
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**
**DEPARTMENT OF COMPUTER SYSTEMS**


# DIGITÁLNÍ OBRAZOVÁ STEGANOGRAFIE

**DIGITAL IMAGE STEGANOGRAPHY**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** — **TOMÁŠ KOLARČÍK**
**AUTHOR**

**VEDOUCÍ PRÁCE** — **Ing. JOSEF STRNADEL, Ph.D.**
**SUPERVISOR**



**BRNO 2019**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

# Zadání bakalářské práce


Student: **Kolarčík Tomáš**
Program: Informační technologie
Název: **Digitální obrazová steganografie**
**Digital Image Steganography**
Kategorie: Bezpečnost

Zadání:

1. Vytvořte přehled metod z oblasti digitální steganografie pro skrývání informace v obrazových datech (dále jen "obrazová steganografie"), jejich vlastností a shrňte současný stav v této oblasti.
2. Zvolte typ skrývané informace (textová, obrazová apod.) a její vlastnosti. Na základě existující či vlastní analýzy způsobů ukládání obrazových dat a typu skrývané informace zužte a zvolte vhodné způsoby pro ukládání dat a vhodné metody pro obrazovou steganografii.
3. Implementujte několik existujících metod obrazové steganografie, zvažte jejich modifikace, popř. návrh a implementace vlastních metod.
4. Demonstrujte a vyhodnoťte funkčnost a vlastnosti implementovaných metod z hlediska skrývání a odkrývání zvoleného typu informace.
5. Dosažené výsledky diskutujte, navrhněte možné návaznosti a rozšíření předloženého řešení.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Strnadel Josef, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 26. října 2018

# Abstrakt

Tato bakalářská práce se zabývá digitální obrazovou steganografií a implementací některých dostupných metod. V rámci této práce je popsána steganografie na obecné úrovni, její historie i současná podoba. Dále je uvedena její klasifikace dle typu nosiče, klasifikace metod podle stylu ukrývání dat a vliv skrývané informace. Jádro práce poté tvoří samotná obrazová steganografie, kde jsou popsána fakta, která umožňují nepozorovaně ukrýt data do obrázku. Značná část je věnována vlastnostem metod, které pozorujeme a metodám samotným. Všechny popsané metody jsou implementovány do aplikace s grafickým i konzolovým rozhraním. Vytvořená aplikace poskytuje vícero obrazových formátů pro ukrytí. V poslední části je provedeno testování z pohledu kapacity, nepostřehnutelnosti a robustnosti.



# Abstract

This bachelor thesis deals with digital image steganography and implements some of the available methods. This thesis describes steganography on a general level, it's history and it's current form. It also shows it's classification according to the type of carrier, the classification of the methods according to the style of hiding data and the influence of the hidden information. The primary subject of this thesis is image steganography, where facts that allow imperceptibly hide data inside image are described. A great part of this thesis is dedicated to the characteristics of the methods, which we observe and to the methods themselves. All of the methods described are implemented in a graphical and console application. The created application provides multiple image formats for hiding. In the last part, testing is performed in terms of capacity, imperceptibility and robustness.



# Klíčová slova

steganografie, digitální steganografie, obrazová steganografie, historie, metody, steganogram, LSB, PVD, JSTEG, F3, F4, F5, Inverted LSB, Triple-A, EMD, DCT, JPEG, PNG, BMP, JFIF, robustnost, nepostřehnutelnost, testování, kapacita, rozdělení, prostorové metody, frekvenční metody

# Keywords

steganography, digital steganography, image steganography, history, methods, steganogram, LSB, PVD, JSTEG, F3, F4, F5, Inverted LSB, Triple-A, EMD, DCT, JPEG, PNG, BMP, JFIF, robustnes, imperceptibility, testing, capacity, division, spatial methods, frequency methods

# Citace

KOLARČÍK, Tomáš. *Digitální obrazová steganografie*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Josef Strnadel, Ph.D.

# Digitální obrazová steganografie

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Josefa Strnadela, Ph.D. a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Kolarčík
23. dubna 2019


## Poděkování

Hlavní poděkování patří vedoucímu mé bakalářské práce, panu Ing. Josefu Strnadelovi, Ph.D., za cenné poznámky, konzultace a poskytnuté návrhy a připomínky k řešení této práce. Dále bych chtěl poděkovat mé přítelkyni a rodině za podporu při psaní této práce.

# Obsah

# Kapitola 1

# Úvod

V dnešní době většina lidí ke své práci nebo ve volném čase nějakým způsobem využívá Internet. Ať už pro vyhledávání informací, nakupování skrze internetové obchody, sledování videí, publikování, komunikace s přáteli přes sociální sítě anebo mnoho z dalších činností. Při těchto činnostech je důležité myslet na to, že se provoz na síti dá sledovat. Z tohoto provozu se dají vyčíst různé informace. Proto je vhodné při citlivé komunikaci zajistit, aby daná komunikace byla dostupná pouze zamýšleným osobám a žádná další osoba do ní nemohla nepovoleně nahlížet. Pro potřeby utajení smyslu zprávy a dat jako jsou osobní údaje uložené na serverech nebo obsah přenášených dat po síti je tady kryptografie. Kryptografie je nauka o technikách pro bezpečnou komunikaci, a to i za přítomnosti nechtěné třetí strany. Kryptografie do příchodu moderního věku byla ve své podstatě synonymem šifrování, kde z čitelného textu vytváříme text zdánlivě nelogický. Dnes je doplněna například o ověřování neporušenosti, důvěryhodnosti, autentičnosti a nepopíratelnosti dat [36]. Je používána při komunikaci přes nezabezpečené kanály jako je třeba právě zmíněný internet. Šifrování je velmi vhodné pro utajení obsahu zprávy před očima nežádoucího čtenáře. Avšak pokud někdo sleduje naši komunikaci, která normálně není šifrovaná a uvidí zprávu: `tSnWPNnkCfyySKIZTftHPOcd3VEckVG6lGIHpjp75WQ=` vzbudí to v dotyčné osobě podezření na probíhající tajnou komunikaci a zvědavost, co se za tímto řetězcem ukrývá. Tato náhodně vyhlížející posloupnost znaků, která však ve skutečnosti nese zprávu "Budeme tam v 5 hodin", byla vytvořena pomocí šifrování algoritmem AES (128 bitů) a klíče "vutbr".

Tento problém viditelného utajování obsahu zprávy řeší právě Steganografie, kterou se bude zaobírat tato práce. Název steganografie pochází z řečtiny, významem tohoto slova je "skryté psaní" [17]. Steganografie se totiž zabývá ukrýváním tajných dat např. textové zprávy do zdánlivě nevinně vypadajícího nosiče, kterým může být obrázek, text nebo jeden z další typů nosičů [37]. První zmínky o steganografii se nacházejí už ve starověkém Řecku a samotná steganografie se od té doby vyvíjela s rozvojem technologií. Na rozdíl od kryptografie není tolik známá, přestože se používá již stovky let. To je vcelku logické, jelikož bezpečnost závisí na její nenápadnosti a na tom, že by ji na daném místě málokdo hledal. Z tohoto důvodu asi nikdy nedojde k jejímu masovému rozšíření, jelikož by to s sebou neslo zavedení bezpečnostních opatření, které by negativně ovlivnily její použitelnost. Avšak skrze to je steganografie velice zajímavou oblastí bezpečnostních systémů, kterou se má jistě cenu zabývat a stále dochází k vývoji nových metod pro skrývání dat, a hlavně k jejich detekci, jelikož v poslední době byla steganografie využita i v rámci škodlivého softwaru.

Význam steganografie je dle mého názoru dobře pochopitelný na často uváděném vzorovém příkladu steganografie tzv. "Problém vězňů" (Prisoners problem), který uvedl Gustavus J. Simmons na konferenci v roce 1983 [13] a jeho permutace nalezneme v mnoha jiných

pracích. Problém vězňů je založen na přítomnosti dvou vězňů, kteří se spolu snaží tajně domluvit na útěku. Při jejich komunikaci je přítomen dozorce, jenž může obsah kdykoliv kontrolovat a při pojetí podezření potrestat vězně. Pro vězně tedy není možné zprávy šifrovat, jelikož by vypadaly nápadně. Avšak zde přichází ke slovu steganografie. Vězni si začnou posílat kresby a po nějaké době do těchto kreseb začnou vkládat informace, např. pomocí různého významu barev nebo vzorů. Dozorce kresby kontroluje, ale pro něj vypadají naprosto normálně, a tak je nechá doručit. Druhá strana pak obrázek prohlédne a získá tajnou zprávu.

Pro doplnění uvedu předchozí příklad uvedený u kryptografie. Opět komunikují dva uživatelé, kteří si něco chtějí tajně sdělit, ale někdo na ně může dohlížet. Jak již bylo řečeno kryptografie by byla moc nápadná, a tak zvolí obrazovou steganografii. Vezmou tedy opět zprávu "Budeme tam v 5 hodin" a pomocí programu ji ukryjí do obrázku něčeho o čem se často baví a je to pro ně běžné, například fotografii, jako je obrázek 1.1. Tento obrázek pak pošlou v nějaké jejich tematické konverzaci. Třetí strana v tomto případě nečeká, a tudíž ani nedetekuje tuto zprávu, a i kdyby si vybrala pro kontrolu právě tuto část komunikace, tak by to brala jako běžnou komunikaci a hledala dál. Druhá strana však ví, že za tímto obrázkem se skrývá tajná zpráva a tak jej vloží do programu a získá ukrytý text.


Obrázek 1.1: Obrázek obsahující zprávu "Budeme tam v 5 hodin" skrytou algoritmem F5.

Tato práce si klade za cíl informovat o tom, co to je steganografie, jak vytvářet skryté zprávy za pomoci digitální steganografie pro skrývání informace v obrazových datech (dále jen "obrazová steganografie"), seznámit čtenáře s některými jejími metodami a vytvořit aplikaci implementující několik metod umožňujících skrývat data do obrázků a tato data posléze extrahovat. V této práci bude uvedena steganografie obecně, a to v kapitole 2, kde bude nastíněna důležitost steganografie v historii, její aktuální podoby, co to je, jak ji můžeme rozdělit a jaké rozlišujeme typy steganografických metod. Zmíněny budou i jiné možnosti pro skrývání než jsou obrazová data, avšak hlavní zaměření této práce bude věnováno právě obrazové steganografii v kapitole 3. V obrazové steganografii bude rozebráno, co to je vlastně obrázek, různé obrazové formáty, vnímání barev a jejich reprezentace, což je vhodné pro pochopení následného popisu samotných metod a jejich vlastností. Po této části bude následovat kapitola 4, která je věnována popisu implementace výsledné aplikace pro aplikaci obrazové steganografie. V předposlední kapitole 5 bude provedeno vyhodnocení implementovaných metod. Poslední část 6 tvoří závěr.

# Kapitola 2

# Steganografie

Tato kapitola zprostředkovává obecný pohled na steganografii, je tedy jakýmsi úvodem do této problematiky. Definuje, co to steganografie je, jaké jsou používané pojmy, zmíněno bude pár zajímavých historických výskytů steganografie a dále bude uvedeno jak vypadá její moderní podoba a kde se může nacházet. Další část bude věnována steganografie v závislosti na nosiči, jaké se rozlišují typy steganografických metod a jaké typy dat může skrývat a co to obnáší.

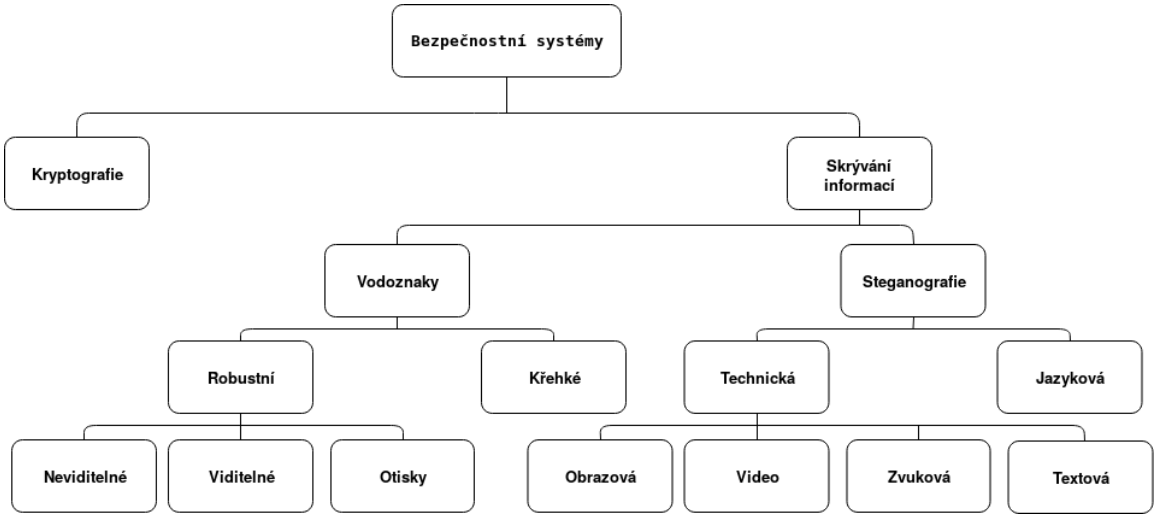
## 2.1 Základní pojmy

Slovo steganografie pochází ze spojení řeckých slov steganos v překladu skrytý a graphein, psaní. Významem slova steganografie, je tedy "skryté psaní" [17]. Steganografie se zabývá ukrýváním tajných dat např. textové zprávy do zdánlivě nevinně vypadajícího nosiče, kterým může být obrázek, text, spustitelný soubor nebo jiný z dalších typů nosiče [37].

Tento obor je jeden z oborů skrývání informací, jak je vidět na obrázku 2.1, zobrazujícím hierarchii jednotlivých oborů bezpečnostních systémů. Dalším oborem zabývajícím se skrýváním informací je vodotisk. Tyto obory spolu úzce souvisí a využívají stejné principy, avšak každý cílí na jiné vlastnosti výsledného steganogramu. Steganografie usiluje o nepozorovatelnost a co možná největší kapacitu [1].

Než bude popsána samotná steganografie a pojmy s ní spojené, je nutné vysvětlit pár základních pojmů, pro správné pochopení následujících informací. Tyto pojmy jsou:

- nosič, cover, krycí médium – označuje médium, do kterého skrýváme tajná data
- tajná data, tajná zpráva, skrytá zpráva, data – označují data určená k ukrytí
- steganogram, stego objekt, stegomédium – je nosič, který již obsahuje skrývaná data




Obrázek 2.1: Obrázek zobrazující hierarchii jednotlivých disciplín bezpečnosti. Převzato z [7].

## 2.2 Steganografie v historii

Steganografie našla v historii své uplatnění i přes existenci šifrování. V historii byla steganografie využívána v těžkých dobách jako byly například války či povstání. V těchto dobách byla sledována komunikace a šifrované zprávy by vzbudily podezření. To je skvělé místo pro steganografii. Mezi tyto známé metody použité v historii patří například přenos tajné zprávy vytetované na oholenou hlavu otroka, která se nechala následně zarůst vlasy. Mezi další známé metody patří přenos pod voskem na voskových destičkách použitých v Řecku, zpráva na hedvábí zavoskovaná do kuličky pocházející z Číny, kterou spolknul posel nebo známý neviditelný inkoust. Tyto metody jsou při nejmenším zajímavé, ale taky demonstrují smysl steganografie. V případě zájmu je více informací o metodách vyskytujících se v historii zmíněno v příloze A, jelikož tato práce je zaměřená na moderní steganografii.

## 2.3 Moderní steganografie

Steganografie jako prostředek pro tajnou komunikaci, sloužila v historii převážně pro doručování důležitých zpráv v období konfliktů. Dnes steganografie najde uplatnění například pro ochranu autorských práv i když tato oblast spadá formálně pod vodoznaky nebo pro udržení soukromí při komunikaci v případě, že vláda zakazuje využití šifrování či pro vedení nepozorované komunikace, například pro komunikaci se škodlivým softwarem.

Společnost BBC na svém zpravodajském webu ve článku s titulkem "How do terrorists communicate?" (Jak komunikují teroristé?) [8], řadí jako jednu ze známých technik teroristické komunikace právě obrazovou steganografii. Byly to teroristické útoky, které vedly některé vlády k myšlence, že by se šifrování, které chrání soukromí lidí, mělo dát prolomit na žádost, a to z důvodu pátrání v rámci vyšetřování teroristické či jiné záškodné činnosti. Příkladem může být Francie a Německo, které tyto změny, co se týče možnosti prolomení šifrování na žádost, prosazovali u Evropské komise v roce 2016, jak se o tom píše v článku [20]. Hlavně se mělo jednat o aplikace využívající end-to-end šifrování. Důvodem bylo zneužití aplikací jako je WhatsApp nebo Telegram využívající end-to-end šifrování,

právě pro dorozumívání teroristických skupin. Když si však uvědomíme, kde všude je naše komunikace chráněna šifrováním a co by tedy znamenalo poskytovat nástroj k jeho jistému prolomení, tak nám rázem dojde, že není v této situaci možné spoléhat na zabezpečení třetí stranou a je dobré data, která mají být opravdu soukromá, řádně zabezpečit proti přečtení někým cizím. Pro představu jeden z nejznámějších prohlížečů Firefox vytváří statistiku načtených stránek, kde bylo využito šifrované spojení pomocí HTTPS. V této statistice viz https://letsencrypt.org/stats/ je k lednu 2018 cca 70% načítaných stránek zabezpečeno a tento graf má jasně rostoucí tendenci.

Skrývání informací nachází velké uplatnění v komerční branži, v podobě digitálních vodoznaků (digital watermark). Vodoznaky jsou ve svém principu trochu odlišné od steganografie, ale základ zůstává stejný. Rozdílem je důraz kladený na jednotlivé vlastnosti, mezi které patří například kapacita, nenápadnost, robustnost, atd. Vodoznaky totiž na rozdíl od skryté informace mohou být zřejmé i na první pohled [17], takzvané viditelné vodoznaky, ale neměly by být odebratelné. Nebo zase naopak existují vodoznaky neviditelné. Ty dělíme na dvě kategorie. Jedny, kterým se říká křehké a ty jsou založeny na tom, že při manipulaci s nosičem je nesená tajná informace zničena a tímto je prokázána manipulace s mediem. Anebo robustní, které by měly jistou míru manipulace ustát, a to nejlépe až do té míry, kdy už by došlo k znehodnocení chráněného média. Digitální vodoznaky v různých formách mají několik využití. Bývají použity k označení kopie pro případnou identifikaci, kdo materiál vypustil do oběhu, například předverze filmů. Tato metoda se dá nalézt pod anglickým názvem fingerprint (otisk palce). Dalším použitím je označení pro účely plnění smluvních požadavků, což se tyká například vysílání reklam, seriálů a podobných medií. Anebo tradiční využití k dokazování autorství, nepozměněnosti a jiné další.

V dále uvedených podčástech bude zmíněno pár známých využití moderní steganografie.

### 2.3.1 Tiskárny

I tiskárny mohou využívat steganografii. Stala se událost, kdy zaměstnanec NSA (National Security Agency) vynesl citlivý dokument do novin. Tento dokument vytiskl v kanceláři na tiskárně, aby jej mohl vynést a předat do novin. Tyto noviny tento dokument celý zveřejnili na webu. Díky tomu byla NSA schopná vystopovat původ dokumentu. Umožnil jí to tajný kód, který tiskárna vytiskla společně s obsahem. Například v kódu od společnosti Xerox jsou obsaženy informace jako sériové číslo tiskárny, čas a datum vytištění dokumentu. Tento kód je znám pod označením MIC (Machine Identification Code), tiskarnová steganografie (printer steganography) nebo žluté tečky (yellow dots). Každý výrobce do těchto teček může ukrývat jiné informace.[5]

### 2.3.2 DNA

V roce 1999 výzkumníci na Mount Sinai School of Medicine v New Yorku zakódovali skrytou zprávu do řetězce lidské DNA. Tato technika je dohledatelná pod anglickým názvem "Genomic steganography". U této metody je dokonce uváděno, že jeden gram DNA by mohl nést až 100 exabytů dat.[15]

Dnes je využívaná metoda značení majetku pomocí syntetické DNA. Technologie mikroteček s obsahem syntetického kódu DNA doplněné výstražnými prvky a ve spojení s rozsáhlou mezinárodní databází chráněných předmětů, tvoří silný nástroj při ochraně majetku a při jeho jednoznačné identifikaci. Označení má předcházet krádežím kol a usnadnit jejich případnou identifikaci, jak se tvrdí v [18]. Tento postup je již nyní dostupný v několika městech. O toto zabezpečení je poměrně velký zájem.

