveden v modulu *columnar.js*. Rozhraní modulu tvoří čtyři funkce – *encrypt*, *encryptByPermutation*, *decrypt* a *decryptByPermutation*. Při použití sloupcové transpozice je třeba získat pro klíč jeho abecední vyčíslení, tento výpočet je vyjmut do samostatné funkce. Protože se abecední vyčíslení netýká jen sloupcové transpozice,

funkce *keyPermutation* je umístěna do samostatného modulu *permutation.js*. Definice funkce je uvedena ve zdrojovém kódu č. 4. Jejím vstupem je klíč a výstupem abecední vyčíslení, které je reprezentováno polem indexů.

```
export function keyPermutation(key, { normalize = true } = {}) {
  if (normalize) key = normalizeKey(key)

  // pole indexů, např. [0, 1, 2, 3] pro délku 4
  const indices = [...Array(key.length).keys()]
  // setřídění indexů podle abecedního pořadí písmen klíče
  // získání umístění indexů ve výsledné permutaci
  indices.sort((x, y) => key.charCodeAt(x) - key.charCodeAt(y))
  // rozmístění indexů do výsledné permutace
  const permutation = Array(key.length)
  for (let i = 0; i < key.length; i++) {
    permutation[indices[i]] = i
  }

  return permutation
}
```

Zdrojový kód 4: Funkce pro výpočet abecedního vyčíslení

Funkce *encrypt* a *decrypt* pro sloupcovou transpozici používají funkci *keyPermutation*. Protože se v uživatelském prostředí pro názornost abecední vyčíslení zobrazuje, dělá se jeho výpočet samostatně. Abecední vyčíslení je v UI k dispozici, a proto jsou do *columnar.js* přidány funkce *encryptByPermutation* a *decryptByPermutation*, které nepřijímají klíč, ale jeho abecední vyčíslení. Toto zajišťuje, že se abecední vyčíslení nepočítá při každém šifrování (dešifrování) znovu. Ve zdrojovém kódu č. 5 je uvedena definice funkcí *encrypt* a *encryptByPermutation*.

Při volání funkcí *keyPermutation* a *encryptByPermutation* (a dalších) je možné nastavit atribut *normalize*, který je součástí nepovinného parametru. Ten specifikuje, zda se mají vstupní hodnoty normalizovat. K tomu se používají funkce definované v dalším samostatném modulu se jménem *normalization.js*. Protože jsou tyto normalizační funkce použity v komponentech a je zajištěno, že normalizace textu probíhá v UI automaticky, když uživatel píše text do formulářů, není třeba, aby se při šifrování a dešifrování prováděla znovu. Tento atribut je tak možné nastavit na *false*. Toto je také důležité při prolamování, protože při něm se opakovaně volá funkce pro dešifrování a je výhodné, aby její vyhodnocení bylo co nejrychlejší. Funkce jsou navrženy tak, aby dělaly jen to, co se od nich požaduje.

3.3.3 Komponenty

Ve složce *views* je pro každý šifrový systém definován jednosouborový Vue.js komponent (dále je nazýván view pro odlišení od běžných komponentů), který

```
export function encrypt(key, text) {
  const permutation = keyPermutation(key)
  return encryptByPermutation(permutation, text)
}

export function encryptByPermutation(permutation, text, { normalize
    = true } = {}) {
  if (normalize) text = normalizeText(text)

  const keyLength = permutation.length
  const cipherCols = Array(keyLength).fill('')

  // vytváření šifrových sloupců v permutovaném pořadí
  for (let col = 0; col < keyLength; col++) {
    const colPosition = permutation[col]
    for (let i = col; i < text.length; i += keyLength)
      cipherCols[colPosition] += text.charAt(i)
  }

  return cipherCols.join('').toUpperCase()
}
```

Zdrojový kód 5: Kód šifrování sloupcovou transpozicí

obsahuje mimo jiné formulář pro šifrování a dešifrování. Například pro sloupcovou transpozici máme soubor *Columnar.vue*. Jednotlivá view mají podobnou strukturu a formulář pro šifrování a dešifrování je často složen ze stejných částí – má pole pro klíč, otevřený text, šifrový text a další části. Pro tyto účely máme několik znovupoužitelným komponentům, jejichž kompozicí výsledný formulář vznikne. Tyto komponenty jsou umístěny v podsložce *input* složky *components*.

