

# **STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST**

**Obor č. 12: Tvorba učebních pomůcek, didaktická technologie**

## **Aplikace pro šifrování ve WPF**

**Daniel Rybář  
Liberecký kraj**

**Liberec 2023**

# STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 12: Tvorba učebních pomůcek, didaktická technologie

**Aplikace pro šifrování ve WPF**

**Application for encryption using WPF**

**Autoři:** Daniel Rybář

**Škola:** Střední průmyslová škola strojní a elektrotechnická a Vyšší odborná škola, Liberec 1, Masarykova 3, příspěvková organizace; Masarykova 460/3, 460 01, Liberec 1

**Kraj:** Liberecký kraj

**Konzultant:** Ing. Marek Pospíchal, Ing. Vladimír Prokeš

Liberec 2023

## Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Liberci dne 15. 3. 2023 .....

Daniel Rybář

## **Anotace**

Tato práce se zabývá vytvořením aplikace pro šifrování dat pomocí frameworku WPF. Bude sloužit jako podpůrný materiál při výuce předmětu Kybernetická bezpečnost. V teoretické části se soustředí na analýzu základních principů šifrování, rozbor nejdůležitějších šifer a také na popis praktické části odborné práce.

## **Klíčová slova**

šifrování; bezpečnost; WPF; aplikace

## **Annotation**

This thesis deals with the development of an application for data encryption using the WPF framework. It will be used as a supporting material in the teaching of the Cyber security subject. The theoretical part is focused on the analysis of the basic principles of encryption, the analysis of the most important ciphers and also on the description of the practical part of this thesis.

## **Keywords**

encryption; security; WPF; application

# Obsah

Úvod .....	1
1 Úvod do kryptologie .....	2
1.1 Vymezení pojmů .....	2
1.2 Historie .....	2
1.2.1 Nejstarší šifrovací algoritmy .....	2
1.2.2 Steganografie .....	3
1.2.3 Enigma .....	3
2 Symetrické a asymetrické šifrování .....	4
2.1 Symetrická šifra .....	4
2.1.1 Proudová šifra .....	4
2.1.2 Bloková šifra .....	4
2.2 Asymetrická šifra .....	5
3 Základní rozdělení šifer .....	6
3.1 Substituční šifry .....	6
3.1.1 Monoalfabetické šifry .....	6
3.1.2 Polyalfabetické šifry .....	7
3.1.3 Ostatní substituční šifry .....	8
3.2 Transpoziční šifry .....	8
3.3 Ostatní druhy šifer a kódování .....	9
3.3.1 AES .....	9
3.3.2 RSA .....	9
3.3.3 Morseovka (Morseova abeceda) .....	9
3.3.4 XOR .....	10
3.3.5 ASCII kód .....	10
3.4 Frekvenční analýza .....	11
4 Hashovací funkce .....	12
4.1 Požadavky hashovacích funkcí .....	12
4.1.1 Kolize hashů .....	12
4.2 Příklady použití .....	12
4.3 Základní druhy hashovacích funkcí .....	13
4.3.1 CRC .....	13

4.3.2	MD5.....	13
4.3.3	SHA .....	13
5	Prolomení šifrovacích algoritmů .....	14
5.1	Útok hrubou silou (brute force) .....	14
5.2	Slovníkové útoky.....	14
5.3	Kvantové počítače .....	16
5.4	Minimální požadavky na kryptografické algoritmy .....	16
6	Popis praktické části odborné práce – aplikace.....	17
6.1	Uživatelské rozhraní.....	18
6.2	Funkce – tabulka .....	18
6.3	Funkce – popis .....	19
6.4	Popis programového kódu .....	20
6.5	Použitá rozšíření a knihovny.....	22
	Závěr.....	24
	Seznam zkratk a odborných výrazů.....	25
	Seznam obrázků .....	26
	Použité zdroje.....	27
A.	Seznam příložených souborů.....	I

# Úvod

Toto téma jsem si zvolil především kvůli tomu, že je zajímavé, a co víc, také velmi důležité. A to nejen z historického hlediska, jak by mohlo mnohé napadnout, ale především v současné době. Dnes se šifruje v podstatě všechno – komunikace na internetu, databáze, soubory na harddiscích a jiných úložištích. Bez šifrování by to zkrátka nešlo. Dalším důvodem výběru bylo to, že se příští rok na této škole bude vyučovat předmět Kybernetická bezpečnost a praktická část této práce se bude moci použít jako podpůrný výukový materiál pro studenty.

S tímto tématem mám docela dost zkušeností, jelikož jsem se zúčastnil několika ročníků Soutěže v kybernetické bezpečnosti a některé šifrovací funkce jsem použil v rámci školních aplikací. Také mám kvalitní materiály z předchozího ročníku předmětu Operační systémy a sítě, kde byla tato problematika rozebírána. K realizaci aplikace jsem si vybral framework WPF, jelikož mi přijde z hlediska programování jednoduchý, srozumitelný i přívětivý, a navíc jsme jej minulý rok taktéž probírali.

Hned na úvod bych rovněž rád poděkoval vedoucímu práce Ing. Marku Pospíchalovi za to, že se uvolil tuto práci vést a Ing. Vladimíru Prokešovi za to, že ji bude oponovat.

# 1 ÚVOD DO KRYPTOLOGIE

Kryptologie je věda, která se zabývá šifrováním a dešifrováním informací. Jejím cílem je poskytovat bezpečnost a soukromí při přenosu dat mezi různými subjekty, a to tak, aby byla zabráněno jejich zneužití nebo ztrátě. Tato věda se využívá v mnoha různých oblastech, například v počítačové bezpečnosti, v komunikaci mezi vojenskými jednotkami nebo v běžném životě, například při bezpečném placení online nebo při ochraně soukromých informací v elektronické poště.

V současné době je kryptologie velmi důležitá, protože se informace přenášejí stále častěji prostřednictvím digitální komunikace a je nutné zajistit, aby byly chráněny před neoprávněným přístupem. Navíc s rozvojem internetu a různých online služeb se stále více otevírají nové možnosti pro hackerské útoky a je nutné tyto hrozby co nejlépe odrazit.

## 1.1 Vymezení pojmů

Jak již bylo zmíněno, **kryptologie** je věda, která se zabývá šifrováním a dešifrováním informací. Jinými slovy se dá říci, že je to věda zabývající se šifrováním, tvorbou a luštěním šifer. Zahrnuje v sobě dvě části, a to kryptografii a kryptoanalýzu. **Kryptografie** je věda o tvorbě šifer. Využívá metody šifrování, aby ukryla citlivé informace a data před nepovoleným přístupem. **Kryptoanalýza** je věda o luštění šifer. Zabývá se hledáním způsobů, jak šifrovaný text odšifrovat (dešifrovat).

Dále je třeba vysvětlit dva pojmy, které jsou často zaměňovány, a to šifrování a kódování. Oba termíny popisují proces transformace určité informace z jedné podoby do druhé, ale zatímco **šifrování** využívá utajené informace (např. klíče), **kódování** tyto informace nevyužívá.

## 1.2 Historie

### 1.2.1 Nejstarší šifrovací algoritmy

Kryptologie, jak ji známe dnes, má své kořeny až v roce 1900 př. n. l., kdy Egypťané používali speciální hieroglyfy pro zápis citlivých informací, aby se zajistilo, že k nim bude mít přístup pouze určená skupina lidí. Je třeba ale poznamenat, že i běžné hieroglyfy mohou být považovány za formu šifrování, protože mnoho lidí neumělo hieroglyfy ani číst, natož psát.

Staří Řekové hrají významnou roli v rozvoji šifrovacích metod. Okolo roku 350 př. n. l. navrhl vojevůdce Aeneas Tacticus asi 20 šifrovacích klíčů, které byly rozděleny do dvou skupin: transpoziční a substituční. Řecký historik Plútarchos zaznamenal vznik prvního transpozičního šifrovacího systému zvaného SKYTALA, který byl používán spartánskými vojevůdci kolem roku 500 př. n. l. Tento systém byl založen na jednoduchém principu: na dřevěnou hůl s přesným průměrem byla navinuta tenká kožená nebo pergamenová páska a na ni byla napsána tajná zpráva. Poté byla páska opět sundána z hole. Pouze osoba s holí



stejného průměru mohla přečíst šifrovanou zprávu. Pokud se zprávu pokusil číst někdo s holí jiného průměru, dostal pouze nesmyslnou změť písmen.

### **1.2.2 Steganografie**

Steganografie je technika, která se používá k utajení komunikace prostřednictvím skrytí zprávy. Toto utajení informací může být provedeno tak, aby si pozorovatel neuvědomil, že komunikace vůbec probíhá. Zpráva může být ukryta v různých formátech, jako jsou obrázky, zvukové soubory nebo videa. V minulosti se často používaly neviditelné inkousty. Steganografie se často používá spolu s kryptografií, kdy se informace nejen skrývají, ale také šifrují, aby se zvýšila úroveň ochrany před neoprávněným přístupem a umožnila tajná komunikace.