### 2.3.3 Útoky z 11. září 2001

Po teroristickém útoku 11. září roku 2001, vznikly obavy, že Al-Káida při spřádání útoku na Světové obchodní centrum v New Yorku používala jako komunikační nástroj steganografii. Původně se myslelo, že se tato komunikace skrývá v obrázcích na stránkách pro dospělé. Na Michiganské univerzitě začali prozkoumávat obrázky na různých stránkách, jako je například eBay a nenašli žádná skrytá data. Při tomto průzkumu prozkoumali přes dva milióny obrázků.[16]

### 2.3.4 Škodlivý software

Tato část vychází ze článku [19], který informuje o sérii kyberútoků na různé organizace. Tyto útoky nesly označení Shady RAT. Útok se skládá ze 3 základních částí. První částí je rozšíření škodlivého softwaru za pomocí příloh emailů. Tyto přílohy jsou nevinně vyhlížející, ale obsahují skrytý kód, který při otevření souboru nakazí zařízení trojským koněm. V druhé fázi, kdy je počítač infikován trojským koněm, se tento program pokusí kontaktovat vzdálenou stránku, kterou má v sobě definovanou. Tato stránka odkazuje na obrázek nebo HTML dokument, což na první pohled není podezřelá činnost. Navíc mnoho firewallů je nastaveno na povolení průchodu obrázků a HTML souborů při HTTP požadavku. Navíc obrázky na příkazovém serveru, který byl při pátrání nalezen nejsou nijak neobvyklé. Mimo jiné zde nalezneme například obrázek Lena oblíbený v počítačové grafice. Právě tyto obrázky však nesou skryté příkazy pomocí steganografie. Fáze třetí je připojení ke vzdálenému počítači a umožnění přístupu k souborům. Poněkud podivné je, že útočníci nezabezpečili řídící server a navíc na něm nechali nainstalované analytické nástroje, což usnadnilo pátrání po ovlivněných organizacích. Cíle těchto útoků, jak se ukázalo, byly od vládních organizací, až po soukromé firmy, a to po celém světě. Motiv tohoto útoku nebyl objasněn, jelikož se jednalo o rozmanité spektrum cílů.

Existují i další škodlivé softwary využívající nějakým způsobem steganografii. Mezi tyto se řadí např. Alueron, Stuxnet, Duqu.

## 2.4 Rozdělení steganografie podle typu nosiče

### 2.4.1 Textová steganografie

V předchozí části této práce, která se zabývala historií, byla většina metod z kategorie textové steganografie. Textový nosič však oproti obrazovému nebo zvukovému příliš nedisponuje nadbytečnými informacemi. Tento fakt trochu znesnadňuje vkládání tajné informace do existujícího textu, kde si musíme dávat pozor na poškození původní zprávy nebo nápadnou podobu steganogramu. Jinak řečeno, změna jednoho písmene v textu, například jeho zvětšení nebo naklonění je mnohem nápadnější než změna jednoho bitu v barevné složce obrázku. Avšak textová steganografie je mnohem méně paměťově náročná, a tudíž i méně náročná na přenos.

Mezi populární metody textové steganogafie, jak uvádí [3], patří posuny slov nebo řádků. Syntaktické metody, které binární reprezentaci dat skrývají za čárky či tečky v textu. Poměrně známá je i reprezentace pomocí daného počtu mezer v textu, kdy jedna mezera za slovem značí třeba nulu a dvě mezery značí jedničku. Zajímavé je také využití neviditelných unikódových znaků a unikódových znaků s nulovou šířkou, jako je třeba "Zero-Width Joiner" a "Zero-Width Non-Joiner". A mnoho dalších metod.

### 2.4.2 Obrazová steganografie

Obrazová steganografie je jedna z nejpoužívanějších variant steganografie. Je to pravděpodobně zapříčiněno velkou popularitou komunikace za pomoci obrázků. Příkladem může být popularita aplikace Instagram, humorných obrázků nebo vyjadřování se pomocí gifů v různých komunikátorech. Obrazová data jsou dnes všude kolem nás a je jich nepřeberné množství. S tím se také váže fakt, že komunikace skrytá v obraze se tím pádem lépe ztratí v nepřeberném množství přenášených obrazových dat. Obrazová steganografie je hlavním tématem této práce a je jí věnována celá kapitola 3, proto v této části nebude více rozebrána.

### 2.4.3 Zvuková steganografie

Zvuková steganografie využívá nedokonalosti lidského sluchu stejně jako obrazová steganografie využívá jemné odchylky ve vizuální podobě nosiče, kterou není oko schopno zaznamenat. Mají dokonce společnou metodu skrývání informace a tou je vkládání do nejméně významného bitu. Dalšími metodami, jak uvádí [14], jsou například kódování do parity, do fáze nebo schování do ozvěny. U zvukové steganografie je nutné myslet na to, že ucho je poměrně citlivé na šum.

### 2.4.4 Souborová steganografie

Jednou z variant je ukrývání do souborů za ukončující značky. Tato metoda je využitelná u formátu souborů, které mají vnitřní strukturu a nějakým způsobem označují konec dat. Tajná zpráva je pak vložena za tuto užitnou část, většinou ukončenou nějakou ukončující značkou, která není zpracovávána zobrazovačem daného souboru.

Dále je možné ukrývání dat ve spustitelných souborech. Zde je využíváno několik možností, jako jsou například nesplnitelné podmínky, místo vytvořené pomocí podmíněného skoku za tajnou oblast, komentáře nebo opět vložení dat za ukončující značku, například v poli znaků za znak `\0` v jazyce C. Zjednodušeně řečeno se využívá mrtvého kódu anebo nezobrazitelných částí v kódu. Složitějšími technikami jsou poté metody pracující přímo s instrukcemi.

Dokonce existují souborové systémy podporující steganogafii například StegFS dostupný na `https://github.com/albinoloverats/stegfs`. Tento systém je ovšem ztrátový a není tudíž moc použitelný jako hlavní souborový systém třeba pro operační systém a cenná data, protože může kdykoliv dojít k přepsání existujících dat. Je tedy spíše využitelný na USB discích a tak podobně.

### 2.4.5 Síťová steganografie

Tato část vychází z `http://stegano.net`, která je projektem zabývajícím se síťovou steganografií a detekcí anomálií. V dnešní době je možné najít využití steganografie i v samotném síťovém komunikačním protokolu. Rozdělujeme možné metody ukrytí na tři skupiny. První skupinou jsou metody, které ukrývají tajnou informaci do nesených dat. Pak jsou tady metody ovlivňující logiku protokolu, jako je například posloupnost zpráv atd. Poslední kategorie jsou metody ukrývající tajná data do nevyužitých částí hlavičky protokolu.[17] Pokud se zaměříme například na metodu, která využívá nevyužitá pole v hlavičce protokolu, tak množství dat, které v této hlavičce můžeme přenést je velice malé, avšak pokud zvážíme kolik takových hlaviček je odesláno, tak už se dá ukrýt docela slušná tajná zpráva. Ve výše uvedených zdrojích je možné najít i několik metod společně s pracemi, které se s nimi pojí.

## 2.5 Typy metod a vliv typu skrývané informace

U steganografie, tak jako rozlišujeme typ nosiče, můžeme dále rozlišovat metody, které se pojí s nosičem. Metody můžeme zařadit do 3 kategorií.[23]

**Injekční** U injekčních metod nová data vkládáme navíc do nosiče. Tento postup se projeví zvětšením velikosti výsledného souboru. K tomuto postupu se využívají tzv. slepá místa neboli místa v nosiči, která se při ovlivnění neprojeví navenek. Příkladem můžou být komentáře, část kódu za příkazem `return` nebo části souborů za značkou ukončující čtení.

**Substituční** Substituční metody nahrazují stávající části dat za části skrývaných dat. Jejich výhodou je zachování podobné velikosti souboru, avšak v důsledku pozměnění původní informace může dojít ke zhoršení kvality. Tyto metody jsou v obrazové steganografii používány nejčastěji.

**Propagační** Posledním typem jsou propagační metody. U těchto metod nepoužíváme stávající nosič, ale přímo jej generujeme ze skrývaných dat. Výsledkem může být například obrazec nějakého fraktálu.

Dnes když mluvíme o digitální steganografii je poměrně jedno jaká data budeme skrývat, podmínkou je, aby se data dala převést do binární podoby a aby byl nosič dostatečně veliký pro ukrytí informace. Navíc v případě uvažování obecných dat nám vznikne několik výhod. Mezi tyto výhody patří například volnost při výběru formátu tajných dat a aplikování různých zpracování jako je třeba vlastní zabezpečení dat.

Jsou však případy a metody, kdy se můžeme zaměřit pouze na určitý typ dat a získat tak něco navíc. Například pokud víme, že budeme chtít skrývat pouze textové zprávy. Při ukládání pouze textových zpráv se můžeme zaměřit na ušetření místa, protože pokud si vystačíme s velkými písmeny anglické abecedy, můžeme redukovat počet bitů pro uložení jednoho znaku. Velká písmena abecedy jsou v ASCII uložena od hodnoty 65 do hodnoty 90, pokud tedy jednoduše odečteme od těchto hodnot 65 jsme schopní těchto 26 znaků uložit jako 5bitovou hodnotu. Tím ušetříme 3 bity na každé písmeno. To znamená, že na původní místo, kde by bylo uloženo 5 písmen, uložíme písmen 8. Dalším příkladem může být ukládání pouze obrazových dat. U těchto dat můžeme například zvolením správné metody docílit toho, že při filtrování vytvořeného obrázku získáme přímo viditelný obrázek, jež byl skrytý. Tato technika nebude sice odolná vůči odhalení, jelikož na první pohled i nezkušený člověk uvidí obrázek v obrázku po vynulování bitů, ale může se nám někdy hodit, například pro demonstraci nebo jednoduchost extrakce.

# Kapitola 3

# Obrazová steganografie

Tato kapitola se bude věnovat již konkrétně obrazové steganografii. Bude zde řečeno, co to obrázek je, jak lidé vnímají barvy a jak jsou tyto barvy reprezentovány v počítači. Poté bude vysvětleno, jak jsou data uloženy do různých obrazových formátů. Zmíněny budou pouze formáty následně využité při implementaci. Porozumění základní logice daného formátu je vhodné pro pochopení následujícího popisu metod nebo pro zkoumání upravených souborů. U formátu JPEG bude rozebrán i proces komprese, z kterého vyplývá nutnost použití metod až po provedení transformace místo úpravy hodnot pixelů, jako takových. Dále bude uvedeno, jaké vlastnosti jsou u metod obrazové steganografie pozorovány a co znamenají. Nakonec bude uvedeno několik vybraných metod obrazové steganografie společně s popisem, jak fungují.

## 3.1 Definice obrázku

Nejprve by bylo vhodné říct si, co chápeme pod pojmem digitální obrázek, do kterého bude následně skryta tajná informace pomocí různých metod. Rozlišujeme dva základní typy digitálních obrázků, rastrové a vektorové. Tato práce se bude zaobírat metodami pro rastrové obrázky, jelikož na internetu a při běžné komunikaci převažují. Většinou jsou velikostně větší, a tudíž poskytují větší kapacitu pro uchování tajné zprávy. Vektorové obrázky jsou ukládány ve formě nějakého jazyka, což neposkytuje takovou možnost nepozorovatelné úpravy, jakou je drobné pozměnění barvy pixelu v případě rastrového obrázku.

Kniha [10] definuje obrázek jako dvourozměrnou funkci $f(x, y)$, kde $x$ a $y$ určují pozici v prostoru a hodnota této funkce v daném bodě udává intenzitu v daném bodě. Pokud jsou $x$, $y$ a hodnota funkce konečné a diskrétní hodnoty, pak tento obrázek nazýváme digitálním obrázkem. Pro naše potřeby si obrázek můžeme představit jako dvojrozměrné pole pixelů, kde šířka je počet sloupců a výška obrázku je počet řádků. Tyto pixely, dle typu obrázku, mohou mít různé formy. Pro černobílý obrázek budou hodnoty těchto pixelů nabývat hodnot 0 nebo 1, u šedotónových obrázků to bude opět jedno číslo a to většinou v intervalu <0,255> anebo například při barevném RGB obrázku tento pixel bude složen právě z těchto tří složek R pro červenou, G pro zelenou, B pro modrou barvu a opět většinou v rozsahu <0,255>. Příklady různých barevných typů obrázků jsou vidět na obrázku 3.1.

Barevný rozsah nám určuje barevná nebo také jinak bitová hloubka (color depth neboli bit depth), u černobílého obrázku je tato hloubka jeden bit, a to nám dává právě dvě barvy, černou nebo bílou. Naopak třeba při zmiňovaném barevném obrázku s hodnotami pro danou barevnou složku v rozsahu <0,255>, tudíž 8 bitů pro každou barvu, se bavíme o takzvané

"True color" barevné hloubce 24 bitů, pomocí které umíme pokrýt 16 777 216 barevných odstínů.

(a) Černobílý (b) Šedotónový (c) Barevný

Obrázek 3.1: Různé typy obrázků.

Jedním z důležitých předpokladů pro tvorbu kvalitního steganogramu je volba krycího obrázku. Například [17] uvádí, že mnoho expertů doporučuje používat jako nosič 8 bitový šedotónový obrázek, protože vzniklé rozdíly po úpravě odstínu jsou pro člověka méně postřehnutelné. Co do detekce změn, tak je vhodné volit nosič z obrázků, které nejsou dostupné někomu, kdo by mohl chtít naši tajnou informaci odhalit. Jelikož pokud dotyčný najde originální obrázek a porovná jej s našim vytvořeným obrázkem nesoucí tajná data, tak mu značně ulehčíme práci. Proto není vhodné volit obrázky z veřejných zdrojů jako jsou fotky herců, plakáty a tak podobně. Vhodné je naopak námi pořízená fotografie, jejíž kopii vlastníme pouze my a nikdo jiný. Vlastní fotografie z mobilního telefonu má výhodu také v tom, že ještě neupravená má už nějaký ten šum a nedokonalosti, proto to každý předpokládá za normální. Dalším vhodným aspektem pro výběr nosiče je absence příliš jednolitých barevných ploch, jako třeba blankytně modré oblohy. Důvodem je, že tyto plochy nám po vložení dat mohou snadno zviditelnit změny a to v podobě viditelného šumu.

## 3.2 Vnímání a reprezentace barev

Aby člověk vnímal barvy, musí kolem něj být světlo, jelikož právě světlo dopadá do našich očí, kde je snímáno a následně v mozku převedeno na danou barvu. Světlo je elektromagnetické záření o určité vlnové délce nebo frekvenci. Lidé jsou schopni vnímat pouze tzv. viditelné světlo, které je mezi vlnovými délkami ultrafialového a infračerveného světla. Světlo vnímáme pomocí sítnice oka, kde se nacházejí světlocitlivé buňky dvojího typu, tyčinky a čípky. Tyčinky nám slouží pro vnímání kontrastu a mimo jiné umožnují vidění ve tmě. Čípky nám na druhou stranu umožnují vidět barvy. Čípky máme v oku třech druhů, každý druh vnímá světlo jiné vlnové délky a to červené, zelené, modré. Citlivost na tyto barvy není stejná, nejcitlivější jsme na zelenou barvu, trochu slaběji vnímáme červenou a nejslaběji poté modrou barvu. Naše oko pomocí čípků tedy vnímá různou intenzitu dané barvy a my jí pak interpretujeme jako specifickou barvu.

Bylo nastíněno, jak člověk barvu vnímá, nyní něco o tom, jak jí reprezentujeme v počítači. Jak již bylo řečeno, cílovou barvu většinou skládáme z několika základních barev. Pro jednotlivé barvy můžeme určit různý počet bitů (barevnou hloubku), který nám udává kolik barev můžeme nakombinovat, jinak řečeno určujeme, jak jemný rozdíl barev umíme vytvořit. Ale důležité je, jak barvy mícháme, jaké jsou naše základní barvy a tak podobně. K tomuto účelu nám slouží barevné modely, které nám umožnují pracovat s barvami a různě je míchat.

Barevných modelů existuje mnoho druhů pro různé potřeby, RGB, CMYK, HSV, HSL, YCbCr a mnoho dalších. Například běžný monitor a tiskárna budou mít rozdílné barevné modely. Je to dáno také tím, jak skládají barvy. Tiskárny využívají subtraktivní míchání, což znamená, že při maximální intenzitě všech barev dostáváme černou. To také odpovídá práci s inkoustem. Známým a často používaným subtraktivním modelem, například v tiskárnách, je CMY, dle anglických názvů základních barev tohoto modelu, tyrkysové, purpurové, žluté nebo CMYK, kde je doplněn o černou, což je lepší pro rychlejší schnutí a šetření inkoustu. Monitory zase využívají aditivní míchání barev, při kterém v maximální intenzitě všech barev dostáváme bílou barvu. Zde je nejznámější RGB barevný model. RGB, opět podle anglických názvů základních barev tohoto modelu, červené, zelené a modré. S tímto modelem, a také s modelem YCbCr bude v této práci pracováno, jelikož jsou využívány u obrazových souborů. YCbCr model rozděluje barvu na 3 složky. Y specifikuje intenzitu dané barvy <0,255>, Cb udává modrost <-127,128> a Cr udává červenost <-127,128>. Tento popis lze dobře využít, jelikož rozděluje od sebe komponenty barevnosti, na kterou je oko méně citlivé a jasu, na které je naopak citlivější více. YCbCr model bude využíván hlavně pro potřeby metod na obrázcích formátu JPEG, jelikož je zde tento model využit při ukládání a kompresi, jak bude zmíněno později v části 3.3.3.

## 3.3 Obrazové formáty

Obrazovou informaci je možné ukládat pod různými formáty pro obrazová data, jako jsou například GIF, BMP, JFIF, PNG, TIFF a další. Formát souboru nám nebo spíše programu, kterým soubor otvíráme říká, jaká data má očekávat a jak by tato data měla být organizována, tudíž i jak je má číst. Jelikož jsou v různých prostředích používány různé formáty souborů, tak i pro steganografické účely bude vhodné aplikovat steganografii na ty hojně používané, které budou méně nápadné při přenosu. Například ve firmě, kde se pracuje výhradně s tiff soubory, bude budit pozornost přenos souboru s příponou jpeg, pokud se jedná o intranetovou komunikaci. Kdybychom sledovali celkovou komunikaci včetně komunikace do internetu, tak by se soubor s příponou jpeg ztratil, jelikož tento formát je na internetu velice populární. My budeme vycházet z toho, že hlavní komunikací je dnes komunikace přes internet a velké množství provozu na síti tvoří prohlížení webu, při kterém se stahuje velké množství obrázků, jako například pozadí stránky, náhledy produktů, ikony a další. Proto bude výsledná aplikace podporovat hlavně formáty, které jsou často používány právě zde. Mezi tyto časté formáty se dle [2] řadí PNG, JPEG, GIF, SVG a z menší části i BMP. V této části budou lehce rozebrány použité formáty.

### 3.3.1 BMP

Informace o tomto formátu byly čerpány z [30, 9]. Níže je popsán formát souboru určený pro Microsoft Windows, existuje také verze pro OS/2, ale ta nebude rozebrána.

BMP je formát souboru umožňujicí nést, jak monochromatický, tak i barevný obraz s různou barevnou hloubkou. Někdy je tento formát také nazýván DIB (Device Independed Format), což je jakési jeho jádro. BMP je totiž příponou při ukládání Windows DIB file souboru. Tento formát patří mezi starší a dnes méně využívané formáty. Od verze 3.0 podporuje run-length encoded formáty pro kompresi bitmap, a to s použitím 4 nebo 8 bitů na pixel. Komprimace se u tohoto formátu, ale většinou nevyužívá. Hlavním účelem tohoto formátu bylo umožnit nezávislý přesun obrazových dat mezi různými zařízeními, jak napovídá zkratka DIB.