Ve view jsou definována data (proměnné), nad kterými šifrování a dešifrování probíhá. Ve view *Columnar* je to *keyValue*, *plainText* a *cipherText*. Každá datová položka je spojena se samostatným komponentem, který zapouzdřuje pole pro zadání její hodnoty a logiku, která je s ní spjatá. Pro otevřený text máme komponent *PlainTextArea*, jehož definice je uvedena ve zdrojovém kódu č. 6. Součástí je HTML šablona a JavaScriptový kód. Šablona je tvořena elementy `<label>`, `<textarea>` a `<p>`. Element `<p>` představuje zpětnou vazbu ve formě chybové hlášky, která se zobrazí, pokud vstup není validní. To je řešeno pomocí atributu *v-show* – element je zobrazen pouze v případě, kdy je jeho hodnota *true*.

Komponent *PlainTextArea* je součástí view *Columnar*. Ve view je zaregistrován a v jeho šabloně specifikován, což je zaznamenáno ve zdrojovém kódu č. 7 (kód view je pro stručnost neúplný). S komponentem je sdílena hodnota *plainText*, která je pak v políčku zobrazena. Pokud uživatel do políčka napíše jinou hodnotu, vyvolá se událost *input*, jejíž obsluhou je metoda *valueChanged*. Nová hodnota se z pole (textarea) získá, normalizuje se a následně se vyvolá další událost, která směřuje do view *Columnar*. V něm je pro *PlainTextArea* speci-

```
<template>
  <div class="input-group">
    <label for="plainTextArea">Otevřený text</label>
    <textarea
      id="plainTextArea"
      :value="value"
      :class="{ error: !isValid }"
      placeholder="zadej otevřený text"
      @input="valueChanged"
      rows="5"
    ></textarea>
    <p v-show="!isValid" class="error-msg error">{{ invalidFeedback }}</p>
  </div>
</template>

<script>
import { normalizePlainText } from '@/ciphers/normalization'
export default {
  props: {
    value: String,
    maxLength: Number,
    isValid: {
      type: Boolean,
      default: true
    },
    invalidFeedback: String
  },
  emits: ['valueChanged'],
  methods: {
    valueChanged(event) {
      const value = event.target.value
      const normalizedValue = normalizePlainText(value).substring(0, this.maxLength)
      event.target.value = normalizedValue
      this.$emit('valueChanged', normalizedValue)
    }
  }
}
</script>
```

Zdrojový kód 6: Komponent PlainTextArea

fikována obsluha události *valueChanged* metodou *plainTextChanged*. Na změnu hodnoty otevřeného textu je v této metodě adekvátně reagováno – dojde k zašifrování. Výsledek se poté uživateli zobrazí pomocí dalšího komponentu, jímž je *CipherTextArea*.

```
<template>
  ...
  <plain-text-area
    :value="plainText"
    @valueChanged="plainTextChanged"
  />
  ...
</template>

<script>
...
import PlainTextArea from '@/components/input/PlainTextArea'
...
export default {
  components: {
    ...
    'plain-text-area': PlainTextArea,
    ...
  },
  data() {
    return {
      isEncrypting: true,
      keyValue: '',
      plainText: '',
      cipherText: ''
    }
  },
  methods: {
    ...
    plainTextChanged(value) {
      this.plainText = value
      this.isEncrypting = true
      this.encrypt()
    },
    ...
  },
  ...
}
</script>
```

Zdrojový kód 7: Použití komponentu PlainTextArea ve view Columnar

3.3.4 Router

Ve zdrojovém kódu č. 8 je ukázáno, jak jsou pomocí nástroje Vue Router specifikovány URL pro jednotlivá view. Pro příklad je zvoleno view Columnar, které reprezentuje sloupcovou transpozici. Součástí je i hodnota *title*, která je zobrazena na záložce v prohlížeči. Vše je uvedeno v souboru *index.js* ve složce *router*.

```
const routes = [
  ...
  {
    path: '/sloupcova-transpozice',
    name: 'Columnar',
    meta: {
      title: 'Sloupcová transpozice'
    },
    component: Columnar
  },
  ...
]
```

Zdrojový kód 8: Nastavení URL pro view Columnar

3.3.5 Prolamování

Kód pro prolomení šifer využívá knihovnu Comlink (konkrétně nástroj comlink-loader[13]), která poskytuje transparentní použití Web Workers API. Tento nástroj umožňuje definovat asynchronní funkci v samostatném souboru s koncovkou .worker.js, kterou je pak možné kdekoli importovat. Vykonání této funkce je provedeno v dalším vlákně, takže neblokuje UI vlákno.

