

BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

BACHELOR'S THESIS

Brno, 2020

Pavla Ryšavá



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

WEB-BASED APPLICATION FOR CRYPTOGRAPHIC PROTOCOLS VISUALIZATION

WEBOVÁ APLIKACE PRO VIZUALIZACI KRYPTOGRAFICKÝCH PROTOKOLŮ

BACHELOR'S THESIS

BAKALÁŘSKÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Pavla Ryšavá

SUPERVISOR

VEDOUCÍ PRÁCE

M.Sc. Sara Ricci, Ph.D.

BRNO 2020

Bachelor's Thesis

Bachelor's study program **Information Security**

Department of Telecommunications

Student: Pavla Ryšavá

ID: 203713

**Year of
study:** 3

Academic year: 2019/20

TITLE OF THESIS:

Web-based application for cryptographic protocols visualization

INSTRUCTION:

At first, the student will study the foundation of cryptology, focusing on monoalphabetic and polyalphabetic cipher (considering also one-time pad scheme). Then, the student will study the different attack techniques that can be applied to these schemes and how information on the plain text can help in the attack.

Then, the student will implement an interface (e.g., web page) where given a plain text, different kind of aforementioned encryption schemes are provided and different attacks to the computed cipher text are interactively applied.

RECOMMENDED LITERATURE:

[1] SINKOV, Abraham; FEIL, Todd. Elementary cryptanalysis. Maa, 2009.

[2] KATZ, Jonathan, et al. Handbook of applied cryptography. CRC press, 1996.

**Date of project
specification:** 3.2.2020

Deadline for submission: 8.6.2020

Supervisor: M.Sc. Sara Ricci, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
Chair of study program board

WARNING:

The author of the Bachelor's Thesis claims that by creating this thesis he/she did not infringe the rights of third persons and the personal and/or property rights of third persons were not subjected to derogatory treatment. The author is fully aware of the legal consequences of an infringement of provisions as per Section 11 and following of Act No 121/2000 Coll. on copyright and rights related to copyright and on amendments to some other laws (the Copyright Act) in the wording of subsequent directives including the possible criminal consequences as resulting from provisions of Part 2, Chapter VI, Article 4 of Criminal Code 40/2009 Coll.

ABSTRACT

The thesis deals with the creation of an interactive web application for substitution ciphers and their interactive cryptanalysis. Six ciphers are implemented in the work. Representatives of monoalphabetic ciphers are Caesar's cipher, Atbash, and Keyword cipher and representatives of polyalphabetic ciphers are Vigenère cipher, Kryptos and Vernam cipher. Frequency analysis, index of coincidence and n-gram statistics as a fitness function are used for interactive cryptanalysis. The result is achieved by using HTML5, CSS and ReactJS scripting language which is a JavaScript library with the ability of variable type-check.

KEYWORDS

Cryptography, substitution cipher, monoalphabetic cipher, polyalphabetic cipher, Caesar cipher, Atbash, Keyword cipher, Vigenère cipher, Kryptos, Vernam cipher, cryptanalysis, frequency analysis, Kasiski's method, index of coincidence, n-gram statistics, HTMLv5, Javascript, ReactJS, web-based application

ABSTRAKT

Práce se zabývá vytvořením interaktivní webové aplikace pro substituční šifry a jejich interaktivní kryptoanalýzu. V práci je implementováno šest šifer a zástupci monoalfabetických šifer jsou Caesarova šifra, Atbaš a substituce s klíčovým slovem. Dále zástupci polyalfabetických šifer jsou Vigenèrova šifra, Kryptos a Vernamova šifra. Pro interaktivní analýzu je použita frekvenční analýza, index koincidence a n-gramová statistika jako fitness funkce. Výsledek byl dosažen za pomoci HTML5, CSS a skriptovacího jazyka ReactJS což je JavaScriptová knihovna s možností typové kontroly proměnných.

KLÍČOVÁ SLOVA

Kryptografie, substituční šifra, monoalfabetická šifra, polyalfabetická šifra, Caesarova šifra, Atbaš, Substituce s klíčovým slovem, Vigenèrova šifra, Kryptos, Vernamova šifra, kryptoanalýza, frekvenční analýza, Kasikiho metoda, index koincidence, n-gramová statistika, HTMLv5, Javascript, ReactJS, webová aplikace

RYŠAVÁ, Pavla. *Web-based application for cryptographic protocols visualization*. Brno, 2020, 66 p. Bachelor's Thesis. Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Advised by M.Sc. Sara Ricci, Ph.D.

ROZŠÍŘENÝ ABSTRAKT

Zadáním této bakalářské práce je implementace webové aplikace zaměřené na tvorbu substitučních šifer a jejich analýzu v interaktivní formě.

V první kapitole 1 se práce zaměřuje na vysvětlení principu substitučních šifer, kde je základním principem utajení zprávy za pomoci určení substitučního pravidla jako je například definice tabulky, podle které jsou písmena nebo skupina písmen z otevřené zprávy následně nahrazována. Tyto šifry se následně dělí podle způsobu šifrování. Nejzákladnějším dělením je dělení podle toho, kolik šifra používá substitučních abeced na *monoalfabetické* a *polyalfabetické*.

Monoalfabetické substituční šifry používají skrze celý proces šifrování pouze jednu substituční abecedu a jsou tak jedny z nejjednodušších ale zároveň nejméně bezpečných principů zabezpečení informací. Jsou totiž jednoduše prolomitelné už jen při použití frekvenční analýzy šifrovaného textu, protože i zde budou viditelné unikátní vzorce výskytu písmen v jednotlivých jazycích.

Mezi notoricky známé zástupce patří např. *Caesarova šifra*, kterou měl používat sám Julius Caesar ve své privátní korespondenci [46]. Principem této šifry je imaginární posun abecedy o definovaný počet znaků dopředu nebo dozadu. Například při použití klíče = 3 je písmeno **A** v celé zprávě nahrazeno písmenem **D**, písmeno **B** je nahrazeno písmenem **E** atd.

Další rozebírané monoalfabetické šifry v rámci této práce jsou *Atbaš* používající jako šifrovou abecedu obrácenou abecedu otevřeného textu a *substituce s klíčovým slovem* což je metoda, kdy je použitá abeceda modifikována heslem, které je vloženo na začátek celé abecedy – tím jsou některé (ne-li všechny) znaky přesunuty.

Polyalfabetické substituční šifry používají v procesu šifrování více šifrových abeced, které jsou na základě předem definovaného pravidla obměňovány a tím dochází k větší pravděpodobnosti, že jeden element z otevřené zprávy bude zašifrován více možnými způsoby a tudíž je potřeba více metod k následné analýze a prolomení šifrovaného textu.

Mezi jedny ze známějších polyalfabetických šifer patří *Vigenèrova* a *Vernamova šifra*. Princip Vigenèrovy šifry je střídání abeced z definované substituční tabulky, která obsahuje všechny možné posuny abecedy po jedné pozici a ve které se pohybuje za pomoci prvních písmen v řadě a sloupci (viz příloha A, kde je vyobrazen celá tabulka pro mezinárodní abecedu). Která abeceda bude použita je definováno heslem, kdy jednotlivá písmena z hesla definují substituční abecedu pro jednotlivé symboly ze zprávy, která je šifrována. Ve zprávě se postupuje po jednotlivých písmenech, stejně tak v hesle, a tudíž je každé písmeno ze zprávy šifrováno jinou abecedou.

Princip Vernamovy šifry je stejný jako u Vigenèrovy, akorát není použito heslo, ale řetězec náhodně generovaných čísel, které definují posun v abecedě a když jsou

dodržena všechna pravidla (jako např. že vygenerovaný klíč se nesmí už nikdy použít) tak je Vernamova šifra neprolomitelná.

Další polyalfabetické substituční šifry, která je v této práci dopodrobna rozebrána je Kryptos (z řeckého slova *skrytý*). Kryptos není tak úplně šifra jako spíše kryptogram vytvořený z kombinace více šifer a je vyobrazený na soše na nádvoří hlavního štábu Central Intelligence Agency (CIA) ve Virginském Langley. Kryptogram se skládá ze čtyř částí a do dnešního dne jsou prolomeny pouze první tři. V aplikaci je implementován způsob šifrování použitých u prvních dvou kryptogramů, a to je Vigenèrova šifra s použitím tabulky, kde použitá abeceda je modifikována klíčovým slovem.

V poslední podkapitole o substitučních šifrách je zmíněna i *Enigma*, což je stroj používaný za druhé světové války Německem k šifrování tajných zpráv. Principem Enigmy je totiž taky substituce definována technickým provedením stroje.

V následující kapitole 2 jsou představeny vybrané metody pro analýzu šifrovaného textu. Jde především o frekvenční analýzu, index koincidence a n-gramovou statistiku jako fitness funkce pro sestavení použitého hesla. Je zde zmíněna i Kasiskiho metoda.

Frekvenční analýza je zvláště účinná na monoalfabetické substituční šifry, jak bylo již zmíněno, protože každý jazyk má unikátní frekvence písmen abecedy a tedy na základě frekvenční analýzy šifrovaného textu je možné určit, které písmeno bylo nahrazeno kterým.

Index koincidence je používán zvláště k odhalení o jaký typ substituční šifry se jedná a jakým jazykem byla původní zpráva napsaná a je tudíž jakýmsi rozšířením frekvenční analýzy. Tento index může být použit i na výpočet pravděpodobné délky hesla.

Kasiskiho metoda je spíše takovým předchůdcem indexu koincidence. Byla hlavně používána k odhalování délky hesla na základě výpočtu společného dělitele počtu prvků mezi opakujícími se skupinami písmen.

Jako poslední zmíněná metoda analýzy je **n-gramová statistika jako fitness funkce**, která je v aplikaci implementována pro analýzu šifrovaného textu, který byl identifikován jako výstup Vigenèrovy metody šifrování. V podstatě se jedná o frekvenční analýzu definovaného n-gramu kdy je sčítána logaritmická pravděpodobnost jeho výskytu.

Ve třetí kapitole 3 jsou představeny technologie a programovací jazyky použité k implementaci webové aplikace. Jsou to: Hypertext Markup Language version 5 (HTML5), Cascading Style Sheets (CSS) a je představen i React JS což je programovací jazyk typu TypeScript což je nadmnožina jazyku JavaScript, která umožňuje typovou kontrolu kódu.

Následující kapitola č. 4 popisuje implementaci vytvořené webové aplikace za

použití zmíněných technologií. Aplikace byla vyvinuta v prostředí Visual Studio Code verze 1.45.1 v operačním systému Windows 8.1 a s použitím prázdného projektu staženého z GitHub pod licencí MIT (odkaz na projekt je pod citací [48]).

Jádrem aplikace je soubor *index.tsx* situovaný ve složce `src` kde je celá aplikace renderována a následně zobrazena do základního webového rozhraní definovaného v HTML5 souboru s názvem *intex.html* který je ve složce `public`. Základní ovládací prvky jako je vstupní a výstupní pole aplikace jsou definované v souboru *basicLayout.tsx* ve kterém je definováno i přepínání aplikačních módů *šifrování* a *analýzy* za pomoci dvou plovoucích tlačítek situovaných v pravém dolním rohu aplikace.

Jednotlivé módy mají své vlastní třídy. Základní třída pro *šifrový mód* je soubor *CipherTab.tsx* ve složce `ciphers` a základní třída pro *analýzační mód* je soubor *Analyzer.tsx* ve složce `analyzer`.

V **šifrovém módu** je na stránce zobrazen kontejner se záložkami, pod kterými se skrývají jednotlivé šifry. I ty mají své vlastní třídy pro lepší orientaci v kódu, kde mají definované i své ovládací prvky, nastavení a rozklikávací položku “ABOUT”, kde se nachází zjednodušený popis dané šifry. Všechny tyto prvky jsou následně zobrazené v záložkovém kontejneru. Jednotlivé záložky šifer jsou i barevně rozlišené na **zelené** které označují monoalfabetické zástupce a **modré** označující polyalfabetické zástupce. Tato práce implementuje webovou aplikaci se zaměřením na substituční šifry a odpovídající útoky. Aplikace byla vytvořena především jako učební pomůcka pro pochopení způsobu šifrování pomocí substitučních šifer, protože substituce je jeden ze základních kamenů používaných v pokročilejších systémech jako je např. AES a pro pochopení nejzákladnějších technik analýzy šifrových textů jako je frekvenční analýza a index koincidence.

Práce byla implementována jako webová aplikace z důvodu jednoduchého přístupu pro každého, kdo by se zajímal o substituční šifry – není potřeba nic instalovat nebo stahovat. Stačí aktuální webový prohlížeč jako např. Google Chrome, Firefox nebo třeba Opera mimo Internet Explorer na kterém nejsou nativně podporované dva JavaScript objekty a to objekt `Map` a objekt `Set`.

DECLARATION

I declare that I have written the Bachelor's Thesis titled "Web-based application for cryptographic protocols visualization" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Bachelor's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno

.....

author's signature

ACKNOWLEDGEMENT

I would like to thank to my supervisor Mrs. M.Sc. Sara Ricci, Ph.D. for professional guidance, consultation, patience and suggestions for my bachelor work. I would like to also thank my family and friends especially to my sister, Tadeáš and Lukáš for undying support and consultations.

Contents

Introduction	14
1 Substitution cipher	16
1.1 Monoalphabetic ciphers	16
1.1.1 Atbash	17
1.1.2 Caesar cipher	18
1.1.3 Keyword cipher	19
1.2 Polyalphabetic ciphers	20
1.2.1 Vigenère cipher	20
1.2.2 Vernam cipher	22
1.2.3 Kryptos sculpture	23
1.2.4 Enigma	25
2 Cryptoanalysis of substitution ciphers	26
2.1 Frequency analysis	26
2.2 Index of coincidence	28
2.3 Kasiski's method	29
2.4 N-gram fitness measure	29
3 Web interface	32
3.1 HTML5 (Hypertext Markup Language version 5)	32
3.2 CSS (Cascading Style Sheets)	33
3.3 JavaScript and TypeScript	34
4 Application implementation	36
4.1 Installation and start up	37
4.2 Application development	38
4.3 Application visualisation	43
4.3.1 Section of the substitution ciphers	44
4.3.2 Section of the cryptoanalysis	48
Conclusion	53
List of abbreviations	62
A Tabula recta (Vigenère tableau)	63
B Application directory tree	64

List of Figures

1.1	Atbash cipher example	17
1.2	Caesar cipher encryption example	18
1.3	Derivation of alphabet with password	19
1.4	Encryption example by Keyword cipher	20
1.5	Alberti cipher disk	20
1.6	Vigenère cipher running key encryption example	21
1.7	Vigenère cipher autokey encryption example	21
1.8	Different readings from tabula recta	22
1.9	Vernam cipher encryption example	22
1.10	Kryptos sculpture	24
1.11	Kryptos encryption example	24
1.12	Photo of Enigma machine	25
4.1	Successful application compilation	38
4.2	Simple application diagram	41
4.3	Division of the application	44
4.4	Detailed Atbash settings	45
4.5	Detailed Caesar settings	45
4.6	Detailed Keyword settings	46
4.7	Detailed Vigenère settings	46
4.8	Detailed Kryptos settings	47
4.9	Detailed Vernam settings	47
4.10	Detailed view of frequency table	48
4.11	Detailed substitution menu	49
4.12	Detailed overview table	50
4.13	Detailed Is this Vigenère cipher?	51
C.1	Analysis mode controls	65
C.2	Cipher mode controls	66

List of Tables

1.1	Numerated alphabet order	18
1.2	Cesar cipher alphabet shift	19
1.3	Keyword substitution table	19
2.1	Frequency result comparison based on source size	27
2.2	English letter frequency	27
2.3	Language index of coincidences	28
2.4	N-fitness text preparation	30
A.1	Tabula recta	63

Listings

3.1	Example of CSS document	33
3.2	Example of HTML file with link to CSS document	34
3.3	Example of JSX syntax in ReactJS	35
4.1	Example of JSX syntax in ReactJS	39
4.2	Fragment of CSS document from the implemented application	40

Introduction

Privacy and secrecy of personal information always played an important role in the history and nowadays, in the “Age of the Internet”, it gains even higher importance, because for earning these information, one does not need to make such effort. All that is needed is connection to the Internet.