Formát BMP umožňuje ukládat rastrová data ve čtyřech variantách:

- 1 bit na pixel s barevnou paletou
- 4 bit na pixel s barevnou paletou
- 8 bit na pixel s barevnou paletou
- 24 bit na pixel bez barevné palety

Soubor typu BMP se skládá z hlavičky BITMAPFILEHEADER, která nese základní informace o souboru a její velikost je 14 bytů. Ihned následuje DIB hlavička, s metainformacemi o obrázku o velikosti 40 bytů. Barevné palety, kde jednotlivé složky jsou uvedeny v RGB a typickou délkou této palety je 2, 16 nebo 256 barev. Jednotlivé barvy však nejsou uloženy na 3 bytech v pořadí červená, zelená, modrá, jak by se dalo očekávat, ale místo toho je uložena na 4 bytech s tím, že čtvrtý byte je ponechán nulový a první tři byty nejsou v pořadí červená, zelená, modrá, ale naopak, a to tedy modrá, zelená, červená. Poslední části jsou samotné rastrová data.

### 3.3.2 PNG

V této části bylo čerpáno z [31, 29, 25, 24].

Grafický formát PNG (Portable Network Graphics) slouží pro uchování, přenos a zobrazení rastrových obrázků. PNG vznikl jako zcela nový formát nepošpiněný minulým vývojem, nezávislý na firmě a patentech, a měl vylepšovat tehdejší vlastnosti stávajících formátů. Tento formát měl nahradit formát GIF a také poskytoval lepší kompresi, jak uvádí [25]. PNG při ukládání využívá bezeztrátovou kompresi. Mezi výhody tohoto formátu patří možnosti jako jsou, využití různých barevných hloubek, průhlednost neboli alfa kanál a to v 1, 8 nebo 16 bitech, prokládání pixelů a další. PNG umožňuje ukládat jak barevné, tak šedotónové obrázky. Šedotónový obrázek lze uložit s barevnou hloubkou 1, 2, 4, 8 nebo 16 bitů. Barevné obrázky je poté možné ukládat dvěma způsoby, a to s barevnou paletou a bez ní. Ty bez barevné palety je možné ukládat s barevnou hloubkou 24 nebo 48 bitů a palety je možné využít 1, 2, 4 nebo 8 bitové.

PNG se skládá z hlavičky, která je neměnná. Za hlavičkou se může nalézat různé množství pojmenovaných bloků, kde každý nese svou délku, typ a je zabezpečen kontrolním součtem CRC. Takovýto blok nazýváme chunk a nese informace o zpracovávaném obrázku, tyto informace jsou zde zapsány v tzv. "network byte order".

#### Významné povinné chunky

**IHDR** Hlavička obrázku o pevné velikosti 13 bytů, kde jsou obsaženy metainformace o daném obrázku jako jsou výška a šírka obrázku, bitová hloubka, typ kódování, metoda komprimace, metoda filtrace a informace, zda je obrázek s prokládáním. Všechny údaje jsou každý na jednom bytu kromě šířky a výšky, která je na 4 bytech. Tento chunk s hlavičkou obrázku se musí nacházet jako první hned za hlavičkou samotného PNG.

**IDAT** Obsahuje samotná obrazová data. Těchto chunků může být více. Podmínkou mnohonásobného použití je, že tyto chunky leží bezprostředně za sebou.

**IEND** Označuje konec PNG souboru. Slouží k označení konce načítání a také pro kontrolu zda je soubor přenesen celý. Je dlouhý 12 bytů a jeho datová část je nulová. Má tedy neměný tvar. Tento chunk musí být posledním chunkem.

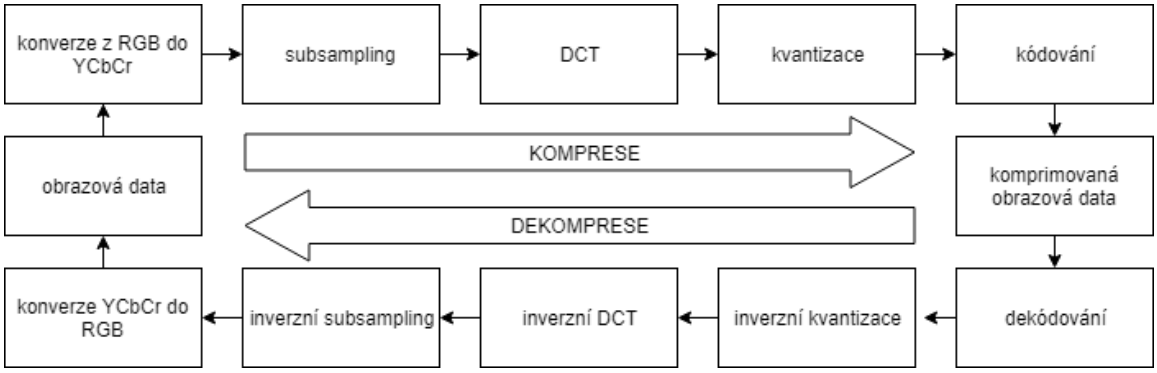
### 3.3.3 JFIF

JPEG File Interchange Format (JFIF) je souborový formát pro přenos obrázku komprimovaného metodou JPEG. Tento JFIF souborový formát bývá často nesprávně označován právě jako JPEG, který je označením pro ztrátovou komprimační metodu pro komprimaci obrazových dat. Celý postup komprese JPEG je vidět na obrázku 3.2 a jeho jednotlivé kroky budou následně lehce popsány pro lepší pochopení toho, co dané kroky obnáší i následných metod pro JPEG. U JFIF se skrývají data do DCT koeficientů, které dostaneme po ztrátové části komprese JPEG. Pokud bychom data skrývali přímo do hodnot pixelů, tak by byly poškozeny hned při prvním zpracováním, jelikož by proběhl proces komprese a hodnoty by se změnily. Výhodou metod pro JFIF formát je mnohem méně předvídatelné ovlivnění výsledného obrázku, protože změna se projeví na celém bloku 8x8 při ovlivnění komponent DCT. Tyto metody však nedosahují takové kapacity jako metody pracující přímo s hodnotami pixelů, což také vyplývá z komprimace tohoto formátu. Pro úplnost doplním, že barevný prostor RGB i YCbCr (CCIR 601-256 levels) je v tomto popisu brán s jedním bytem pro každou komponentu, tedy může nabývat hodnot <0,255> v případě Cb a Cr hodnot <-127,128>.

Nyní však bude popsán samotný souborový formát JFIF. Tento formát neobsahuje žádné pokročilé funkce a ani by neměl, jelikož jeho jedinou úlohou je umožnit přenos obrázků komprimovaných metodou JPEG [12]. Tento formát opět pro vícebytové hodnoty používá tzv. "network byte order" jako tomu je u formátu PNG. Obrazová data jsou v tomto formátu uložena tradičně zleva doprava a řádky poté shora dolů. To například při konverzi z BMP do JFIF formátu obnáší převedení obrazových dat do správné orientace před samotným zpracováním JPEG kompresí.

JFIF soubor je rozdělen do segmentů. Maximální délka jednoho segmentu je $2^{16}$ bytů. Každý segment začíná značkou, dvěma byty, kde první je vždy nastaven na `0xff` a druhý byte podle typu značky nabývá hodnot `0x01` až `0xfe`. Krajní hodnoty jsou vynechány z důvodu synchronizace, při výskytu chyby. V samotných datech se výskyt `0xff` musí doplňovat bytem s hodnotou `0x00`, aby vznikl rozdíl mezi značkou a daty. Mezi nejdůležitější značky, které najdeme prakticky v každém obrázku, patří SOI (`0xffd8` - Start Of Image), SOS (`0xffda`- Start Of Scan), APP0 (`0xffe0` - Application Marker), DHT (`0xffc4` - Define Huffman Table), DQT (`0xffdb` - Define Quantization Table) a nakonec také značka EOI (`0xffd9` - End Of Image) [33].

APP0 označovač je použit k identifikaci formátu a následuje bezprostředně za označovačem SOI. Obsahuje délku segmentu APP0. K identifikaci formátu je opět použit jeho název, tedy řetězec "JFIF", avšak zde je navíc doplněn ukončujícím nulovým znakem. APP0 označovač dále poskytuje některé informace, které nejsou poskytovány přímo ve výstupu JPEG. Jako jsou náhled, hustota pixelů vertikálně, hustota pixelů horizontálně, typ jednotky rozlišení, verze JFIF.




Obrázek 3.2: Obrázek kroků při komprimaci a dekomprimaci.

### Převod z RGB do YCbCr

V tomto kroku jsou obrazová data převedena z prostoru RGB do YCbCr a to za pomocí vzorců 3.1, 3.2, 3.3, jak uvádí specifikace JFIF formátu [12]. Tento krok je bezeztrátový, avšak umožní nám díky oddělení světelnosti od barevnosti jednu z podstatných věcí, která nám ušetří na celkové velikosti výsledného souboru a tou je subsampling viz další krok.

$$\mathtt{Y} = \mathtt{0.299} \cdot \mathtt{R} + \mathtt{0.587} \cdot \mathtt{G} + \mathtt{0.114} \cdot \mathtt{B} \tag{3.1}$$

$$\mathtt{Cb} = -\mathtt{0.1687} \cdot \mathtt{R} - \mathtt{0.3313} \cdot \mathtt{G} + \mathtt{0.5} \cdot \mathtt{B} + 128 \tag{3.2}$$

$$\mathtt{Cr} = \mathtt{0.5} \cdot \mathtt{R} - \mathtt{0.4187} \cdot \mathtt{G} - \mathtt{0.0813} \cdot \mathtt{B} + 128 \tag{3.3}$$

### Chroma subsampling

Princip chroma subsamplingu spočívá v ukládání informace o barvě (Cb a Cr) v menším rozlišení, než je použito pro komponentu světelnosti (Y). Využívá tak vlastností lidského vnímání pro zmenšení velikosti výsledného souboru a také ke snížení množství prostředků nutného ke zpracování. Tento proces je ztrátový, jelikož místo původních několika barev uchováváme pouze hodnotu jejich průměru. Míra subsamplingu se značí pomocí tří čísel oddělených dvojtečkou. My si tyto čísla pro potřeby vysvětlení zaznačme jako A:B:C. Toto označení specifikuje oblast širokou A sloupců a dva řádky vysokou. Číslo B poté značí počet barevných složek v prvním řádku a číslo C počet políček, který může být rozdílný od prvního řádku. V aplikaci při ukládání do JPEG formátu budeme využívat subsampling s označením 4:2:0 tedy horizontálně i vertikálně budeme ukládat jen polovinu barevné informace.

### DCT

DCT je zkratkou pro discrete cosine transform. DCT se provádí pro každou složku modelu YCbCr zvlášť a je bezeztrátový. Základním krokem je rozdělit nejprve obrázek na bloky o velikosti 8x8 hodnot. Pokud není šířka či výška násobkem osmi je tento obrázek rozšířen na požadovanou velikost, a to většinou kopírováním posledního pixelu. Nyní je nutné upravit hodnoty pro DCT. Máme hodnoty v intervalu <0,255> a my je chceme dostat do hodnot okolo nuly v intervalu <-127,128>, proto odečteme hodnotu 128 od všech hodnot. Dostaneme 8x8 posunutých hodnot, tak jak je potřebujeme soustředěné kolem 0. Vypočteme DCT za pomoci vzorce 3.4 a 3.5 převzatého z [38]. Po výpočtu DCT dostaneme opět 8x8 hodnot, nyní se však jedná o hodnoty DCT, které nám udávají, jak moc je daná složka

zastoupena v naší zpracovávané části obrázku. Složky jsou uspořádány tak, že v horním levém horu získané 8x8 matice se nachází hodnota značící průměrnou hodnotu v bloku 8x8 a označujeme jí jako DC složku. Zbylých 63 složek označujeme jako AC a jsou uspořádány tak, že s každým narůstajícím indexem řádku narůstá horizontální frekvence a s každým indexem sloupce narůstá vertikální frekvence. To znamená, že vpravo dole nalezneme hodnotu komponenty s největší vertikální i horizontální frekvencí. Většinou po výpočtu zjistíme, že právě čísla vlevo nahoře jsou zásadně větší než čísla vpravo dole, což nám napovídá, že vysokofrekvenční kosinusovky se moc nepodílí na konečném výsledku a toho využívá kvantizace.

$$F_{u,v} = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f_{x,y}\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right] \tag{3.4}$$

$$\alpha(x) = \begin{cases} \frac{1}{\sqrt{2}}, \text{pro } x = 0 \\ 1, \text{pro ostatní} \end{cases} \tag{3.5}$$

$x$...je index řádku pixelu v rozsahu $< 0, 8)$

$y$...je index sloupce pixelu v rozsahu $< 0, 8)$

$u$...jsou horizontální frekvence v rozsahu $< 0, 8)$

$v$...jsou vertikální frekvence v rozsahu $< 0, 8)$

$f_{x,y}$...je hodnota pixelu na souřadnici $(x, y)$

$F_{u,v}$...je hodnota DCT na souřadnici $(u, v)$

### Kvantizace

Tento proces je opět jedním z velkých kroků k menší velikosti výsledného souboru, avšak je také opět ztrátový. Využívá se již zmíněného faktu o rozložení získaných DCT hodnot. V tomto kroku komprese se odeberou vysokofrekvenční data, která nejsou tolik důležitá pro výsledné zobrazení. K rozdělení toho, jak moc které frekvence potřebujeme, nám slouží kvantizační tabulky. Tyto tabulky jsou různé, a to podle požadované kvality výsledného obrazu a podle daného kompresoru. Navíc jsou různé tabulky pro složku jasu a složky barevnosti. Práce s touto tabulkou je jednoduchá. Vezmeme naše hodnoty DCT, tedy tabulku hodnot 8x8 a vydělíme každou hodnotu této tabulky hodnotou na stejné pozici v kvantizační tabulce a získanou hodnotu zaokrouhlíme na nejbližší celé číslo. Kvantizační tabulka je většinou tvořena hodnotami, které jsou větší v pravé dolní části, než v levé horní, to znamená, že tyto části se po vydělení budou více blížit k nule a většina dokonce bude nulová. Tato skutečnost se následně využije při kódování.

### Kódování

Než začne serializace je provedena ještě jedna úprava, a to DC koeficientů. Tato úprava spočívá v odečtení aktuální DC komponenty bloku od předchozí. Díky tomu, že obrázky obsahují souvislé barevné plochy, je rozdíl průměrných hodnot těchto dvou bloků malý, a tak jej bude možno zakódovat do menšího počtu bitů. Získané hodnoty jsou poté převedeny do posloupnosti hodnot, tak že je načítáme z původního pole cik cak z levého horního rohu do spodního pravého rohu. Cik cak, abychom dostali co nejdelší nulovou sekvenci, nad kterou následně provedeme run-length encoding a nakonec použijeme zakódování pomocí huffmanova kódování. Namísto huffmanova kódování JPEG standard dovoluje použít také

aritmetické kódování. Huffmanovo kódování je však častější. Jak tvrdí [32], "Ukazuje se, že snížení velikosti souboru o cca 5% – 10% v případě aritmetického kódování výrobci oželí a raději se soustředí na kvalitně provedené Huffmanovo kódování s možnou optimalizací kódovacích tabulek a se zárukou, že vytvořené soubory bude možné bez problémů prohlížet, či editovat v mnoha dalších aplikacích."

### Dekódování

Jelikož je kódovací proces bezztrátový, tak po provedení zpětného procesu opět dostaneme původní kvantizovanou tabulku.

### Inverzní kvantizace

Provedeme opačný proces kvantizace, tedy vynásobíme hodnoty získané dekódováním kvantizační tabulkou.

### Inverzní DCT

Provedení inverzní DCT za použití vzorců 3.6 a 3.7 uvedených v [38]. Výsledné hodnoty jsou zaokrouhleny na nejbližší celé číslo. Následuje posunutí hodnot z okolí nuly zpět do rozsahu <0,255>. Toho je docíleno přičtením hodnoty 128 ke všem získaným hodnotám. Po tomto kroku mohou být získané hodnoty mimo zmíněný rozsah a je nutné tyto přesahy rozsahu vrátit do mezí oříznutím.

$$f_{x,y} = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7}\alpha(u)\alpha(v)F_{u,v}\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right] \tag{3.6}$$

$$\alpha(x) = \begin{cases} \frac{1}{\sqrt{2}}, \text{pro } x = 0 \\ 1, \text{pro ostatní} \end{cases} \tag{3.7}$$

$x$...je index řádku pixelu v rozsahu $< 0, 8)$

$y$...je index sloupce pixelu v rozsahu $< 0, 8)$

$u$...jsou horizontální frekvence v rozsahu $< 0, 8)$

$v$...jsou vertikální frekvence v rozsahu $< 0, 8)$

$f_{x,y}$...je hodnota pixelu na souřadnici $(x, y)$

$F_{u,v}$...je hodnota DCT na souřadnici $(u, v)$

### Inverzní subsamplingu

Následuje provedení inverzního subsamplingu. Použijeme tedy podvzorkovanou hodnotu barvy pro více hodnot jasu.

### Převod YCbCr do RGB

V tomto kroku jsou obrazová data převedena z prostoru YCbCr do RGB a to za pomocí vzorců 3.8, 3.9, 3.10 jak uvádí specifikace JFIF formátu [12]. Tento krok je bezztrátový.

$$\mathtt{R} = \mathtt{Y} + \mathtt{0.587} \cdot \mathtt{G} + \mathtt{0.114} \cdot \mathtt{B} \tag{3.8}$$

$$\mathtt{G} = \mathtt{Y} - \mathtt{0.3313} \cdot \mathtt{G} + \mathtt{0.5} \cdot \mathtt{B} + \mathtt{128} \tag{3.9}$$

$$\mathtt{B} = \mathtt{Y} - \mathtt{0.4187} \cdot \mathtt{G} - \ \mathtt{0.0813} \cdot \mathtt{B} + \mathtt{128} \tag{3.10}$$

## 3.4 Vlastnosti metod

Dále budou uvedeny některé důležité vlastnosti, které od steganografických metod požadujeme, jak uvádí například [22].

### 3.4.1 Nepostřehnutelnost

Nepostřehnutelnost je nejdůležitějším parametrem při hodnocení steganografického algoritmu. Celá síla steganografie stojí na její nepostřehnutelnosti, jak pro lidské oko, tak i pro odhalení strojem. Jakmile je totiž vidět nějaká anomálie v obraze a zjistí se, že byl obraz upraven, je celá steganografie prozrazena. Nejedná se nám totiž, jako v případě kryptografie, aby nebyla skrytá zpráva jako taková odhalena, ale o to, aby nebyl odhalitelný samotný průběh tajné komunikace.

### 3.4.2 Kapacita

Jedním z důležitých kritérií je ovšem také kapacita. Kapacita nám udává, kolik dat můžeme maximálně do daného nosiče uložit při použití dané metody. Na rozdíl od steganografii příbuznému vodotisku, kde je potřeba vložit jen malé množství informace, je u steganografie vítána co největší kapacita.

### 3.4.3 Odolnost proti statistickým útokům

Statistické útoky jsou využívány v steganoanalýze pro detekci steganografických úprav. Při vkládání tajných dat do nosiče u některých metod vznikne nestandartní rozložení hodnot. Toho využívá například Chi-squared útok, který porovnává předpokládané statistické vlastnosti obrázku s vlastnostmi testovaného obrázku. Pokud není metoda odolná proti těmto útokům, vystavuje se opět nebezpečí snadné detekovatelnosti.

### 3.4.4 Robustnost

Dalším parametrem je robustnost neboli odolnost proti manipulaci s obrázkem. Při přeposílání steganogramu může nastat situace, že bude s obrázkem nějak manipulováno, bude otočen, ořezán, a tak podobně. K tomu může dojít buď záměrně s úmyslem poškodit případnou tajnou informaci v obrázku nebo nezáměrně, například v důsledku přenosu přes nějakou aplikaci. Každopádně odolnost vůči těmto úpravám je opět kladnou vlastností u steganografie.

### 3.4.5 Časová náročnost

V některých případech nás může zajímat doba za kterou daná metoda skryje anebo odkryje tajná data.

## 3.5 Metody obrazové steganografie

Pro obrazová média je dnes možné dohledat velké množství metod pro ukrytí tajné zprávy. Některé tyto metody vznikly tzv. od píky, některé pouze rozvíjejí existující metody a některé jsou vystavěny na spojení jiných existujících metod a využívají jejich vlastností k vylepšení výsledků vytvořené metody.