Pro každou šifru je ve složce *solve* definován soubor s příponou .worker.js, který exportuje funkci pro její prolomení. Vyřešit toto formou vykonávání v jiném vlákně je zásadní, protože za určitých podmínek může prolamování zabrat i minuty. Pokud by po tu dobu nebyla stránka interaktivní, bylo by to nepřijatelné. Při prolamování se pro uživatele zobrazuje zpětná vazba, takže má přehled o tom, že prolamování probíhá, nic mu však nebrání se stránkou interagovat.

Způsoby prolamování šifer a jejich úspěšnost je popsána v sekci 2.3.

3.3.6 Testy

Pro ověření a zachování funkčnosti aplikace jsou napsány testy. Nacházejí se ve složce *tests*. Zaměřené jsou především na otestování kódu šifrování a dešifrování, protože to je stěžejní složka aplikace. Opomenuty však nejsou ani testy základních komponentů.

Testy jsou napsány pomocí populárního JavaScriptového testovacího frameworku Jest[14]. Příklad testu komponentu *PlainTextArea* je uveden ve zdrojovém kódu č. 9. Testuje, zda komponent generuje událost při zadávání otevřeného textu a posílá s ní novou hodnotu. V prvním kroku je vytvořen „obal“ komponentu. Získá se z něj HTML element políčka (textarea) a vepíše se do něj nová hodnota. Následně dochází ke kontrole, že událost byla vyvolána a její parametr odpovídá nové hodnotě.

[13] https://github.com/GoogleChromeLabs/comlink-loader
[14] https://jestjs.io/

```
1  test('emits valueChanged event with value', async () => {
2    const wrapper = mount(PlainTextArea)
3    const textarea = wrapper.find('textarea')
4
5    await textarea.setValue('secretmessage')
6
7    const valueChangedEvent = wrapper.emitted('valueChanged')
8
9    expect(valueChangedEvent[0]).toEqual(['secretmessage'])
10 })
```

Zdrojový kód 9: Ukázka testu komponentu PlainTextArea

3.3.7 Nasazení

Aplikace je nasazena pomocí platformy Netlify[15], která podporuje kontinuální nasazení. Webové rozhraní platformy umožňuje jednoduché propojení GitHub[16] repozitáře s Netlify stránkou. Commit do repozitáře automaticky spouští nasazení nové verze.

4 Uživatelská část

Tato kapitola popisuje naprogramovanou webovou aplikaci z uživatelského pohledu. Pro použití postačuje základní znalost interakce s webovými stránkami. K aplikaci se přistupuje prostřednictvím webového prohlížeče zadáním URL transpozicnisifry.cz – nutnou podmínkou je, aby v prohlížeči byl aktivovaný JavaScript. Pro zajištění správné funkčnosti jsou doporučeny prohlížeče Google Chrome, Firefox, Safari nebo Microsoft Edge[17]. Aplikace je responzivní – přizpůsobuje se velikosti obrazovky zařízení.

Aplikace má jednoduchou strukturu. V horní části se nachází hlavička v tmavě šedé barvě. Součástí hlavičky je navigace. Pod hlavičkou je umístěn hlavní obsah aplikace. Úvodní stránka poskytuje základní informace o šifrování a tématu práce. Stránky, na které odkazuje navigace, se věnují jednotlivým transpozičním šifrovacím systémům – obsahují základní popis a formulář, pomocí kterého je možné šifrovat (dešifrovat) text. Poslední položka navigace odkazuje na stránku umožňující prolamování transpozičních šifer.

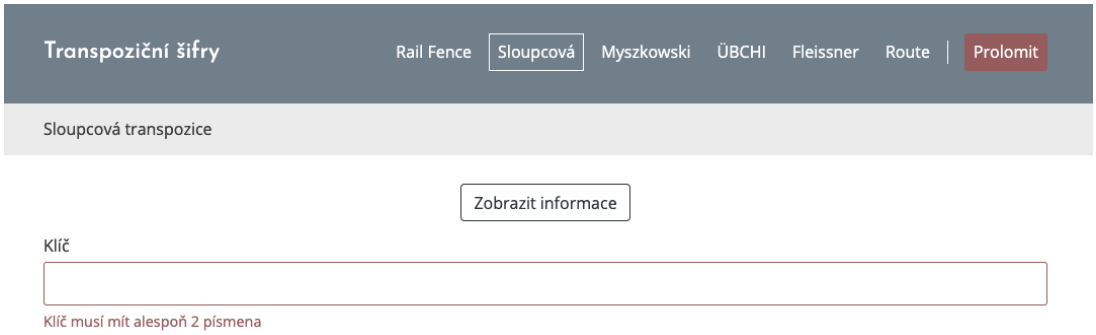
4.1 Šifrování a dešifrování

Pomocí navigace si zvolíme šifrovací systém, který chceme použít. Např. sloupcovou transpozici vybereme tak, že klikneme na odkaz *Sloupcová*. Následně se