### **1.2.3 Enigma**

Enigma je šifrovací stroj, který byl používán za druhé světové války německou armádou k šifrování tajných informací. Název pochází z latiny a znamená „hádanka“ nebo „záhada“. Byla vyvinuta německým inženýrem, který se jmenoval Arthur Scherbius, a byla patentována v roce 1918. V letech 1926 a 1928 ji německé námořnictvo používalo v upravené verzi.

Enigma se skládá z klávesnice, svítící desky s písmeny a šifrovacího mechanismu, který obsahuje dvě krajní kola s 26 kontakty a tři vnitřní kola s pružinovými a plochými konektory. Při stisknutí písmena na klávesnici se první kolo otočí o jednu pozici. Poté, co se první kolo otočí 26x, se otočí druhé a nakonec třetí. Tím se získá celkem 17 576 různých stavů. Pro ztížení práce kryptoanalytikům byla délka zprávy omezena na několik písmen.

Enigma byla velmi účinným nástrojem pro šifrování informací během války a její dešifrování vyžadovalo velké úsilí a vynalézavost ze strany kryptoanalytiků. V roce 1941 se britským kryptoanalytikům podařilo dešifrovat Enigmu díky týmu, který se skládal zejména z matematiků a logiků a který pracoval v tajném středisku Bletchley Park. Tento tým vyvinul komplikované dešifrovací zařízení zvané Bomba, které dokázalo procházet velké množství možných kombinací klíčů a dešifrovat tajné zprávy. Díky dešifrování Enigmy se britským tajným službám podařilo získat cenné informace o německých plánech.

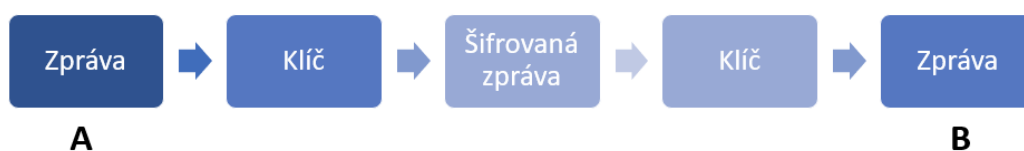
Enigma je dodnes považována za jednu z nejúčinnějších šifrovacích metod a stala se inspirací pro mnoho pozdějších šifrovacích systémů.

## 2 SYMETRICKÉ A ASYMETRICKÉ ŠIFROVÁNÍ

Šifrování se obecně rozděluje na dva základní typy, a to symetrické a asymetrické. Tyto dva druhy se liší ve způsobu použití šifrovacích klíčů.

### 2.1 Symetrická šifra

Symetrická šifra je založena na použití **jednoho klíče** pro šifrování a dešifrování zprávy. Patří mezi nejstarší a nejběžnější druhy šifrování. Mezi její hlavní výhody patří rychlá a snadná implementace, schopnost zpracovat velké množství dat v krátkém čase a nízké nároky na výkon systému. Symetrické šifry se dají použít jako základ pro vytvoření různých kryptografických mechanismů jako např. generátor pseudonáhodných čísel nebo hashovací funkce. Mezi tento druh šifer patří mj. Vigenérova a Vernamova šifra, DES nebo AES.



Obrázek 1 Schéma symetrického šifrování

#### 2.1.1 Proudová šifra

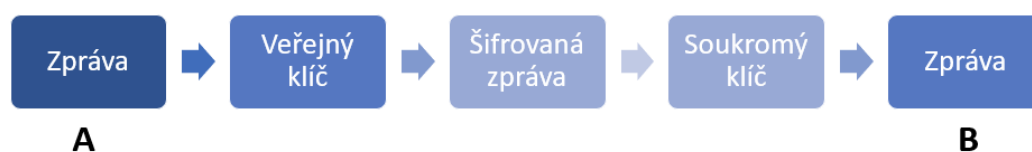
Proudová šifra se používá pro šifrování **proudů dat**, jako jsou například audio nebo videosoubory. Na začátku se provede inicializace pomocí krátkého klíče (*seed*) a tento klíč je posléze „natažen“ do dlouhého proudu klíče (*keystream*). Principem je tedy šifrování dat v reálném čase po malých částech (např. bit po bitu, příp. bajt po bajtu). Mezi proudové šifry se řadí např. *RC4*, či šifry z projektu **eSTREAM**.

#### 2.1.2 Blokovaná šifra

Blokovaná šifra se používá pro šifrování **pevně velkých částí dat (bloků)**, jako jsou například textové nebo binární soubory. Tyto bloky mohou být dlouhé např. 64, 128 nebo 256 bitů. Mezi blokové šifry patří mj. *DES*, *AES* či *RC5*.

## 2.2 Asymetrická šifra

Asymetrická šifra využívá pro šifrování a dešifrování **dva různé klíče**, veřejný klíč a soukromý klíč. Veřejný klíč slouží k šifrování dat a soukromý klíč k dešifrování. Vztah mezi soukromým a veřejným klíčem je výpočetně velice náročný, a tak není možné v reálném čase získat z veřejného klíče soukromý. Často je veřejný klíč kratší než soukromý. Metoda asymetrického šifrování je tedy bezpečnější, protože pokud by útočník získal pouze veřejný klíč, nemohl by data dešifrovat. Nevýhodou je rychlost, asymetrická šifra je mnohonásobně pomalejší než symetrická. Další nevýhodou je nutnost ověření pravosti klíče, ale pro tyto účely existují certifikační autority. Nejznámějším zástupcem asymetrických šifer je *RSA*.



Obrázek 2 Schéma asymetrického šifrování

### 3 ZÁKLADNÍ ROZDĚLENÍ ŠIFER

Šifry se rozdělují na několik základních druhů podle použitého způsobu šifrování.

#### 3.1 Substituční šifry

Substituční šifry jsou jedním z nejjednodušších typů šifer. Používá se zde pouze jeden klíč, jedná se tedy o symetrické šifrování. Princip substitučních šifer je takový, že každý znak zdrojového textu je nahrazen jiným znakem podle určitého klíče. Výhodou je jednoduchá implementace. Nevýhodou je snadnost, s jakou lze šifru prolomit. Proto se používá spíše jako ukázka jednoduchého principu šifrování než jako zabezpečení skutečných informací. Pro většinu účelů se dnes používají složitější šifry, které jsou odolnější vůči útokům.

##### 3.1.1 Monoalfabetické šifry

Monoalfabetické šifry jsou typem substitučních šifer, kde se používá jeden klíč pro všechny znaky ve zdrojovém textu. To znamená, že při šifrování je každý znak ve zdrojovém textu nahrazován jedním pevně stanoveným znakem v šifrovaném textu.

###### 3.1.1.1 Caesarova šifra

Caesarova šifra je jednou z nejstarších a nejjednodušších monoalfabetických šifer. Princip spočívá v posunutí každého znaku v abecedě o určitý počet míst. Klíčem je zde tedy číslo – počet míst. Tuto šifru používal Julius Caesar při vojenské komunikaci.

Jako příklad mějme slovo „AHOJ“. Pokud je klíčem např. číslo 9, posuneme každé písmeno ve slově o devět míst.

původní slovo	A	H	O	J
posun o 9 míst	J	Q	X	S

Variantou Caesarovy šifry je např. ROT13. Tato šifra posouvá každý znak vždy o 13 míst v abecedě.

###### 3.1.1.2 Augustova šifra

Augustova šifra je velmi podobná Caesarově šifře. Liší se v tom, že každý znak se vždy posouvá o jedno místo v abecedě dopředu. Písmeno „Z“ se překládá na „AA“. O vznik této šifry se zasloužil Caesarův synovec Augustus.

Jako příklad mějme slovo „ZIMA“.

původní slovo	Z	I	M	A
šifrovaný text	AA	J	N	B

### 3.1.1.3 Atbash

Princip této šifry spočívá v tom, že se každé písmeno v abecedě nahradí písmenem, které se nachází na druhém konci abecedy. Ve své podstatě se jedná o zrcadlový efekt. Funkce pro šifrování i dešifrování jsou tedy ekvivalentní. Atbash šifra byla používána v židovské tradici k šifrování hebrejských textů.

Jako příklad mějme slovo „PANDA“.

původní slovo	P	A	N	D	A
šifrovaný text	K	Z	M	W	Z

### 3.1.2 Polyalfabetické šifry

Polyalfabetické šifry jsou typem šifer, které používají více než jednu abecedu (šifrovací tabulku) pro šifrování. Dochází k několika různým substitucím v různých pozicích. Tyto šifry jsou složitější než monoalfabetické šifry, a tím pádem odolnější vůči útokům. Používaly se pro komunikaci během válek a pro utajování důležitých informací. Mezi polyalfabetické šifry patří i Enigma.