### 3.5.1 Primitivní injekční metoda

Tato metoda je zde pouze pro zajímavost, že to může fungovat. Jak pod operačním systémem Windows, tak Linux je možné vytvořit steganogram, a to bez jakéhokoliv steganografického programu. Tato metoda se dá zařadit mezi injekční metody, kde vkládáme za značku ukončení souboru. Mějme obrázek, do kterého chceme ukrývat s názvem `image.jpeg` a data, která chceme ukrývat s názvem `text.txt`. Data zkomprimujeme do archívu s názvem `archiv`. Ve Windows pomocí pravého tlačítka a možnosti odeslat, komprimovaná složka. Pod Linuxem příkazem `tar -cvf archiv.tar text.txt`. Poté stačí pomocí příkazové řádky daného systému spojit obrázek a archív. Pro Windows spustíme následující příkaz: `copy /b image.jpeg+archiv.zip steganogram.jpeg`. Pro Linux spustíme následující příkaz: `cat image.jpeg archiv.tar > steganogram.jpeg`. Tímto jsme v obou případech vytvořili soubor s názvem `steganogram.jpeg`, tento soubor se v prohlížeči chová jako normální obrázek. Pokud jej však ve Windows dáme otevřít jako archív dostaneme obsah skrytého archívu. Pod Linuxem nám postačí v příkazové řádce zadat příkaz `tar -xvf archiv.tar` pro extrakci dat.

Tato metoda krásně ukazuje nevýhodu injekčních metod. Pokud se podíváme na vzniklý obrázek, tak jeho velikost je rovna velikosti obrázku před skrytím plus velikost skrytých dat. Tímto při větším souboru vznikne podezřele veliký obrázek, a tudíž vzniká nápadný steganogram.

### 3.5.2 LSB - Least Significant Bit

V angličtině tuto metodu najdeme jako least significant bit nebo pod zkratkou LSB. Tato metoda je vhodná pro nekomprimované nebo bezztrátově komprimované obrázky, jelikož ztrátová komprimace by mohla poškodit zapsaná data. Metoda spadá pod substituční metody, kde nahrazujeme bity krycího obrázku, bity skrývaných dat [17]. Princip nahrazení je předveden na ukázce 3.3, kde je vloženo písmeno A za použití metody LSB s využitím posledních dvou bitů. Z ukázky navíc můžeme pochopit také to, že je poměrně pravděpodobné, že bity, které chceme zapsat se již shodují se zapsanými, a tudíž nedochází ke změně.

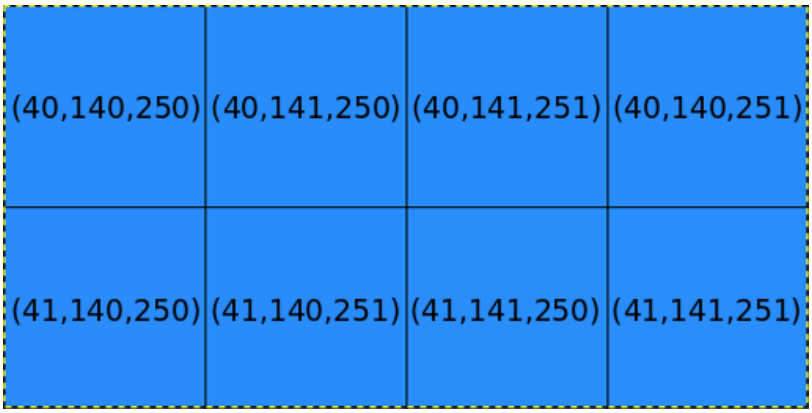
Ukládáme znak A –> dekadicky ASCII 65 –> binárně 01000001
Mějme dva pixely:
První –> (R,G,B) binárně (00110111,01010101,01101111)
Druhý –> (R,G,B) binárně (00010011,00111010,01011011)

Původně tedy máme:
00110**11** 01010**01** 01101**11** 00010**11** 00111010 01011011
Tučně jsou vyznačeny nahrazované bity a poslední 2 byty zůstaly nedotčené, jelikož nezapisujeme více než písmeno A.

Vznikne nám po zápisu bitů písmena A:
00110**0**1 01010**0**0 01101**00** 00010**0**1 00111010 01011011
Tučně jsou vyznačeny bity, které se změnily oproti originálu.

Obrázek 3.3: Ukázka vložení znaku A metodou LSB pri použití 2 bitů na barevném obrázku s barevnou hloubkou 24 bitů.

U metody LSB využíváme toho, že změna nejméně významných bitů nezpůsobí velké změny v finální hodnotě barvy a tudíž pro lidské oko nebude příliš nápadná, ne-li přímo nepostřehnutelná. Obrázek 3.4 zobrazuje 8 barevných čtverců jedné barvy umístěných vedle sebe, které mají v hodnotě barvy obměněny právě nejméně významný bit. Je patrné, že změna těchto bitů je nepostřehnutelná.




Obrázek 3.4: 8 barev lišících se v posledním bitu. Hodnoty RGB viz. hodnoty v závorkách.

Tato metoda může být různě implementovaná, jelikož nám dovoluje měnit hned několik faktorů, které zároveň ovlivní konečné množství dat, které budeme moci umístit do krycího obrázku. Jedním z těchto faktorů je počet bitů použitých pro skrývání dat. Pokud bychom využili u barevného obrázku s rozlišením 1920x1080 pouze poslední bit každé barevné složky, pak bychom dostali prostor pro data o velikosti 259,2 kB, avšak pokud bychom využili 2 poslední bity, tak by kapacita vrostla na 518,4 kB a s každým dalším bitem bychom kapacitu zvyšovali o 259,2 kB. Ovšem nic není zadarmo a s každým bitem použitým pro skrytí dat ubíráme počet bitů, které zůstanou nezměněny, a tudíž nesou originální hodnotu barvy obrázku. Právě změna většího počtu posledních bitů má za následek zkreslení obrazu, a tak zapříčiní větší nápadnost v důsledku toho i větší šanci na odhalení. V tabulce 3.1 je vidět v procentech, jak moc jednotlivé bity ovlivňují konečnou hodnotu barevné složky.

Z tabulky je patrné, že při změně pouze posledního bitu ovlivníme konečnou barvu jen o 0.3906% a to jen v případě, že se liší hodnota tohoto posledního bitu a bitu, který chceme uložit ze skrývaných dat. Pokud budeme provádět ukládání do posledních dvou bitů, tak zdvojnásobíme kapacitu a stále změníme jen něco málo přes jedno procento hodnoty barvy. Avšak čím více bitů měníme, tím se nám zvětšuje dopad změn, jak demonstruje obrázek 3.5, kde jsou postupně nulovými bity od nejméně po nejvíce významný. Dalším faktorem, který můžeme implementovat různě je například rozložení bitů na jednotlivé barevné složky, můžeme například bity přepisovat po barvách postupně anebo nejprve naplnit jednu barvu a pak postoupit k další.

| **N-tý bit** | **Procentuální vliv** |
|---|---|
| 7 | 50% |
| 6 | 25% |
| 5 | 12.5% |
| 4 | 6.25% |
| 3 | 3.125% |
| 2 | 1.5625% |
| 1 | 0.7813% |
| 0 | 0.3906% |

Tabulka 3.1: Procentuální vliv jednotlivých bitů na 8 bitovou hodnotu.

Ve vytvořené aplikaci je tato metoda implementována tak, aby využívala rovnoměrně všechny barevné komponenty a je umožněno volit počet využitých bitů pro skrývání. Dodaným prvkem je možnost náhodné redistribuce tajných dat po celém obrázku. Toto rozložení dat po celém obrázku pomocí náhodnosti přináší dodatečné zabezpečení. Při sekvenčním ukládání tajných dat do po sobě jdoucích pixelů může útočník jednoduše zkusit vyčíst poslední bity a složit tajnou zprávu. Pokud použijeme náhodné rozložení, tak útočník musí navíc vědět, jak získané bity za sebou následují.

Obrázek 3.5: Nulování posledních bitů, kde vlevo nahoře je zobrazen originál a postupně jsou nulováno více bitů, obrázky jsou seřazeny v prvním sloupci směrem dolů a pokračují v druhém sloupci dolů a poslední vpravo dole je obrázek s nenulovaným pouze 8. bitem každé barvy.

### 3.5.3 PVD - Pixel Value Differencing

Da-Chun Wu a Wen-Hsiang Tsai, představili v [39] metodu PVD pro ukrývání dat do obrázku. Tato metoda byla představena na šedotónových obrázcích, avšak lze ji modifikovat pro barevné obrázky. Skrytá data mohou být obnovena bez původního obrázku. Při představení teto metody šlo především o vylepšení parametru nepostřehnutelnosti bez ohledu na robustnost. Tato metoda je založena na vlastnostech lidského vidění. Jak již bylo zmíněno u LSB, tak ne všechny oblasti obrázku snesou stejné množství změn. Tyto oblasti jsou například obloha, voda, zdi, obecně řečeno jednolité plochy. Právě s tímto faktem se snaží metoda PVD vypořádat výpočtem rozdílu hodnot zvolených pixelů. Pokud je rozdíl malý jedná se o jednolité oblasti, a pokud je velký, jedná se o různorodé oblasti, kde jsou změny pro člověka nepatrnější. Tímto metoda vylepšuje nepostřehnutelnost. Dále jsou dohledatelné i jiné verze PVD, které staví na stejném základu, avšak přidávají jisté úpravy. Mezi tyto úpravy patří například vkládání pomocí LSB při určité podmínce nebo práce s více pixely místo původních dvou následujících. Poměrně pěkně jsou některé upravené verze PVD stručně popsány v [6], společně s tabulkou, která umožňuje jejich porovnání. Tato práce bude vycházet z metody popsané v [39], která pracuje tak, že vybírá vždy dva sousedící pixely a určí rozdíl mezi hodnotami těchto pixelů. Podle tohoto rozdílu pak určí příslušnou kategorii v kvantizační tabulce. Jednotlivé kategorie pak určují, jaké množství dat (kolik bitů) může být uloženo. Tyto data se poté zakomponují jako nové hodnoty zvolených pixelů.

Skrývání probíhá následovně. Obraz je rozčleněn na dvojice po sobě jdoucích (sousedících) pixelů, které se vzájemně nepřekrývají. Tyto pixely budeme značit $\mathtt{p}_i$ a $\mathtt{p}_{i+1}$ a jsou vybírány cikcak tzn. první řádek zleva do prava a následující zprava do leva a s následujícími řádky postupujeme zase od začátku. Vypočítáme rozdíl hodnot dvou po sobě jdoucích pixelů jako $\mathtt{g}_{i+1} - \mathtt{g}_i$ a získáme `d`, nabývající hodnot v rozsahu `<-255,255>`. Pomocí `d` jej zařadíme do jedné z kategorií označme jí `k`. Pro tyto účely můžeme hodnotu `d` převést na její absolutní hodnotu a tím získáme hodnoty z rozsahu `<0,255>`. Kategorie jsou rozsahy hodnot rozdílů, které můžeme získat a vzájemně na sebe navazují neboli je to interval možných hodnot `d` rozdělený do několika subintervalů. Tyto jednotlivé kategorie tedy mají svou spodní hraníci `L` a horní hranici `U`. Z toho jednoduše vypočteme šířku dané kategorie jako $\mathtt{U} - \mathtt{L} + 1$. V metodě představené v [39] jsou tyto šířky druhou mocninou. Jednotlivé šíře jsou pak zvoleny v závislosti na lidském vidění, hodnoty spadající do jedné kategorie jsou považovány za vzájemně nahraditelné bez vzbuzení podezření, že došlo ke změně. Pro menší rozdíly, které značí plynulé přechody, je tato šířka menší a pro větší rozdíly, které značí barevné přechody, mají naopak větší šířku, a to znamená i větší kapacitu. Nejdříve tedy musíme vypočítat kolik bitů, označme jejich počet `n`, můžeme uložit dle dané kategorie pomocí výpočtu $\mathtt{n} = \log_2(\mathtt{U}_k - \mathtt{L}_k + 1)$. Poté vyjmeme těchto n bitů z ukrývané zprávy a jejich dekadickou hodnotu nazvěme `b`. Vypočítáme nový rozdíl `d'` pomocí

$$\mathtt{d'} = \begin{cases} \mathtt{L}_k + \mathtt{b} & \text{pro } \mathtt{d} \geq 0 \\ -(\mathtt{L}_k + \mathtt{b}) & \text{pro } \mathtt{d} < 0 \end{cases} \tag{3.11}$$

Poté provedeme inverzní výpočet podle násedujícího vzorce, kde $\mathtt{m} = (\mathtt{d'} - \mathtt{d})/2$ , abychom z nového rozdílu získali patřičné nové hodnoty pixelů, z kterých poté můžeme obnovit data.

$$f\big((\mathtt{g}_i, \mathtt{g}_{i+1}), m\big) = (\mathtt{g'}_i, \mathtt{g'}_{i+1}) = \begin{cases} (\mathtt{g}_i - \lceil \mathtt{m} \rceil, \mathtt{g}_{i+1} + \lfloor \mathtt{m} \rfloor) & \text{pro sudé} \\ (\mathtt{g}_i - \lfloor \mathtt{m} \rfloor, \mathtt{g}_{i+1} + \lceil \mathtt{m} \rceil) & \text{pro liché} \end{cases} \tag{3.12}$$

Problém nastává pokud se pohybujeme příliš u hranic intervalu <0,255>, jelikož bychom se mohli dostat mimo něj. To by se dalo řešit posunem hodnoty, která vypadla za hranici na danou hranici a patřičně upravit druhou hodnotu, avšak to by mohlo vyvolat viditelné změny. Proto se v [39] používá mechanismus "falling-off-boundary checking", který zkontroluje, zda může dojit k přesažení hranic. Pokud ano, tak není daná dvojice pixelů využita pro kódování, a tudíž ani pro dekódování. Tento mechanismus je prakticky předchozí funkce, avšak s $\mathtt{m} = \mathtt{U}_k - \mathtt{d}$.

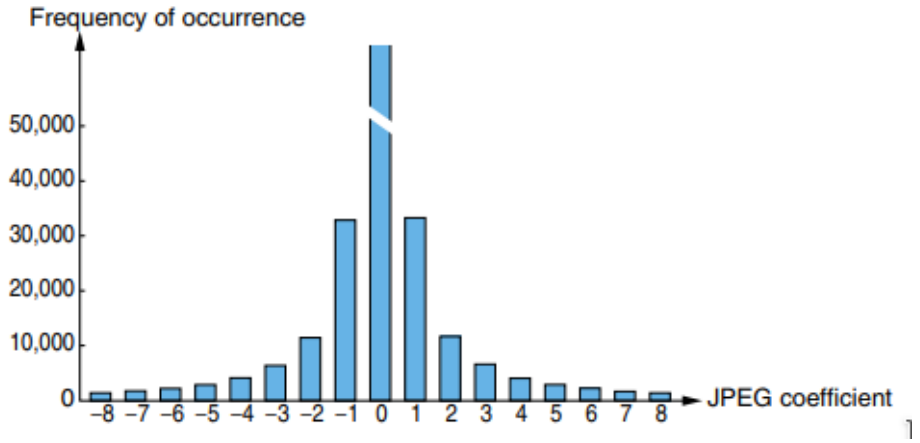
Odtajnění skrytých dat probíhá podobně jako skrývání. Při odkrývání se opět vybírají stejným stylem pixely a získává se rozdíl jejich hodnot `d` provede se "falling-off-boundary checking", který rozhodne jestli se jedná o použitelnou dvojici pixelů a zda provádět odtajnění nebo nikoliv. Pokud ano, zjistí do které kategorie rozdíl dvou pixelů spadá a samotné získání skrytých dat se pak provádí výpočtem dle vzorce

$$\mathtt{b} = \begin{cases} \mathtt{d} - \mathtt{L}_k & \text{pro } \mathtt{d} \geq 0 \\ -\mathtt{d} - \mathtt{L}_k & \text{pro } \mathtt{d} < 0 \end{cases} \tag{3.13}$$

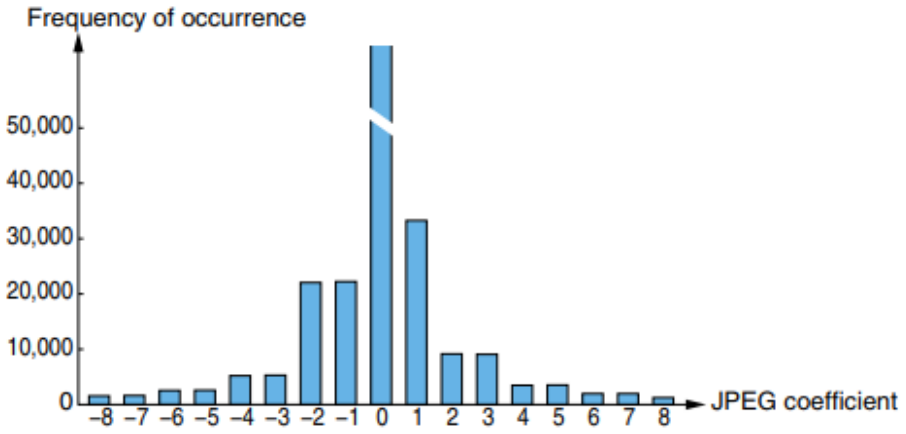
kde získané `b` je dekadická hodnota, která byla skryta. Pomocí určené kategorie, do které spadá rozdíl `d` vybraných pixelů a výpočtu počtu bitů `n`, který může být do dané kategorie skryt poté určíme kolik bitů z dané dekadické hodnoty tvoří nesená data.

### 3.5.4 JSTEG

JSTEG je jednou ze steganografických metod pro JPEG, kterou představil Derek Upham v [35]. Princip je velice podobný s principem LSB, jak byl popsán výše. U metody JSTEG se však místo se samotnými hodnotami pixelů pracuje s hodnotami kvantizované tabulky hodnot DCT. K ukládání se využívají všechny AC koeficienty, kromě těch s hodnotou 0 a 1. Využití nuly by sice značně zvýšilo kapacitu, ale přineslo by nežádoucí viditelné změny oproti původnímu obrázku a zároveň by se projevilo na zvětšení velikosti daného souboru. Jednička poté nemůže být využita z důvodu jejího rozpoznání při zápisu hodnoty 0 do posledního bitu, jelikož se v tomto případě jednička změní na 0 a tu nelze rozpoznat od nepozměněné nuly vyňaté ze zpracování. Tato metoda by měla mít dobrou odolnost vůči vizuálním útokům, avšak zároveň se výsledný obrázek odchyluje od očekávaného histogramu DCT koeficientů, a tak se vystavuje snadné odhalitelnosti pomocí statistického útoku. Porovnání rozložení hodnot po použití této metody a očekávaného steganogramu můžeme vidět na obrázku 3.6 a 3.7.




Obrázek 3.6: Ukázka standartního rozložení hodnot pro JPEG po kvantizaci převzato z [35].

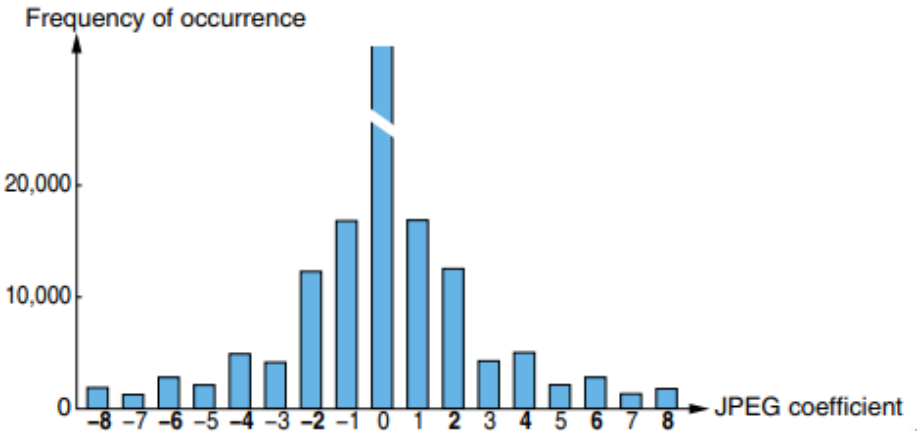



Obrázek 3.7: Ukázka rozložení hodnot pro JPEG po kvantizaci a vložení dat metodou JSTEG převzato z [35].