That is why diverse ciphers were invented. To understand the complex ciphers as DES, AES or RSA it is important to understand the simplest ones, because they include functions based on the idea of these primary ciphers, e.g. usage of secret keys, substitution etc. That is also why substitution ciphers can be considered as the ancestors of *stream ciphers* (i.e. ciphers using XOR function for encryption, e.g. A5/1 used for mobile phone communication) or *block ciphers* (i.e. ciphers operating above a group of characters with fix length, e.g. above mentioned DES or AES).

First and most known way how to encrypt plaintext is with the substitution method. In fact, the monoalphabetic ciphers came to be for this purpose. However, this method was not very secure. Even in the antiquity people were able to decipher the content. Because of this, inventors began to add complexity into their ciphers, e.g. usage of secret keys to somehow shift the used alphabet. This process caused the creation of polyalphabetic ciphers.

The thesis focuses on the implementation of several substitutions ciphers and show how they can be attacked. This is done by the creation of a web application where the user can interactively choose which cipher to use and how the attack should be carried out. Goal of this bachelor work is to implement three monoalphabetic, and three polyalphabetic substitution ciphers and interactive attacks on chosen ciphers using HTML5 and JavaScript. Chosen monoalphabetic ciphers are Atbash, Caesar cipher and Keyword cipher. I case of polyalphabetic ciphers we consider are Vernam cipher, Vigenère cipher and Kryptos (I. and II. part). Moreover, the cryptanalysis consists on the implementation of frequency analysis, index of coincidence and n-gram fitness password construction.

This is achieved by developing an interactive web application. We consider a web-based method since is a widely used way and can easily reach users without the necessity of being either downloaded or installed.

For the application creation was used HTML5 (Hypertext Markup Language version 5), CSS (Cascading Style Sheets) and JavaScript library ReactJS. ReactJS was chosen because it is so called “*TypeScript*”, which is actually JavaScript which has the ability to check variable type. For the GUI (Graphical User Interface) implementation was used Material-UI library in combination with CSS for deeper customization of the predefined components. HTML5 was used just to create basic environment for the ReactJS application. Application itself was implemented so

that the user could play also with the settings of each cipher and that the process of deciphering would not be automatic. User has to try different settings to uncover the original message. For these reasons this application can be used as a teaching tool, e.g. it can be used for demonstration of the encryption progress.

Sections 1 and 2 contain necessary theory for understanding of the problematic – acquaintance with substitution ciphers with special focus on the chosen ciphers and in the Chapter 3 is a short introduction to web interface and to used technologies to create and implement the application (HTML5, CSS and ReactJS).

Section 4 contains the description of the application implementation. First how to install and boot up the application is described. Then there is an inside view into the application itself – how does the Graphical User Interface looks like, what are the functions behind and how is the whole interface controlled.

This thesis also includes three Appendixes:

- Appendix A that contains full Vigenère substitution table A (alternatively so called *tabula recta*),
- Appendix B contains diagram of the application directory tree for better visualisation of the code structure, e.g. how the application is coordinated in within the directories, what each file does etc., and
- Appendix C that contains description of the application controls.

1 Substitution cipher

Substitution ciphers belong to the *symmetric cryptography*. Marking “symmetric” is because ciphers belonging to this category use same secret key during encryption and decryption. In the encryption process is somehow used of the key to hide original message (so called *plaintext*) and then in order to achieve this plaintext from the created *ciphertext* (encrypted message), is then the process of the encryption inverted.

The principle of substitution ciphers is to encrypt plaintext to ciphertext by replacing (*substituting*) a certain element (e.g. letter or a group of letters) with another element by defined rules.

Substitution ciphers have two ways how to differ them. First, they can be divided by the amount of elements that are substituted during the encryption process on *simple*, *homophonic* and *polygraphic substitution ciphers*. In polygraphic substitution cipher, the algorithm operates above a group of elements (many-to-many) while in simple substitution algorithm operates above only one symbol (one-to-one). To simple substitution ciphers belongs e.g. Morse alphabet or pigpen cipher and an example of polygraphic substitution cipher is Playfair cipher that was the first encrypting two letters together. *Homophonic cipher* uses one-to-many mapping based on the frequency of used elements. That is one symbol (e.g. number) represents a group of symbols (e.g. letters **Q**, **W** and **X**) because they appear scarcely in the message and during the decryption it is then not hard to appoint the correct letter.

Another way how to differ substitution ciphers can be by whether the elements are replaced by the same element in the whole process of encryption or not. In this case, they are split in *monoalphabetic* and *polyalphabetic*.

A special substitution ciphers are the **nomenclators** and **codes**, which is some sort of an “extension” of the nomenclators. Both use large homophonic substitution tables usually containing graphical representatives mapped to letter, words or even entire phrases. Example of a nomenclator can be for example The Rossignols’ Great Cipher used during the reign of Louise XIV [29].

1.1 Monoalphabetic ciphers

Monoalphabetic ciphers use only one cipher alphabet that does not change through out the whole process and thus are the simplest substitution ciphers but also very insufficient in secrecy. Even if one randomly assigns all letters from international alphabet, the number of all combinations is 4.033×10^{26} thus created cryptogram can be easily attacked by frequency analysis and can be broken without the knowledge of the used secret key.

One of the first description of monoalphabetic ciphers appeared in *Kámasútra* (book from 4th century B.C.) but the concept is probably even older. The reason, why is substitution cipher mentioned in the “Book of Love” is because author recommends especially to woman to learn to encrypt their messages so they could write love letters to their lovers [14].

Mapping of the plaintext element to ciphertext element can be defined by algorithms or by substitution tables where is defined by what will be element substituted by. For example defined substitution table replaces letter **A** by letter **J** and letter **N** by letter **K**. Now if one would like to encrypt name “ANNA” by this table the result would be “JKKJ”.

1.1.1 Atbash

Atbash (also *Temurah* or *Kabbalah*) is a simple monoalphabetic substitution cipher originally used for Hebrew alphabet. The transliterated name “*Atbaš*” is formed from first, last, second and penultimate letter from the Hebrew alphabet – *Aleph-Taw-Bet-Shin*.

The principle of this cipher is that each letter from plaintext alphabet is substituted by its alphabet counterpart. For instance in international alphabet the letter **A** is substituted by letter **Z**, letter **B** by letter **Y**, etc. Therefore, the encryption process can be described as in the following formula 1.1,

$$c = N - m + 1 \tag{1.1}$$

where c is the ciphertext letter position in alphabet, m is the plaintext letter position in alphabet and N is the total amount of letters in used alphabet (in international alphabet it is equal to 26) [55].

MESSAGE:	HELLO WORLD!	ALPHABET:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
CIPHERTEXT:	SVOOL DLIOW!	NEW ALPHABET:	ZYXWVUTSRQPONMLKJIHGFEDCBA

Fig. 1.1: Example of plaintext encryption by Atbash cipher.

This cipher was probably invented by small Jewish and is also used in some passages in Bible, for instance in Jeremiah 51:1¹ where the word “Leb-kamai” translated in English as “The Midst of Those who rise up against Me” is actually encrypted in Atbash. When the word “Leb-kamai” is deciphered, one gets the word “Chaldea” that is a country that existed that time and was part of Babylonia [27].

¹Jeremiah 51:1: “*This is what the Lord says: ‘See, I will stir up the spirit of a destroyer against Babylon and the people of Leb-Kamai.’*” (Bible, New International Version).

Another interesting myth with the Atbash cipher, was the case of the worship of so called “Baphomet” by the Templar knights (also known as “Sabbatic Goat”). Though, when Atbash is used the world changes to something, that can be represented as Greek word *Sophia* (*wisdom* in English) [32].

1.1.2 Caesar cipher

Caesar cipher is one of the most known representative of the monoalphabetic ciphers. As the name hints, this cipher is named after Julius Caesar, who used it to encrypt his private correspondence (according to biography “*De vita Caesarum*” written by Gaius Suetonius Tranquillus [46]).

The principle of this cipher is very simple. It uses a secret key – number that shift the set of used elements (in this case alphabet) in some direction (forwards or backwards) as shown in the Figure 1.2. Shifted alphabet with $k = 3$ with corresponding plaintext letters can be seen in Table 1.2.

PLAINTEXT :	HELLO WORLD!	ALPHABET :	ABCDEFGHIJKLMNOQRSTUVWXYZ
KEY :	+3 (A → D)	NEW ALPHABET :	DEFGHIJKLMNOQRSTUVWXYZABC
CIPHERTEXT :	KHOOR ZRUOG!		

Fig. 1.2: Example of plaintext encryption by Caesar cipher.

Since each letter of the alphabet can be represented by a number, the whole process of encryption could be described as shown in 1.2

$$E(M) = (m + k) \bmod n = C \tag{1.2}$$

where M represents the message in plaintext, m is the number representing element from the message, k the chosen secret, n the amount of elements in the used alphabet (e.g. for regularly used alphabet - A to Z - it would be 26) and C is then the received ciphertext, seen in Table 1.1.

Tab. 1.1: Corresponding numbers to letters in English alphabet.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

The process of decryption could be mathematically described as shown in 1.3

$$D(C) = (c + k) \bmod n = M' \tag{1.3}$$

where C represents the ciphertext, c is the number representing ciphertext element, and M' is then the decrypted message [54].

Tab. 1.2: Example of Ceasar alphabet shift with $k = 3$.

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

1.1.3 Keyword cipher

This section presents another substitution cipher which is a bit more sophisticated with respect to Caesar one. The Keyword cipher uses two different methodologies: shifting the letter as Caesar cipher and the usage of a secret keyword (password) as a permutation. In particular, the chosen password is inserted at the beginning of the alphabet. Then the alphabet is browsed from the start and all letters that had already occurred are erased i.e. all the doubles are erased.

The creation process of the alphabet (so called *transposition*) is depicted in the Figure 1.3.

PASSWORD:	SECRET	
ALPHABET:	ABSDEFGHIJKLMNOPQRSTUVWXYZ	
INSERTION:	SECRET ABCDEFGHIJKLMN OP QRSTU VWXYZ	// doubles
ERASURE:	SECRET ABDFGHIJKLMN OP QU VWXYZ	

Fig. 1.3: Transposition of the alphabet with the password “SECRET”.

The encryption is then equal to the Caesar cipher process. Letters are then during the process of encryption replaced by the corresponding letters from the derived alphabet. Table 1.3 shows the created substitution table with the method explained above.

Tab. 1.3: Substitution table for the Keyword cipher with password “SECRET”.

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

Figure 1.4 shows an example of the Keyword cipher encryption process with the chosen password “SECRET”. The decryption process is just inverted encryption algorithm using the same derived alphabet (shifted with the same password).

PLAINTEXT:	HELLO WORLD!	ALPHABET:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
KEY:	SECRET	NEW ALPHABET:	SECRET ABDFGHIJKLMNOPQUVWXYZ
CIPHERTEXT:	DTIIL WLOIR!		

Fig. 1.4: Keyword cipher with password “*SECRET*” encryption example.

1.2 Polyalphabetic ciphers

Polyalphabetic ciphers use more than one alphabet through the encryption process. These alphabets are usually derived from the used alphabet more than once. Actually if it is looked from a different angle, polyalphabetic ciphers contain multiple different monoalphabetic substitutions. These ciphers were invented as a reaction on the low security level of the simple substitution ciphers. In fact, monoalphabetic substitution ciphers can be decrypted with just a paper, pen and one hour of spare time.



Fig. 1.5: Alberti cipher disk [6].

The invention of the first polyalphabetic cipher is attributed to Italian architect Leon Battista Alberti. The *Alberti cipher* was described in his treatise *De componendis cifris* in 1466. This cipher uses different mixed alphabets, variable periods of usage of given mixed alphabet and it also includes four positions, that can be used as nulls within text. The complexity of the algorithm also led to the invention of the so-called *Alberti cipher disk* depicted in Figure 1.5, which consists of two attached concentric disks that can rotate one with respect to the other and thus, the encryption and decryption process of the Alberti cipher was simplified [10].

1.2.1 Vigenère cipher

Vigenère cipher was first introduced by Giovan Battista Bellaso in his book *La cifra del. Sig. Giovan Battista Bellaso* published in 1553. This cipher was then misattributed to French cryptographer Blaise de Vigenère, that had in 1586 presented a similar cipher known as *autokey* or *autoclave cipher* [52]. Since then, the Bellaso’s cipher is known as *Vigenère’s cipher*.

Vigenère cipher uses passwords for the determination of the alphabet derivation through the process and thus uses multiple monoalphabetic ciphers [34]. There are two modes of passwords: *running key* and *autokey*.

PLAINTEXT:	H E L L O W O R L D!
PASSWORD:	B C D A B C D A B C . . .
<hr/>	
CIPHERTEXT:	I G L O P Y R R M F!

Fig. 1.6: Example of plaintext encryption using *running key* Vigenère cipher.

In the **running key** variant (or the “basic” variant), alphabets are derivated through position of a letter in plaintext alphabet from given password. It is actually Caesar cipher, where each element of the plaintext, that is being encrypted, has different key based on the password. When one gets at the end of the password and the plaintext is not yet fully encrypted, the password is “restarted” which results in password repetition. Encryption example is depicted in Figure 1.6.

In the **autokey** variant (also known as *autoclave cipher*) is the chosen password used only once and the rest of the cipher is encrypted using the plaintext itself [7]. Example is shown in Figure 1.7.

PLAINTEXT:	H E L L O W O R L D!
PASSWORD:	P R E F I X
USED PASSWORD:	P R E F I X H E L L O W O R L D // unused
<hr/>	
CIPHERTEXT:	W V P Q W T V V W O!

Fig. 1.7: Example of plaintext encryption using *autokey* Vigenère cipher.

Table that holds all derivated alphabets and it is called “*tabula recta*”. This table is shown in Appendix A. This term was first used, and thus invented, in book *Polygraphiae*² by German monk Johannes Trithemius who used it within a cipher of his own – *Trithemius cipher*. The only difference between this cipher and Vigenère is, that Trithemius cipher uses the alphabets in the order as they are written in (thus password is “ABCD...ZABC...”).