#### 3.1.2.1 Vigenérova šifra

Vigenérova šifra používá k šifrování klíč, který je tvořen jednoduchým slovem nebo frází. Klíč se používá k určení, která abeceda se má použít pro šifrování každého písmena ve zdrojovém textu. Jinými slovy se dá říci, že každé písmeno zprávy se posouvá o tolik pozic, o kolik je posunuto příslušné písmeno v klíči (vzhledem k písmenu „A“ v abecedě). Pokud je délka klíče kratší než délka zprávy, klíč se řetězí za sebou – to je často nevýhodné, jelikož různé části zprávy mohou být zašifrovány stejným klíčem. Výhodou je, že je tato šifra odolná proti útokům hrubou silou, jelikož potenciálních klíčů je opravdu mnoho.

Osvědčenou pomůckou používanou pro šifrování a dešifrování je tzv. Vigeněrův čtverec.

	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

Jako příklad mějme slovo „POTEMNICEK“. Klíčem, necht' je slovo „BROUK“. Jak jsem již zmínil, pokud je klíč kratší než text k zašifrování, musíme jej zřetěžit.

původní slovo	P	O	T	E	M	N	I	C	E	K
klíč	B	R	O	U	K	B	R	O	U	K
šifrovaný text	Q	F	H	Y	W	O	Z	Q	Y	U

Písmeno „B“ v klíči je posunuto o jednu pozici, proto musím posunout o jednu pozici i písmeno „P“. Vyjde mi „Q“.

Písmeno „R“ v klíči je posunuto o sedmnáct pozic, proto musím o stejný počet pozic posunout i písmeno „O“. Vyjde mi „F“.

A takto bych mohl pokračovat dále. Není na škodu pro tyto účely použít výše zmíněný čtverec, ale lepší je samozřejmě použít již hotový nástroj.

### 3.1.3 Ostatní substituční šifry

#### 3.1.3.1 Vernamova šifra (one-time-pad)

Vernamova šifra, také známá jako one-time-pad, je typem šifry, která používá jednorázový klíč k šifrování zdrojového textu. Tento klíč je stejně dlouhý jako zdrojový text a je tvořen náhodnými čísly (většinou je to posloupnost čísel). Pro každé písmeno zdrojového textu se provede posun o odpovídající počet znaků, které jsou v klíči. Jedná se o proudovou šifru. Existuje také binární varianta, kde je klíčem posloupnost nul a jedniček.

Tato šifra se často považuje za neprolomitelnou a nejbezpečnější, protože pokud je klíč správně vygenerován a použit, není možné šifrovaný text prolomit ani pomocí nejmodernějších kryptoanalytických metod. Nicméně, jednou z největších slabin Vernamovy šifry je skutečnost, že obě strany potřebují zabezpečený kanál pro přenos klíče. Pokud klíč unikne, šifrovaný text se stává bezcenným.

Vernamova šifra se dnes používá především v oblasti zabezpečeného přenosu informací, jako například v bankovníctví. Je také základem pro některé typy moderních šifer, jako například pro šifry z projektu eSTREAM.

### 3.2 Transpoziční šifry

Transpoziční šifry jsou jednou z nejstarších a nejjednodušších forem šifrování. Tyto šifry fungují tak, že se původní text převede na jiný text prostřednictvím přesouvání jednotlivých písmen. Často se používají pro šifrování krátkých textů a jsou docela snadno odhalitelné. Nicméně, v kombinaci s jinými šiframi mohou být transpoziční šifry užitečným nástrojem pro šifrování a ochranu dat.

### 3.3 Ostatní druhy šifer a kódování

#### 3.3.1 AES

AES je standardizovaný algoritmus, který byl oficiálně schválen v listopadu roku 2001. Výsledkem je symetrická bloková šifra, která šifruje a dešifruje data rozdělená do bloků dané délky (128 bitů). Jsou zde podporovány tři různé délky klíče, a to 128, 192 a 256 bitů. Obecně platí, že čím delší klíč, tím složitější a bezpečnější je šifra.

Proces šifrování a dešifrování se skládá ze čtyř základních částí – **expanze klíče**, **inicializace**, **iterace** a **závěr**. Expanze klíče se stará o vytvoření podklíčů z původního klíče. V inicializační části se každý stavový bajt zkombinuje s podklíčem pomocí operace XOR. Iterační část se skládá z 9, 11 nebo 13 iterací (závisí na délce klíče). V této fázi se provádějí operace jako záměna bajtů, prohození řádků a kombinování sloupců. Tyto kroky zajišťují nelineárnost a složitost šifrování. Závěrečná část se skládá z kroku záměny bajtů, prohození řádků a přidání podklíče. Tyto kroky zajišťují, že šifrovaná data jsou správně dešifrovatelná pomocí stejného klíče.

AES se používá v mnoha oblastech, kupř. k šifrování dat na pevných discích, při přenosu dat přes internet, pro zabezpečení Wi-Fi sítí pomocí WPA2 nebo k zabezpečení platebních transakcí.

#### 3.3.2 RSA

**RSA** (zkratka ze jmen autorů Rivest-Shamir-Adleman) je asymetrický šifrovací systém. Používá dva klíče, a to soukromý a veřejný. Veřejný klíč se používá k šifrování a soukromý k dešifrování. Tyto klíče spolu matematicky souvisí, ale není možné jednoduše získat z jednoho klíče druhý. Vždy je ovšem nutné mít klíčový pár vygenerovaný. Princip fungování RSA je celkem složitá záležitost, obecně je založen na faktorizaci velkých čísel, což je v nejčastější podobě rozklad na součin prvočísel. Dále se zde používá například Eulerova funkce. Pro obecný popis ale není potřeba rozpitvávat tento algoritmus do hloubky.

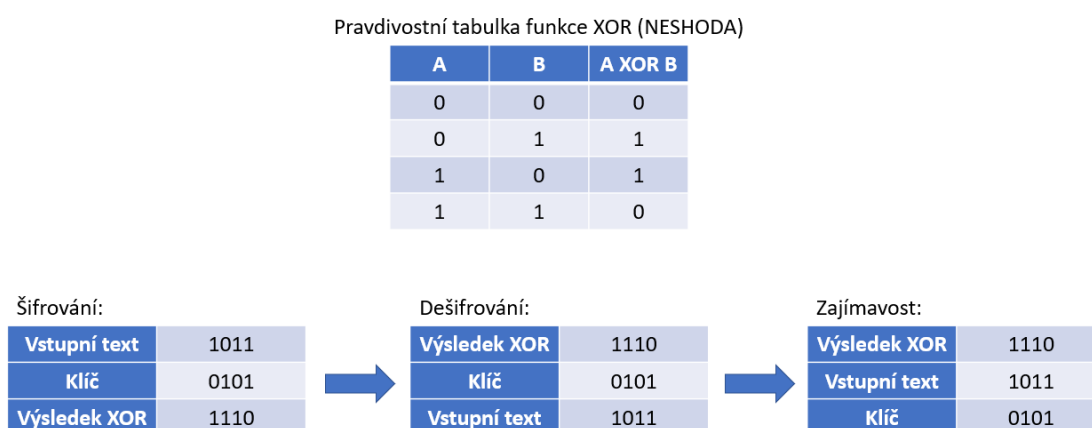
RSA je jedním z nejstarších a nejpobulárnějších kryptosystémů a často se používá k zabezpečení SSL/TLS připojení, při zabezpečení a ověření digitálního podpisu a šifrování e-mailů. Klíče v RSA mohou mít velkou délku a tím se zvyšuje jejich bezpečnost. Navzdory tomu, že RSA není tak rychlý jako některé jiné systémy, stále se jedná o silný a spolehlivý šifrovací systém.

#### 3.3.3 Morseovka (Morseova abeceda)

Morseovka není přímo šifra, nýbrž kódovací metoda, jelikož nevyužívá žádných utajených informací. Jedná se o převod písmen, čísel a symbolů do kombinace teček a čárek podle slovníku. Tyto kombinace představují krátké a dlouhé signály, které se mohou přenášet zvukově (vytūkávání), opticky (světelné záblesky), nebo elektricky (telegraf).

### 3.3.4 XOR

XOR neboli exkluzivní součet je logická operace, která se často používá při šifrování dat. Při použití XOR se původní text šifruje s použitím klíče, který se používá jako druhý operand při provádění operace. Výsledkem šifrování je nový text, který není možné jednoduše dešifrovat bez použití stejného klíče. Toto šifrování je založeno na vlastnosti exkluzivního součtu, kdy použití stejného klíče pro šifrování a dešifrování způsobí, že původní text je obnoven. Šifrování pomocí XOR je jednoduché a rychlé, nicméně není považováno za bezpečné pro šifrování velkých množství dat, protože existuje mnoho způsobů, jak tuto šifru prolomit.