### 3.5.5 F3

F3 je také steganografickou metodou pro JPEG. Navrhl jí, stejně jako u F4 a F5, Andreas Westfeld [35]. Tato metoda na rozdíl od metody JSTEG nepřepisuje bity. Namísto přímého přepisování bitů odečítá jedničku od absolutní hodnoty koeficientu DCT tak, aby se poslední bit rovnal ukládané hodnotě. Opět nevyužíváme nulu a komponentu DC. O nevyužití komponenty DC se v mnohých pracích nemluví. Dokonce sám Andreas Westfeld v [35] tuto skutečnost zmiňuje až v ukázkovém kódu. Vynechání DC složky je však zcela logické, jelikož tato hodnota je průměrem hodnot v celém 8x8 bloku a jen drobná změna této komponenty by se podstatně projevila na konečném obrázku. Zároveň však může dojít k jevu nazývanému "shrinkage". K tomuto jevu dochází, pokud je hodnota DCT 1 nebo -1 a hodnota dat 0, pak dojde ke změně na nulu. Opět nejsme v tomto případě schopni rozlišit mezi nulami nepoužitými a těmi, které vznikly v důsledku ukládání dat. V případě, že při ukládání dat dojde k tomuto jevu, je nutné pokračovat v uložení stejné hodnoty znovu, dokud nebude řádně uložena. Tento proces sebou nese další nevýhodu a tou je, že není možné dopředu určit kapacitu daného obrázku, jelikož ta je vyvozena až podle dat. Tato metoda má za následek nerovnoměrné rozložení koeficientů. Vyskytuje se zde statisticky

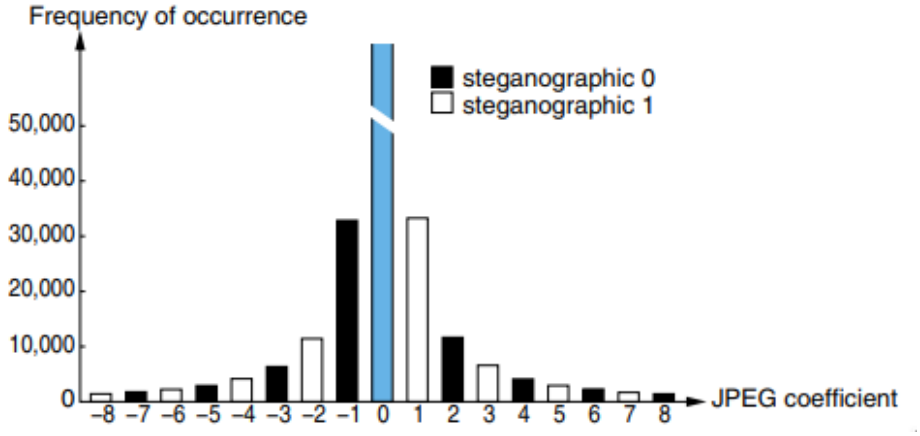
více sudých než lichých hodnot koeficientů. F3 je metoda odolná vůči vizuálním útokům, avšak je náchylná k statickému útoku. Porovnání rozložení hodnot po použití této metody a očekávaného steganogramu můžeme vidět na obrázku 3.6 a 3.8.




Obrázek 3.8: Ukázka rozložení hodnot pro JPEG po kvantizaci a vložení dat metodou F3 převzato z [35].

### 3.5.6 F4

F4 je vylepšenou verzí metody F3. Metoda F4 odstraňuje neduh nerovnoměrného rozložení koeficientů, tedy převahu sudých koeficientů nad lichými. Tento problém odstraňuje pomocí změny logiky ukládání dat. Při kladných koeficientech jsou data uložena stejně jako tomu bylo u F3, ale u záporných koeficientů ukládáme invertovanou hodnotu bitu dat. F4 je metoda odolná vůči vizuálním i statistickým útokům. Porovnání rozložení hodnot po použití této metody a očekávaného steganogramu můžeme vidět na obrázku 3.6 a 3.9.




Obrázek 3.9: Ukázka rozložení hodnot pro JPEG po kvantizaci a vložení dat metodou F4 převzato z [35].

### 3.5.7 F5

F5 je ve svém principu F4 s přidanými mechanismy. Jedním z nich je rozdistribuování ukládaných dat po celém obrázku tzv. "permutative straddling" a nikoliv jejich sekvenční

zápis jako tomu bylo u metod popsaných výše. Tento postup při nezaplnění celé kapacity měl za následek zhoršení kvality obrázku v jeho vrchní části a tím ubíral na nepozorovatelnosti ukrytí dat. Dalším mechanismem je takzvaný "matrix encoding", který snižuje počet provedených změn v obrázku nutných pro zapsání dat. Pro ukrytí 2 bitů využívá hodnot 3 koeficientů, to také znamená, že platíme zmenšením kapacity za snížení počtu nutných změn, které však vedou k horší detekovatelnosti steganografie. Ve vztazích uvedených v 3.14 převzatých z [35] máme dva bity x1 a x2 a tři bitové pozice, na které je chceme uložit. Pomocí exkluzivní disjunkce pak zakódujeme dané datové bity na tyto pozice a v nejhorším případě dojde maximálně ke změně na jedné pozici. F5 je metoda odolná vůči vizuálním i statistickým útokům.

$$\begin{aligned} x_1 = a_1 \oplus a_3,\ x_2 = a_2 \oplus a_3 &\implies \text{beze změny} \\ x_1 \neq a_1 \oplus a_3,\ x_2 = a_2 \oplus a_3 &\implies \text{změň } a_1 \\ x_1 = a_1 \oplus a_3,\ x_2 \neq a_2 \oplus a_3 &\implies \text{změň } a_2 \\ x_1 \neq a_1 \oplus a_3,\ x_2 \neq a_2 \oplus a_3 &\implies \text{změň } a_3 \end{aligned} \tag{3.14}$$

### 3.5.8 Inverted LSB

Je metoda předložena v [4], která je založena na metodě LSB s cílem snížit zhoršení kvality výsledného steganogramu. V [4] byly představeny dva přístupy využívající inverzi bitů. Celý princip je zjednodušeně založen na inverzi LSB při nalezení jistého vzoru v daném pixelu. Tento proces je vykonáván s cílem snížit počet pozměněných bitů, a tak zvýšit nepostřehnutelnost a kvalitu.

Jako první bude popsán první přístup k inverzi LSB. Tento přístup spočívá v aplikaci metody LSB a vytvoření kategorií dle druhého a třetího nejméně významného bitu. U těchto kategorií poté spočítáme u kolika z nich byl změněn poslední bit metodou LSB oproti originálu. Pokud je více změněných než nezměněných bitů, tak invertujeme všechny poslední bity dané kategorie. Tento přístup vede k nutnosti někde uchovávat informaci o tom, která kategorie ze 4 byla invertována.

Druhý přístup. Tento přístup vychází z předpokladu, že obě strany mají dostupný originál obrázku do kterého byla ukryta data. V tomto přístupu se k třetímu a druhému bitu přidává i nejméně významný bit originálu. Postupujeme podobně jako u prvního přístupu a vytvoříme jakousi kategorizaci, kde opět musíme nějak přenést informaci o překlopení bitů, zde však z důvodu přidání nejméně významného bitu je těchto kategorií 8. Pro každou kombinaci 3. a 2. nejméně významného bitu můžeme pozorovat 4 stavy. První, kdy LSB byl změněn z 0 na 1. Druhý, kde LSB byl 0 a zůstal 0. Třetí, kde se LSB změnil z 1 na 0. Poslední je stav kdy LSB byl a zůstal nastaven na 1. Pravidlo pro inverzi LSB je pak následující. Pokud je počet výskytů prvního stavu větší než druhého stavu, tak invertujeme všechny LSB v příslušné kategorii, a těch kde zůstala 0. Dále, pokud je výskyt třetího stavu větší, než výskyt čtvrtého stavu, invertujeme všechny LSB v příslušné kategorii a LSB těch, kde zůstala 1.

Jednoduše řečeno rozdíl mezi prvním a druhým přístupem invertování spočívá ve větší přesnosti rozlišování změn. V případě prvního přístupu nás zajímají celkově změny jako takové. V druhém přístupu pak rozlišujeme mezi změnou z 0 nebo z 1, a tak máme větší počet možných inverzí.

Přístup 2 testovaný v rámci představení metody v [4] vykazovalo větší nárust kvality obrázku než přístup 1, které však také oproti čistému LSB zlepšilo kvalitu obrázku. Nevý-

hodou druhého principu je nutnost vlastnit originál obrázku do kterého jsou ukryty data pro odtajnění zprávy.

V rámci vytvořené aplikace je použit první přístup, jelikož nevyžaduje, aby obě strany komunikace měli původní obrázek. Což osobně vnímám jako výhodnější pro komunikaci tohoto typu.

### 3.5.9 Triple-A

Tato metoda byla představena v [11] a je založena na metodě LSB. Měla by díky zavedení náhodnosti zvýšit bezpečnost a kapacitu. Uvedené tvrzení autora práce o zvětšení kapacity je pravděpodobně bráno v rámci nejzákladnějšího LSB s použitím jednoho bitu. V případě použití metody LSB využívající 3 nejméně významné bity bychom dosahovali stejné nebo vyšší kapacity.

Metoda pracuje na základě náhodnosti, a navíc je v popisu této metody obsaženo i šifrování. Z tohoto důvodu je nutné mít v procesu heslo a nejlépe od uživatele. V této metodě se nejdříve zašifruje zadaná tajná informace a vytvoří se dva pseudonáhodné generátory čísel. První generátor vrací celá čísla v intervalu <0,6> a druhý v intervalu <1,3>. První slouží pro zvolení složek, do kterých se bude ukládat a druhý pro zvolení kolik nejméně významných bitů se bude používat u vybraného pixelu. Výběr pixelu jsem v této práci nezaznamenal, avšak předpokládám základní výběr popořadě.

Tato metoda by se dala udělat obecněji než je prezentována. Navrhuji tedy postup, kde by byla zachována tabulka výběru komponent, avšak tabulku počtu bitů bych vytvořil celou až do 8 bitů a omezil tuto volbu při vytvoření dané instance této metody. Požadavek na šifrování dat automaticky nemusí být úplně vhodný zejména pro zmiňované využití při zákazu šifrování. Proto bych možnost šifrování nechal volitelnou. Poslední navrhovanou úpravou je možnost umožnit volbu pixelů náhodně namísto sekvenčního průchodu.

### 3.5.10 EMD - Exploiting Modification Direction

Hlavní myšlenkou metody představené na šedotónových obrázcích v [41] je, že každá tajná číslice v systému s základem $(2n + 1)$ je nesena na $n$ pixelech a v nejhorším případě bude hodnota jednoho pixelu zvýšena nebo snížena o 1. Velikost základu $(2n+1)$ vychází z faktu, že pro skupinu $n$ pixelů existuje $2n$ možností modifikace tak, aby byla změněna hodnota pouze jednoho pixelu, k tomu je přičtena 1, jelikož je zde možnost shody s hodnotou pixelů jak jsou.

Nejprve bude popsáno společné pro skrývání i odkrývání tajných dat. Tím je převedení dat z dvojkového základu (binární reprezentace dat) na části s $\mathtt{L}$ bity a každá decimální hodnota těchto $\mathtt{L}$ bitů může být reprezentována $\mathtt{K}$ číslicemi systému se základem $(2n + 1)$. Toto rozdělení splňuje vztah 3.15. Tento postup je aplikován u skrývání pro odkrývání je proveden obráceně. Dále náhodnou procházkou skrze všechny pixely v obrázku vytvoříme skupiny pixelu, tak aby odpovídaly zmíněným vztahům. Nyní tedy jsou pixely rozděleny do skupin po $n$ pixelech.

$$\mathtt{L} = \lfloor \mathtt{K} \cdot \log_2(2n + 1) \rfloor \tag{3.15}$$

Pro skrytí dat mějme hodnoty těchto pixelů označeny jako $g_1, g_2, g_3 \ldots, g_n$ a vypočítáme extrakční funkci $f$ jako zbytek po dělení váženého součtu hodnotou $(2n + 1)$ dle vzorce 3.16. Pro skrytí tajného čísla $d$ v systému o základu $(2n + 1)$ do obrázku se postupuje

následovně. Pokud je $d = f$, neděláme nic. Pokud $d \neq f$, vypočítáme hodnotu $s$ podle vzorce $s = (d - f) \bmod (2n + 1)$ a postupujeme podle funkce 3.17. Při skrytí může nastat situace, kdy máme přičíst nebo odečíst 1, ale tím bychom vypadli z rozsahu hodnot pixelu. Tuto situaci řešíme opačnou operací a novým přepočítáním celé skupiny tedy i extrakční funkce. Tímto krokem se nám změní i pozice, kde dojde ke změně, a tudíž může opět nastat stejný problém, v tomto případě postupujeme stejným způsobem a po konečném počtu kroků se nám povede data vložit. I přes úpravu více pixelů nedojde k velkému dopadu na kvalitu, a navíc tyto krajní podmínky se vyskytují v přirozeném obrázku zřídka.

$$f(g_1, g_2, g_3 \ldots, g_n) = \left[ \sum_{i=1}^{n} (g_i \cdot i) \right] \bmod (2n + 1) \tag{3.16}$$

$$h(g_1, g_2, g_3 \ldots, g_x, s) = \begin{cases} g_s = g_s + 1 & \text{pro } s \leq n \\ g_{2n+1-s} = g_{2n+1-s} - 1 & \text{pro } s > n \end{cases} \tag{3.17}$$

Pro odkrytí dat u příjemce stačí mít vypočítanou hodnotu funkce $f$ z dané skupiny pixelů. Tato hodnota reprezentuje skryté číslo, avšak stále v systému o základu $(2n + 1)$. Zbývá získané číslo do dvojkového základu. Po tomto převodu získáme zpět $L$ bitů tajných dat.

Tato metoda byla v rámci [41] testována a porovnána s metodami skrývání v [35, 40, 21]. V rámci tohoto testování bylo zjištěno, že představená metoda má větší efektivitu vkládání, která přináší menší narušení obrazu a větší bezpečnost.

# Kapitola 4

# Rozbor k implementaci

## 4.1 Průzkum existujících aplikací

Na internetu je mnoho dostupných nástrojů, které se zabývají aplikací steganografie, a to na různé typy nosičů. Tato práce se zaobírá obrazovou steganografií, a proto také tato část, bude zaměřena na nástroje, které pracují s obrazovým nosičem. Většina nástrojů je uzpůsobena pro práci jen s určitými formáty souborů, kde se nejčastěji setkáme s formáty jako jsou BMP, JPEG, GIF, PNG, méně časté jsou pak například formáty TIFF, TGA nebo MNG. V mnoha případech je pak výstupní formát pouze jeden. Dalším rozdílem mezi nástroji je jejich rozhraní. Nástroje nabízí více či méně uživatelsky přívětivé rozhraní a to buď ve formě grafického uživatelského rozhraní (GUI - Graphical User Interface), rozhraní příkazového řádku (CLI - Command Line Interface) nebo obojího. Mnou testované nástroje nabízely ve většině případů čistě grafické rozhraní. Testovány byly programy: Hide'N'Send, SteganPEG, OpenStego, SSuite Piscel, Crypture, Steghide, Xiao Steganography, SilentEye. Všechny tyto nástroje jsou zdarma a některé z nich mají dostupný zdrojový kód. Některé nabízí možnost dodatečného zabezpečení dat pomocí šifrování a některé jsou i multiplatformní. Nepříliš často se aplikovaná steganografie dala více ovlivňovat uživatelem.

## 4.2 Požadavky na aplikaci

Jelikož požadavky na aplikaci k využití steganografických metod nebyly blíže specifikovány v zadání práce, stanovil jsem tyto požadavky sám v závislosti na vyzkoušených dostupných aplikacích a vlastní představě o finální aplikaci. Vytvořený návrh jsem konzultoval s vedoucím bakalářské práce.

Co se tedy týká cílů kladených na aplikaci, tak bude aplikace ve verzi s GUI a také s CLI. CLI hlavně z důvodu spouštění pomocí skriptů na testování a porovnání implementovaných metod. Grafické rozhraní výsledné aplikace by mělo být rozděleno na logické části, a to minimálně na část sloužící pro skrývání dat a na část pro odkrývání dat. Toto oddělení mi osobně u některých dostupných nástrojů chybělo a ubíralo to na celkové přehlednosti kvůli většímu množství ovládacích prvků. Na vstupu bude jako krycí medium možno vložit různé obrazové formáty. Aplikace bude navržena tak, aby podporovala více výstupních obrazových formátů a k nim příslušné steganografické metody. U těchto metod bude umožněna jejich volba. Pokud metoda bude pracovat s generátorem náhodných čísel, bude uživateli umožněno zadání hesla pro vytvoření počátečního stavu (seed). V aplikaci bude dostupný ukazatel kapacity a zaplnění zvoleným souborem. Jako skrytá data bude možno

zvolit jakýkoliv soubor, který bude uživatel moci dodatečně opatřit kontrolním součtem, pro ověření nepoškozenosti nesených dat a bude moci tyto data před uložením zašifrovat s volbou vlastního hesla. Aplikace by dále měla být spustitelná pod operačními systémy Windows a Linux.

## 4.3 Použité technologie

**Java** Programovací jazyk nebyl v zadání této bakalářské práce specifikován. Programovacích jazyků je dnes velké množství a počet parametrů, dle kterých se dají porovnávat je také mnoho, proto jsem omezil výběr pouze na populární jazyky. Popularnost jazyka se dá vyvodit podle mnoha různých pohledů. Někdo jí bere podle množství pull požadavků na githubu, který je udáván zde https://octoverse.github.com/ nebo podle různých průzkumů jako je například ten, provedený na stackoverflow https://insights.stackoverflow.com/survey/2017 nebo TIOBE Index https://www.tiobe.com/tiobe-index/. Všechny se ale v konečném důsledku shodují na pětici nejpopulárnějších jazyků, kterými jsou Java, C++, Python, C# a C. Jelikož bych chtěl aplikaci psát objektově a multiplatformě, tak jsem zmíněné jazyky omezil pouze na Javu, C++ a Python, které toto umožňují. Dále jsem uvažoval nad rychlostí výsledné aplikace, přestože aplikace nemá za účel být extrémně rychlá při skrývání a odkrývání tajných dat. Z toho důvodu a také z důvodu osobních preferencí syntaxe jazyka jsem se rozhodl vyřadit jazyk Python. Nakonec jsem ze zbylých dvou jazyků zvolil jazyk Java, přestože jsem ho dosud v žádném projektu nevyužil, na rozdíl od C++ a Pythonu. Hlavními důvody bylo pěkné GUI pomocí JavaFX, které se zobrazuje stejně na různých platformách a jeho přenositelnost bez nutnosti překompilování. Kvůli JavaFX byla zvolena Java ve verzi 8 pro usnadnění distribuce.

**Commons CLI** Tato knihovna přináší API pro zpracování parametrů zadaných při spuštění aplikace z příkazového řádku. Dále tato knihovna umožňuje zobrazovat vygenerovanou nápovědu pro dané přepínače. Její použití se dá rozdělit do tří fází. V první fázi jsou definovány požadavky na parametry. V druhé fázi dochází k předání parametrů a argumentů příkazové řádky ke zpracování. A v poslední fázi je již možné přistupovat ke zpracovaným hodnotám přepínačů nebo se například tázat, zda byly zadány a tak podobně. Knihovna je dostupná ke stažení na stránce https://commons.apache.org/proper/commons-cli/ a to pod licencí Apache Licence, verze 2.0.

**JavaFX** JavaFX nabízí širokou sadu nástrojů pro tvorbu Java aplikací s moderním, hardwarově akcelerovaným a multiplatformním rozhraním. JavaFX byla obsažena naposledy ve verzi Java JDK 8, poté byla oddělena a nyní je spravována jako OpenJFX projekt. Je licencovaná jako "GNU General Public License with Classpath Exception" a je možné jí stáhnout na stránce https://openjfx.io/. Podle https://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html#6 má JavaFX nahradit starší Swing. JavaFX nabízí dvě možnosti ke tvorbě uživatelského rozhraní. První možností je tvořit jednotlivé instance komponent a ty poté přidávat do layoutů. Druhou a mnou zvolenou variantou je využít FXML jako jazyk pro návrh formuláře. Tento přístup je navíc podle mě čistější, jelikož lze více oddělit logiku od uživatelského rozhraní a dojít, tak k návrhovému vzoru MVC (Model-View-Controller).