[15] https://www.netlify.com/

[16] webová služba podporující vývoj softwaru za pomoci verzovacího nástroje Git

[17] otestován byl Google Chrome verze 89, Firefox verze 87, Safari verze 14 a Microsoft Edge verze 89



Obrázek 10: Stránka sloupcová transpozice

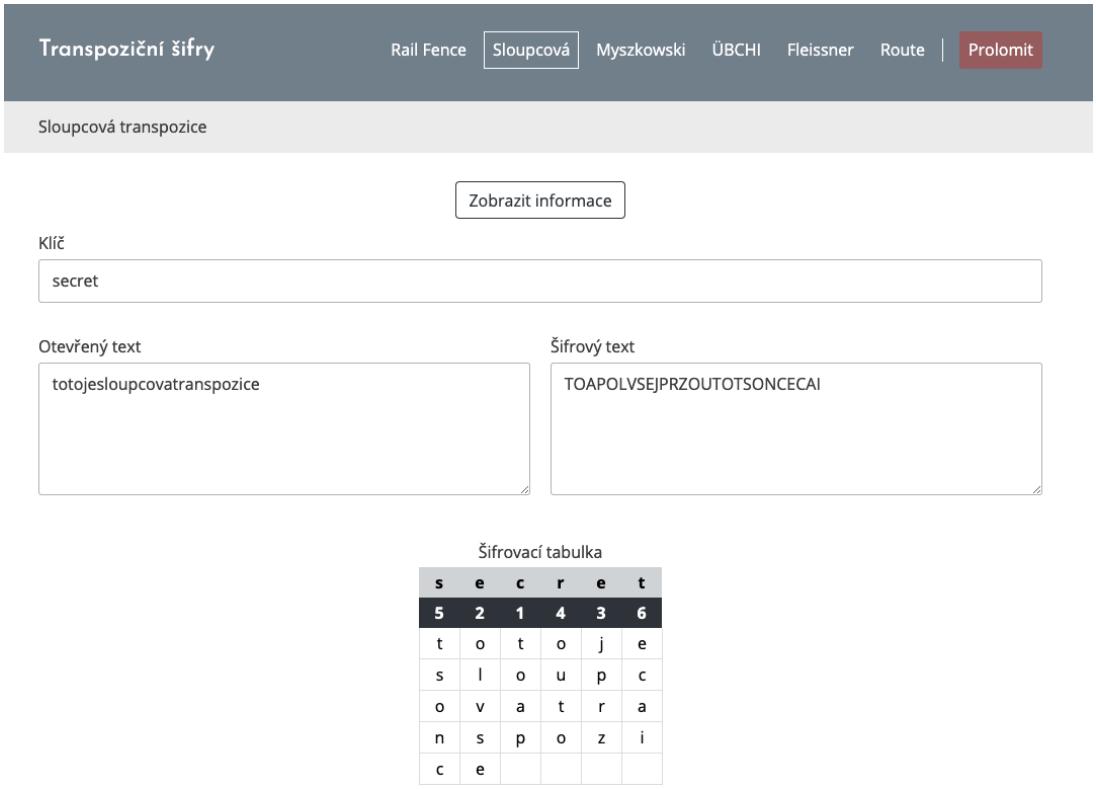
zobrazí stejná stránka jako na obrázku č. 10. Pod hlavičkou se nachází tlačítko *Zobrazit informace*. Pokud na něj klikneme, zobrazí se informace o sloupcové transpozici. Ty poté můžeme skrýt kliknutím na tlačítko *Skrýt informace*.

Pod tlačítkem se nachází formulář, který slouží k šifrování (dešifrování) zpráv. V prvním kroku je potřeba zadat klíč. V případě sloupcové transpozice musí mít klíč alespoň dvě písmena, o čemž nás informuje hláška pod polem (podobná zpětná vazba se objevuje napříč celou aplikací). Zadáme např. klíč „secret“. Jakmile bude v poli validní klíč, zobrazí se zbývající část formuláře – pole pro zadání a zobrazení otevřeného a šifrového textu. Pod formulářem se dále zobrazí *Šifrovací tabulka*, což je tabulka, která by vznikla, pokud bychom šifrovali manuálně na papíře – tento prvek je čistě informačního charakteru.