Obrázek 4 Ukázka jednoduchého šifrování pomocí XOR

### 3.3.5 ASCII kód

ASCII kódování je standardní kódování znaků, které se používá k reprezentaci textu v informatice. Jedná se v podstatě o tabulku, která definuje znaky anglické abecedy, čísla, symboly a řídicí kódy. Těchto ASCII znaků je celkově 128. Každý z nich má přiřazený jednoznačný číselný kód v rozsahu 0 až 127. Tyto číselné kódy se používají mj. i k přenosu textu přes síť. ASCII kód se stále často používá jako základní kódování pro textové soubory, i když existují i jiná, více rozšířená a výkonnější kódování, jako například **Unicode**.



Níže je přiložena tabulka se shrnutím skupin znaků ASCII tabulky.

Desítkově	Hexadecimálně	Význam
0 až 31	0 až 1F	řídící kódy, netisknutelné znaky
32 až 47	20 až 2F	symboly
48 až 57	30 až 39	číslíce
58 až 64	3A až 40	symboly
65 až 90	41 až 5A	velká písmena
91 až 96	5B až 60	symboly
97 až 122	61 až 7A	malá písmena
123 až 126	7B až 7E	symboly
127	7F	DEL (delete)

### 3.4 Frekvenční analýza

Frekvenční analýza je kryptoanalytická technika, která využívá statistických vlastností jazyka k odhalení šifrovaného textu. Funguje na základě skutečnosti, že určitá slova nebo znaky se v jazyce vyskytují častěji než jiná. Tyto statistické vlastnosti se používají k porovnání šifrovaného textu s očekávanými frekvencemi jednotlivých znaků a k určení, jaký znak se nachází na určité pozici v šifrovaném textu.

Frekvenční analýza je často používána k odhalení monoalfabetických šifer, jako je například Caesarova šifra, ale může být také účinná proti určitým druhům polyalfabetických šifer, jako je například Vigenérova šifra. Je důležité si uvědomit, že frekvenční analýza není všemocná a že existují šifry, proti kterým není účinná. Tyto šifry jsou složitější a odolnější vůči útokům.

## 4 HASHOVACÍ FUNKCE

Hashovací funkce jsou matematické funkce (resp. algoritmy), které převádějí vstupní data (jako např. text, soubory atd.) na unikátní a pevně daný výstup, který se nazývá **hash** nebo též **otisk**. Tyto funkce se používají k zabezpečení a ověření integrity dat.

### 4.1 Požadavky hashovacích funkcí

Hashovací funkce a výsledný hash musí splňovat několik požadavků, aby byly použitelné pro zabezpečení informací.

1. **Jednoznačnost** – hashovací funkce musí pro každý vstup produkovat jedinečný výstup – hash (otisk)
2. **Fixní délka výstupu** – hashovací funkce musí fungovat pro libovolně dlouhý vstup, ale výstup musí mít stále stejnou délku
3. **Nepřevoditelnost** – není možné algoritmickou cestou z hashe získat zpět původní vstup
4. **Unikátnost** – malé změny ve vstupu způsobují velké změny ve výstupu
5. **Rychlost** – hashovací funkce by měly být rychle výpočetně proveditelné, aby se mohly používat v praktických aplikacích

Pokud jsou všechny tyto požadavky splněny, hash je bezpečný a můžeme jej použít pro verifikaci autenticity a integrity informací.

#### 4.1.1 Kolize hashů

Kolize hashů nastává, když hashovací funkce vypočítá z více zdrojových dat shodný hash. Tato data samozřejmě nemusí být nijak podobná. Ke kolizím by rozhodně docházet nemělo, a pokud se tak stane, znamená to, že použitá hashovací funkce není bezpečná.

### 4.2 Příklady použití

Hash má mnoho využití, ale nejčastěji se využívá při ukládání hesel do databáze. Kdyby se totiž do databáze uložilo heslo tak, jak ho zadává uživatel (např. „bobik“), útočník by se při napadení databáze okamžitě heslo dozvěděl. Proto se z hesla před uložením vypočítá hash (např. MD5 4ddc3a9aaa4de72ea38333044cbecaaa), a ten se do databáze uloží. Toto by kdysi stačilo, ale dnes už nikoliv, jelikož existují poměrně rozsáhlé slovníky, které obsahují seznam prolomených hashů (např. CrackStation, HashCat atd.). Proto se ještě používají metody jako je tzv. salting („posolení“), kdy se k hashovanému heslu navíc přidá náhodný řetězec (sůl), aby se zabránilo slovníkovým útokům.

Dalším využitím je již výše zmíněné ověření integrity dat. Je to v podstatě proces, při kterém zjišťujeme, zda data nebyla během jejich přenosu či ukládání poškozena nebo změněna. Hashovací funkce se zde využívají ke srovnání stavu původních dat a stavu po přenosu.

Pokud se hodnoty hashů shodují, je téměř jisté, že data nebyla nijak upravována. Naopak, pokud se budou hodnoty hashů lišit, mohlo dojít k poškození či napadení dat při přenosu.

### 4.3 Základní druhy hashovacích funkcí

Níže jsou stručně popsány základní druhy hashovacích funkcí. Možná se ptáte, proč jsem nepopisoval tyto druhy podrobněji. Je to z toho důvodu, že algoritmy, které se k vytvoření těchto hashů používají, jsou technicky i výpočetně velice náročné a jejich detailní rozbor by razantně přesahoval rozsah této odborné práce.

#### 4.3.1 CRC

**Cyklický redundantní součet** je hashovací funkce, která se používá výhradně k detekci chyb během přenosu či ukládání dat. Jedná se o realizaci kontrolního součtu. Obecně platí, že kontrolní součet se odesílá s daty a po převzetí dat je znovu nezávisle spočítán. Pokud dojde k chybě při přenosu dat, bude kontrolní součet přijatých dat odlišný od výpočtu, a tím se zjistí, že došlo k chybě. Nejčastěji je používána varianta **CRC-32**.

#### 4.3.2 MD5

**MD5** je jednou z nejběžnějších hashovacích funkcí. Výsledný hash má délku 128 bitů, které jsou zapsány v hexadecimální soustavě. Funkce je velmi rychlá a efektivní při ověřování dat, ale má hlavní bezpečnostní nedostatek, a to výslednou velikost hashe. Kvůli relativně malé délce výsledného hashe je tu vyšší pravděpodobnost kolize, proto se nedoporučuje tuto funkci nadále používat.

#### 4.3.3 SHA

SHA je skupina celkem bezpečných hashovacích funkcí, které byly vyvinuty americkou Národní bezpečnostní agenturou (NSA). Dělí se na tři základní typy, a to SHA-1, SHA-2 a SHA-3. Je důležité uvést, že SHA-2 je souhrnné označení pro algoritmy SHA-224, SHA-256, SHA-384 a SHA-512. Níže je uvedena přehledná tabulka s výslednou délkou hashe u jednotlivých funkcí.

Typ	Délka
SHA-1	160 bitů
SHA-224	224 bitů
SHA-256	256 bitů
SHA-384	384 bitů
SHA-512	512 bitů
SHA-3	není přesně definována, nejméně 224 bitů, nejvíce 512 bitů

## 5 PROLOMENÍ ŠIFROVACÍCH ALGORITMŮ

Prolomení šifrovacích algoritmů je proces, při kterém se útočník pokouší dešifrovat zašifrovaný text bez znalosti klíče použitého k jeho šifrování. Často se jedná o velice obtížný proces, protože většina šifrovacích algoritmů, především hashovací funkce, jsou navrženy tak, aby bylo velmi těžké dešifrovat zašifrovaný text bez znalosti klíče. Existují různé metody, jak lze šifrovací algoritmy prolomit, používá se například útok hrubou silou (brute force), slovníkové útoky, nebo se využívá chyb v samotném algoritmu. Je důležité mít na paměti, že prolomení šifrovacích algoritmů může mít závažné důsledky pro soukromí a bezpečnost, a proto se často používají různé metody, jako je například kryptografická kontrola integrity (třeba kontrolní součty), aby se zabránilo neoprávněnému prolomení šifrovacích algoritmů.

### 5.1 Útok hrubou silou (brute force)

Brute force útok je prolamovací metoda, při které se útočník pokouší dešifrovat zašifrovaný text tím, že postupně zkouší všechny možné kombinace klíčů, dokud nenajde ten správný. Tento způsob je časově velmi náročný a může trvat i desítky nebo stovky let, v závislosti na síle šifrovacího algoritmu a délce klíče. Navzdory tomu, že útok hrubou silou může být účinný proti slabším šifrovacím algoritmům nebo krátkým klíčům, většina moderních šifrovacích algoritmů je navržena tak, aby byly odolné proti těmto útokům.