There are also different ways of reading from the tabula recta resulting in differently named ciphers. One is **Beaufort cipher** which was invented by Irish admiral sir Francis Beaufort and the other is **Variant Beaufort cipher**. For better orientation between readings, all methods assigned to each cipher (Vigenère, Beaufort and Variant Beaufort) are shown in Figure 1.8 □.

²Scan of this book is freely available online on web page of **Library of congress**. [50]

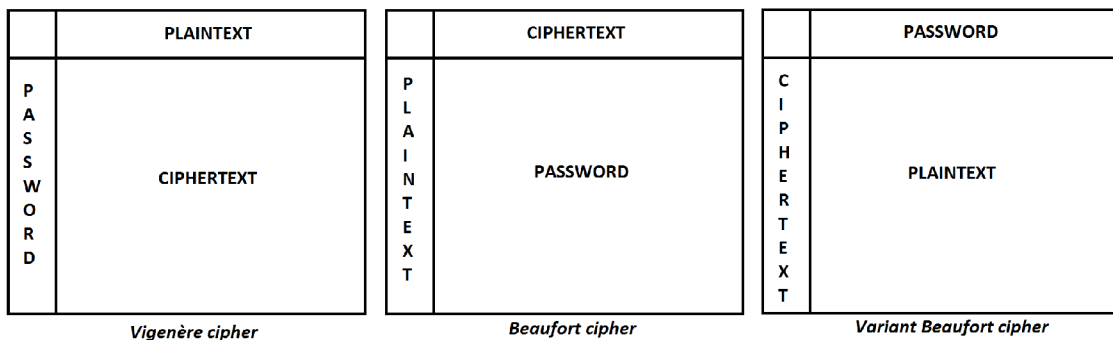


Fig. 1.8: Differences between readings from tabula recta resulting in different ciphers.

1.2.2 Vernam cipher

Vernam cipher (also known as the *one-time pad*) is the safest substitution cipher ever invented. The encryption algorithm of this cipher is similar to Vigenère cipher (see Chapter 1.2.1 for more details) with one major difference: the key used during encryption is *randomly generated* in the length of the message or longer. Example of such encryption is shown in Figure 1.9.

```

MESSAGE:   H E L L O W O R L D!
KEY:       10 8 16 10 7 24 13 5 6 7           // randomly generated key
           // H will be ciphered as if using Caesar cipher with key - 10, E with key - 8 etc.
-----
CIPHERTEXT: R M B V V U B W R K!

```

Fig. 1.9: Example of plaintext encryption by Vernam cipher.

The proof of the ciphers secrecy was mathematically proven by mathematician Claude Elwood Shannon in his paper “A Mathematical Theory of Communication” (commonly known as *information theory*) in 1948 where the key to all is entropy. The theory claims that the ciphertext gives no additional information about the plaintext if certain rules are applied [38].

In order for Vernam cipher to be truly safe one needs to abide four rules during encryption:

- password used for encryption is **randomly generated**,
- length of the password is **of the same length** as the plaintext or longer,
- password is used for the encryption **only once** and
- there should be maximally two copies of the key and both should be destroyed immediately after use [36].

Of course there are multiple problems. At first most of the commonly accessible random generators are *pseudorandom*, which mimic random generators using math-

ematical formulas. Unluckily, these pseudo-generated numbers can be predetermined [37]. In order to ensure true secrecy a *true random* generator must be used but these generators first use hardware device in order to generate a random value and second they are significantly slower than pseudorandom generators [56].

At second the key needs to be safely distributed. In fact, if an attacker gets the key then it is equal to sending the message to receiver in plaintext. And at third, there is also problem with the message authenticity. However the manipulation of the ciphertext by attacker is possible only if attacker knows the content of the original message [39].

The cipher was first described by American cryptographer Frank Miller in 1882. 35 years later it was reinvented by Gilbert Standford Vernam who had patented his binary version of the cipher in 1919 [4]. In particular, the binary variant uses the *Boolean "exclusive or" function* (short XOR). The encryption process is then the plaintext converted to binary form XORed to the bit-stream key again, randomly generated. This is also why this cipher is also a representative of the *stream ciphers* (ciphers where input is combined, usually by XOR function, with random bit-stream key).

1.2.3 Kryptos sculpture

Kryptos sculpture comes from the workshop of Jim Sanborn – a Washington, D.C. born artist. This two-part “S” shaped sculpture was installed in 1990 and holds four cryptograms (photo can be seen in Figure 1.10). To this day are known only three messages (deciphered in 1999) written on the plastic standing at the entrance to the Central Intelligence Agency (short CIA) in Langley, Virginia [28].

Sanborn accepted the advise of Edward M. Scheidt, a retiring Chairman of CIA Cryptographic Center and during four months he created four ciphertexts by combining different techniques [2] that are know just by Jim Sanborn and William Webster³ [9].

The cryptogram cut through the copper plate in the first half of the sculpture is divided into four parts by question marks. The second part of the sculpture is a modified Vigenère substitution table (so called *tabula recta*, see Appendix A) using international alphabet transposed by password “KRYPTOS” (see Chapter 1.1.3 for reminder of alphabet transposition).

The first and the second cryptograms of the Kryptos are encrypted with the Vigenère cipher using the modified Vigenère substitution table created with the transposed alphabet by password “KRYPTOS” (see Chapter 1.2.1 for reminder of the encryption process). The passwords used for Vigenère cipher itself are in the

³That day head of the CIA and to whom had Jim Sanborn provided a key to deciphering.



Fig. 1.10: Kryptos sculpture by Jim Sanborn at the CIA Headquarters Langley, Virginia [40].

first part “PALIMPSEST” and in the second part “ABSCISSA”. An example of encryption can be seen in Figure 1.11.

The third ciphertext is more complicated. It again uses word “KRYPTOS” as the key, but the encryption process uses first Route Transposition followed by Key Columnar Transposition [45]. Since the application that is the result of this Bachelor work implements just the method used within first and second part of the cryptogram, this part will not be described in detail.

PLAINTEXT:	HELLO WORLD!	
VIGENÈRE KEY:	PALIMPSEST	USED ALPHABET: KRYPTOSABCDEFGHIJLMNQVWXZ
<hr/>		
CIPHERTEXT:	LMBSW KEFWH!	

Fig. 1.11: Example of Kryptos 1st part encryption with password “PALIMPSEST”.

And what do the deciphered messages contain? The first message is on a philosophical base composed by Jim Sanborn himself. It reads: “Between subtle shading and the absence of light lies the nuance of iqlusion.”. The misspelling of *illusion* at the end was intentional according to author to make the deciphering more difficult. The second bears hints at something buried, holds the location of CIA by latitude and longitude and also contains reference on William Webster, who is supposed to know the exact location of the buried mystery on which the message refers to. The third part contains a section from diary of archaeologist Howard Carter, that describes the opening of King Tutankhamen’s tomb on 26th of November 1922 again with few misspellings.

Since the last part is still unknown Jim Sanborn released three clues to this day: “BERLIN” (2010), “CLOCK” (2014) and “NORTHEAST” (2020) [41].

The “celebrity” within the Kryptos fandom is a game developer and cryptologist Elonka Dunin, that maintains a web-page of background information on the Kryptos and an online community of Kryptos fans. Her web-page can be found under Reference [8]. Another personality in the Kryptos community is for instance Craig Bauer who wrote book *Unsolved!* where the first three parts of Kryptos ciphertext are described.

1.2.4 Enigma

Enigma (or precisely *Enigma machine*, pictured in Figure 1.12) was used during the World War II by Germans to encrypt their internal communication (e.g. information about the movement of the military force or diplomatic communication).

The first machines were independently developed in the period from 1917 to 1921 in several countries, but these “Enigmas” were meant for commerce usage.

The construction of Enigma is very complex. The machine consists of five rotors, panel of bulbs, keyboard and since 1930 also a plugboard. This upgrade increased the amount of possible key combinations (2^{75} of combinations).

Introductory settings of the machine played an important role. In the period 1930 to 1938 German army used following settings to determine the daily key:

- order of three middle rotors inside of the machine,
- plugboard connections,
- initial positions of rotors and
- position of rings that sets reflecting positions of first and last rotors [18].



Fig. 1.12: Enigma Machine at the Imperial War Museum, London [44].

2 Cryptoanalysis of substitution ciphers

Substitution ciphers are nowadays easy to decrypt with the exception of Vernam cipher as long as all the rules described in Chapter 1.2.2 are followed otherwise, there is a slight chance of breaking the ciphertext. These days there are multiple ways how to break the encrypted messages – from *brute-force* to complex mathematical analysis for gaining information. But basically all methods used for ciphertext analysis are the most effective upon a larger amount of data – the bigger the examined sample is, the more information about the ciphertext is gained and thus the decryption process is easier.

In this chapter will be described multiple ways how to analyse ciphertext to achieve as much as possible information to break the cryptogram especially those, that have been used or studied for the application implementation.

2.1 Frequency analysis

Frequency analysis is a very simple analysis method that compares occurrences of all elements in the ciphertext to occurrences of elements in a large sample of plaintext data sourced for instance from books or speeches. This method is based on unique patterns of occurrences of letters in given language. Examined elements during the frequency analysis can be single or grouped letters for example two letters (so called *bigrams*), three letters (so called *trigrams*) – in general so called **n-grams**. The bigger is the n-gram the easier is the analysis, but also: the higher n the bigger is the database of permutations of stated n-gram.

If one has the knowledge of in which language is the original plaintext written in and the cryptogram is identified as *monoalphabetic substitution cipher* (on polyalphabetic and homophonic ciphers must be used different techniques), then next steps can reveal, what text is hidden in the ciphertext including what key is used during encryption in case of for instance Caesar cipher or transposed alphabet by password cipher.

Example of reference frequency database can be seen in Table 2.2. The size of the sample plays an important role. For instance, William F. Friedman had published in his *Military Cryptoanalysis* an comparison of 75.000, 100.000 and 136.257 letters long source frequency analysis. The differences can be seen in Table 2.1.

Once one have the frequencies of each element in ciphertext can one determine what is used to represent each letter from the original message. Already this step can, for instance, provide the key for Caesar cipher. However, for more sophisticated ciphers the analysis can go on and, e.g. one can then take the results from the analysis of the most used words and decide weather in the analysed sample are not

Tab. 2.1: Frequency analysis result comparison based on the size of the text sample [16].

	Source size	Result
1.	75.000 letters	E T R N I O A S D L
2.	100.000 letters	E T R I N O A S D L
3.	136.257 letters	E T R N A O I S L D

similar structures of words. For instance, the most used word in English is “the”, in the analyzed sample there is ciphered word “AOL”, and from the previous analysis was found, that letter **L** represents letter **E** – from this knowledge can be deduced, that letter **A** represents **T** and letter **O** represents **H**).

Of course result of the frequency analysis can be distorted when applied on a short text. For instance in the given text “HELLO WORLD!” is most common letter **L** and if it would be substituted by any letter it would be mapped to letter **E** because of its highest occurrence in the ciphertext (while letter **E** is the most common letter in English – see Table 2.2).

Tab. 2.2: English letter frequency based on a sample of 40.000 words [12].

Letter	Frequency (%)	Letter	Frequency(%)
E	12.02	M	2.61
T	9.10	F	2.30
A	8.12	Y	2.11
O	7.68	W	2.09
I	7.31	G	2.03
N	6.95	P	1.82
S	6.28	B	1.49
R	6.02	V	1.11
H	5.92	K	0.69
D	4.32	X	0.17
L	3.98	Q	0.11
U	2.88	J	0.10
C	2.71	Z	0.07

In extreme cases the frequency pattern can be deliberately adjusted to make the analysis harder or even impossible. An excellent example in this case can be book *La Disparition* by Georges Perec written in 1969 in which the occurrence of letter **E** is omitted except for the name of the author. In 1995 was the book translated to English as *A Void* by Gilbert Adair under the same constraints [15].

2.2 Index of coincidence

The index of coincidence is another method used to mine information from ciphertext. This method was first published in 1922 in a technical paper *The Index of Coincidence and its applications in cryptanalysis* by William F. Friedman that was written as a further study of a concept of coincidences. The index of coincidence detects how likely is to draw two same characters from text if randomly chosen. It can be summarized by the following formula 2.1:

$$IC = \frac{\sum_{n=1}^c n_i \cdot (n_i - 1)}{N \cdot (N - 1) \cdot \kappa_r} \quad (2.1)$$

where N is the total amount of letters in ciphertext, c the number of letters in the alphabet (in international alphabet it is 26), n_i the number of appearance of each letter (where i identifies the letters position in the alphabet) and κ_r is the **kappa-random** (the *normalizing denominator* of the index) and is counted as $\kappa_r = \frac{1}{c}$.

Sometimes are the results reported without the normalizing denomination. Then the result is rather referred to as **kappa-plaintext** (κ_p) [33].

The IC can be used to gain information about **language** and **type of substitution cipher** (whether the cipher is monoalphabetic or polyalphabetic). Both information are gained based on the assimilation to pre-calculated values. For instance in case of the *type detection* will the IC for monoalphabetic substitution ciphers oscillate around $\kappa_p = 0.07$ and higher and for polyalphabetic will be the value declining to κ_r . If the index reveals that the ciphertext was created by simple substitution it can also reveal in what *language* was the prime plaintext written in [13]. Some values are presented in Table 2.3.

Tab. 2.3: κ_p results for various languages [3].

Language	IC	Language	IC
<i>English</i>	0.0667	<i>Swedish</i>	0.0681
<i>French</i>	0.0694	<i>Polish</i>	0.0607
<i>German</i>	0.0734	<i>Danish</i>	0.0672
<i>Spanish</i>	0.0729	<i>Icelandic</i>	0.0669
<i>Portuguese</i>	0.0824	<i>Finnish</i>	0.0699
<i>Turkish</i>	0.0701	<i>Czech</i>	0.0510

When the ciphertext is qualified as a polyalphabetic cipher encryption result, then the index is used for password length computation. In general in the polyalphabetic substitution cipher each n th, $2n$ th, $3n$ th, etc. letter fall in the same cipher alphabet [17]. When is counted the average index of coincidence for each probable

password length (i.e. IC for each position in password through out the whole ciphertext divided by the password length) then can be determined which password length was probably used. The one that has the closest result to the “general” IC (the one counted above the whole ciphertext) would be the most probable. Once again, the precision of the result depends on how large the ciphertext is [5].

2.3 Kasiski’s method

Kasiski’s method was a predecessor of the index of coincidence (see previous Chapter 2.2 for reminder). It was first described by Charles Babbage in 1846 in an unpublished paper, but in 1863 German military officer Friedrich W. Kasiski published a book *Die Geheimschriften und die Dechiffrierkunst* (in English *Cryptography and the Art of Decryption*) thus the method was named after him [42].