Pro zašifrování zadáváme otevřený text do pole se štítkem *Otevřený text*. Když do tohoto pole píšeme, v poli se štítkem *Šifrový text* se začne objevovat výsledný šifrový text. S každým zadaným písmenem se také průběžně aktualizuje šifrovací tabulka. Po zadání otevřeného textu „totojesloupcovatranspozice“ bude stránka vypadat jako na obrázku č. 11. Formulář by reagoval stejně na zadávání šifrového textu – v poli se štítkem *Otevřený text* by se objevoval otevřený text.

Text v polích formuláře je automaticky normalizován. Ve většině případů jsou podporována pouze písmena abecedy – výjimkou jsou pole pro klíč některých šifrovacím systémů. Pokud do pole pro otevřený text zadáme např. „toto je transpoziční šifra“, ve výsledku v něm bude „totojetranspozicnisifra“.

Stránky ostatních šifer jsou stránce sloupcová transpozice podobné. Největší rozdíly lze nalézt u Fleissnerovy mřížky a šifry Route. Při použití Fleissnerovy mřížky se nezadává klíč v podobě posloupnosti písmen, ale je třeba specifikovat mřížku. Proto je na stránce její interaktivní reprezentace. Pomocí pole *Velikost mřížky* se určí její rozměry a pak se políčka mřížky vybírají tak, že se na ně klikne myší. Políčka, která byla zvolena, jsou bílá. Červená políčka již nelze vybrat. Pro zrušení výběru stačí na bílé políčko kliknout znovu. Tímto způsobem se mřížka nastaví, případně je možné kdykoli použít tlačítko *Náhodně doplnit mřížku*. Jakmile je mřížka kompletní, objeví se pole pro otevřený a šifrový text. V tomto okamžiku je možné použít tlačítko *Resetovat mřížku* pro sestavení nové



s	e	c	r	e	t
5	2	1	4	3	6
t	o	t	o	j	e
s	l	o	u	p	c
o	v	a	t	r	a
n	s	p	o	z	i
c	e				

Obrázek 11: Ukázka šifrování sloupcovou transpozicí

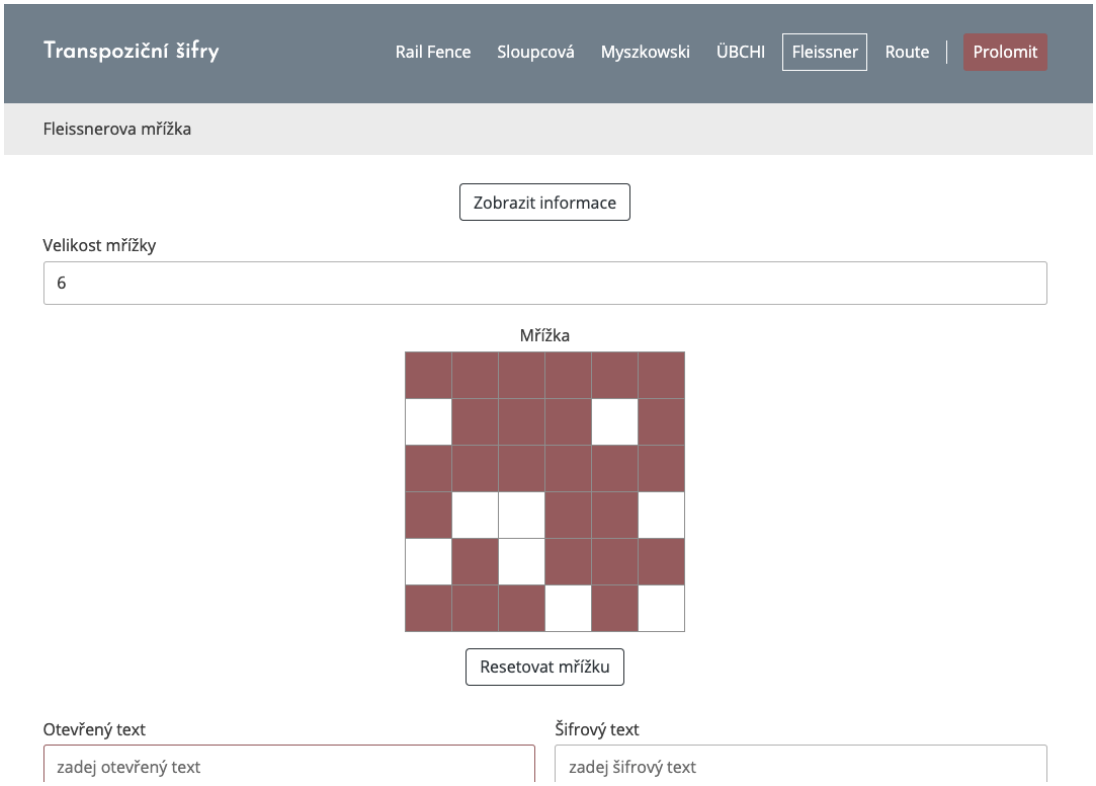
mřížky. Ukázka je na obrázku č. 12.