**ControlsFX** Doplňuje komponenty JavaFX o další užitečné ovládací prvky. Knihovna je k dispozici pod licencí "3-bodová BSD licence" a je dostupná na stránkách http://fxexperience.com/controlsfx/.

**libjpeg-turbo** Knihovna libjpeg-turbo slouží ke kompresi a dekompresi obrázku ve formátu JPEG. Implementuje jak libjpeg API, tak TurboJPEG API. Výhodou této knihovny je zpřístupnění DCT koeficientů po ztrátové kompresi prostřednictvím vlastního filtru. Jeho výhodou je API pro jazyk Java. Je ke stažení na https://libjpeg-turbo.org/. Licence je složena ze 3 kompatibilních licencí více informací na https://github.com/libjpeg-turbo/libjpeg-turbo/blob/master/LICENSE.md.

## 4.4 Zvolený typ skrývané informace

Po zvážení byl vybrán jako typ skrývané informace všeobecně jakýkoliv soubor. Jinak řečeno, metody implementované v aplikaci nebudou nijak vázány na daný typ skrývané informace a budou tyto data brát jako posloupnost jedniček a nul. Tato volba se mi osobně zdála nejlepší, jelikož nedojde k omezení uživatele na daný typ souboru. Tento přístup má další výhodu. Touto výhodou je možnost zpracování souboru v jiném programu a jeho následné skrytí. Příkladem může být vlastnost některých existujících aplikací, ukládat více souborů do jednoho obrázku. Tato vlastnost ve vytvořené aplikaci nemusí být implementována přímo, jelikož v případě, že budeme chtít uložit více souborů najednou, můžeme tyto soubory komprimovat do jednoho souboru a ten poté skrýt do obrázku pomocí vytvořené aplikace.

## 4.5 Koncepce aplikace

Při vývoji aplikace bylo dbáno na oddělení logické a uživatelské části. Pro demonstraci steganografických metod byly implementovány dvě aplikace sdílející stejný logický základ, popsaný níže v části "Logické jádro". Tyto aplikace se liší použitým uživatelským rozhraním. První z nich využívá grafické uživatelské rozhraní a bude popsána v části "Grafické uživatelské rozhraní" uvedené níže. Druhým rozhraním je rozhraní příkazového řádku, které bude taktéž podrobněji popsáno v samostatné části a to části "Rozhraní příkazového řádku". Jak již bylo zmíněno, tak pro implementaci aplikací byl zvolen jazyk Java doplněný o některé volně dostupné knihovny. Knihovny byly využity pro tvorbu grafického rozhraní, načítání argumentů v rozhraní příkazového řádku a pro usnadnění práce s formátem JPEG, který jak již bylo zmíněno je celkem komplexní. Kde však knihovny využity nebyly je část zabývající se samotnými steganografickými metodami. Tyto části byly implementovány podle dostupných zdrojů. U zdrojů jsem se snažil nalézt informace přímo od autora metody, avšak v některých případech byly těžko dohledatelné z důvodu zrušení odkazů a podobných patálií. V těchto případech bylo čerpáno z popisů dané metody v jiných pracích, které čerpaly z původní práce popisující danou metodu.

## 4.6 Logické jádro

Tato část se blíže zabývá implementací logického jádra aplikace společného pro obě implementované rozhraní. U této části byla co největší snaha odstínit logiku celé aplikace,

umožňující vkládat tajná data do obrazového nosiče, od jejího rozhraní. Z tohoto důvodu je celá logika umístěna zvlášť v balíčku *logic*.

### 4.6.1 Výjimka

Skrze celou aplikaci je využívaná výjimka třídy *UserException*, která je zděděná od třídy *Exception*. Tato výjimka se používá pro předávání hlášení o chybách uživateli. Prakticky to znamená, že pokud vznikne nějaká chyba v aplikaci, například uživatel chce vytvořit steganogram a přitom nezadal vstupní obrázek, tak logika aplikace vyšle uživatelskou výjimku, kterou pak dané rozhraní zachytí a vypíše uživateli. V případě GUI pomocí vyskakovacího okna s oznámením a u CLI jednoduchým výpisem na chybový výstup a ukončením programu s návratovou hodnotou 1.

### 4.6.2 Kontrolery pro formáty

Ke každému výstupnímu formátu připadá jeden kontroler, který je zděděný od abstraktní třídy jádra kontroleru. Toto jádro nese pole použitelných steganografických metod, aktuálně zvolenou metodu, výchozí heslo a načtený krycí obrázek. Hlavními metodami jsou `saveTo`, `getImageData`, `setImage`, `getFormatName`, `isRandomEnabled`, `isRandomSupported`, `getMethods`, `setMethod`. Třída obsahuje i další metody a to `getEstimatedCapacity`, `getRealCapacity`, `enableRandom`, `setPassword`, `hide`, `uncover`, avšak tyto metody pouze zprostředkovávají práci se zvolenou steganografickou metodou. Hlavní metody budou popsány níže.

Metody `isRandomEnabled`, `isRandomSupported` slouží pro zjištění, zda zvolená metoda umožňuje náhodnou redistribuci dat a jestli je již aktuálně zapnuta. Metoda `getFormatName` vrací název formátu pro který daný kontroler slouží. Tato metoda je využita hlavně z důvodu vynechání mapování textového názvu pro uživatele a následného zpětného vyhledání o jaký kontroler se jedná. Další metody jsou `getMethods` a `setMethod`, první zmíněná slouží pro vrácení seznamu dostupných metod a druhá pro nastavení jedné z těchto metod. Poté je zde metoda `setImage` sloužící pro načtení nosiče a `saveTo` pro uložení souboru po skrytí dat pomocí metody `hide`. Jako poslední jsem si nechal metodu `getImageData`, tato metoda slouží pro získání dat, do kterých jsou následně vložena skrývaná data pomocí metody `hide` nebo odkrývána skrytá data pomocí `unhide`. Tato metoda je důležitá zejména z důvodu jednotného rozhraní mezi metodami pracujícími s různými daty, jako tomu je například v případě JPEG a PNG, kde u prvního ukládáme do DCT koeficientů a v druhém případě přímo do hodnot pixelů. Tato metoda tedy vytvoří matice cílových dat do kterých u daného formátu skrýváme a vrátí je jako pole 2D polí, kde jednotlivé 2D pole jsou separátně oddělené složky barevného modelu. V případě BMP a PNG tedy hodnoty pixelů jednotlivé barvy, v případě JPEG jsou to komponenty DCT pro složky YCbCr.

Nyní něco málo o konkrétních kontrolerech. V případě BMP a PNG formátů se v rámci kontroleru nekoná nic jiného než specifická implementace abstraktních metod rodičovského kontroleru. Tyto abstraktní metody jsou: `hide`, `uncover`, `saveTo`, `getFormatName` a `getImageData`. V případě kontroleru pro JPEG, který také specificky implementuje abstraktní metody nadřazeného kontroleru, se zde nachází několik metod navíc. Tyto metody jsou `setQuality`, `runDCTProcess`, `customFilter`. Metoda setQuality se stará o nastavení kvality do kompresoru JPEG formátu. Další metodou je `runDCTProcess`, která je jakýmsi pomocníkem pro spouštění metody `customFilter` v jednom ze dvou režimů, a to čtení nebo zápisu DCT koeficientů. Metoda `customFilter`, která je nutná k implementaci rozhraní `TJCustomFilter` z knihovny turboJPEG, umožňuje přistupovat k DCT koeficientům

po procesu ztrátové komprese JPEG. Tato metoda vrací výřezy DCT koeficientů k jejich úpravě, avšak já jsem tuto metodu využil s drobnou kličkou ke skládání a získání celé DCT matice a následně zapsání celé DCT matice po částech zpět.

### 4.6.3 Metody

Při implementaci steganografických metod nebyly využity žádné knihovny. Jednotlivé metody byly již popsány dříve v sekci zabývající se steganografickými metodami 3.5 a pokud byly modifikovány nebo rozšířeny je to uvedeno v rámci jejich popisu. Metody byly implementovány podle dostupných zdrojů. U zdrojů jsem se snažil nalézt informace přímo od autora metody, avšak v některých případech to nebylo možné. V těchto případech bylo čerpáno z popisů dané metody v jiných pracích, které čerpaly z původní práce popisující danou metodu viz uvedené zdroje.

### 4.6.4 Pomocné třídy

#### AES

Třída *AES* zapouzdřuje práci s šifrováním pomocí algoritmu AES. Obsahuje pouze 3 metody. Metodu `setPassword` sloužící k zadání hesla pro šifrování, dále poté metody `encrypt`, `decrypt` sloužící pro vytvoření šifry a pro její zpětné dešifrování.

#### BitGetter

Tato třída slouží pro načítání zadaného počtu bitů, aby tento postup nemusel zbytečně znepřehledňovat jednotlivé steganografické metody. Při inicializaci jsou do konstruktoru předána načítaná data a poté pomocí obsažených metod je možné s těmito daty pracovat. Třída poskytuje možnost data pouze přečíst metodou `checkNextNBits`, přečíst a označit jako přečtená metodou `popNextNBits` nebo získat aktuální nenačtenou délku dat pomocí `getUnreadLength`.

**BitSetter** Třída BitSetter slouží stejně jako třída BitGetter pro zpřehlednění práce jednotlivých steganografických metod. Na rozdíl od BitGetteru však slouží k načítání bitů a složení cílových dat. Při inicializaci je zadána požadovaná kapacita alokovaného prostoru pro data. Dostupné metody slouží pro zjištení ještě volné kapacity (`getUnwrittenLength`), zjištění zapsané kapacity (`getCountOfWrittenBits`), zápisu N bitů (`writeNBits`) a vrácení zapsaných dat (`getData`).

**ByteToBitConvertor** Jedná se o jednoduchou třídu, která obsahuje pouze dvě pomocné metody. Obě tyto metody jsou statické. Metoda `byteToBits` bere jako jediný parametr typu byte a tento byte převede na pole 8 bytů, kde každý nese hodnotu daného bitu z předaného bytu. Byte na indexu 0 odpovídá MSB a byte na indexu 7 LSB předaného bytu. Druhou metodou je poté `bitsToByte`, která provádí opačný proces, tedy převádí pole bytů reprezentující jednotlivé bity na jeden složený byte.

**ParsedFile** Tato třída slouží pro načtení a zapouzdření datového souboru pro skrytí. Načtení je možné dvěma způsoby. Prvním je načtení předáním souboru a druhou načtení z pole bytů. První způsob slouží pro načtení dat a jejich zapouzdření, tedy při skrývání. Druhý poté k načtení již zapouzdřených dat, pří odkrývání. Tato třída dále obsahuje 14

metod. Z těchto 14 jsou čtyři pro interní použití. První je metoda pro získání přípony z názvu souboru a druhá slouží pro spojování bytových polí do jednoho bytového pole, předposlední je metoda pro vyčištění načtených dat z proměnných a poslední získává hodnotu kontrolního součtu dat. Zbylých 10 metod je veřejných. Z těchto metod dvě slouží pro práci s kontrolním součtem. Přesněji jeho zapnutí nebo vypnutí a ověření stavu, tedy zda je nebo není zapnut. Další tři pro práci s šifrováním, jeho zapnutí a vypnutí, ověření stavu a nastavení hesla pro šifrování. Další dvě poté slouží pro práci s daty. Jedna pro uložení načtených dat a druhá pro serializaci načtených dat. Poslední tři metody jsou již zmíněné metody pro načtení dat z souboru nebo z pole bytů a metoda pro zjištění jestli jsou již data načtena do objektu.

Zapouzdřená data se skládají z 6-ti částí. První částí je délka přípony v bytech. Toto pole má velikost jednoho bytu, tedy nabývá hodnot v rozsahu <0,255>. Druhou částí je samotná přípona. Toto pole má velikost v závislosti na předchozí hodnotě. Dále se nachází pole označující, zda je nebo není použit kontrolní součet. Tato hodnota je uložena na jednom bytu a pokud je nastavena na 1, tak je kontrolní součet aktivní, pokud na 0, tak není. Za tímto polem v případě zapnutého CRC nalezneme samotnou hodnotu CRC uloženou na 4 bytech. Pokud CRC zapnuto není, je vynecháno pole s hodnotou CRC neboli má délku 0 bytů. Následuje velikost samotných dat. Pro tyto účely jsou vyhrazeny 4 byty, což dává možnost uložení souboru o velikosti 4 294 967 295 bitů. RGB obrázek o velikosti 8K, tedy 7680x4320 s využitím všech bitů by byl schopný teoreticky přenést 99 532 800 bitů dat. Což ukazuje na to, že 4 byty by měly být dostačující pro zaznamenání velikosti skrytého souboru. Poslední částí je samotná část s daty o velikosti specifikované předchozím polem.

## 4.7 Uživatelská část

### 4.7.1 Grafické uživatelské rozhraní

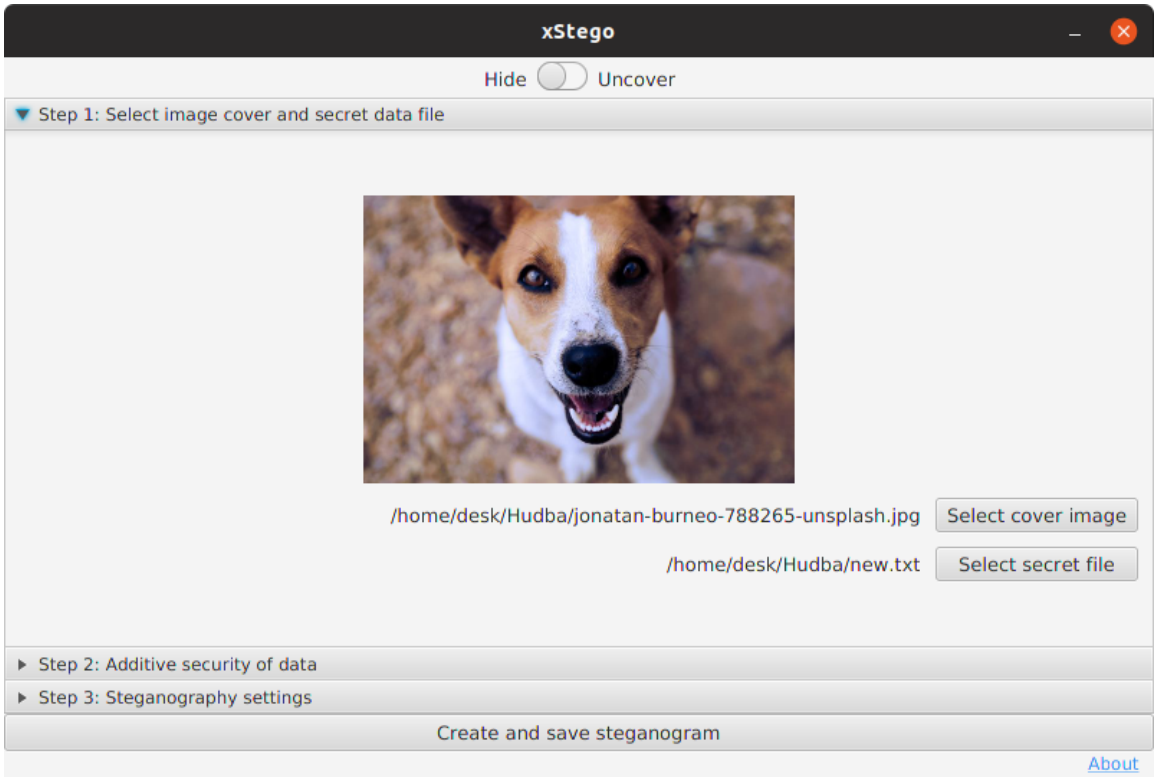
Jak již bylo zmíněno výše, GUI bylo tvořeno pomocí JavaFX s doplněním o komponenty ControlsFX. JavaFX byla využívána stylem, kdy používáme k návrhu rozhraní FXML. Snahou bylo dodržet oddělení logiky aplikace od samotného uživatelského rozhraní a jeho obsluhu. Aplikace se tedy skládá ze dvou hlavních kontrolerů a jednoho pomocného, který se stará pouze o přepínání mezi skrýváním a odkrýváním. Zmíněné dva hlavní kontrolery poté slouží právě pro rozhraní skrývání a odkrývání. K těmto kontrolerům vždy náleží jeden FXML soubor s definovaným uživatelským rozhraním. Funkci tzv. modelů zde představuje dříve zmíněné jádro aplikace, které nese zapouzdřenou všechnu logiku steganografické aplikace a jeho jednotlivé funkce jsou poté volány z obslužných metod kontroleru v reakci na interakci uživatele.

Při návrhu rozhraní bylo provedeno několik návrhů. Návrhy byly obměněny zejména kvůli zvětšení přehlednosti rozhraní a dodávání funkcí. Nejvhodnějším rozložením rozhraní se ukázalo využití přepínače mezi režimy "Hide", "Uncover" a harmonikového systému hlavního rozložení těchto režimů. Přepínač zajištuje úplné oddělení dvou režimů a harmonikový systém poté umožňuje zpřehlednit zadávání jednotlivých parametrů a oddělit je pomyslně do jednotlivých fází. Oddělení do jednotlivých kroků neurčuje však nutnost kroky procházet v daném pořadí, je pouze doporučující, slouží jako jednoduchý průvodce pro nového uživatele a má usnadnit používání aplikace pro uživatele a zvýšit přehlednost. Harmonika je rozdělena do 3 kroků. Prvním krokem je načtení potřebných souborů. Druhým krokem je nastavení dodatečného zabezpečení jako je povolení CRC, které však ubere na maximální kapacitě z důvodu přidání CRC do zapouzdření dat a možnost šifrování dat s volbou

vlastního hesla. Pokud heslo není zadáno, je použité heslo výchozí. Posledním krokem je nastavení samotné steganografie.

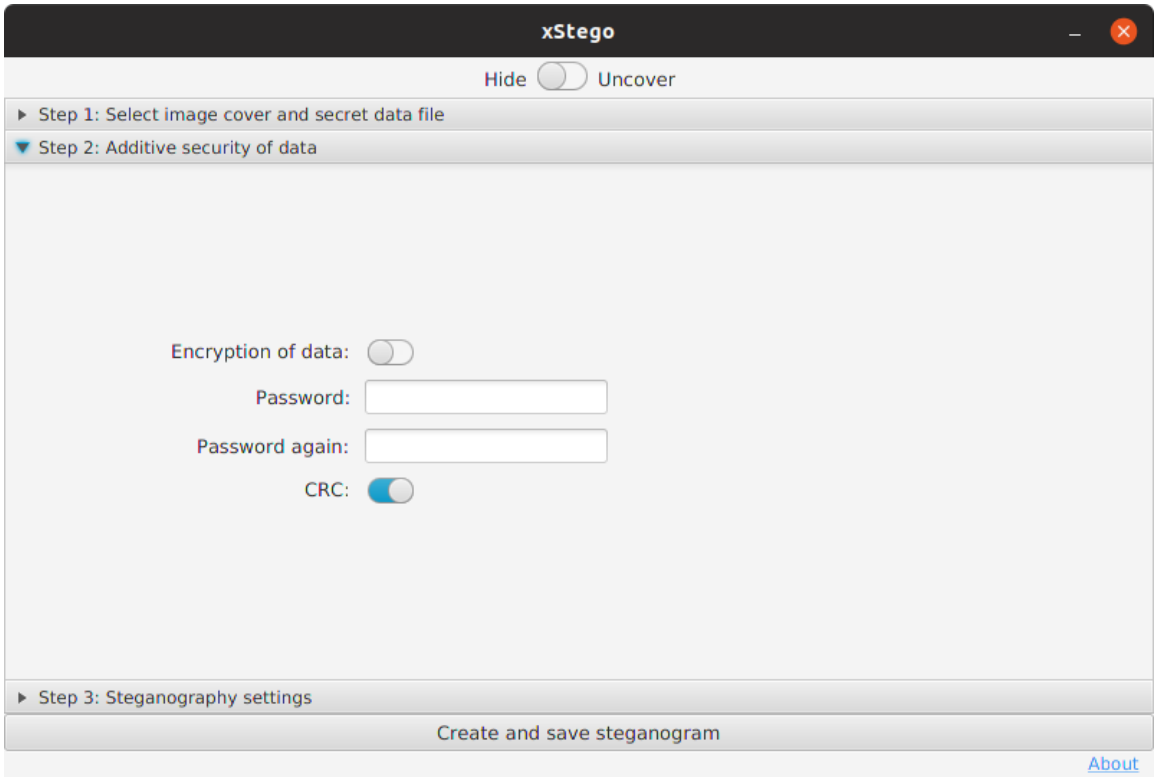
Rozhraní režimu "Hide" a "Uncover" je téměř totožné. Odlišnost najdeme pouze v absenci přepínače CRC a posuvníků kvality JPEG, jelikož jsou zjištěny přímo ze zapouzdření. Dále v části výběru skrývaných dat v režimu "Uncover" a s ním se pojícím polem velikosti souboru a poslední rozdílnou částí je absence ukazatele zaplnění. Tyto režimy by mohly být teoreticky spravovány jako jeden kontroler s jedním FXML souborem a naprogramovaným skrýváním daných komponent při přepnutí režimů. Avšak pro případné budoucí úpravy jsem se rozhodl implementovat tyto dva režimy duplicitně.