The principle of the Kasiski’s analysis is search of repeated keywords based on the fact that identical plaintext encrypted by identical key results in the same cryptogram. It is also expected that the number of characters in between the repeating groups is a multiple of the key length thus it would be possible to compute Greatest Common Divisor (short GCD) of all these “spaces” to find the probable length of the used key. The most useful repetitions are groups consistent of more than four letters. Of course, in the ciphertext can also appear coincidental repetitions that makes the analysis harder, however the analysis of all encountered “spaces” is still possible because probability, that in the text will occur more than one group of four and more letters, that does not come from identical plaintext and password is almost impossible.

Once the letters in between each identical group are counted divisors for each gained values are found. Divisors 1 and 2 are usually omitted from considerations, because password of length 1 is equal to Caesar cipher, so it is usually already discovered by performing the frequency analysis (see Chapter 2.1 for reminder) and passwords of length 2 are insufficient thus they are not commonly used. Then “gaps” whose divisors differ too much are again omitted from consideration and from the rest is counted the GCD for the most probable password length¹ [33].

2.4 N-gram fitness measure

For recapitulation n-gram is a continuous group of n elements from one defined group. In cryptoanalysis the n-gram frequency databases can be used for a fitness

¹Sometimes happen, that the GCD is bigger than the real password length but this only results in multiplication of the password in the final length of GCD.

function when searching for a password used in the polyalphabetic substitution ciphers. The whole process is build on a logarithmic probability and on the use of the n-gram statistical database.

The first step in the process of recreation of the password is to find the probable password length. This can be obtained for instance by the Kasiski’s method (see previous Chapter 2.3 for reminder) or one can try to count the average index of coincidence for each password length (see Chapter 2.2 for reminder).

Once the password length is known then the ciphertext is divided into groups in the length of the counted password length. Each group is then divided into the n-grams so that the following n-gram consists of the last $n - 1$ elements from the previous and the last element would contain the first element from the group. For example, if we consider the message “HELLO WORLD” and a password of length 5, then the message is divided in bigrams (n-gram where $n = 2$), i.e. [HE, EL, LL, LO, OH] and [WO, OR, RL, LD, DW]. For a better visualisation of the process see Table 2.4.

Tab. 2.4: Bigram division of “HELLO WORLD” if the probable password length would be equal to 5.

Index	0.	1.	1.	2.	2.	3.	3.	4.	4.	0.
0.	H	E	E	L	L	L	L	O	O	H
1.	W	O	O	R	R	L	L	D	D	W

The penultimate step takes the most time. This step consists of trying all the combinations of the n-gram-length password on all the n-grams from one column and count for each the fitness. The fitness is counted as a sum of decimal logarithmic probabilities of each n-gram in the column. For instance, column 0. 1. from the Table 2.4 consists of bigrams “HE” and “WO”. The mathematical formula would be:

$$\log(p(HEWO)) = \log(p(HE)) + \log(p(WO)) \quad (2.2)$$

where probabilities of each bigram are counted as

$$p(HE) = \frac{n_{HE}}{N} \quad p(WO) = \frac{n_{WO}}{N} \quad (2.3)$$

where n_{HE} and n_{WO} are the number of occurrences of each bigram in ciphertext and N is the total number of all the bigrams from the database.

The final step is to take the tried password fragments with the smallest fitness result and put them together. Note that if the following password fragment has smaller fitness than its elements are preferred to than the one that preceded it. For

example if indexes 0. 1. have counted fitness higher than in the indexes 1. 2., the element that would be used for the index 1 would be from the password fragment from indexes 1. 2.

In practise at least two n-gram fitness functions are usually combined – one for password composition and one for the fitness of the decrypted text especially if a range of password lengths is given. One fitness is used to put the password together, then this password is applied on the ciphertext and with the other fitness function is counted the final result that helps to determine which password in case of the password length range is the best [19].

3 Web interface

Who would not know these days what is a Web interface? Modern people use it every day, but better question is, if they know something about the used technologies that allows them to access large number of different Web sites.

Everybody knows that to access the Internet one must have a Web browser. In order to access requested Web site browsers one can use WWW information system (World Wide Web) that identifies web resources by their URL (Uniform Resource Location). According to the statistic from July 2019 the most three popular browsers nowadays are *Google Chrome*, *Apple Safari* and *Internet Explorer and Edge* [30].

The basis of Web pages are HTML or XHTML documents, that can be viewed on the monitors of PCs or Smartphones through Web browsers and WWW. These documents can also include scripts, that can be implemented in many programming languages such as PHP, Javascript or Actionscript. In order to run these variable scripts, one must install and allow in Web browser the necessary extension packages. For instance, Javascript can not run without JVM (Java Virtual Machine) [43].

Web pages can be divided based on usage of scripts into *static* and *dynamic* pages. By *static* web page is meant that its content does not change, on the contrary in *dynamic* web page, multiple features allow to generate content automatically while running on URL. Nowadays dynamic pages are very popular, because the interactivity gives the page better appeal than the static one [49]. Of course, too much is always bad. If there are too many dynamic elements on the page, it can slow down the loading of the page and also the browsing itself.

This Section focus on HTML5, CSS and Javascript, because these languages were chosen for implementation of the assigned application.

3.1 HTML5 (Hypertext Markup Language version 5)

Hypertext Markup Language version 5 is an open format markup language developed by WHATWG (Web Hypertext Application Technology Working Group). HTML5 was introduced to the public in 2008 and in 2014 was this standard updated by the “*W3C Recommendation*” group [20]. Before HTML5 was introduced there were also some attempts to cross XML and HTML syntaxes, because HTML4 was not upgraded since 2000. This experiment is referred as “XHTML” [24].

As it was mentioned before, HTML5 is a markup language. This means that it consists of a tree of elements and text. Each component is marked with a *tags*. There are two types of tags within HTML: *start tag* and *end tag*. Example of such pair can look like this: `<html> ... </html>`. Some tags can also be during the writing of the code omitted and implied by different tags, e.g. tag `<html></html>`

(for closer specification of optional tags see source [20] Section “12.1.2.4 Optional tags”).

HTML5 had brought several innovations compare to HTML4. To these innovations belongs e.g.:

- new elements (e.g. `section`, `header`, `figure` or `embed`),
- new attributes for several elements (e.g. `hreflang`, `type` and `ref` for `area` element),
- content model (how elements may be nested) and
- new APIs (e.g. API for enabling the offline Web application within application cache).

Of course this new standard obsoleted several elements and attributes, e.g. elements `frame` or attribute `border` on element `object`. For complete list of changes among HTML5 compare to HTML4 see source [21].

3.2 CSS (Cascading Style Sheets)

Cascading Style Sheets is a language that define style of the Web documents written in e.g. HTML. CSS documents can define uniform color, width, height, align, margins, padding and many other style related information for each element implemented in the whole Web page. Example of how can such CSS document look like is shown in Listing 3.1.

There are three ways how to use CSS within the HTML document:

1. define style of element with attribute `style` (e.g. `<p style="color: #FF0000">`),
2. inside the `<head>` tag inside `<style>` tags, or
3. with extra CSS file (with `*.css` suffix).

Listing 3.1: Example of CSS document

```
1  /* Example of used CSS file */
2
3  p {                               // customization of a paragraph
4      color: #FF0000,               // red colour
5      text-align: center;
6  }
```

The CSS file is than implemented within the HTML document with tag `<link>` with attributes `rel` that specify the relationship with HTML file (in this case it would be “stylesheet”). Moreover, `href` defines the CSS document (this attribute can also

contain the full path to the CSS file). An example of such implementation is shown in Listing 3.2.

Listing 3.2: Example of HTML file with link to CSS document

```
1  <!DOCTYPE html>
2
3  <!-- Example of link tag in HTML file -->
4
5  <html>
6      <head>
7          <link rel="stylesheet" href="style.css">
8      </head>
9      <body>
10         ...
11     </body>
12 </html>
```

What is important to know is that there is a rule that says that the last used argument within CSS applies, because CSS styles can layer definitions [23].

3.3 JavaScript and TypeScript

JavaScript is a cross-platform object-oriented programming language, that allows to create scripts within a Web page, and thus creating its dynamic content e.g. menu animation, slide show etc. This language can also be used to create mobile applications, or can be used in non-browser applications such as Adobe Acrobat, but since this thesis is focused on the implementation of a Web-based application, further content will be concentrated on dynamic Web page development [25].

There are two ways how to implement Javascript scripts in created Web document. Either is the script embed inside the HTML (or XHTML) document inside `<script>` tags, or it is included in a `*.js` file in the Web source directory. The script needs to be downloaded from server into the clients computer in order to be run. Moreover, client has to have JVM (Java Virtual Machine) environment installed within the Web browser [53].

TypeScript is an open-source superset of Javascript created and maintained by Microsoft. It allows for example code refactoring, variable type check and many other useful functions. This superset allows to create not just client-side application, but also server-side application, which means, that the script does not have to be

downloaded into the clients computer in order to run. TypeScript files have *.ts* appendix and are used the same as Javascript files [51].

Javascript contains many frameworks and libraries, that developer can work with such as jQuery, AngularJS, Node.js or ReactJS. Because ReactJS has been selected to implement the specified application, this library will now be introduced.

ReactJS

As it was mentioned before, **ReactJS** is a JavaScript library dedicated to build user interfaces maintained by Facebook. Main advantage of this language is, that it can build “single page” applications i.e. there is no necessity to reload the page for viewing another content. Another advantage is that ReactJS allows to reuse already created user interface components, renders just code, that is needed. It is also faster, because it uses virtual DOM (Document Object Model) elements which tries to most effectively update the browser’s DOM [35].

In ReactJS developer can also use JSX syntax. This syntax is similar to the HTML’s (it uses tags), but it is all “hidden” inside a variable or component [22]. An example of JSX syntax use is shown in the following Listing 3.3:

Listing 3.3: Example of JSX syntax in ReactJS

1

```
var myVariable = <h1> Example of JSX syntax. </h1>;
```

In the implemented application ReactJS package *Material-UI* is used, that enables easy definition of GUI elements. Material-UI is an open-source crowd-funded project licensed under MIT license. Installation of this package is simple. User just assigns `npm install @material-ui/core` into the command line. The default CSS design is then implemented with `<link>` tag. This project also contains many “free to use” templates just as prepared Web page templates available in their shop on the official page of Material-UI. Link to the GitHub project site with all documentation is found under source [31].

4 Application implementation

In this chapter the complete implementation of the bachelor work is described.

At first how to install the created application with essential packages for the application to run and display correctly is explained. In particular, this chapter briefly describes the system requirements and browser support. Then, we pass to the description of the implemented scripts that allows to start up the application on the local server. At last there is an inside view into the application itself, i.e. its GUI (Graphical User Interface) and implemented functions (chosen ciphers described in the previous chapters and methods allowing the ciphertext analysis) with closer description of their customizable settings.

For the implementation of the application was used an empty template from GitHub that is licensed under the terms of MIT licence [48]. The whole application was developed on Microsoft Windows 8.1 operating system in text editor Visual Studio Code (version 1.45.1) with installed following extensions:

- *Code Runner* by Jun Han (version 0.10.0),
- *Debugger for Chrome* by Microsoft (version 4.12.8),
- *ESLint* by Dirk Baeumer (version 2.1.5),
- *TSLint* by Microsoft (version 1.2.3) and
- *Typescript React code snippets* by infeng (version 1.3.1).

All named extensions are available for free in the Visual Studio Marketplace. Extensions are also downloadable right in the Visual Studio Code menu under **View > Extensions** module.

As programming languages we chose HTML5 and JavaScript library ReactJS. The main reason of this choice is that ReactJS is more user-friendly in building user interfaces, and unlike JavaScript it also checks the variable type by using Typescript (for more details see Chapter 3.3).

The main file of the application is *index.tsx* and the general layout of the page is defined in *basicLayout.tsx*. Both files are located in directory `src`. Further each ciphers and analyzer methods has its own basic *.tsx files where all components are implemented. They are located in directories `analyzer` and `ciphers`. These files are then displayed in the corresponding layout based on the users choice. For a better visualisation of the structure of the code see Appendix B and for inside application controls see Appendix C where are described print-screens of the implemented application.

4.1 Installation and start up

As mentioned in Chapter 3.3, ReactJS uses `npm` scripts for installation and creation of empty project, and manipulation with created applications that are defined in the `package.json`. For their implementation is used **Node.js** environment – asynchronous JavaScript runtime. In the implemented application are all these `npm` scripts referenced in the `react-app.d.ts` file in the general application directory and are imported during the application installation.

Installation

All what is needed to do is to get into the directory with the source, open command line in this directory and run `npm install` command. This command installs all needed packages for the application and the environment ReactJS. File where the created application will be stored can be anywhere on users computer.

JavaScript (and thus TypeScript) applications development is possible on any device because JavaScript Integrated Development Environment (short IDE) can run on any operating system also including the phone operation systems. Even hardware requirements for applications development are not that crucial. According to Dani Akash S, author of the *JavaScript by example* guide is recommended Linux or Windows OS with minimum of 4 GB RAM or any Mac machine but only to achieve better development experience [1]. The only problem that can be encountered is with accessing and running the JavaScript applications in some older versions of web browsers.

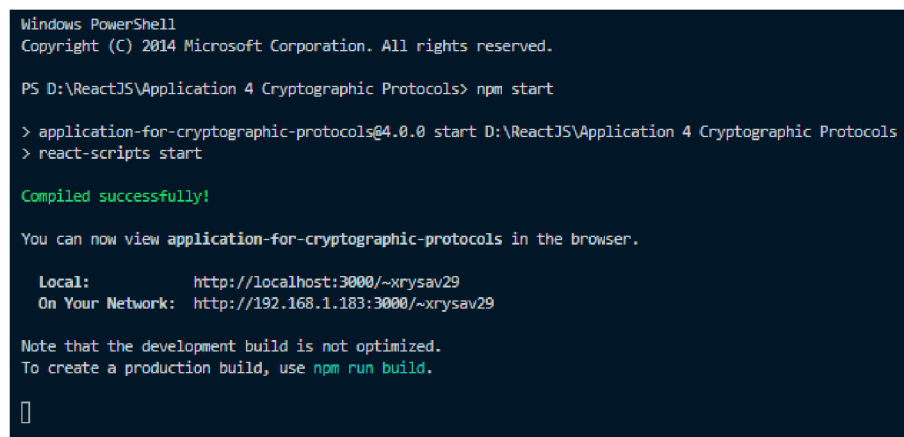
The implemented application uses the latest **ReactJS** (16.13.1) and the latest **Material-UI** (4.10.1) and ReactJS supports it from version 16.8.0. The latest ReactJS version depends on JavaScript Keyed collection *Map* and *Set* (definition of both collections is described in standard ECMAScript¹ ECMA-262) that are widely supported by all latest builds of all major web browsers such as Google Chrome, Firefox, Opera or Safari with the exception of Internet Explorer version 11, where multiple essential functions described in the mentioned collections does not have native support [26]. In case of Material-UI it supports all the stable builds of the major browsers including Internet Explorer 11 [47].

¹“ECMA is an industry association dedicated to standardization of informational and communication systems.” – official ECMA International® introduction [11]. JavaScript is ECMA standard since 1997 under official name ECMAScript.