Útok hrubou silou se v praxi používá například k prolomení hesla u šifrovaných souborů a archivů.

### 5.2 Slovníkové útoky

Slovníkový útok (anglicky dictionary attack) je metoda, při které se použije seznam předem připravených hesel (tzv. slovník) k postupnému zkoušení jednotlivých hesel, dokud se nenajde to správné. Tento útok je často účinný, protože mnoho lidí používá hesla, která jsou snadno uhodnutelná nebo se nacházejí ve slovníku. Slovníkový útok se může použít k prolomení různých typů hesel, jako je například heslo od e-mailové schránky, sociální sítě nebo počítačové heslo. Navzdory své účinnosti proti snadno uhodnutelným heslům je slovníkový útok méně účinný proti silným a náhodně vygenerovaným heslům. Pro zvýšení bezpečnosti je proto doporučeno používat silná a náhodná hesla, která nejsou snadno napadnutelná.

Slovníkový útok se dá použít i proti archivu. Níže je přiložen kód v jazyce C#, který tuto operaci provede. Je nutné doinstalovat balíček *DotNetZip*.

```

using Ionic.Zip;

string zipPath = "HowMakeAbomb.zip";
string[] passwords = File.ReadAllLines("dict.txt");

int passwordCount = passwords.Length;
int currentPassword = 0;
foreach (string password in passwords)
{
    currentPassword++;
    using ZipFile archive = new(zipPath);
    archive.Password = password;
    archive.Encryption = EncryptionAlgorithm.PkzipWeak;
    try
    {
        archive.ExtractAll("dir", ExtractExistingFileAction.Throw);
        Console.WriteLine("Heslo " + password);
        break;
    }
    catch (BadPasswordException)
    {
        Console.WriteLine("Špatné heslo " + password + " (" +
currentPassword + "/" + passwordCount + ")");
    }
    catch
    {
        Console.WriteLine("Jiná chyba");
    }
}

```

Na začátku je deklarována a inicializována proměnná *zipPath*, která obsahuje cestu k archivu. Proměnná *passwords* obsahuje cestu ke slovníku, ve kterém se nachází seznam předem připravených hesel, která budou zkoušena jako možné heslo pro archiv. Kód dále obsahuje smyčku *foreach*, ve které se prochází každé heslo v poli *passwords*. Proměnné *currentPassword* a *passwordCount* slouží k zobrazení informace o tom, kolikáté heslo se právě zkouší a kolik hesel je celkem v seznamu. Uvnitř smyčky se vytvoří nový objekt typu *ZipFile*, který představuje archiv, a k němu se nastaví vlastnost *Password* na aktuálně zkoušené heslo. Dále se nastaví vlastnost *Encryption* na **PkzipWeak**, která se používá ke klasickému slovníkovému útoku. Následuje blok *try-catch*, v jehož první větvi se snažíme extrahovat archiv do složky **dir**. Pokud se to povede, heslo se vypíše na konzoli a program se ukončí. Jestliže je aktuální heslo chybné, skočí se do první větve *catch*, vypíše se chybné heslo a pokračuje se ve zkoušení. Kdyby nastala jiná chyba, tak je ošetřena v poslední větvi bloku.

Tento algoritmus je názornou ukázkou slovníkového útoku. Bohužel ale není tak rychlý a efektivní, jak bychom potřebovali. Poněkud vhodnějšími prostředky pro slovníkové i jiné útoky jsou nástroje obsažené v **Kali Linuxu**. Mezi ně patří například *John the Ripper*, *fercrackzip*, *hashcat* nebo *hydra*. Tyto nástroje nepracují tak, že by vnitřně zkoušely rozbalovat archiv (viz výše), ale porovnávají pouze vypočítaný hash z archivu s hashem aktuálního slova z databáze. Tímto se tedy celý proces zrychlí, a pokud použijeme kvalitní slovník, tak máme heslo do několika málo minut či sekund.

### 5.3 Kvantové počítače

Kvantové počítače jsou novým typem počítačů, které využívají zákony kvantové mechaniky k výpočtům. Na rozdíl od klasických počítačů, které pracují s bity, kvantové počítače pracují s kvantovými bity, nazývanými **qubity**. Qubity mohou být v různých stavech současně a dokáží provést více výpočtů najednou, což kvantovým počítačům dává možnost vyřešit určité problémy mnohem rychleji než klasické počítače.

Kvantové počítače mají také obrovský potenciál prolomit některé šifry a hashe, u kterých by prolamování na běžném počítači zabralo věky. Mohly by prolomit například RSA šifrování, které je založeno na obtížnosti faktorizace velkých čísel. Další použití by našly při prohledávání velkých databází hesel nebo při útoku hrubou silou.

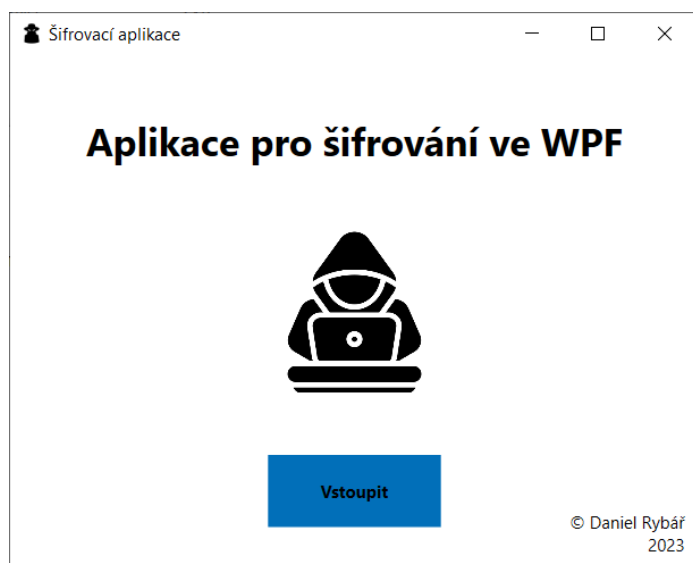
Nicméně je důležité poznamenat, že v současné době jsou kvantové počítače stále v rané fázi vývoje a praktické využití pro prolomení šifer a hashů může být ještě vzdálené. Kromě toho se v poslední době objevují nové algoritmy a šifry, které jsou odolnější proti kvantovým výpočtům a mohou být využity pro budoucí zabezpečení dat.

### 5.4 Minimální požadavky na kryptografické algoritmy

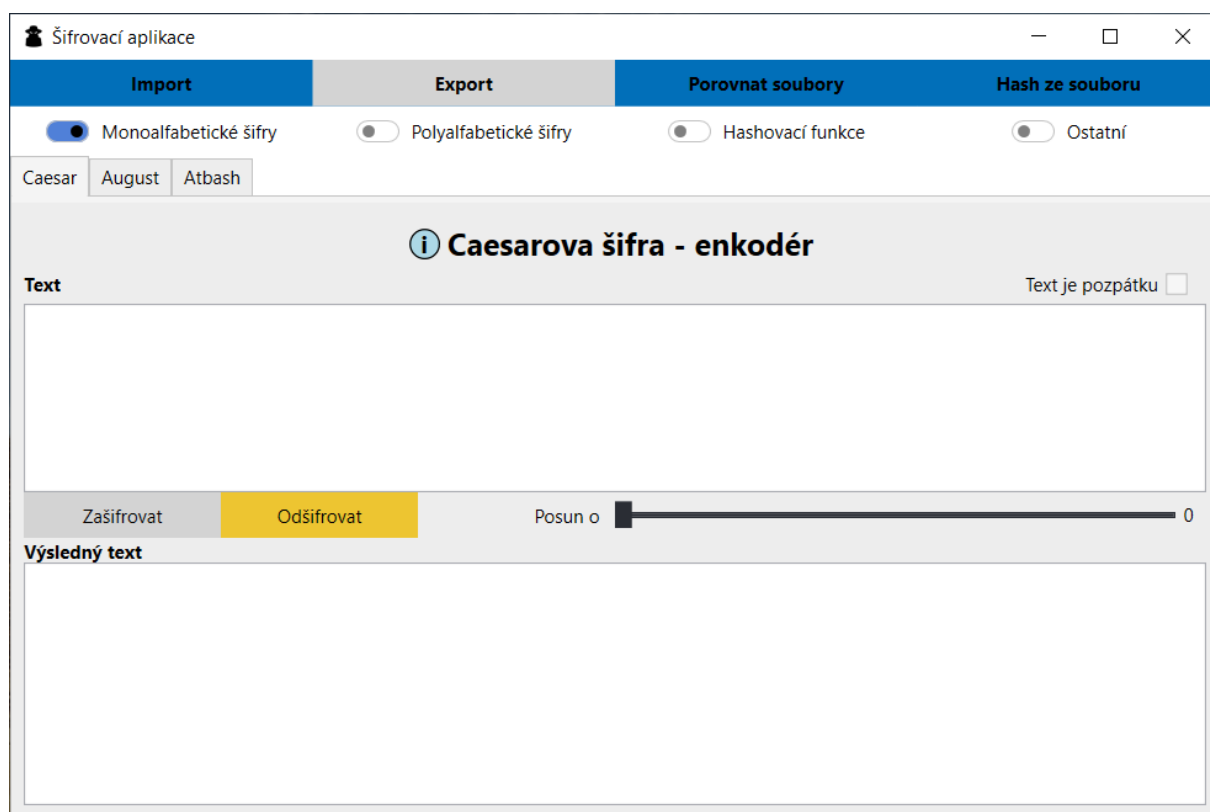
Podle NÚKIB je doporučeno upřednostňovat blokové šifry před proudovými, tj. používat např. *AES*, a to s optimální délkou klíče 256 bitů. Mezi dosluhující šifry patří *3DES*, *Blowfish* nebo *Kasumi*. Mezi schválené hashovací funkce patří mj. *SHA-2* nebo *Whirlpool*. Obecně se nedoporučuje používání hashů s výstupem menším než 160 bitů, může zde totiž docházet ke kolizím. Mezi takovéto „zastaralé“ hashovací funkce patří kupř. MD5 nebo MD4. Dále je doporučeno přejít od funkce SHA-1 (160 bitů) na novou generaci hashovacích funkcí SHA-2 (jelikož mají délku větší než 224 bitů).