Nyní bude popsáno rozhraní režimu "Hide", jelikož je téměř totožné jako u druhého režimu pouze obsahuje prvky navíc. Zachycené rozhraní je vidět na obrázcích 4.1, 4.2, 4.3.



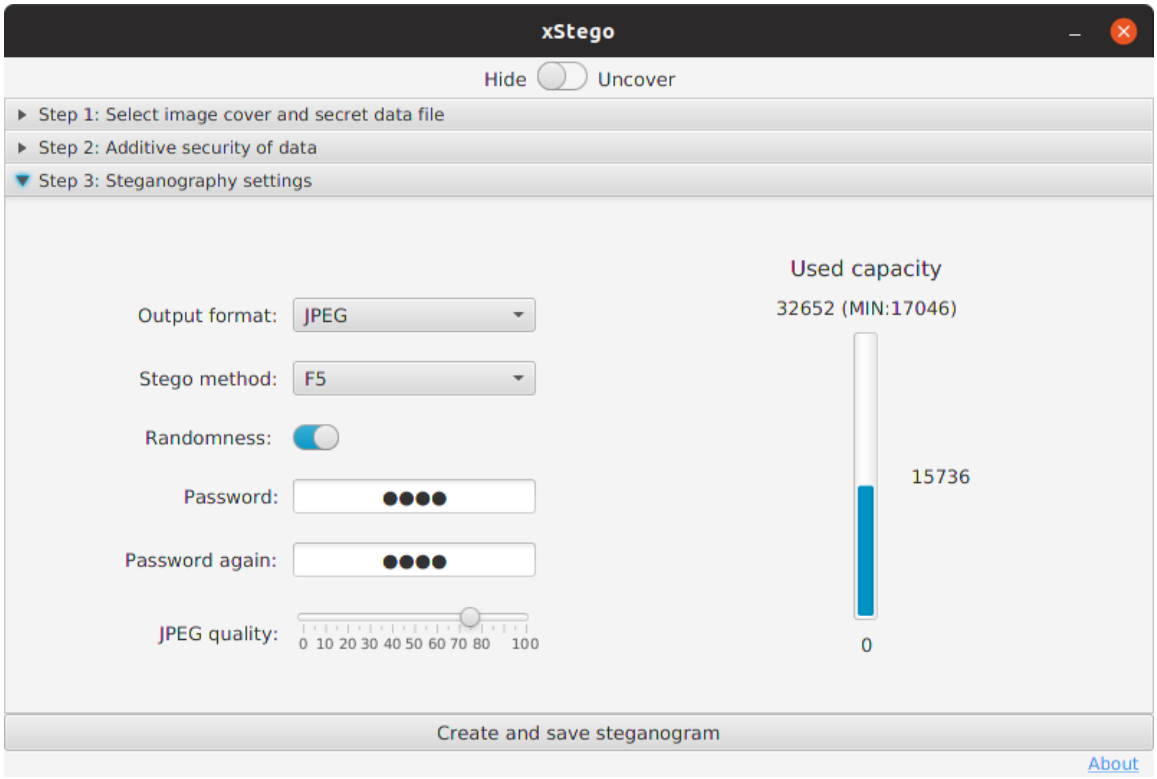

Obrázek 4.1: Krok 1

V prvním kroku viz obrázek 4.1 jsou vidět 3 části. První část je náhled zvoleného obrázku, který pokud není žádný obrázek vybrán zaplňuje výchozí obrázek. Druhá část je tvořena popiskem, kde je cesta k souboru a tlačítko "Select cover image". Tlačítko slouží pro otevření okna pro výběr souborů, kde uživatel zvolí obrázek, do kterého mají být skryta data. Pokud je obrázek vybrán, vyplní se popisek cestou k tomuto obrázku. Poslední částí je opět cesta k souboru, vedle něj poté tlačítko pro výběr souborů. Toto tlačítko funguje podobně jako tlačítko pro výběr obrázku, pouze s tím rozdílem, že při výběru souboru uživatel není omezen pouze na obrazové formáty souborů.




Obrázek 4.2: Krok 2

V druhém kroku na obrázku 4.2 je vidět možnost volby šifrování s dvěmi poli pro heslo. Šifrování je provedeno alogritmem AES. Pokud zadáváme vlastní heslo je nutné vyplnit obě pole "Password" a "Password again". Dvojice hesel je zde pro ověření překlepů. V případě nevyplnění polí pro heslo, je použito heslo výchozí zabudované v aplikaci. Posledním prvkem je přepínač zapnutí CRC, jedná se o 32 bitové CRC. Při jeho zapnutí je pro data spočítán kontrolní součet a ten je přidán do zapouzdření při jejich ukrytí. V důsledku zapnutí CRC můžeme pozorovat zvýšení využité kapacity. Toto zvýšení je však minimální, přesněji 4 byty.




Obrázek 4.3: Krok 3

V třetím kroku zachyceném na obrázku 4.3 přecházíme na nastavení steganografie, která bude následně po stisknutí tlačítka "Create and save steganogram" aplikována po zvolení adresáře pro výstupní soubor. Můžeme zde tedy vidět nastavení výstupního formátu, ke kterému se následně vztahuje výběr metody pro daný formát. Další částí je přepínač umožňující zapnutí náhodného rozložení dat po celém obrázku. Tato možnost je dostupná pouze u některých metod. Po zvolení tohoto přepínače do pozice zapnuto je možné zadat vlastní heslo, které bude použito pro generování náhodnosti. A opět je zde vyžadováno shodné vyplnění obou polí pro ověření překlepů. Poslední nastavitelnou částí je posouvátko s názvem "JPEG quality". Jak název napovídá, toto nastavení je dostupné pouze při výstupním formátu JPEG a nastavuje se s ním kvalita, tedy míra komprese, která může být v rozsahu <1,100>. 0 je na ose přítomná z důvodu vzhledu posuvníku a při zadání nuly je tato hodnota automaticky přenastavena na 1. Vpravo poté ještě můžeme vidět ukazatel zaplnění nosiče. Nahoře je maximální a minimální kapacita a vpravo velikost souboru s zapouzdřením. Tato část není přímo ovlivnitelná uživatelem a její hodnoty jsou vypočítány s nastavováním jednotlivých parametrů.

### 4.7.2 Rozhraní příkazového řádku

CLI aplikace při spuštění ověří, jestli je v zadaných argumentech spuštění přítomný přepínač -h nebo --help. Pokud je tento přepínač nalezen, je vypsána nápověda včetně podporovaných formátů, metod a nápověd dostupných režimů. CLI obsahuje 3 režimy, kde každý režim spravuje samostatný kontroler. V počáteční fázi je tedy z prvního parametru rozpoznán režim a následující parametry jsou poté předány obslužnému kontroleru ke zpracování. Daný kontroler si parametry zkontroluje a pokud jsou zadány všechny povinné parametry, tak je začne zpracovávat a nastavovat aplikaci. Po dokončení nastavení se spustí výpočet kapacity, skrývaní dat nebo odkrývání dat v závislosti na zvoleném režimu. V příloze B naleznete více informací o použití. Formát spuštění je následující:

```
java -jar xStegoC.jar <režim> <volby>
<režim> ::= hide | uncover | capacity
<volby> ::= (kombinace voleb dostupných pro daný režim)
```

# Kapitola 5

# Vyhodnocení implementace

Tato kapitola se zabývá testováním a vyhodnocením implementovaných metod. Tyto metody budou testovány z pohledu steganografických vlastností. V první části bude testována nepostřehnutelnost, poté bude porovnána kapacita pro různě velké obrázky. Následně se tato práce bude věnovat odolnosti z pohledu manipulace se steganogramem i přenosu steganogramu. Všechny testy jsou psány pro Linux.

## 5.1 Testovací sada obrázků

Testovací sady byly složeny ze stejných obrázků zmenšených na požadovanou velikost. Obrázky byly získány na stránce https://unsplash.com/. Tyto obrázky byly zvoleny tak, aby pokrývaly časté typy fotografií a zároveň byly dosti různorodé. Tematika těchto obrázků je proto zaměřena na fotografie automobilů, lidí, zvířat, přírody, a navíc byla přidána sada s šedotónovými obrázky různých scén. Každá tematická sada obsahuje 30 fotografií a celá testovací sada, pak tedy skýtá 150 testovacích obrázků ve formátu JPEG. Tato sada byla následně pomocí pomocného skriptu převedena na různé rozlišení. Spuštění skriptu je možné pomocí:

```
./createSpecificResolution.sh složkaObrazky šířka výška
```

kde `složkaObrazky` je složka se zdrojovými obrázky a `šířka`, `výška` udávají požadované rozlišení. Šířka by měla být při spuštění větší než výška, skript si opačně orientované obrázky sám zjistí a přehodí si zadané rozměry.

## 5.2 Test (ne)závislosti na vstupním formátu

V rámci tohoto testování byla otestována závislost na vstupním formátu ve spojení s kapacitou obrázku. V rámci tohoto testování jsem došel k závěru, že pokud pracujeme s totožnými vstupními obrazovými daty, a to ať už byl vstupní formát jakýkoliv, tak dostaneme stejnou kapacitu pro stejný výstupní formát a metodu. Problémem je dostat v různých formátech totožná vstupní data. V případě formátů BMP a PNG, je transformace mezi těmito formáty bezproblémová a kapacita zde zůstane stejná pro oba formáty pro všechny výstupní formáty i metody. Problém nastává při převodu z nebo do formátu JPEG, kde dochází ke změně obrazových dat. Tato rozdílnost byla odhalena pomocí porovnání rozdílů hodnot pixelů. První situace, kde se změna vyskytne, je převod z PNG a BMP na JPEG, což by se dalo očekávat v důsledku fungování procesu komprese JPEG, a to například kvantizace a podvzorkování.

Poněkud nečekanější je projev při převodu z JPEG do PNG a BMP. V tomto případě jsou obrazová data souborů PNG i BMP totožná, avšak neshodují se přesně s obrazovými daty JPEG, což v konečném důsledku vede k drobné odchylce kapacity aplikovaných metod v porovnání s formáty BMP a PNG.

### Testovací skript

Testovací skript provede převod zadaného vstupního obrázku do ostatních formátů s výjimkou formátu ve kterém je aktuálně. Pro aktuální formát se použije vstupní obrázek. Pro každý konvertovaný obrázek je pak provedeno měření kapacity a to pro všechny možné výstupní formáty v kombinaci s dostupnými metodami. V průběhu testu jsou tvořeny pomocné soubory potřebné pro testování, do složky se skriptem, které na konci skriptu budou smazány. Výsledkem tohoto spuštěného testu je poté CSV soubor, který je možné převést do tabulky, která obsahuje v prvním řádku záhlaví složené z názvu výstupního formátu a metody a první sloupec obsahuje vždy název vstupního formátu a další sloupce poté naměřené hodnoty. Jeho spuštění je následující:

```
./inputFormatIndependency.sh vstupníObrázek názevVystupníhoCSV
```

Kde `vstupníObrázek` je cesta k obrázku, který bude konvertován a otestován. Část `názevVystupníhoCSV` je cesta k souboru, kde bude zapsáno CSV s výsledky testu.

## 5.3 Kapacita

Tato část bude věnována měření kapacity implementovaných metod. Kapacita bude testována na třech různých rozlišeních obrázků. V tabulkách 5.1, 5.2, 5.3, je vždy uveden výstupní formát a metoda následována minimální, průměrnou a maximální naměřenou kapacitou. Jednotlivé kapacity jsou pak u každé metody rozděleny do dvou řádků, minimální a maximální kapacity. Tento postup byl zvolen, jelikož některé metody jsou závislé na skrývaných datech pro přesné určení kapacity. Kapacita v řádku "min" je tedy kapacita v nejhorším případě a v řádku "max" se jedná o maximální možnou kapacitu. U steganografie je vítána co největší kapacita.

Testování kapacity dopadlo dle očekávaných vlastností zkoumaných metod. To jest, že metody na obrázcích formátu JPEG mají značně menší kapacitu, než metody pro BMP a PNG. To je dáno ukládáním tajných dat až do DCT koeficientů po procesu ztrátové komprese z čehož vyplývá, že pro uložení máme významně méně komponent v důsledku podvzorkování a kvantizace, než v případě formátů BMP a PNG, kde pro skrytí využijeme všechny pixely a barvy. Dalším očekávaným chováním je, že s počtem využitých bitů u metod LSB a JSTEG roste i jejich kapacita. U LSB poté můžeme vidět shodnou minimální, průměrnou i maximální kapacitu. Tento jev způsobuje princip LSB, který je nezávislý na daném obrázku a využívá jej vždy stejně. Dalším viditelným zjištěním, z níže uvedených výsledků testů kapacity, je shodná kapacita metod F3 a F4, které využívají stejnou logiku pro rozhodnutí, zda je místo vhodné ke skrývání, ale liší se v logice skrývání ve smyslu jak skrývají, ale ne kolik toho skrývají. Metoda F5 přestože využívá stejnou logiku má menší kapacitu. To je dáno využitím mechanismu kódování dat do matice, jak již bylo popsáno v části věnující se metodám obrazové steganografie. Tento mechanismus totiž pro ukrytí 2 bitů zprávy využije 3 bity kapacity. Proto nám kapacita klesne o 1/3 kapacity metod F3 nebo F4, ale na oplátku snížíme množství změn v obrázku. Dále je vidět totožná kapacita

LSB pracujícího s jedním bitem a EMD pracujícího po dvou pixelech. K tomuto dojde při jakékoliv velikosti obrázku. I přestože EMD pracuje po 2 pixelech, tak při dosazení do rovnic pro výpočet kapacity těchto dvou pixelů dostaneme 2 bity a tím se dostaneme na stejnou kapacitu jako má jednobitové LSB. Při metodě Inverted LSB vidíme podobnou kapacitu s jednobitovým LSB, přesněji nižší o 4 bity. Tento rozdíl je způsoben implementací značení inverzí, jinak tato metoda kapacitně odpovídá metodě jednobitového LSB. U metod TripleA je vidět velký rozdíl mezi minimální a maximální kapacitou. Tento rozdíl je velký z důvodu náhodného výběru komponenty a počtu bitů. Je totiž možné se v nejhorším případě dostat na vkládání jednoho bitu a to pouze do jedné komponenty. V reálném použití bude tato kapacita ve většině případů vyšší než zmíněná minimální.

### Testovací skript

Pro testování kapacity byl vytvořen skript, který provede test kapacity na všech obrázcích v zadané složce pomocí všech metod pro zvolený výstupní formát. Výsledky tohoto testu jsou poté uloženy do CSV souboru. Přesněji dvou souborů, protože je vytvořen soubor pro minimální a maximální kapacitu. Jméno výstupních CSV souboru je potom předsazeno o min_ a max_. Jeho spuštění jé následujicí:

```
./capacityToCSV.sh složkaObrázky názevVystupníhoCSV výstupníFormát
```

Kde `složkaObrázky` je cesta ke složce obsahující sadu obrázků pro zpracování. Část `názevVystupníhoCSV` je cesta k souboru, kde bude zapsáno CSV s výsledky testu. Poslední část `výstupníFormát`, je jeden z názvů podporovaných formátů, tedy BMP, PNG nebo JPEG.

| Formát | Metoda | | Minimální | Průměrná | Maximální |
|---|---|---|---|---|---|
| BMP a PNG | LSB 1b | min | 1036800 | 1036800 | 1036800 |
| | | max | 1036800 | 1036800 | 1036800 |
| | LSB 2b | min | 2073600 | 2073600 | 2073600 |
| | | max | 2073600 | 2073600 | 2073600 |
| | LSB 3b | min | 3110400 | 3110400 | 3110400 |
| | | max | 3110400 | 3110400 | 3110400 |
| | LSB 4b | min | 4147200 | 4147200 | 4147200 |
| | | max | 4147200 | 4147200 | 4147200 |
| | Inverted LSB | min | 1036796 | 1036796 | 1036796 |
| | | max | 1036796 | 1036796 | 1036796 |
| | TripleA 2b | min | 345600 | 345600 | 345600 |
| | | max | 2073600 | 2073600 | 2073600 |
| | TripleA 3b | min | 345600 | 345600 | 345600 |
| | | max | 3110400 | 3110400 | 3110400 |
| | TripleA 4b | min | 345600 | 345600 | 345600 |
| | | max | 4147200 | 4147200 | 4147200 |
| | EMD 2 | min | 1036800 | 1036800 | 1036800 |
| | | max | 1036800 | 1036800 | 1036800 |
| | EMD 4 | min | 777600 | 777600 | 777600 |
| | | max | 777600 | 777600 | 777600 |
| | EMD 8 | min | 518400 | 518400 | 518400 |
| | | max | 518400 | 518400 | 518400 |
| | PVD | min | 1018909 | 1574632 | 1779822 |
| | | max | 1018909 | 1574632 | 1779822 |
| JPEG | JSTEG 1b | min | 18484 | 58909 | 125406 |
| | | max | 18484 | 58909 | 125406 |
| | JSTEG 2b | min | 32320 | 98635 | 207648 |
| | | max | 32320 | 98635 | 207648 |
| | JSTEG 3b | min | 45831 | 134373 | 277788 |
| | | max | 45831 | 134373 | 277788 |
| | JSTEG 4b | min | 58832 | 169021 | 348008 |
| | | max | 58832 | 169021 | 348008 |
| | F3 | min | 13030 | 41141 | 87017 |
| | | max | 23665 | 76850 | 164009 |
| | F4 | min | 13030 | 41141 | 87017 |
| | | max | 23665 | 76850 | 164009 |
| | F5 | min | 8686 | 27427 | 58010 |
| | | max | 15776 | 51232 | 109338 |

Tabulka 5.1: Tabulka kapacit pro obrázky s rozlišením 720x480 s hodnotami uvedenými v bitech.

| Formát | Metoda | | Minimální | Průměrná | Maximální |
|---|---|---|---|---|---|
| BMP a PNG | LSB 1b | min | 2764800 | 2764800 | 2764800 |
| | | max | 2764800 | 2764800 | 2764800 |
| | LSB 2b | min | 5529600 | 5529600 | 5529600 |
| | | max | 5529600 | 5529600 | 5529600 |
| | LSB 3b | min | 8294400 | 8294400 | 8294400 |
| | | max | 8294400 | 8294400 | 8294400 |
| | LSB 4b | min | 11059200 | 11059200 | 11059200 |
| | | max | 11059200 | 11059200 | 11059200 |
| | Inverted LSB | min | 2764796 | 2764796 | 2764796 |
| | | max | 2764796 | 2764796 | 2764796 |
| | TripleA 2b | min | 921600 | 921600 | 921600 |
| | | max | 5529600 | 5529600 | 5529600 |
| | TripleA 3b | min | 921600 | 921600 | 921600 |
| | | max | 8294400 | 8294400 | 8294400 |
| | TripleA 4b | min | 921600 | 921600 | 921600 |
| | | max | 11059200 | 11059200 | 11059200 |
| | EMD 2 | min | 2764800 | 2764800 | 2764800 |
| | | max | 2764800 | 2764800 | 2764800 |
| | EMD 4 | min | 2073600 | 2073600 | 2073600 |
| | | max | 2073600 | 2073600 | 2073600 |
| | EMD 8 | min | 1382400 | 1382400 | 1382400 |
| | | max | 1382400 | 1382400 | 1382400 |
| | PVD | min | 2643235 | 4158229 | 4736099 |
| | | max | 2643235 | 4158229 | 4736099 |
| JPEG | JSTEG 1b | min | 46056 | 140780 | 327946 |
| | | max | 46056 | 140780 | 327946 |
| | JSTEG 2b | min | 80750 | 236486 | 539036 |
| | | max | 80750 | 236486 | 539036 |
| | JSTEG 3b | min | 114753 | 323854 | 728640 |
| | | max | 114753 | 323854 | 728640 |
| | JSTEG 4b | min | 148600 | 409009 | 927744 |
| | | max | 148600 | 409009 | 927744 |
| | F3 | min | 30077 | 96570 | 225784 |
| | | max | 61064 | 185552 | 437088 |
| | F4 | min | 30077 | 96570 | 225784 |
| | | max | 61064 | 185552 | 437088 |
| | F5 | min | 20050 | 64379 | 150522 |
| | | max | 400708 | 123701 | 291392 |