Start up of the application server

For starting up the server with this application, the user types down the command line `npm start` in the source directory. In Visual Studio Code the server runs, as default, on local server on port 3000 (start up settings can be changed in file *launch.json* located in directory `/.vscode`).

First the script obtains all necessary information form the application and environment. Then it starts up the development server and emits all implemented files. After this step follows the type-check of the code itself. If the type-check is successful the command line will report a successful compilation and the rendered application will be available for view in the browser on url `http://localhost:3000`. Success message inside the Windows PowerShell command line is depicted in Figure 4.1.



```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS D:\ReactJS\Application 4 Cryptographic Protocols> npm start

> application-for-cryptographic-protocols@4.0.0 start D:\ReactJS\Application 4 Cryptographic Protocols
> react-scripts start

Compiled successfully!

You can now view application-for-cryptographic-protocols in the browser.

Local:            http://localhost:3000/~xrysav29
On Your Network:  http://192.168.1.183:3000/~xrysav29

Note that the development build is not optimized.
To create a production build, use npm run build.

█
```

Fig. 4.1: Print screen of the success message written by the `npm start` script.

Once one finishes with the web page inspection, the development server can be closed in the command line first by pressing the key combination of `Ctrl + C` and then by confirmation of the batch job termination or just by quitting the Visual Studio Code environment.

4.2 Application development

the main file *index.html* is located in the `public` directory. This file contains just plain HTML environment with definition of the web icon, manifest, Google font stylesheet and a statement in case the JavaScript is not enabled in the browser. The rendering file of the page is defined in the *index.tsx* file, that is located in the `src` directory. The rendering part of the code is shown in the Listing 4.1.

In the above listing, in the `ReactDOM.render()` function is first declared element that is supposed to be rendered and then placed in the *index.html* file, where the

Listing 4.1: Example of JSX syntax in ReactJS

```

1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import { ThemeProvider } from '@material-ui/styles';
4   import { CssBaseline, /*...*/ } from '@material-ui/core';
5   import theme from './theme';
6   import styles from './styles.css';
7
8   /* ... */
9
10  ReactDOM.render(
11    <ThemeProvider theme={theme}>
12      <CssBaseline classes={styles}/>
13      <App />
14    </ThemeProvider>,
15    document.querySelector('#root'),
16  );

```

element or in this case application should be shown both separated by comma. All elements are closed in tags of `<ThemeProvider>` that implements the created *theme.tsx* file which is almost equal to the CSS file but it allows more user-friendly environment for elements customization than in **.css* files especially when it comes to customization of the Material-UI elements. Inside the theme-rendering tags is the `<CssBaseline/>` tag that implements the CSS classes that are imported from the *styles.css* file and the `<App/>` tag that refers to the implemented application.

As it was already mentioned before, the GUI the application uses Material-UI by default and files *theme.tsx* and an *styles.css* are implemented for a better customization of the components. Both are located in `/src` directory. The *theme.tsx* uses a `createMuiTheme` provider that allows to effectively override the predefined looks of the Material-UI elements. It also allows an implementation of a basic color palette that is then easier accessible in the code. The only problem with this solution is that the defined customization is *general* thus is then applied to all the modified elements. For instance, if the implementation requires two fields that have different colors while they are used, then the CSS is a better solution. Fragment of the *style.css* file is shown in the Listing 4.2.

In the Listing 4.2 was used the method of the external definition described in the Chapter 3.2. In this particular file are used two types of element description – in form of a **general** description (the upper example of the `body` and `img`) and

Listing 4.2: Fragment of CSS document from the implemented application

```
1      /* HTML elements customization */
2
3      body {          // background definition
4          background-image: url("background.png");
5          background-color: white;
6          background-position: top;
7          background-position-x: right;
8          background-attachment: fixed;
9          min-inline-size: 500px;
10         background-repeat: no-repeat;
11     }
12     img {          // image customization
13         align-content: center;
14     }
15
16     /*
17     * CSS classes used for Material-UI elements
18     * customization (implemented under element
19     * attribute "className").
20     */
21
22     .title {      // custom title
23         font-size: 80px;
24         text-align: center;
25         text-shadow: 0 0 5px #9a9a9a;
26     }
27
28     /* ... */
```

in form of a **class** description (the lower example of a `.title`). The *general form* is applied on all defined instances in the rendered application without any further mention in the code but in case of the *class description* have to be the class name put in the corresponding attribute field. In case of the Material-UI elements it is assigned to the `className` attribute. These CSS classes have to be imported from the `*.css` file – this is done in the rendering file `index.tsx` as it was mentioned few paragraphs above.

The characteristic of the CSS is that these files contain only the description of each element, that is then reproduced by the browser, when the code is rendered. It

also allows to have all the graphical definitions in one place thus a code duplication is avoided and the maintenance becomes also easier. Another advantage of the CSS is that it is widely supported even on the older browsers so the layout of the application does not scatter when is the CSS file correctly implemented.

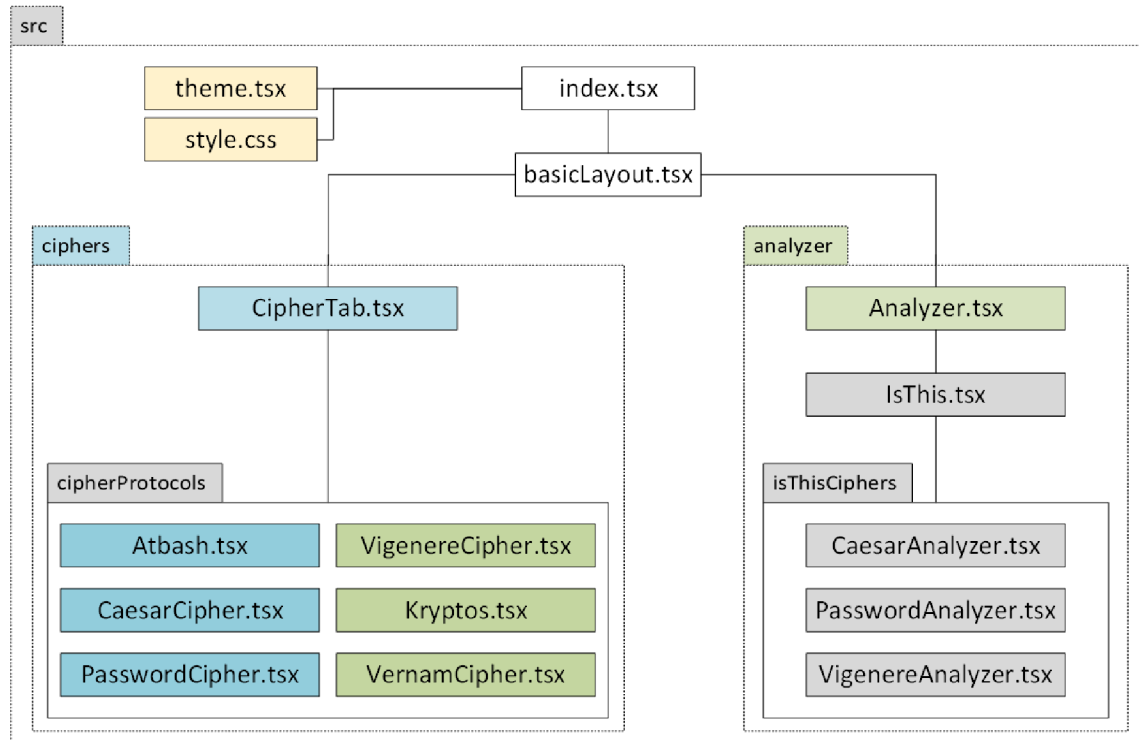


Fig. 4.2: Simple diagram of the application hierarchy between files.

Now it is possible to pass to the description of the application implementation itself, which represents the most complex part. Note that the complete tree directory with all the application files is shown in Appendix B. This tree directory is also simply reproduced in the Figure 4.2 where is also shown the hierarchy of the class files and their bindings.

As it was mentioned before the file that renders the whole application is the *index.tsx* where is also defined the general layout of the application:

- the **title of the application** (“Application for Cryptographic Protocols”),
- **application work field** hidden under the `<basicLayout/>` tag that is referring to the application controls and functions and
- the **application footer** that contains the copyright reference to the page icon and a hyperlink to the Material-UI documentation page.

In the file *index.tsx*, files containing the element customization are also imported – the previously described *styles.css* and *theme.tsx* files.

The core file that puts together all the implemented functions and elements is the *basicLayout.tsx*. In this file are defined all the general functions such as

language change, JSON databases loader, fields loader etc. This class also holds important global variables such as the loaded JSON databases, alphabet arrays and regular expression variable for the input purification from undesired characters. Most importantly this file implements the input and output field, the menu bar located above these two fields and two floating buttons located in the right corner of the page that are switching between the two environments – *Cipher* mode and *Analysis* mode. This two “modes” are actually just two classes implementing the content under the input/output field that are changed when one of the floating button is pressed. This “change” is implemented in the `basicLayout` class in the public function `render()` and is based on private class property of type `boolean` named `fabSwitch`.

The *CipherTab.tsx* contains the class defining the tab-container with the implemented ciphers. Each cipher have its own class and they are merged into the final application layout by the `CipherTab` class.

All the cipher classes has three general properties from the `CipherTab` class (that assumes it from the `basicLayout` class) which are:

1. identifier of the input field,
2. identifier of the output field and
3. identifier of the encryption/decryption switch.

Each cipher class also always contains a method defining the cipher algorithm with the ability to run in the encryption mode just as in the decryption mode. This ability was implemented because the processes are the same with the only exception of the processed text. These methods names can be generalized as `<cipher_name>Cipher(<parameters>` and are also always constituted from two steps:

1. **input load** to `string` array from the input field (eventually also the key load) that is cleared from undesirable characters and definition of other variables necessary for the progress – at least a variable where the encrypted or decrypted text is stored and
2. the own **cipher algorithm** performed above the loaded input array with the `forEach()` function.

Each cipher also defines it own control elements used for the settings visualization (these are described in the following chapter 4.3).

The *Analyzer.tsx* contains class defining the first half of the implemented analyzer – the frequency analysis table and substitution menu and overview table containing various values, that can be found useful during the ciphertext analysis. This class also assume three identifiers of the input and output fields and encryption/decryption switch and distributes it to the further implemented classes. In the `Analyzer` class, the file *IsThis.tsx* is imported. This file holds the implemented “cipher checkers”. In particular, each “checker” has again its own implementa-

tion file containing custom settings element and `isThis<cipher_name>()` method, that implements public methods from corresponding cipher files in fixed decryption mode with the exception of the Atbash cipher, that is implemented directly in the *IsThis.tsx* selector container. Further details on how the settings of each cipher or analyzer function looks like see the following Chapter 4.3.

4.3 Application visualisation

The application is thought for both creating encrypted data and analyzing them. This allows users to understand the hardness of attacking the different implemented ciphers even if they have knowledge either on the secret key or on the text that has been encrypted. Moreover, users can personally try to attack the cipher.

At the first boot of the application, the user will be directed to the default page containing these implemented substitution ciphers:

1. **Monoalphabetic substitution ciphers** (green tabs)
 - Atbash cipher,
 - Caesar cipher,
 - Password-modified cipher,
2. **Polyalphabetic substitution ciphers** (blue tabs)
 - Vigenère cipher,
 - Kryptos, part I & II,
 - Vernam cipher.

The basic GUI division can be seen in Figure 4.3. In the upper part of the application layout are always located two fields: customizable **Input field** and **Output field** available only for reading. Both fields are working with `string` type. User can also find above these two fields three general controls described from left to right:

- The encryption and decryption **switch** that defines whether will be the action buttons under each cipher tab encrypting or decrypting the given `string` in the input field.
- **Two buttons** for *resetting* the fields and *copying* the output to input (these two buttons are disabled in the *Analyzer mode*). The “RESET FIELDS” button throws everything in the application into the state that is loaded at the first access to the page. The “COPY OUTPUT TO INPUT” button allows the user to copy the encrypted text into the input field. This step is necessary since the encrypted text can be then pass to the analyzer in order to try to be decrypted.
- The **language selector** that enables change between *English* (international) and *Czech* alphabet, that are within the implemented functions.

The context of the lower part is based on the mode of the application. Modes can be changed with buttons that are always situated at the right bottom of the page. In the **Cipher mode** user can encrypt (or decrypt) plaintext that can be defined in the input field. In the **Analyzer mode** user can play with created cryptogram and try variable analytical methods.

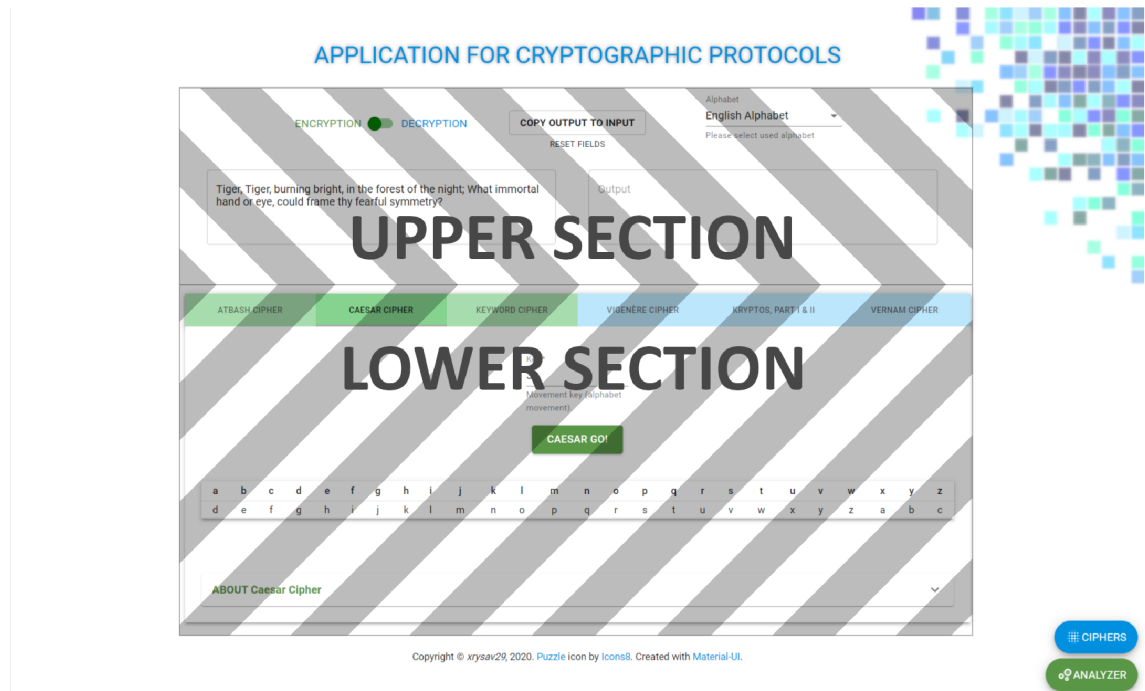


Fig. 4.3: Basic layout division of the web application.