V případě šifry Route se volí cesta, což se realizuje kliknutím na tlačítko *Zobrazit cesty* a volbou konkrétního obrázku, který reprezentuje požadovanou cestu. Poté je již možné šifrovat (dešifrovat) text.

4.2 Prolamování

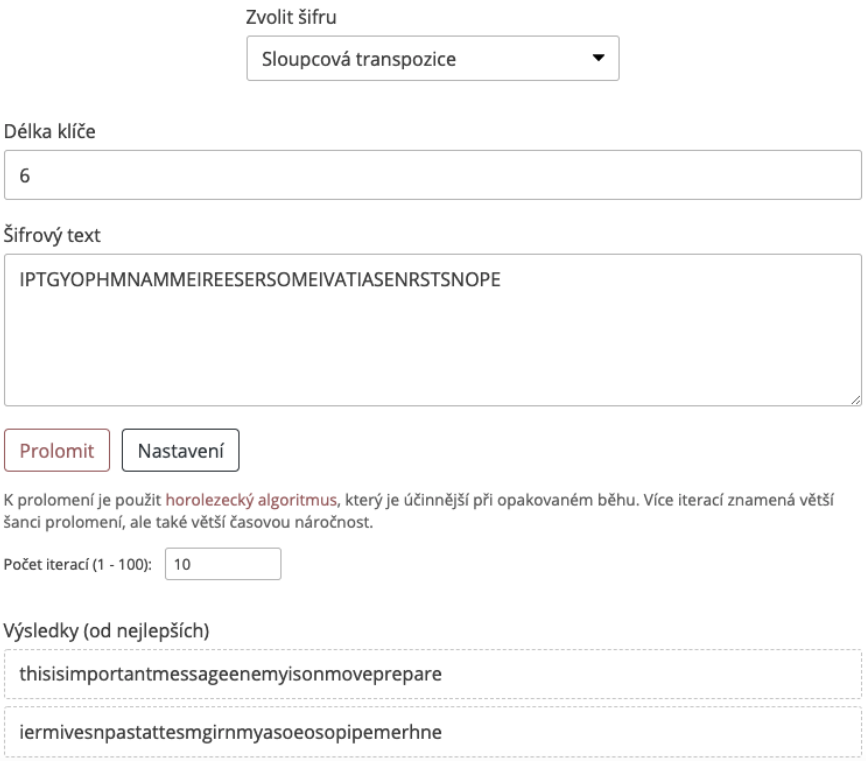
Kromě šifrování a dešifrování aplikace umožňuje transpoziční šifry také prolamovat. Pro tento účel je součástí samostatná sekce, která je přístupná pomocí odkazu *Prolomit* v navigaci. V horní části této sekce se nachází tlačítko *Zobrazit informace*, které po kliknutí odhaluje informace o prolamování – jakým způsobem jsou jednotlivé šifry prolamovány a za jakých podmínek je prolamování úspěšné. Výsledkům prolamování se podrobněji věnuje sekce 2.3. Prolomení je pro řešení situace, kdy máme k dispozici šifrový text, ke kterému neznáme klíč.

Pomocí nabídky *Zvolit šifru* se vybírá konkrétní transpoziční kryptosystém, který byl pro zašifrování zprávy použit. Dále máme pole pro zadání šifrového textu. Tlačítkem *Prolomit* proces spustíme. Pokud prolamování aktuálně probíhá, je zobrazena červená „lišta“ s nápisem *Probíhá prolamování...* Výsledky se zobrazují pod formulářem. Jak z nadpisu *Výsledky (od nejlepších)* vyplývá, jsou řazeny od nejlepších v závislosti na tom, jak moc dávají „smysl“.