Je důležité se zamyslet nad tím, že všechna výše zmíněná tvrzení jsou pouze doporučení a není nutné je bezpodmínečně dodržovat. Vždy je třeba zvážit konkrétní situaci a zvolit nejoptimálnější šifrovací algoritmus.

## 6 POPIS PRAKTICKÉ ČÁSTI ODBORNÉ PRÁCE – APLIKACE



Obrázek 6 Úvodní stránka a logo aplikace



Obrázek 5 Hlavní okno aplikace

Aplikaci jsem vytvořil v relativně moderním a inovativním *.NET* frameworku verze 6, který byl oficiálně uveden v listopadu roku 2021. Jako vývojové prostředí jsem použil **Visual Studio 2022**. Grafické uživatelské rozhraní je implementováno pomocí osvědčeného frameworku **WPF**, který zajišťuje intuitivní design a taktéž umožňuje snadnou interakci s uživatelem.

## 6.1 Uživatelské rozhraní

Vzhled aplikace je sladěný do dvou barev – modré a žluté. V horní části je menu složené z komponent *Button* a *Menu*, které umožňují vybrat souborovou operaci. Pod ním jsou přepínací tlačítka (*ToggleButton*), která navigují mezi jednotlivými druhy šifer. Při najetí myši na nějaké z těchto tlačítek se zobrazí stručný popis daného typu šifer. Hlavní komponentou aplikace je *TabControl*, který umožňuje docela elegantní přepínání mezi konkrétními šiframi. V podstatě to funguje jako navigace mezi jednotlivými kartami ve webovém prohlížeči. Většina těchto karet (*TabItem*) obsahuje čtyři základní komponenty, a to vstupní textové pole, dále tlačítka pro šifrování a dešifrování, a nakonec výstupní textové pole. U některých druhů šifer jsou umístěny zaškrťovací pole, tzv. *CheckBoxy*, která určují, zda je vstupní text napsán pozpátku. Dále je možné se setkat s kombinovanými poli (*ComboBox*), která umožňují výběr cílové číselné soustavy či počtu bitů. Někde jsou také umístěny posuvníky (*Slider*), pomocí kterých můžeme např. posouvat znaky u Caesarovy šifry. Všechny druhy šifer obsahují nadpis (*Label*) s informačním kolečkem. Po najetí myši na toto kolečko se zobrazí popis dané šifry. Tato komponenta se nazývá *ToolTip*.

Když se provádí nějaká operace, která trvá déle, tak se přes celé okno zobrazí načítací stránka. Tuto funkci s animační stránkou framework implicitně neobsahuje, takže je převzatá. Jmenuje se *busyIndicator*.

U tlačítek pro šifrování a dešifrování je možné si všimnout, že jedno tlačítko je vždy aktivní (*enabled*) a druhé zablokováno (*disabled*). Když je tlačítko zablokováno, znamená to, že je aktuálně vybraná tato funkce. Když je tedy kupř. zablokováno tlačítko pro zašifrování, tak je vybraný mód enkodéru. Když je zablokováno tlačítko pro dešifrování, je vybrán mód dekodéru. Podobný princip je použit i u tlačítek pro import a export souborů. Tlačítko pro export je aktivní pouze tehdy, když není výstupní pole prázdné. U šifer, kde nejsou tyto funkce podporovány, jsou obě tlačítka vždy zablokována.

## 6.2 Funkce – tabulka

Aplikace obsahuje několik základních funkcí, které souvisí s kryptologií. Přehled těchto funkcí je v následující tabulce.



Monoalfabetické šifry	Polyalfabetické šifry	Hashovací funkce	Ostatní	Soubory
Caesarova	Vigenérova	SHA-1 a SHA-2	Morseovka	Komparace souborových hashů
Augustova		MD5	ASCII kód	SHA-1 a SHA-2
Atbash			RSA	MD5
			Generátor čísel	CRC-32
			Ukázka brute-force	

### 6.3 Funkce – popis

U šifer je možné přepínat mezi módy enkodéru (pro zašifrování) a dekodéru (pro odšifrování). Výchozím stavem je enkodér. U většiny šifer také můžeme zvolit, zda je vstupní text pozpátku či nikoli.

Caesarova šifra obsahuje posuvník, který určuje o kolik znaků abecedy má být text posunut. U šifry Atbash jsou funkce pro šifrování a dešifrování ekvivalentní, jelikož se písmena převádějí na písmena opačná v abecedě. Vigenérova šifra obsahuje navíc textové pole, do kterého se zadává klíč.

U SHA se vybírá výsledná délka hashe. Ta může být 160 bitů (SHA-1), 256 bitů (SHA-256), 384 bitů (SHA-384) nebo 512 bitů (SHA-512). Nejkomplikovanější funkcí z celé aplikace je nepochybně výpočet RSA. Pro použití šifrovací funkce je nutné nejprve vygenerovat klíčový pár, který se skládá z veřejného a soukromého klíče. Je důležité říci, že tento pár nelze do polí nakopírovat, musí být opravdu vygenerován přímo v aplikaci. Můžeme si ale zvolit délku klíčů, a to v rozmezí od 520 do 4096 bitů. Dále je možné klíče samostatně exportovat do souboru. Pak už stačí jen přepnout mezi šifrovacími režimy. U RSA systému je soukromý klíč používán k dešifrování zpráv a jejich podepisování, zatímco veřejný klíč slouží pro šifrování zpráv a ověřování podpisů.

U Morseovky je možné pro větší přehlednost zvolit oddělování jednotlivých slov lomítkem. Na stránce pro ASCII kódování je možné zvolit si cílovou soustavu (případně vstupní v režimu dekodéru), a to buď desítkovou, hexadecimální, binární, nebo osmičkovou. Implicitně je vybrána binární soustava.


Na další stránce pro generování pseudonáhodných čísel je nutné zvolit rozsah ve kterém se má číslo vygenerovat. Je možné tak učinit pomocí posuvníků, které mají limit 10 000, či napsáním hodnoty do textového pole. Dá se také zvolit, zda chcete generovat desetinné číslo. V takovém případě se generuje desetinné číslo, které je zaokrouhlené na tisíce. Další funkcí

je generování číselné řady. Rozsah se opět nastaví pomocí posuvníků či textových polí. Je také samozřejmě nutné zadat počet čísel v této řadě, jinak generace není možná.

V další sekci je znázorněna ukázka brute-force útoku. Do prvního textového pole se zadají znaky, ze kterých mají být následně sestaveny kombinace. Do dalšího pole se zadá délka slov, které mají být sestaveny. Je důležité upozornit, že maximální délka slov, která se mají generovat, je šest znaků. Je to zapříčiněno časovou náročností funkce pro vytváření kombinací. Jako výsledek po generování se zobrazí všechny možné kombinace, které jdou ze znaků sestavit. Dále je uveden počet těchto kombinací a celková doba trvání demonstrace.