4.3.1 Section of the substitution ciphers

Application in the *Cipher mode* contains in the lower section box with six tabs of two colors: **green** that represents *monoalphabetic ciphers* and **blue** that represents *polyalphabetic ciphers*. Under each tab are situated the settings for each cipher. Each setting will be described in the following sub-chapters.

Atbash cipher

Since this cipher that is described in Chapter 1.1.1 does not have any possible settings then there are situated only two elements in the box under tab “Atbash cipher”:

- **button** “ATBASH GO!” that encrypts the text given in the input field and shows the result in the output field and
- **table** that shows mapping of the plaintext alphabet to the ciphertext alphabet used during the encryption (or decryption).

Atbash tab layout can be seen in Figure 4.4.

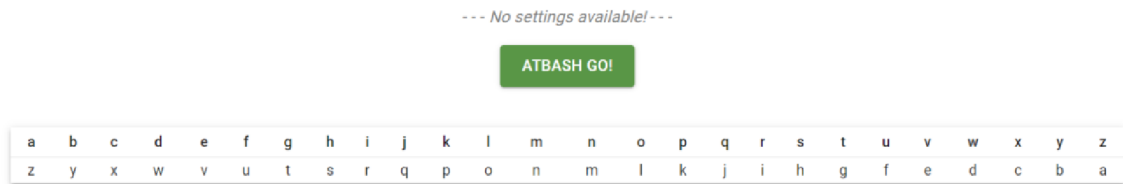


Fig. 4.4: Detailed view of the Atbash cipher settings.

Caesar cipher

In case of the Caesar cipher described in the Chapter 1.1.2, the tab section with one customizable input field for the cipher key definition labeled as a “Movement key” field (type of **number**) with default value $key = 3$. This field is a required parameter so if the field is empty, then the application shows a prompt box requiring a value when the action button (labelled as “CAESAR GO!”) is pressed. Button is located under the key field. When no value is given to the prompt box, application will use the default value of $key = 3$.

The last component of the tab section is a table underneath all named components, that enables better visualization of the defined movement within the used cipher alphabet.

Caesar cipher settings are shown in the Figure 4.5.



Fig. 4.5: Detailed view of the Caesar cipher settings.

Keyword cipher

Under the Keyword cipher tab can be found settings for the implemented cipher described in Chapter 1.1.3. In this settings are again three components: action button labelled “CIPHER GO!”, table showing the mapping of plaintext alphabet to by password modified alphabet and a required field of type **string** for custom password with default value **SECRET**. Once again if an action button is pressed and

the password field is empty, application will ask for the password or it will use the default value.

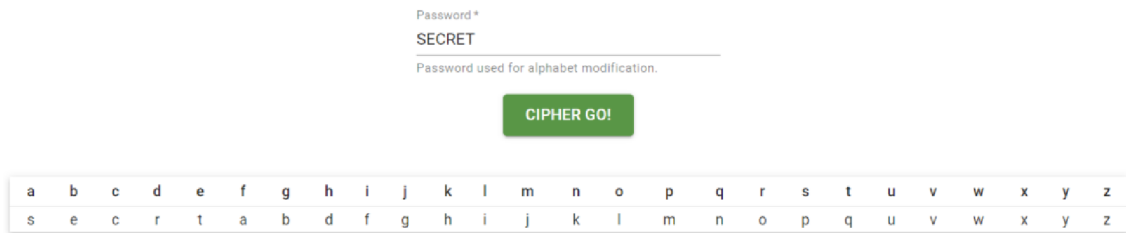


Fig. 4.6: Detailed view of the Keyword cipher settings.

Password field is also treated with protection against punctuation and other characters that are not included in the used alphabet (like for instance numbers) chosen by the selector located in the upper part of the application. All described elements are depicted in the Figure 4.6.

Vigenère cipher

Vigenère cipher settings consists only from action button labeled “VIGENÈRE GO!” and of a customizable field of type `string` that is once again required. The default password is `SECRET` and when is this field left empty just as the following prompt box then this default value is used. The algorithm of the cipher is described in the Chapter 1.2.1.

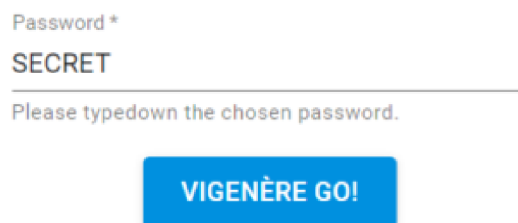


Fig. 4.7: Detailed view of the Vigenère cipher settings.

Custom password field is again treated from usage of different characters than those that are included in the alphabet chosen in the general settings in the upper part of the application. Depicted Vigenère settings are shown in the Figure 4.7.

Kryptos cipher, part I & II

In the Kryptos cipher tab are prepared setting for the cipher combination used in the part I and II of the ciphertext engraved in the Kryptos sculpture. This

sculpture and its cryptogram is described in the Chapter 1.2.3. In this settings are defined two customizable fields of type `string` (both are required) with default values of `PALIMPSEST` for Vigenère cipher and `KRYPTOS` for the used cipher alphabet modification, both again protected from invalid characters. Action button located beneath these fields is labeled “KRYPTOS GO!” and all these elements are shown in the Figure 4.8.

Fig. 4.8: Detailed view of the Kryptos, part I & II settings.

Vernam cipher

Vernam cipher (also known as the *one-time pad*) described in the Chapter 1.2.2 have in the tab section prepared once again two fields: one labeled as “USED PASSWORD”, second labeled as “CUSTOM PASSWORD”. Both fields operate again with the `string` type. At the beginning of the encryption (or decryption) process is checked, if any value is given in the “CUSTOM PASSWORD” field. If the value is given then is this value loaded into an array of `number` type. If the field is empty a random key is generated. The used password then appears in the “USED PASSWORD” field. Also the “CUSTOM PASSWORD” field is protected from loading different characters – this time from other characters than numbers. Action button located beneath the fields is labeled “VERNAM GO!”.

Fig. 4.9: Detailed view of the Vernam cipher settings.

If the encryption process is launched several times (more than once) an expansion panel labeled as “History of used keys” appears beneath the action button where

all the previously used keys are saved. Figure 4.9 shows the Vernam settings in the *Cipher mode*.

4.3.2 Section of the cryptoanalysis

When the application is switched to the *Analyzer mode* the lower part of the layout changes. There are two general sections:

1. upper part where:
 - the *frequency analysis result table* (see Chapter 2.1 for reminder) that is counted immediately once the mode is changed or when the input field changes (depiction of the exemplar table can be seen in the Figure 4.10),
 - *substitution menu* and
 - an *overview table* holding values such as index of coincidence (described in Chapter 2.2), counted Ceasar cipher key from the performed substitutions and other values, that can be useful during the cryptogram analysis,
2. the lower “Is this...?” section, where customized settings are located for quick check if the ciphertext was not encrypted with some of the implemented cipher.

Character analysis

T	I	G	E	R	B	U	N	H	F	O	S	W	A	M	L	D	Y	C
11.83 %	7.53 %	5.38 %	10.75 %	10.75 %	2.15 %	3.23 %	5.38 %	7.53 %	5.38 %	5.38 %	2.15 %	1.08 %	5.38 %	5.38 %	3.23 %	2.15 %	4.3 %	1.08 %

Fig. 4.10: Exemplar result table with the result of the frequency analysis formed from the given ciphertext.

Frequency analysis

The frequency analysis of the text in the input field is performed when the application mode is changed, or when the input field is modified. The results that are shown in the table labeled as “Character analysis” are then mirrored in the “Frequency analysis menu” shown in the Figure 4.11.

The menu is constituted of two selectors – one for ciphertext letters (the ones that are mirrored from the frequency analysis result) and one for the letters used for substitution that are mirrored from the formed database created from a large sample of text (see Chapter 2.1 for analysis process reminder). Values in the selectors are sorted by the falling occurrence.

Both selectors have to contains chosen values otherwise the button “SUBSTITUTE!” will throw an information alert. This action button will convert all the chosen cipher letters to upper case plaintext letters and the modified text is then

shown in the output field. First substitution is created from the `string` in the input field but next steps consider the modified `string` in the output field.

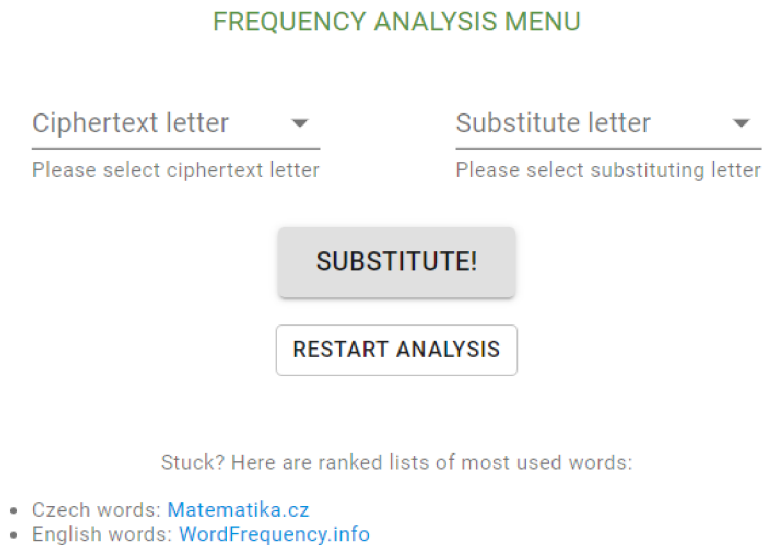


Fig. 4.11: Detailed view of the substitution menu in the web application.

Under the button “SUBSTITUTE!” is located button for the analysis restart. This button erase all achieved information and trows the analyzer in the default state. Important is that the `string` in the input field stays unmodified the whole time.

Also in case the user is stuck with the analysis then under the substitution menu are two links that refer to pages with frequency analysis of the most used words in Czech and English language.

Overview table

The overview table is situated next to the Frequency analysis menu and is shown in the Figure 4.12. This table provides the user at the beginning a few basic information about the cipher: counted **index of coincidence** (see Chapter 2.2 for reminder) and information gained from IC about whether is the cipher *monoalphabetic*, *polyalphabetic* or if it is just random text and information about *probable plaintext language* (in this application are differed five languages: Czech, English, French, Spanish and German). Other information are gained through executed substitutions.

Counted Caesar key is computed from the executed substitution by deduction of the cipher letter position in alphabet from the substituting letter position. The best key is then decided based on the number of key occurrences. This function was implemented by using `Hash Map` package, its implemented functions and the use of the native function `sort()` above arrays in JavaScript. **Atbash counter**

Index of coincidence:	1.592332865825152
Probable type of substitution cipher:	monoalphabetic
Probable language:	English
Counted Caesar key:	0
Atbash correspondence counter:	0
Password alphabet:

Fig. 4.12: Detailed view of the overview table in the web application.

simply counts the occurrences of matches between the executed substitution and the Atbash alphabets mapping and finally the **Password alphabet** shows cipher alphabet created from all past substitutions.

***Is this...?* section**

The *Is this...?* section is similar to the cipher setting as they were described in the Chapter 4.3.1 in some cases with small differences. The idea of this section is, that user does not have to switch between the *Analysis mode* and *Cipher mode* in order to confirm whether an examined ciphertext is not encrypted by one of the given substitution ciphers.

On the first sight can be seen that some ciphers are missing – Kryptos and Vernam. In case of Vernam there is a simple answer since it is unbreakable. In case of Kryptos it is because the analysis of the Kryptos is not simple since this cipher uses a mixture of two encrypting method. Analysis have to be “handmade” since no method was yet encountered that could make the Kryptos analysis easier.

As it was already mentioned, the cipher setting implemented in this section have slight anomalies in order to make the analysis more comfortable. The only setting that does not change is in case of Atbash since it has none.

In case of the **Caesar cipher** an extra button labeled “OTHER POSSIBILITIES” is located under the action button that is just renamed to “IS THIS CAESAR?” but it implements the same function as in the *Cipher application mode* (but just the decryption). The “OTHER POSSIBILITIES” button is at the beginning

disabled because first have to be pressed the action button that executes the decryption of the ciphertext by a given key. This button holds a dialogue window, which contains the decrypted ciphertext by all the possible keys (in the length of the chosen alphabet).

In case of the **Password-modified cipher** the only change except the renamed action button to the “IT THIS PASSWORD-MODIFIED?” is in the table situated under the settings. It maps ciphertext letters to plaintext letters and at the beginning it the first row filled with dashes. This table fills with each executed substitution.

The screenshot shows the 'Is this...?' section with four cipher options: ATBASH CIPHER, CAESAR CIPHER, KEYWORD CIPHER, and VIGENER CIPHER. The VIGENER CIPHER option is selected. Below the options, there are two input fields: 'Password *' with the instruction 'Please typedown the chosen password.' and 'Password length *' with the instruction 'e.g. 3-8 or 5' and 'Password length for computation (min 2, max 99)'. A button labeled 'IS THIS VIGENERE?' is present, along with a 'GUESS PASSWORD?' button. At the bottom, a table displays the following data:

Input length:	118
Computed password length:	5
Best computed password:	--

Fig. 4.13: Detailed view of the Vigenère cipher settings in the Is this...? section.

The greatest change is in the **Vigenère cipher** settings. **This section works only when the chosen language** in the language selector in the menu bar above the input and output field **is English** because there is no free n-gram database for Czech language!

The settings are shown in the Figure 4.13. Three new elements had appeared: new customizable field for password length range, button “GUESS PASSWORD?” and a table with information about *input length*, *computed password length* and *best computed password*. These all new elements appear here because for the Vigenère cipher analysis was chosen to implement the **N-gram fitness** method described in the Chapter 2.4.

In this particular case are implemented bigram fitness for password construction and quadgram fitness for final check of the decrypted text using the build password. The databases used in the process are implemented with use of JSON files located in the `\src` directory (for the application tree directory see Appendix B).

The *probable password length* is computed using the index of coincidence for all possible password lengths considering that each letter from alphabet was used only once (see Chapter 2.2 for the principle reminder). For example for international alphabet (English language) would be the average IC counted for lengths 1 to 26.

The length range field is protected from other characters than one number or two numbers separated by dash and from empty field. Default value for this field is the computed password length. Value written into this field is read as a **string** and then loaded to a **number** array that is then used in the password construction function hidden under the “GUESS PASSWORD?” button. The password construction function is also partially protected from “freezing” since the password construction is a very demanding operation. Partially protected it is because if the input is too long and the user does not give a password length range or gives the range too wide, a prompt box will appear informing the user about the complexity of the operation. Though if the user will not respond or will insist on the wide range there is a high probability that the application will freeze.

Conclusion

This thesis focuses on the creation of an interactive web-based application for monoalphabetic and polyalphabetic ciphers and their cryptanalysis. Within the application were implemented three monoalphabetic substitution ciphers in particular Atbash, Caesar and Password-modified cipher and three polyalphabetic ciphers precisely Vigenère and Vernam cipher and cipher combinations used in Kryptos sculpture.