Obrázek 12: Ukázka vyplnění Fleissnerovy mřížky

Prolamování nemusí být na první pokus úspěšné, proto se jej může vyplatit spouštět opakovaně. U některých šifrovacích systémů také máme tlačítko *Nastavení*, které umožňuje specifikovat počet iterací prolamovacího algoritmu. Pokud chceme větší šanci prolomení, může pomoci nastavit tuto hodnotu na vyšší číslo. Více iterací však znamená, že prolomení bude trvat déle. Můžeme zkoušet měnit i ostatní parametry – např. délku klíče. Výsledky jsou promazány pouze v případě, kdy zasáhneme do zadaného šifrového textu. Ukázka prolomení je na obrázku č. 13.



Obrázek 13: Ukázka prolomení sloupcové transpozice

Závěr

Teoretická část začíná úvodem do problematiky šifrování a dále podrobně popisuje klasické transpoziční šifrovací systémy. Čtenář je seznámen se základními pojmy kryptologie a její stručnou historií. Na příkladech je demonstrováno použití transpozice pro šifrování a dešifrování textu. Pozornost je věnována i kryptoanalýze transpozičních šifer – je prověřeno prolomení hrubou silou a pro složitější případy prolomení horolezeckým algoritmem.

V praktické části je popsán postup implementace webové aplikace, která uživateli umožňuje šifrovat a dešifrovat text pomocí transpozičních systémů a šifrový text také prolamovat. V kapitole jsou vyjmenovány použité technologie a na příkladech zdrojového kódu ukázáno jejich použití pro realizaci aplikace.

Stanovených cílů práce bylo dosaženo. Klasické transpoziční šifrovací systémy jsou důkladně popsány a naprogramovaná webová aplikace je umožňuje jednoduše používat. Předností aplikace je její jednoduchost a vysoká interaktivita.

Vypracování zahrnovalo seznámení se s webovými technologiemi a moderním způsobem vývoje webových aplikací pomocí JavaScriptového frameworku Vue.js. Do hloubky bylo také třeba pochopit samotný JavaScript, což je v současnosti velmi populární programovací jazyk.

Výzvou bylo prolamování transpozičních šifer. Vyžadovalo nastudování velkého množství materiálů a experimentování s různými postupy. Jako efektivní metoda prolamování se ukázal horolezecký algoritmus. Protože prolamování může být časově náročná operace, bylo třeba zajistit její vykonávání v dalších vláknech. JavaScript je v základu jedno-vláknový a spuštění náročné operace by stránku zablokovalo. To se podařilo vyřešit pomocí Web Workers API.

Práci a aplikaci je možné ještě vylepšovat. Existují další transpoziční systémy nebo varianty systémů, které by mohly být v práci popsány. Příkladem je dvojitá sloupcová transpozice, která pro každou transpozici používá jiný klíč. Také prolamování by se dalo zdokonalit. Mohlo by se optimalizovat i pro další jazyky, nejen angličtinu. Pro Myszkowskiho transpozici by šlo vymyslet lepší způsob hledání sousedů při prolamování horolezeckým algoritmem, aby bylo možné šifru prolomit i pro delší klíče.

Conclusions

The theoretical part begins with an introduction to the issue of encryption and further describes in detail the classical transposition cryptosystems. The reader is acquainted with the basic concepts of cryptology and its brief history. The examples demonstrate the use of transposition to encrypt and decrypt text. Attention is also paid to the cryptanalysis of transposition ciphers – breaking by brute force and for more complex cases a hill climbing algorithm is tested.

The practical part describes the process of implementing a web application that allows the user to encrypt and decrypt text using transposition systems and also to break the ciphertext. The chapter lists the technologies used and examples of source code show their use for application implementation.

The goals of the work were achieved. Classical transposition systems are thoroughly described and the programmed web application makes them easy to use. The advantage of the application is its simplicity and high interactivity.

The elaboration included acquaintance with web technologies and a modern way of web application development using the Vue.js JavaScript framework. It was also necessary to understand in depth JavaScript itself, which is currently a very popular programming language.

The challenge was breaking the transposition ciphers. It required studying a large amount of materials and experimenting with different methods. The hill climbing algorithm proved to be an effective method of breaking. Because breaking can be a time-consuming operation, it was necessary to ensure that it was performed in other threads. JavaScript is in its basic form single-threaded, and running a time-consuming operation would block the page. This was solved using the Web Workers API.

The work and application can be further improved. There are other transposition systems or variants of systems that could be described in the work. An example is double columnar transposition, which uses a different key for each transposition. Cipher breaking could also be improved. It could be optimized for other languages, not just English. For Myszkowski transposition, it would be possible to come up with a better way of finding neighbors when breaking by the hill climbing algorithm, so that it is possible to break the cipher even for longer keys.