Jak již bylo zmíněno, je možné použít funkce pro import a export textových souborů. Obsah souboru se importuje do vstupního textového pole stránky, na které se uživatel aktuálně nachází. Export je možný pouze tehdy, když není výstupní textové pole prázdné. Do textového souboru se u většiny zaznamená název funkce, režim, vstupní text, výstupní text a který z těchto textů je šifrovaný. Dále se zaznamenají dodatečné informace, např. jestli je text pozpátku nebo cílová číselná soustava.

Funkce pro výpočet hashe ze souboru vytvoří příslušný hash ze zvoleného textového souboru a do stejné složky exportuje výstupní soubor. Tento výstupní soubor obsahuje samotný hash a jeho typ, dále název, cestu a velikost vstupního souboru v bajtech. Vybere-li si uživatel variantu „Všechny předchozí“, ze vstupního souboru se vypočítají hashe, které jsou uvedeny v tabulce výše (sloupec Soubory). Po dokončení a kliknutí na tlačítko „OK“ se výstupní soubor hned otevře.

 md5.txt – Poznámkový blok

Soubor   Úpravy   Formát   Zobrazení   Nápověda

Hash: fed56340f3892c0ef0e9d27b184a4392

Typ hashe: MD5

Název souboru: Bez názvu.png

Cesta k souboru: C:\Users\danos\Downloads\mp\Bez názvu.png

Velikost souboru: 22721 B

*Obrázek 7 Ukázka výpočtu MD5 ze souboru*

Funkce pro porovnání souborů dle hashů vyžaduje zvolení dvou vstupních souborů. Z každého tohoto souboru se vypočítají všechny hashe a výsledek se exportuje do výstupního souboru. Ten mimo jiné i obsahuje verdikt, jestli se hashe souborů shodují či nikoli. Tento příklad by měl demonstrovat unikátnost hashů a detekovat možné kolize.

## 6.4 Popis programového kódu

Celý projekt obsahuje několik složek, které jsou pojmenovány podle konvencí. Ve složce *Converters* jsou umístěny konvertory, které slouží k převodu mezi jednotlivými datovými typy. Nejdůležitější konvertor v této složce je **BoolToModeConverter.cs**. Do této třídy se

posílají dvě hodnoty, a to hodnota aktuálního indexu (tj. aktuální karta), na které se uživatel zrovna nachází a režim šifrování (tj. dekodér – hodnota *True* či enkodér – hodnota *False*). Na základě těchto dvou informací nám konvertor vrátí název dané šifry a režim šifrování. Do konvertoru tedy posílám datové typy **integer** a **boolean**, zpět dostávám **řetězec** s názvem a režimem. Je důležité poznamenat, že je to velmi efektivní způsob, jak dosáhnout výsledku.

V dalším adresáři *Helpers* se nachází samotné kryptografické funkce. Jsou zde třídy pro výpočet souborových hashů či generování náhodného čísla. Je zde také soubor **BruteForce.cs**, který generuje již zmíněné kombinace ze znaků. Dále je tady soubor **Ciphers.cs**, což je třída, která obsahuje všechny textové kryptografické funkce. Pro ukázkou zde bude uvedena funkce pro šifrování pomocí Caesarovy šifry.

```
public static string CaesarEncode(string input, int shift)
{
    RemoveDiacritics(ref input);
    string output = String.Empty;
    foreach (char ch in input)
    {
        if (Char.IsLetter(ch))
        {
            char offset = char.IsUpper(ch) ? 'A' : 'a';
            output += (char)((((ch + shift - offset) % 26) + offset));
        }
        else
            output += ch;
    }
    return output;
}
```

Tato funkce přijímá dva parametry, a to vstupní text a posun. Jako první se odebere ze vstupního textu diakritika, poté se deklaruje a inicializuje výstupní proměnná. Následně se pomocí cyklu *foreach* prochází každý znak vstupního textu a zkoumá se, zda se jedná o písmeno. Pokud ano, písmeno se posune o hodnotu posunu pomocí výpočtu, který zajišťuje, že výsledné písmeno bude vždy v rozsahu A-Z nebo a-z, a to pomocí proměnné *offset*. Výsledné písmeno se pak přidá k výstupnímu řetězci. Pokud se znak vstupního textu nenachází v rozsahu A-Z nebo a-z, přidá se beze změny k výstupní proměnné. Na konci funkce se vrací výstupní řetězec.

V následujícím adresáři s názvem *Models* jsou výčtové typy, které definují druhy použitých šifer. Na těch nic zajímavého není.

Nejdůležitější adresář se jmenuje *ViewModels*. Jsou zde obecné třídy pro tzv. *Commandy*, které slouží k propojení akcí, které uživatel provede v uživatelském rozhraní (např. kliknutí na tlačítko), s logikou aplikace. Nachází se zde také hlavní třída **MainViewModel.cs**, ve které jsou *Commandy* inicializovány a která propojuje všechny šifrovací funkce a souborové operace s uživatelským rozhraním.

Poslední složkou jsou *Views*, kde jsou umístěny soubory pro definici uživatelského rozhraní. Obsahují kód pro zpracování událostí a jsou zodpovědné za zobrazování dat a za odesílání událostí z uživatelského rozhraní do *ViewModelu*.

Posledním důležitým souborem je **App.xaml**, kde jsou podle konvence umístěny styly jednotlivých komponent, které aplikace obsahuje. Níže je uveden příklad stylování komponenty *TextBox*.

```
<Style TargetType="{x:Type TextBox}" x:Key="minmax">
  <Setter Property="TextWrapping" Value="Wrap" />
  <Setter Property="HorizontalAlignment" Value="Stretch" />
  <Setter Property="VerticalAlignment" Value="Stretch" />
  <Setter Property="FontSize" Value="15"/>
  <Setter Property="TextAlignment" Value="Center" />
  <Setter Property="Margin" Value="5" />
  <Setter Property="VerticalContentAlignment" Value="Center" />
</Style>
```

Atribut *TargetType* znamená, pro jakou komponentu je styl určen. Následuje kolekce prvků *Setter*, které definují jednotlivé vlastnosti stylu. Pro přehlednost je uvedena tabulka.

Vlastnost	Funkce	Hodnota
TextWrapping	zalamování textu	zalamovat
HorizontalAlignment	vodorovné zarovnání	celá šířka (roztáhnout)
Vertical Alignment	svislé zarovnání	celá výška (roztáhnout)
FontSize	velikost písma	15
TextAlignment	zarovnání textu	střed
Margin	posun	5
VerticalContentAlignment	zarovnání obsahu komponenty	střed

## 6.5 Použitá rozšíření a knihovny

V aplikaci jsou použity celkem čtyři knihovny, které dodávají do aplikace různé funkce. Jelikož jazyk C# implicitně nepodporuje výpočet kontrolního součtu, je zde použita knihovna **Crc32.NET**, která tuto funkci nabízí. Dále je zde použit **BusyIndicator**, který vytváří animační stránku s načítacím logem, pokud je aplikace zaneprázdněna. Bohužel byl k němu nedostatek dokumentace, takže bylo nutné zanalyzovat, jak vlastně funguje. Nakonec jsem přišel na to, že se musí řídit pomocí paralelního programování v bloku *Task.Run(() => {...})*. Posledními rozšířeními jsou knihovny **AdonisUI** a **AdonisUI.ClassicTheme**. Tyto knihovny dodávají aplikaci jemnější vzhled a mají hezčí stylování některých komponent.

K vytvoření instalačního souboru aplikace jsem použil program **Advanced Installer**, který zdarma umožňuje export aplikace pouze do souboru MSI. Prvním krokem je, že se ve Visual Studiu musí vytvořit publikace projektu do složky. V instalačním programu se pak se zvolí, kde je tato publikace umístěna a které knihovny se mají do MSI souboru zabalit. Následně se dopíše název a autor aplikace, připojí se ikona a program vytvoří instalační soubor. Ten pak stačí otevřít a provede se klasická instalace se vším všudy.

Aplikace funguje v **64bitovém operačním systému Windows ve verzi 10 a vyšší**. Funkcionalitu ve starších verzích jsem z důvodu nedostupnosti jiného operačního systému netestoval.

## Závěr

Téma odborné práce jsem si zvolil, abych si vyzkoušel hlubší práci s frameworkem WPF a abych si rozšířil své znalosti o šifrování, které je v dnešní době opravdu důležité. Trochu jsem se obával, že moje znalosti nebudou k vytvoření této komplexní aplikace dostačující, ale nakonec jsem použil principy, které jsem znal ze školy. Na začátku jsem nevěděl, které šifry bude aplikace obsahovat, ani které funkce bude uživatelům nabízet. Moje představa byla okolo 10. Nakonec se mi povedlo vytvořit celkem 15 funkcionalit, které aplikace nabízí a všechny fungují tak, jak by měly.