The most challenging part was the creation of a methodology that allows users to directly attack (cryptanalysis) the aforementioned ciphers. The attacks consist on 3 parts:

1. an interactive frequency analyses field which allows user to attack the encrypted message,
2. an overview table with index of coincidence and counters that helps in the attack choice, and
3. the possibility to check if the cipher is of a determined form for some selected cases.

In the frequency analysis, the user can compare the statistical letter occurrences in the selected language with the one in the ciphertext. This plus an interactive interface allow the user to directly attack the cipher keeping trace of the change made.

The aforementioned “possibility check” cases are functional for Atbash, Caesar cipher, Keyword cipher or Vigenère cipher. These ciphers have its own settings that allow user to try if the ciphertext is not created by one of the chosen ciphers.

In the overview table, for instance **Atbash counter** informs the user if some of the executed substitutions are equal to the one in Atbash cipher as well as **Counted Caesar Key** informs the user about what key was probably used (again counted from the executed substitutions).

Moreover, the n-gram fitness is an important part of the statistical analysis which allows one to gain the most accurate results in password reconstruction in case the cipher text is identified as a result of an polyalphabetic cipher.

The project was implemented as a web-application because it is easily reachable by a large spectrum of users and it does not have to be installed nor downloaded. All that is required is latest version of any modern web browser such as Google Chrome, Firefox or Safari (with the exception of Internet Explorer).

The whole web application was developed using the Visual Studio Code (version 1.45.1) environment on Windows 8.1 operating system.

Languages used for implementation of the application itself are HTML5, CSS and JavaScript. HTML5 was chosen because of several reasons:

- it is one of the commonly used (and supported) languages among the WWW (World Wide Web),
- it is the current version of HTML language and
- it is easy to learn and to use.

JavaScript was chosen since it is widely used and supported, easy to learn and user-friendly. The precised JavaScript library that was employed to implement the application was TypeScript library ReactJS². CSS was applied alongside with the Material-UI, which is ReactJS package for easier GUI implementation.

²For reminder – TypeScript is a superset for JavaScript, with many more functions, especially with ability to check variable type

Bibliography

- [1] AKASH S, D. *JavaScript by Example*. 2017. URL: https://subscription.packtpub.com/book/web_development/9781788293969/1/011v11sec8/system-requirements.
- [2] APPL, F. *Hádanka jménem Kryptos: Rozluští konečně někdo největší tajemství CIA?* 2019. URL: <https://enigmaplus.cz/hadanka-jmenem-kryptos-rozluستي-konecne-nekdo-nejvetsi-tajemstvi-cia/>. last revision 14. 5. 2019, [cit. 2020-25-05].
- [3] BARTER, A. *Index of Coincidence*. 2016. URL: <http://alexbarter.com/statistics/index-of-coincidence/>. last revision 22. 12. 2016, [cit. 2020-02-06]. Cryptography, Alex Barter.
- [4] BELLOVIN, S. M. “Frank Miller: Inventor of the One-Time Pad”. In: *Cryptologia* 35.3 (2011), pp. 203–222. DOI: 10.1080/01611194.2011.583711. eprint: <https://doi.org/10.1080/01611194.2011.583711>. URL: <https://doi.org/10.1080/01611194.2011.583711>. [cit. 2020-25-05]. An earlier version is available as technical report CUCS-009-11.
- [5] BROWN, D. C. “A Cryptanalysis of the Autokey Cipher Using the Index of Coincidence”. In: *Proceedings of the ACMSE 2018 Conference*. ACMSE '18. Richmond, Kentucky: Association for Computing Machinery, 2018. ISBN: 9781450356961. DOI: 10.1145/3190645.3190679. URL: <https://doi.org/10.1145/3190645.3190679>. [cit. 2020-03-06].
- [6] BUONAFALCE, A. *File:Alberti cipher disk.JPG*. [online]. 2008. URL: https://commons.wikimedia.org/wiki/File:Alberti_cipher_disk.JPG. last revision 18. 3. 2008, [cit. 2020-19-05]. Wikimedia Available under CC BY-SA.
- [7] CrypTool contributors. *Autokey*. 2020. URL: <https://www.cryptool.org/en/cto-ciphers/autokey>. last revision 4. 6. 2020, [cit. 2020-04-06]. Copyright © 1998 - 2020 CrypTool Contributors.
- [8] DUNIN, E. *Kryptos*. 2002. URL: <https://elonka.com/kryptos/>. last revision 29. 1. 2020, [cit. 2020-30-05].
- [9] DUNIN, E. *This is a transcript of a portion of ABC's World News Tonight broadcast from April 2, 1991*. 2004. URL: <https://elonka.com/kryptos/mirrors/WNT.html>. last revision 11. 3. 2004, [cit. 2020-30-05]. From page dedicated to Kryptos <https://elonka.com/kryptos/>, created by Elonka Dunin.

- [10] DUPONT, Q. *The Printing Press and Cryptography*. 1st Edition. Routledge, 2017. ISBN: 978-1-138-24464-1. DOI: <https://doi.org/10.4324/9781315267449>. URL: http://iqdupont.com/wp-content/uploads/2018/06/DuPont-2018-The_Printing_Press_and_Cryptography.pdf. [cit. 2020-19-05]. U.S.
- [11] ECMA International®. *ECMA International - Standards@Internet Speed*. 2020. URL: <https://www.ecma-international.org/>. last revision 13. 3. 2020, [cit. 2020-05-06]. Ecma International®.
- [12] *English Letter Frequency (based on a sample of 40,000 words)*. URL: <http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html>. [cit. 2020-01-06]. Math Explorer's Club, National Science Foundation supported project. Cornell Department of Mathematics.
- [13] FOJTOVÁ, L. "Softwarová podpora výuky klasické kryptoanalýzy". Master thesis. Brno University of Technology. The faculty of Electrical engineering and Communication. Department of Telecommunications, 2010. URL: <http://hdl.handle.net/11012/6354>. [cit. 2019-12-12]. Supervisor doc. Ing. Karel Burda, CSc.
- [14] FOLTÝNEK, T.; PŘICHYSTAL, J. *Komprimace a šifrování*. 2008. URL: <https://is.mendelu.cz/eknihovna/opory/index.pl?opora=621>. [cit. 2020-27-05]. Elektronické studijní materiály, Mendelova univerzita v Brně.
- [15] *Frequency Analysis: Breaking the Code*. 2019. URL: <https://crypto.interactive-maths.com/frequency-analysis-breaking-the-code.html>. [cit. 2020-01-06]. Crypto Corner, © 2013-2019 Daniel Rodriguez-Clark, All rights reserved Interactive Maths, © 2012-2019 Daniel Rodriguez-Clark, All rights reserved.
- [16] FRIEDMAN, W. F. *Military cryptanalysis, part I*. 4th Edition. 1952, p. 36. URL: https://www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/publications/FOLDER_241/41748389078762.pdf. [cit. 2020-01-06]. REF ID:A56895, National Security Agency Washington 25, D.C. Declassified and approved for release by NSA on 02-03-2014 pursuant to E.O. 135226.
- [17] FRIEDMAN, W. F. "The index of coincidence and its applications in cryptanalysis". In: 1935. URL: https://www.nsa.gov/Portals/70/documents/news-features/declassified-documents/friedman-documents/publications/FOLDER_233/41761039080018.pdf. [cit. 2020-02-06]. REF ID:A64722, United States, Government printing office, National Security Agency, Washington D.C.

- [18] GAJ, K.; ORŁOWSKI, A. “Facts and Myths of Enigma: Breaking Stereotypes”. In: (2003), pp. 106–109. DOI: 10.1007/3-540-39200-9. URL: https://link.springer.com/content/pdf/10.1007%2F3-540-39200-9_7.pdf. [cit. 2019-12-13]. Advances in Cryptology — EUROCRYPT 2003. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, 2656.
- [19] GUBALLA, J. *Implementierung eines Vigenere Solvers*. 2015. URL: <https://www.guballa.de/implementierung-eines-vigenere-solvers>. [cit. 2020-03-06]. www.guballa.de, © 2015 Jens Guballa.
- [20] *HTML, Living standard*. 2019. URL: <https://html.spec.whatwg.org/>. last revision 13. 12. 2019, [cit. 2019-16-12]. Web Hypertext Application Technology Working Group (WHATWG) Copyright © 2018 WHATWG.
- [21] *HTML5 Differences from HTML4*. 2014. URL: <https://www.w3.org/TR/html5-diff/>. last revision 9. 12. 2014, [cit. 2019-16-12]. W3C: Leading the web to its full potential Copyright © 2019 W3C®.
- [22] *Introducing JSX*. 2019. URL: <https://reactjs.org/docs/introducing-jsx.html>. last revision 21. 2. 2019, [cit. 2019-17-12]. React – A JavaScript library for building user interfaces Copyright © 2019 Facebook Inc.
- [23] JANOVSKEÝ, D. *CSS styly - úvod*. 2019. URL: <https://www.jakpsatweb.cz/css/css-uvod.html>. last revision 13. 6. 2019. [cit. 2019-17-12]. Jak psát web: o tvorbě, údržbě a zlepšování internetových stránek.
- [24] JANOVSKEÝ, D. *Syntaxe XHTML*. 2019. URL: <https://www.jakpsatweb.cz/html/xhtml.html>. last revision 13. 6. 2019. [cit. 2019-16-12]. Jak psát web: o tvorbě, údržbě a zlepšování internetových stránek.
- [25] *JavaScript*. 2019. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. last revision 9. 12. 2019, [cit. 2019-17-12]. MDN Web Docs © 2005-2019 Mozilla and individual contributors. Content is available under CC BY-SA 2.5.
- [26] *JavaScript Environment Requirements*. 2020. URL: <https://reactjs.org/docs/javascript-environment-requirements.html>. [cit. 2020-05-06]. React - A JavaScript library for building user interface. Copyright © 2020 Facebook Inc.
- [27] *Jeremiah 51*. 2020. URL: <https://www.biblegateway.com/passage/?search=Jeremiah+51%5C&version=NIV>. last revision 1. 1. 2020, [cit. 2020-22-05]. BibleGateway.com: A searchable online Bible in over 150 versions and 50 languages Holy Bible, New International Version®, NIV® Copyright ©1973, 1978, 1984, 2011 by Biblica, Inc.® Used by permission. All rights reserved worldwide.

- [28] *Kryptos*. 2007. URL: <https://www.cia.gov/about-cia/headquarters-tour/kryptos>. last revision 29. 1. 2020, [cit. 2020-30-05]. Central Intelligence Agency: The work of a Nation. The center of Intelligence.
- [29] LYONS, J. *Codes and Nomenclators Cipher*. 2012. URL: <http://practicalcryptography.com/ciphers/codes-and-nomenclators-cipher/>. [cit. 2020-30-05]. Practical Cryptography, Copyright James Lyons © 2009-2012.
- [30] MARTIN, J. *The best web browsers for 2019*. 2020. URL: <https://www.techadvisor.co.uk/test-centre/software/best-web-browsers-3635255/>. last revision 6. 8. 2019, [cit. 2019-17-12]. Tech Advisor - technology reviews, advice, videos, news and forums Copyright © 2019 IDG Communications Ltd. All Rights Reserved.
- [31] *Material-UI*. 2019. URL: <https://github.com/mui-org>. last revision 14. 12. 2019. GitHub. The world's leading software development platform.
- [32] MATUŠINSKÝ, P. *Šifry v Bibli – Atbaš*. 2007. URL: <http://www.myty.info/view.php?cislocclanku=2007110003>. last revision 26. 11. 2007, [cit. 2020-21-05]. Mýty a skutečnost.
- [33] MENEZES, A. J.; VANSTONE, S. A.; OORSCHOT, P. C. Van. *Handbook of Applied Cryptography*. 1st Edition. USA: CRC Press, Inc., 1996. ISBN: 0849385237. DOI: 10.5555/548089. [cit. 2020-02-06].
- [34] PILLÁR, J. “Kryptografie a šifrovací algoritmy”. Bachelor thesis. University of Pardubice, 2011, pp. 25–27. URL: <http://hdl.handle.net/10195/39635>. [cit. 2019-13-12]. Supervisor RNDr. Iva Rulićová.
- [35] *React.js (Introduction and Working)*. 2019. URL: <https://www.geeksforgeeks.org/react-js-introduction-working/>. last revision 17. 5. 2018, [cit. 2019-17-12]. GeeksforGeeks | A computer science portal for geeks @geeksforgeeks, CC BY-SA 4.0.
- [36] REUVERS, P.; SIMONS, M. *One-Time Pad (OTP)*. 2013. URL: <https://web.archive.org/web/20140314175211/http://www.cryptomuseum.com/crypto/otp.htm>. last revision 29. 1. 2013, [cit. 2020-25-05]. Wayback Machine, Internet archive Copyright 2009-2013.
- [37] RIVEST, R. L. “CHAPTER 13 - Cryptography”. In: *Algorithms and Complexity*. Ed. by JAN [VAN LEEUWEN]. Handbook of Theoretical Computer Science. Amsterdam: Elsevier, 1990, pp. 717–755. ISBN: 978-0-444-88071-0. DOI: <https://doi.org/10.1016/B978-0-444-88071-0.50018-7>. URL: <http://www.sciencedirect.com/science/article/pii/B9780444880710500187>. [cit. 2020-28-05].

- [38] RYABKO, B. “The Vernam cipher is robust to small deviations from randomness”. In: *CoRR* abs/1303.2219 (2013), pp. 2–5. arXiv: 1303.2219. URL: <http://arxiv.org/abs/1303.2219>. [cit. 2020-25-05]. Ithaca: Cornell University Library ProQuest Central.
- [39] SAFAVI-NAINI, R. *Information Theoretic Security: Third International Conference, ICITS 2008, Calgary, Canada, August 10-13, 2008, Proceedings*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-85093-9. DOI: 10.1007/978-3-540-85093-9. URL: <https://link.springer.com/content/pdf/10.1007%2F978-3-540-85093-9.pdf>. [cit. 2020-04-06].
- [40] SANBORN, J. *File:Kryptos sculptor.jpg*. [online]. 1991. URL: https://commons.wikimedia.org/wiki/File:Kryptos_sculptor.jpg. last revision 17. 4. 2020, [cit. 2020-30-05]. CC BY-SA (<https://creativecommons.org/licenses/by-sa/3.0>).
- [41] SCHWARTZ, J.; CORUM J. “This Sculpture Holds a Decades-Old C.I.A. Mystery. And Now, Another Clue.” In: *The New York Times* (2020). URL: <https://www.nytimes.com/interactive/2020/01/29/climate/kryptos-sculpture-final-clue.html>. last revision 29. 1. 2020, [cit. 2020-30-05].
- [42] SHENE, C.-K. *Kasiski’s method*. 2014. URL: <https://pages.mtu.edu/~shene/NSF-4/Tutorial/VIG/Vig-Kasiski.html>. last revision 15. 6. 2015, [cit. 2020-03-06]. Cryptography Visualization Tools: A Tutorial Department of Computer Science Michigan Technological University © 2014 C.-K. Shene.
- [43] SIKORA, M. “Webová vizualizace kryptografických systémů”. Bachelor thesis. Brno University of Technology. The faculty of Electrical engineering and Communication. Department of Telecommunications, 2014, p. 10. URL: <http://hdl.handle.net/11012/6354>. [cit. 2019-14-12], Supervisor Ing. Jan Hajný, Ph.D.
- [44] SPERLING, K. *File:EnigmaMachine.jpg*. [online]. 2005. URL: <https://commons.wikimedia.org/wiki/File:EnigmaMachine.jpg>. last revision 15. 4. 2005, [cit. 2020-19-05]. Available under public domain.
- [45] STEIN, D. D. *The Puzzle at CIA Headquarters: Cracking the Courtyard Crypto*. 2009. eprint: https://nsarchive2.gwu.edu/NSAEBB/NSAEBB431/docs/intell_ebb_010.PDF. URL: <https://www.elonka.com/kryptos/mirrors/daw/steinarticle.html>. last revision , [cit. 2020-30-05]. Originally an article at <http://www.csi.cia/studies/vol43no1/art5.html> (nowadays inaccessible). From page dedicated to Kryptos <https://elonka.com/kryptos/>, created by Elonka Dunin.