A Obsah přiloženého CD

`bin/`
: Aplikace připravená na umístění na server. V souboru *readme.txt* je popsáno, jak aplikaci spustit lokálně.

`doc/`
: Text bakalářské práce ve formátu PDF a všechny soubory potřebné pro vygenerování tohoto dokumentu.

`src/`
: Kompletní zdrojové texty aplikace.

`readme.txt`
: Instrukce pro spuštění aplikace. Uvádí webovou adresu, na které je aplikace nasazena. Obsahuje také instrukce pro lokální spuštění aplikace.

Literatura

[1] VONDRUŠKA, Pavel. *Kryptologie, šifrování a tajná písma*. 2006. 346 s. ISBN 80-00-01888-8.

[2] PAAR, Christof; PELZL, Jan. *Understanding Cryptography: A Textbook for Students and Practitioners*. 2010. 382 s. ISBN 978-3-642-04101-3.

[3] SINGH, Simon. *Kniha kódů a šifer: Tajná komunikace od starého Egypta po kvantovou kryptografii*. 2. vyd. Praha: Dokořán, 2009. 382 s. ISBN 978-80-7363-268-7.

[4] BURDA, Karel. *Úvod do kryptografie*. Brno: Akademické nakladatelství CERM, 2015. 110 s. ISBN 978-80-7204-925-7.

[5] LOZOVSKY, Alexei. *Theory of attacks and cryptanalysis* [online]. 2020 [cit. 2021-2-2]. Dostupný z: ⟨https://docs.cossacklabs.com/themis/crypto-theory/theory-of-attacks-and-cryptoanalysis/⟩.

[6] LASRY, George. *A Methodology for the Cryptanalysis of Classical Ciphers with Search Metaheuristics*. 2018. 247 s. ISBN 978-3-7376-0459-8.

[7] WRIXON, Fred B. *Codes, ciphers & other cryptic & clandestine communication: Making and breaking secret messages from hieroglyphs to the Internet*. New York: Black Dog & Leventhal Publishers, Workman Publishing, 1998. ISBN 978-1-57912-040-5.

[8] WIKIPEDIA. *History of cryptography* [online]. 2020 [cit. 2020-11-5]. Dostupný z: ⟨https://en.wikipedia.org/wiki/History_of_cryptography⟩.

[9] WIKIPEDIA. *Cryptographie indéchiffrable* [online]. 2020 [cit. 2020-11-28]. Dostupný z: ⟨https://w.wiki/37Pj⟩.

[10] KLÍMA, Vlastimil. Utajené komunikace – 5. díl: Šifry první světové války a další rozvoj kryptologie. *Chip* [online]. 1994, vol. 9, [cit. 2020-11-29]. Dostupný z: ⟨https://cryptography.hyperlink.cz/1994/1994.html⟩. ISSN 1210-0684.

[11] GAINES, Helen. *Cryptanalysis; a study of ciphers and their solution*. New York: Dover Publications, 1956. ISBN 978-0-486-20097-2.

[12] KAHN, David. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. 1996. 1200 s. ISBN 978-0-684-83130-5.

[13] SWEIGART, Al. *Cracking Codes with Python: An Introduction to Building and Breaking Ciphers*. 2018. 416 s. ISBN 978-1-59327-822-9.

[14] LYONS, James. *Quadgram Statistics as a Fitness Measure* [online]. [cit. 2021-3-9]. online. Dostupný z: ⟨http://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/⟩.

[15] *Vue.js Guide*. [online]. 2021 [cit. 2021-3-27]. Dostupný z: ⟨https://v3.vuejs.org/guide/introduction.html⟩.

[16] *Vue Router Guide*. [online]. 2021 [cit. 2021-3-27]. Dostupný z: ⟨https://router.vuejs.org/guide/⟩.

[17] *Vue CLI Guide*. [online]. 2021 [cit. 2021-3-27]. Dostupný z: ⟨https://cli.vuejs.org/guide/⟩.

[18] MDN CONTRIBUTORS. *Web technology for developers* [online]. 2021 [cit. 2021-3-27]. Dostupný z: ⟨https://developer.mozilla.org/en-US/docs/Web⟩.

[19] *comlink-loader README.md*. [online]. 2021 [cit. 2021-3-27]. Dostupný z: ⟨https://github.com/GoogleChromeLabs/comlink-loader⟩.