Při testování aplikace jsem nejprve použil validní parametry, abych vyzkoušel, zda aplikace dělá, co se od ní očekává. Když toto fungovalo, zkusil jsem použít „zákeřné“ parametry jako například zadávání velkého množství bitů, vložení klíče s neplatnými znaky nebo třeba zadávání rozšiřující diakritiky. Nakonec jsem nezjistil žádný problém, a vše by tedy mělo fungovat správně.

Nejtěžší částí při vývoji aplikace bylo použití načítací stránky, respektive její řízení. Nikde jsem k tomu nenašel žádnou adekvátní dokumentaci, a tak jsem musel chvílemi improvizovat. Nakonec jsem přišel na to, že spuštění a skrytí této stránky se musí řídit pomocí paralelního programování.

Na závěr bych chtěl podotknout, že jsem velmi rád, že jsem při vytváření odborné práce neměl žádné vážnější potíže a splnil jsem všechny cíle, které jsem si na začátku dal.

# Seznam zkratk a odborných výrazů

## **WPF**

Windows Presentation Foundation – aplikační rámec, slouží jako podpora při programování, díky němu můžeme vytvořit aplikaci s grafickým uživatelským rozhraním.

## **NÚKIB**

Národní úřad pro kybernetickou a informační bezpečnost

## **Kali Linux**

Distribuce operačního systému Linux, která byla speciálně vytvořena pro potřeby bezpečnostních expertů, testerů a dalších profesionálů pracujících v oblasti IT bezpečnosti.

## **ASCII**

American Standard Code for Information Interchange („americký standardní kód pro výměnu informací“)

## **AES**

Advanced Encryption Standard („standard pokročilého šifrování“)

## **Wi-Fi**

Skupina protokolů pro bezdrátové sítě založená na standardech IEEE 802.11

## **CRC**

Cyclic Redundancy Check (cyklický redundantní součet)

## **MD5**

Message-Digest algorithm 5

## **SHA**

Secure Hash Algorithm

## Seznam obrázků

Obrázek 1 Schéma symetrického šifrování .....	4
Obrázek 2 Schéma asymetrického šifrování.....	5
Obrázek 3 Vigenèrův čtverec.....	7
Obrázek 4 Ukázka jednoduchého šifrování pomocí XOR .....	10
Obrázek 5 Hlavní okno aplikace .....	17
Obrázek 6 Úvodní stránka a logo aplikace .....	17
Obrázek 7 Ukázka výpočtu MD5 ze souboru .....	20



## Použité zdroje

1. **AdisonCavani**. AdisonCavani/RSA-WPF. *GitHub*. [Online] [Citace: 20. 11. 2022.]  
<https://github.com/AdisonCavani/RSA-WPF>.
2. **Rühl, Benjamin**. Adonis UI. *Adonis UI*. [Online] 2020. [Citace: 25. 11. 2022.]  
<https://benruehl.github.io/adonis-ui/>.
3. **Příspěvatelé, Wikipedie**. Advanced Encryption Standard. *Wikipedie: Otevřená encyklopedie*. [Online] 16. 11. 2022. [Citace: 11. 02. 2023.]  
[https://cs.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard](https://cs.wikipedia.org/wiki/Advanced_Encryption_Standard).
4. **Příspěvatelé, WikiSofia**. Asymetrická kryptografie. *WikiSofia*. [Online] 2013. [Citace: 10. 01. 2023.] [https://wikisofia.cz/wiki/Asymetrick%C3%A1\\_kryptografie](https://wikisofia.cz/wiki/Asymetrick%C3%A1_kryptografie). ISSN 2336-5897.
5. **Příspěvatelé, Wikipedie**. Augustova šifra. *Wikipédia: Slobodná encyklopédia*. [Online] 20. 09. 2012. [Citace: 11. 02. 2023.]  
[https://sk.wikipedia.org/wiki/Augustova\\_%C5%A1ifra](https://sk.wikipedia.org/wiki/Augustova_%C5%A1ifra).
6. **TUL**. ELearning TUL. *TUL*. [Online] [Citace: 28. 12. 2022.]  
<https://elearning.tul.cz/mod/resource/view.php?id=330002>.
7. **Štráfelda, Jan**. Hash. *Jan Štráfelda*. [Online] [Citace: 14. 10. 2022.]  
<https://www.strafelda.cz/hash>.
8. **Bitto, Ondřej**. Historie kryptologie. *fi.muni*. [Online] [Citace: 14. 10. 2022.]  
<https://www.fi.muni.cz/usr/jkucera/pv109/2003/xbitto.htm>.
9. **NÚKIB**. Informace. *Národní úřad pro kybernetickou a informační bezpečnost*. [Online] 16. 12. 2022. [Citace: 09. 01. 2023.] <https://nukib.cz/cs/ochrana-ui-v-ict/kryptograficka-ochrana/informace/>.
10. **Příspěvatelé, Wikipedie**. Kryptografie. *Wikipedie: Otevřená encyklopedie*. [Online] 14. 6. 2022. [Citace: 14. 10. 2022.] <https://cs.wikipedia.org/wiki/Kryptografie>.
11. **Stehlík, Michal**. MichalStehlik/WpfExamples2022. *GitHub*. [Online] 2022. [Citace: 05. 11. 2022.] <https://github.com/MichalStehlik/WpfExamples2022>.
12. **NÚKIB**. Minimální požadavky pro kryptografické algoritmy. *NÚKIB*. [Online] 8. 6. 2022. [Citace: 15. 10. 2022.]  
[https://www.nukib.cz/download/publikace/podpurne\\_materialy/Kryptograficke\\_prostredky\\_doporuceni\\_v2.0.pdf](https://www.nukib.cz/download/publikace/podpurne_materialy/Kryptograficke_prostredky_doporuceni_v2.0.pdf).
13. **Peoky**. Peoky/BusyIndicator. *GitHub*. [Online] [Citace: 18. 11. 2022.]  
<https://github.com/Peoky/BusyIndicator>.

14. **Ing. Tomáš Vaněk, Ph.D.** Proudové šifry. *ČVUT*. [Online] [Citace: 10. 01. 2023.]  
[http://rantos.cz/IBE-prezentace/P05b.prednaska\\_-  
\\_Proudove\\_sifry\\_1.2.6SHORT.pdf](http://rantos.cz/IBE-prezentace/P05b.prednaska_-_Proudove_sifry_1.2.6SHORT.pdf).
15. **Příspěvatelé, WikiSofia**. Symetrická kryptografie. *WikiSofia*. [Online] 2013. [Citace: 10. 01. 2023.] [https://wikisofia.cz/wiki/Symetrick%C3%A1\\_kryptografie](https://wikisofia.cz/wiki/Symetrick%C3%A1_kryptografie). ISSN 2336-5897.
16. —. Šifry. *WikiSofia*. [Online] 2013. [Citace: 14. 10. 2022.]  
<https://wikisofia.cz/wiki/%C5%A0ifry>. ISSN 2336-5897.
17. **Esham, Benjamin D.** Wikipedie: Otevřená encyklopedie. *Vigenère square.svg*. [Online] 08. 09. 2007. [Citace: 11. 02. 2023.]  
[https://cs.m.wikipedia.org/wiki/Soubor:Vigen%C3%A8re\\_square.svg](https://cs.m.wikipedia.org/wiki/Soubor:Vigen%C3%A8re_square.svg).
18. **Příspěvatelé, Wikipedie**. Cyklický redundantní součet. *Wikipedie: Otevřená encyklopedie*. [Online] 04. 12. 2022. [Citace: 13. 02. 2023.]  
[https://cs.wikipedia.org/wiki/Cyklick%C3%BD\\_redundantn%C3%AD\\_sou%C4%8Det](https://cs.wikipedia.org/wiki/Cyklick%C3%BD_redundantn%C3%AD_sou%C4%8Det).
19. **Services, Amazon Web**. What is Quantum computing? *AWS*. [Online] 2023. [Citace: 19. 02. 2023.] <https://aws.amazon.com/what-is/quantum-computing/>.

## A. Seznam příložených souborů

Na přiloženém datovém nosiči se nacházejí následující soubory a složky:

- **Dokumentace** – adresář s textovou částí odborné práce
  - **SOC-2023-P4-Rybar-Daniel.docx** – editovatelná verze dokumentace odborné práce
  - **SOC-2023-P4-Rybar-Daniel.pdf** – tisknutelná verze dokumentace odborné práce
  - **Rybar\_MP\_Casovy\_harmonogram.xlsx** – časový harmonogram odborné práce
- **Šifrovací aplikace.msi** – instalační soubor aplikace
- **CipherApp** – zdrojové kódy aplikace
- **Testovací data** – adresář s pokyny k jednotlivým testovacím souborům a testovací soubory