- [46] SUETONIUS. *The Lives of the Twelve Caesars – Julius Caesar*. Trans. by M. IHM. Loeb Classical Library, 1913. URL: https://penelope.uchicago.edu/Thayer/E/Roman/Texts/Suetonius/12Caesars/Julius*.html.
- [47] *Supported platforms*. 2020. [cit. 2020-05-06]. Material-UI Released under the MIT License. Copyright © 2020 Material-UI.
- [48] TASSINARI, O. *Create React App example with TypeScript*. URL: <https://github.com/mui-org/material-ui/tree/master/examples/create-react-app-with-typescript>. Ver. 4.0.0, last commit 2019-23-05. Licensed under MIT License.
- [49] *The Difference Between Dynamic & Static Web Pages*. 2018. URL: <https://web.archive.org/web/20190320233700/https://smallbusiness.chron.com/difference-between-dynamic-static-pages-69951.html>. last revision 10. 8. 2018. [cit. 2019-17-12]. Chron.com Copyright © 2019 Hearst Newspapers, LLC.
- [50] TRITHEMIUS, J.; Early Printing Collection & George Fabyan Collection. *Polygraphiae libri sex Ioannis Trithemij, abbatis Peapolitani quondam Spanheimensis, ad Maximilianum Caesarem*. [Reichenau] : Impressum ductu Ioannis Haselberg de Aia, bibliopolae, anno a Christo nato 1518, men. Iulio., 1518, p. 471. URL: <https://www.loc.gov/resource/rbc0001.2009fabyan12345/?sp=471>. 22. 10. 1996, [cit. 2020-22-05]. [Washington, D.C.] : Library of Congress, 2000. LCCN: 32017914.
- [51] *TypeScript Overview*. 2019. URL: <https://www.tutorialsteacher.com/typescript/typescript-overview>. last revision 12. 12. 2019, [cit. 2019-17-12]. TutorialsTeacher Online Web Tutorials © 2019 TutorialsTeacher.com. All Rights Reserved.
- [52] *Vigenère Cipher*. 2019. URL: <https://crypto.interactive-maths.com/vigenere-cipher.html#>. [cit. 2020-19-05]. Crypto Corner, © 2013-2019 Daniel Rodriguez-Clark, All rights reserved Interactive Maths, © 2012-2019 Daniel Rodriguez-Clark, All rights reserved.
- [53] *What is Javascript?* 2019. URL: <https://skillcrush.com/2012/04/05/javascript/>. last revision 17. 12. 2019 , [cit. 2019-17-12]. Learn to Code | Digital Skills are Job Skills | Skillcrush © 2012 - 2019 Skillcrush, Inc. All Rights Reserved.
- [54] YADAV, G. S.; OJHA, A. *A novel visual cryptography scheme based on substitution cipher*. IEEE, 2013, pp. 640–643. ISBN: 978-1-467-36101-9. DOI: 10.110

9/ICIIP.2013.6707673. [cit. 2019-10-12]. 2013 IEEE 2nd International Conference on Image Information Processing, IEEE ICIIP 2013. © 2014 Elsevier B.V., All rights reserved.

- [55] ZAPECHNIKOV, S. V. and a group of NRNU students. *Atbash*. 2013. URL: <http://cryptowiki.net/index.php?title=Atbash>. last revision 2. 12. 2003, [cit. 2020-20-05]. CryptoWiki: Encyclopedia of Theoretical and Applied Cryptography © Security department at National Research Nuclear University (NRNU) at Moscow, Russia.
- [56] ZOUHAR, P. “Generátor náhodných čísel”. Master thesis. Brno University of Technology. The faculty of Electrical engineering and Communication. Department of Telecommunications, 2010, pp. 12–13. URL: https://www.vutbr.cz/studenti/zav-prace?zp_id=32065. [cit. 2020-25-05] . Supervisor Ing. Jiří Sobotka.

List of abbreviations

CSS Cascading Style Sheets

CIA Central Intelligence Agency

DOM Document Object Model

GCD Greatest Common Divisor

GUI Graphical User Interface

HTML5 Hypertext Markup Language version 5

IDE Integrated Development Environment

JVM Java Virtual Machine

NSA National Security Agency

URL Uniform Resource Location

WHATWG Web Hypertext Application Technology Working Group

WWW World Wide Web

XOR Boolean "exclusive or" function

A Tabula recta (Vigenère tableau)

Tab. A.1: Substitution table for Vigenère cipher.

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

B Application directory tree

For better navigation through the semestral work, is in this chapter viewed the directory tree of the whole application. This code was created and tested with software Visual Studio Code v1.45.1.

```
/ ..... root directory of the application
├── public ..... public directory of the application
│   ├── favicon.png ..... icon of the web-page
│   ├── index.html
│   └── manifest.json ..... web-page manifest
├── src ..... source directory of the application
│   ├── analyzer ..... directory containing implemented analysis features
│   │   ├── isThisCiphers
│   │   │   ├── CaesarAnalyzer.tsx
│   │   │   ├── PasswordAnalyzer.tsx
│   │   │   └── VigenereAnalyzer.tsx
│   │   ├── Analyzer.tsx ..... analyzer main features and layout
│   │   └── isThis.tsx ..... Is this ...? tab-box implementation
│   ├── ciphers ..... directory containing implemented ciphers
│   │   ├── cipherProtocols
│   │   │   ├── Atbash.tsx
│   │   │   ├── CaesarCipher.tsx
│   │   │   ├── Kryptos.tsx
│   │   │   ├── PasswordCipher.tsx
│   │   │   ├── VernamCipher.tsx
│   │   │   └── VigenereCipher.tsx
│   │   └── CipherTab.tsx ..... implementation of the cipher tab-box
│   ├── background.png
│   ├── basicLayout.tsx
│   ├── index.tsx ..... main file for application render
│   ├── JSON_english_bigram_data.json ..... JSON database
│   ├── JSON_english_quadrams_data.json ..... JSON database
│   ├── module.d.ts ..... created modules for API
│   ├── react-app-env.d.ts
│   ├── styles.css ..... css file for customization of the GUI
│   └── theme.tsx ..... Material-ui elements themes customization
├── .gitignore ..... files, that should be ignored by git
├── package-lock.json
├── package.json ..... project metadata (containing dependencies, scripts etc.)
├── README.md . README file created by the author of the application template
└── tsconfig.json ..... configuration of Typescript
```

C Application controls

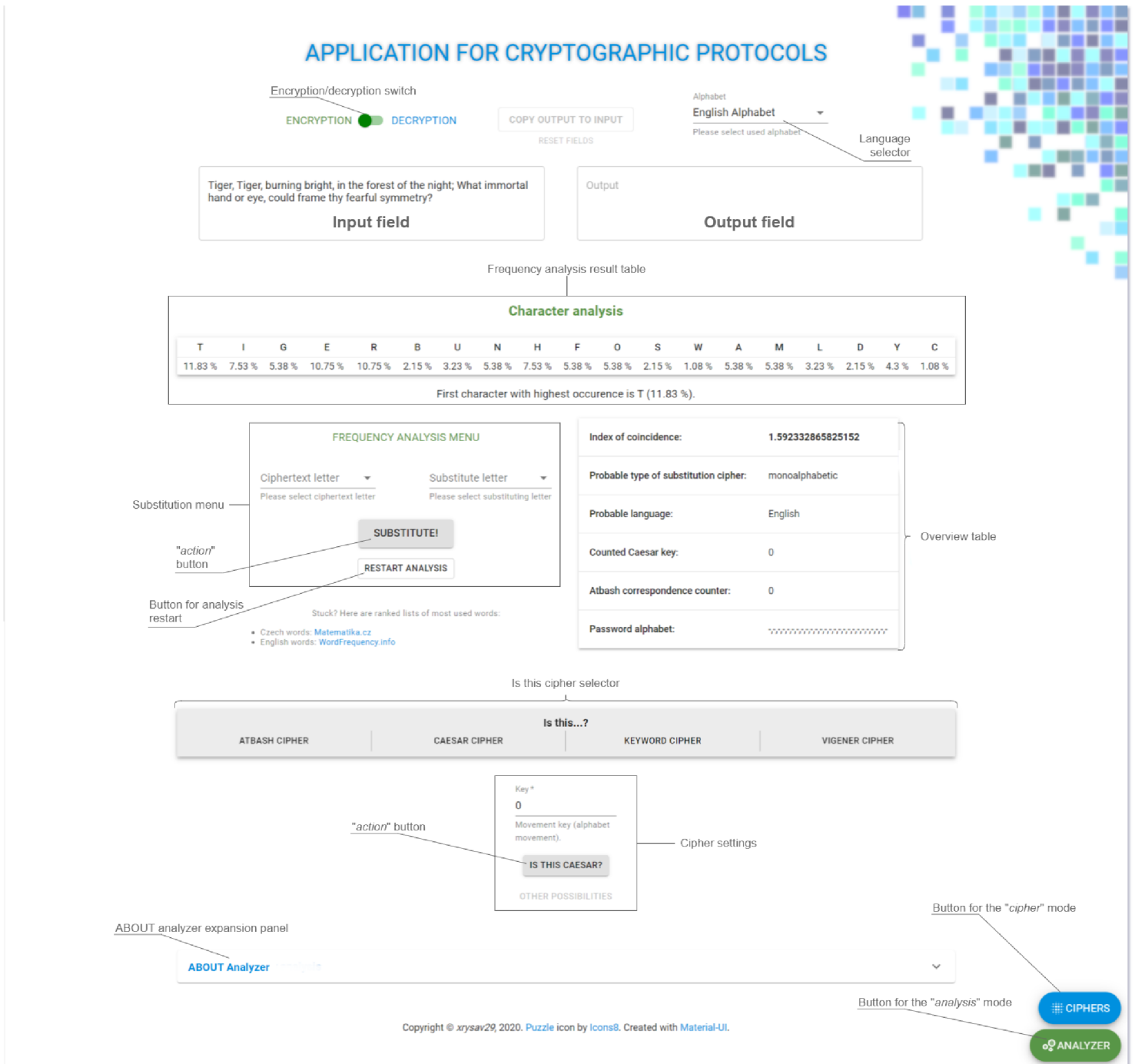


Fig. C.1: Controls of the web application in *Analysis mode*.

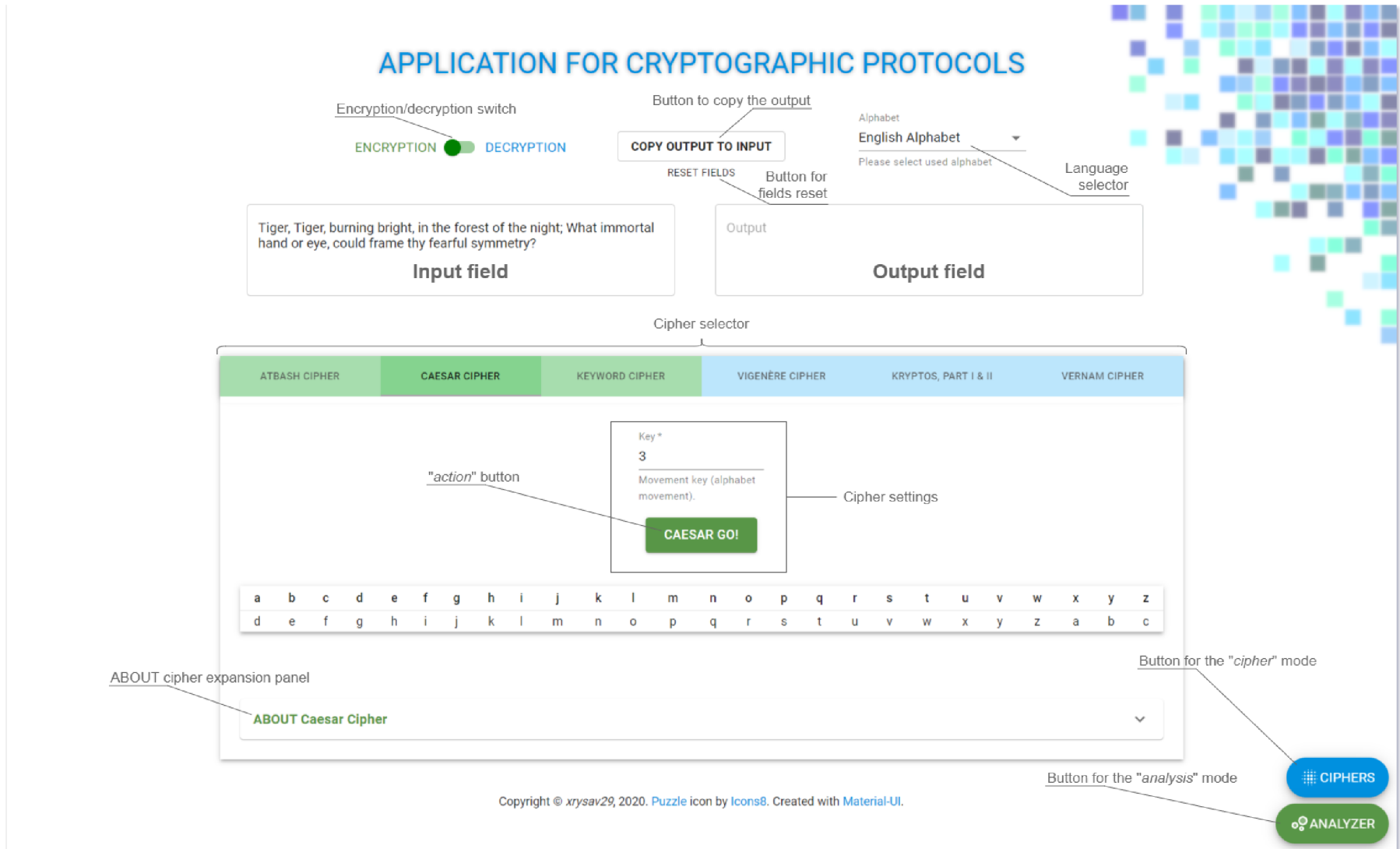


Fig. C.2: Controls of the web application in *Cipher mode*.