Tabulka 5.2: Tabulka kapacit pro obrázky s rozlišením 1280x720 s hodnotami uvedenými v bitech.

| Formát | Metoda | | Minimální | Průměrná | Maximální |
|---|---|---|---|---|---|
| BMP a PNG | LSB 1b | min | 6220800 | 6220800 | 6220800 |
| | | max | 6220800 | 6220800 | 6220800 |
| | LSB 2b | min | 12441600 | 12441600 | 12441600 |
| | | max | 12441600 | 12441600 | 12441600 |
| | LSB 3b | min | 18662400 | 18662400 | 18662400 |
| | | max | 18662400 | 18662400 | 18662400 |
| | LSB 4b | min | 24883200 | 24883200 | 24883200 |
| | | max | 24883200 | 24883200 | 24883200 |
| | Inverted LSB | min | 6220796 | 6220796 | 6220796 |
| | | max | 6220796 | 6220796 | 6220796 |
| | TripleA 2b | min | 2073600 | 2073600 | 2073600 |
| | | max | 12441600 | 12441600 | 12441600 |
| | TripleA 3b | min | 2073600 | 2073600 | 2073600 |
| | | max | 18662400 | 18662400 | 18662400 |
| | TripleA 4b | min | 2073600 | 2073600 | 2073600 |
| | | max | 24883200 | 24883200 | 24883200 |
| | EMD 2 | min | 6220800 | 6220800 | 6220800 |
| | | max | 6220800 | 6220800 | 6220800 |
| | EMD 4 | min | 4665600 | 4665600 | 4665600 |
| | | max | 4665600 | 4665600 | 4665600 |
| | EMD 8 | min | 3110400 | 3110400 | 3110400 |
| | | max | 3110400 | 3110400 | 3110400 |
| | PVD | min | 5953944 | 9289090 | 10646091 |
| | | max | 5953944 | 9289090 | 10646091 |
| JPEG | JSTEG 1b | min | 97154 | 293544 | 740476 |
| | | max | 97154 | 293544 | 740476 |
| | JSTEG 2b | min | 174346 | 494720 | 1219284 |
| | | max | 174346 | 494720 | 1219284 |
| | JSTEG 3b | min | 247533 | 681401 | 1639896 |
| | | max | 247533 | 681401 | 1639896 |
| | JSTEG 4b | min | 314468 | 863919 | 2075712 |
| | | max | 314468 | 863919 | 2075712 |
| | F3 | min | 63345 | 197182 | 508215 |
| | | max | 117641 | 391228 | 973060 |
| | F4 | min | 63345 | 197182 | 508215 |
| | | max | 117641 | 391228 | 973060 |
| | F5 | min | 42230 | 131454 | 338810 |
| | | max | 78426 | 260818 | 648706 |

Tabulka 5.3: Tabulka kapacit pro obrázky s rozlišením 1920x1080 s hodnotami uvedenými v bitech.

## 5.4 Nepostřehnutelnost

Nepostřehnutelnost průběhu staganografické komunikace pomocí obrazového nosiče je založena na nepostřehnutelnosti změn v tomto nosiči a na velikosti výsledného souboru. Jinými slovy by obrazový nosič po vložení tajných dat měl vypadat co nejvíce totožně jako před

vložením a nemělo by dojít k pozorovatelnému zhoršení jeho kvality. Proto se v této oblasti jako hodnocení používá srovnání kvality obrázku. Degradace kvality obrázku je popisována ve smyslu viditelného narušení obrázku jako jsou například rozkostičkování obrazu, rozmazanost obrazu nebo změna barev viz [28]. Metody pro hodnocení kvality obrazového média je možné dělit na dva druhy:

- subjektivní metody (sekce 5.4.1)
- objektivní metody (sekce 5.4.2)

V rámci testování bude využito pouze objektivních metod testování, ale pro úplnost budou popsány níže oba způsoby. Důvodem pro vyřazení subjektivních metod byly dva faktory. Prvním faktorem je, že subjektivní metody jsou časově náročnější a vyžadují velké množství hodnotitelů pro získání relevantních výsledků. Tento fakt by zesložiťoval opětovné provedení testů při změně implementace nebo přidání metody. Druhým faktorem bylo zamyšlení se nad vhodností subjektivních metod testování pro porovnání steganogramů s původním obrázkem. Jelikož původně bylo zamýšleno zamíchat steganogramy do sady normálních obrázků a vytvořit webovou aplikaci s dvěmi režimy, jedním pro porovnání dvou obrázků proti sobě na viditelnost rozdílů a druhým pro hodnocení osamoceného obrázku na podezřelost. Nakonec jsem došel k názoru, že bude lepší aplikovat pouze subjektivní metody. Časová náročnost by nebyla ani tak závažná, avšak obavy vznikly z výsledků objektivního testování. Uživatele by bylo nutné seznámit s důvodem testování a samotnou steganografií, což by jej mohlo ovlivnit při hodnocení. Při neseznámení by zase naopak uživatel nevěděl, jak moc drobnou změnu hledá a pravděpodobně by docházelo k průchodu testovacími obrázky s negativními výsledky na pozorovatelnost změn.

Dalším prvkem, který bude z pohledu nepozorovatelnosti sledován je velikost souborů před a po vložení dat. Pokud by byl odklon od původní velikosti velký, pomohlo by to odhalení steganografické komunikace.

### 5.4.1 Subjektivní metody

Tyto metody jsou založeny na přítomnosti člověka v procesu vyhodnocování. Člověk zde hraje roli hodnotitele kvality daného obrázku. Subjektivní hodnocení je nejspolehlivější a nejpřesnější způsob vyhodnocování kvality obrázku [28]. Avšak tyto metody jsou časově náročné, drahé a nevyhovující pro praktické využití. Subjektivní metody můžeme dále rozlišovat na jednostimulové nebo dvoustimulové. O dvojstimulových se bavíme v případě, je-li dostupný původní obrázek pro porovnání [28].

### 5.4.2 Objektivní metody

Tyto metody jsou vytvořeny pro automatické vyhodnocování kvality obrázku [28]. Jsou méně časově náročné a úkolem těchto metod je co nejvíce se přiblížit subjetivnímu hodnocení. Objektivní metody můžeme dále rozčlenit na ty s plnou referencí (full-reference), redukovanou referencí (reduced-reference) a bez reference (no-reference). Toto rozdělení se opět odvíjí od toho, do jaké míry máme dostupný původní obrázek. [28]. Objektivní metody s plnou referencí spočívají v porovnání dvou obrázků, kde jeden z nich je považován za původní, v plné kvalitě a bez narušení. Což v případě této práce bude původní obrázek, do kterého byly data ukryta. Druhým obrázkem je hodnocený obrázek, u kterého chceme zjistit pokles kvality, v našem případě narušení obrázku v důsledku ukrytí tajných dat. To

znamená, že roli druhého obrázku bude mít vytvořený steganogram. Výsledky testování pomocí objektivních metod PSNR a SSIM jsou uvedeny v tabulce 5.4.

Při hledání objektivních metod jsem zjistil, že jsou metody MSE a PSNR implementovány v programu compare spadající pod ImageMagick. Po dalším pátrání jsem našel i metodu SSIM, která je dostupná ve verzi 7.0.8-19. Z tohoto důvodu nebudou metody pro testování implementovány, ale bude využito zmíněného programu.

### MSE - Mean Squared Error

Široce přijímaným předpokladem při zkoumání poklesu kvality obrázku je, že viditelný pokles kvality přímo souvisí s viditelností chybového signálu [34]. Nejjednodušší aplikací tohoto přístupu je metoda MSE, která objektivně vypočítává sílu chybového signálu v obraze [34]. Avšak dva obrázky se stejným MSE mohou vykazovat různě viditelné chyby [34]. MSE je založené na porovnání změn v bodě a je tedy nezávislé na struktuře obrazu [34]. Níže uvedený vzorec 5.1 pro výpočet MSE je určen pro jednu složku obrázku. V případě RGB obrázku je MSE vypočteno pro každou barevnou komponentu a celkovým výsledkem je průměr výsledků MSE pro jednotlivé složky.

$$MSE = \frac{1}{WH}\sum_{x=1}^{W}\sum_{y=1}^{H}(C(x,y) - S(x,y))^2 \tag{5.1}$$

$W$...šířka obrázku

$H$...výška obrázku

$C$...nosič

$S$...steganogram

Obrázek 5.1: Vzorec pro výpočet MSE čerpaný z [27].

### PSNR - Peak Signal to Noise Ratio

PSNR je nejpopulárnější a zároveň nejvyužívanější metodou z objektivních metod, ale moc dobře neodpovídá s výsledky subjektivních metod [28]. Člověk není schopen rozeznat žádný rozdíl v obrázcích s PSNR větším než 36 dB [27]. PSNR je vypočítáno na základě MSE, proto při testování nepostřehnutelnosti budou uváděny pouze výsledky PSNR. Navíc výsledky PSNR jsou v tabulce přehlednější, jelikož jsou kratší než výsledky MSE.

$$PSNR = 10\log\left(\frac{Max^2}{MSE}\right) \tag{5.2}$$

$Max$...maximální hodnota v obrázku

Obrázek 5.2: Vzorec pro výpočet PSNR čerpaný z [27].

### SSIM - Structural Similarity index

SSIM (Structural SIMilarity) metoda měření kvality obrázku představená v [34], bude jednou z metod testování aplikovanou na steganogram a původní obrázek. Přirozený obraz je strukturovaný, jeho pixely vykazují závislost, obzvláště pixely, které jsou si blízké. Tohoto faktu využívá právě metoda SSIM, která na rozdíl od předchozích metod není založená na hodnotách jednotlivých bodů obrazu. Metoda SSIM, měří zhoršení kvality obrázku pomocí tří faktorů, a to ztráty struktury, narušení kontrastu, narušení jasu. Tato metoda je na rozdíl od předchozích dvou metod více uzpůsobená pro vnímání kvality, tak jak jí vnímá člověk.

### Testovací skript

Pro testování byl vytvořen skript, který ukryje vygenerovaná náhodná data do obrázku, vypočítá PSNR a SSIM a výsledky uloží do CSV. Náhodná data jsou generována podle zjištěné maximální kapacity každého ze zpracovávaných obrázků. Maximální kapacita je vhodná, jelikož je při plném zaplnění pozměněno nejvíce původních dat a tím by se mělo jednat o největší viditelnost změn, což se pro demonstraci schopností implementovaných metod hodí. V tomto testu jsem se snažil přibližně maximálně využít krycí médium. Jelikož některé metody svou opravdovou kapacitu zjistí až podle zadaných dat bylo třeba tuto skutečnost řešit ve skriptu. Konečným řešením je načtení maximální kapacity, pokud se nepovede soubor s takovou velikostí ukrýt je vygenerován nový soubor který je o 1/10 velikosti menší a takto se postupuje do úspěšného ukrytí. Skript vytvořený pro tento test se spouští následovně:

```
./imperceptibilityTest.sh. složkaObrázky názevVystupníhoCSV
```

Část `složkaObrázky` je cesta ke složce obsahující sadu obrázků pro zpracování. Poslední část `názevVystupníhoCSV` je cesta k souboru, kde bude zapsáno CSV s výsledky testu.

| Formát | Metoda | PSNR | SSIM |
|---|---|---|---|
| BMP | LSB 1 BIT | 51,1413 | 0,9954 |
| | LSB 2 BIT | 44,1426 | 0,9778 |
| | LSB 3 BIT | 37,8838 | 0,9208 |
| | LSB 4 BIT | 31,8122 | 0,7773 |
| | Inverted LSB | 51,1411 | 0,9954 |
| | TripleA 2b | 49,2312 | 0,9928 |
| | TripleA 3b | 44,3830 | 0,9786 |
| | TripleA 4b | 39,1698 | 0,9354 |
| | EMD 2 | 52,5362 | 0,9965 |
| | EMD 4 | 55,0705 | 0,9981 |
| | EMD 8 | 57,8478 | 0,9990 |
| | PVD | 41,2138 | 0,9714 |
| PNG | LSB 1 BIT | 51,1413 | 0,9954 |
| | LSB 2 BIT | 44,1425 | 0,9778 |
| | LSB 3 BIT | 37,8840 | 0,9208 |
| | LSB 4 BIT | 31,8125 | 0,7774 |
| | Inverted LSB | 51,1410 | 0,9955 |
| | TripleA 2b | 49,2309 | 0,9928 |
| | TripleA 3b | 44,3827 | 0,9785 |
| | TripleA 4b | 39,1694 | 0,9354 |
| | EMD 2 | 52,5364 | 0,9965 |
| | EMD 4 | 55,0702 | 0,9981 |
| | EMD 8 | 57,8481 | 0,9990 |
| | PVD | 41,2150 | 0,9714 |
| JPEG | JSTEG 1 BIT | 37,8887 | 0,9740 |
| | JSTEG 2 BIT | 30,9324 | 0,9034 |
| | JSTEG 3 BIT | 24,6592 | 0,7474 |
| | JSTEG 4 BIT | 19,2428 | 0,5380 |
| | F3 | 34,5663 | 0,9371 |
| | F4 | 35,7074 | 0,9528 |
| | F5 | 37,6668 | 0,9686 |

Tabulka 5.4: Tabulka výsledků nepostřehnutelnosti zaokrouhleno na 4 desetinná místa. Vetší hodnota je lepší.

### 5.4.3 Velikost souborů

V rámci testování byla porovnána velikost souborů s obrázkem před vložením a po vložení. Jako výsledek je brána absolutní hodnota rozdílu těchto dvou velikostí a dále je v tabulce výsledků 5.5 uvedena velikost největšího vytvořeného souboru.

Výsledky testování na obrázcích velikosti 720x480, jako byly použity v předchozích testech a tajná data jsou opět generována pro dosažení co největšího zaplnění. Výsledky jsou v tabulce 5.5. Jak je vidět rozdíl velikostí souborů před a po není nikterak drastický oproti maximální dosažené velikosti souboru. U formátu BMP můžeme vidět konstantní rozdíl velikostí, jelikož formát BMP je bezkompresní a rozdíl je způsoben rozdílným způsobem uložení. U ostatních formátů pak vidíme rozdílný rozdíl velikostí. Toto chování může jednak způsobit rozdílná implementace ukládání, ale spíše také změna bitů v obraze pomocí

tajné zprávy v důsledku skrývání. Samotná změna bitů nezmění velikost souboru, ke změně velikosti dochází až důsledkem komprese před zapsáním souboru, kde nemusí být možné provést kompresi stejně jako před uložením tajných dat.

### Testovací skript

```
./fileSizeTest.sh. složkaObrázky názevVystupníhoCSV
```

Pro testování je spuštěn skript, kterému je předána složka s obrázky (`složkaObrázky`) a cesta k výstupnímu CSV souboru (`názevVystupníhoCSV`), který bude vytvořen. V tomto skriptu je pomocí všech metod v kombinaci se všemi formáty skrytá tajná zpráva do obrázku a ten je dočasně umístěn do aktuální složky se skriptem. Původní a výsledný soubor je následně porovnán na rozdíl velikostí a výsledek společně s velikostí výstupního souboru je zapsán do CSV. Jelikož implementovaný program převádí všechny obrázky na zvolený výstupní formát a zároveň na barevný obrázek, tak bylo tyto kroky nutné vložit jako předzpracování vstupního souboru do skriptu z důvodu následného porovnání velikostí. Ve skriptu se nachází ještě drobná pomoc pro získání vhodného rozdílu velikostí. Tato pomoc se týká formátu PNG, jelikož v implementovaném programu, je při uložení dosaženo značně odlišné velikosti souboru i přes shodnost nesené obrazové informace v porovnání s výstupem programu `convert` využitého pro předzpracování obrázku. Ve skriptu je tento fakt řešen dodatečným převedením výstupního formátu programem convert pro dosažení stejného uložení obrazových dat. Tento pomocný krok je proveden čistě pro umožnění rozumného porovnání rozdílu velikostí souborů před a po. Vytvořený souboru implementovanou aplikací nijak svou velikostí nevyčnívá a v porovnání s některými jinými dostupnými konvertory formátů se velikost výstupního souboru řadí mezi průměr.

| Formát | Metoda | průměrný rozdíl | Maximální velikost |
|---|---|---|---|
| BMP | LSB 1 BIT | 84 | 1036854 |
| | LSB 2 BIT | 84 | 1036854 |
| | LSB 3 BIT | 84 | 1036854 |
| | LSB 4 BIT | 84 | 1036854 |
| | Inverted LSB | 84 | 1036854 |
| | TripleA 2b | 84 | 1036854 |
| | TripleA 3b | 84 | 1036854 |
| | TripleA 4b | 84 | 1036854 |
| | EMD 2 | 84 | 1036854 |
| | EMD 4 | 84 | 1036854 |
| | EMD 8 | 84 | 1036854 |
| | PVD | 84 | 1036854 |
| PNG | LSB 1 BIT | 96494 | 805515 |
| | LSB 2 BIT | 148788 | 808686 |
| | LSB 3 BIT | 197498 | 815566 |
| | LSB 4 BIT | 264818 | 845181 |
| | Inverted LSB | 96500 | 805575 |
| | TripleA 2b | 119347 | 806393 |
| | TripleA 3b | 150643 | 808538 |
| | TripleA 4b | 187950 | 812187 |
| | EMD 2 | 100297 | 804965 |
| | EMD 4 | 80526 | 803529 |
| | EMD 8 | 53600 | 801852 |
| | PVD | 150290 | 815947 |
| JPEG | JSTEG 1 BIT | 566 | 116267 |
| | JSTEG 2 BIT | 3463 | 125501 |
| | JSTEG 3 BIT | 8250 | 132727 |
| | JSTEG 4 BIT | 14623 | 146485 |
| | F3 | 11519 | 92941 |
| | F4 | 8437 | 98185 |
| | F5 | 5633 | 102925 |

Tabulka 5.5: Tabulka výsledků testů velikosti souborů v bytech.

## 5.5 Robustnost

V této části testování je zkoumána robustnost implementovaných metod v oblasti manipulace s médiem. Následující testy jsou věnovány odolnosti vůči otočení, změně velikosti a přeposlání skrze některé oblíbené komunikátory.

### 5.5.1 Otočení

Otočení je jedna ze základních úprav obrázků, která se dá provádět i v běžném prohlížeči obrázků. Z tohoto důvodu byla vybrána pro testování odolnosti steganogramu.

Testy byly prováděny na sadě obrázků o rozměru 720x480 z testu kapacity. Tyto obrázky byly nejprve předány na předzpracování do skriptu, který do těchto obrázků ukryl tajná data a rozdělil je do složek s názvem metody a výstupního formátu. Skrývaná data představoval textový soubor obsahující hru Romeo a Julie, zkrácenou na velikost nejmenší získané kapacity v předchozím testu kapacity této testovací sady. Následně byly obrázky předány do skriptu pro samotné testování odolnosti vůči rotaci. Toto testování bylo složeno ze 3 testů. Prvním bylo pouhé otočení o 90 stupňů, bez zpětného otočení. V tomto případě mělo dojít k nečitelnosti skryté zprávy vzhledem ke změně pořadí pixelů nesoucích tajná data. Dalším testem je otočení o 90 stupňů s návratem do původní pozice pomocí otočení o 90 stupňů zpět. Posledním testem bylo postupné otočení o 360 stupňů po 90 stupních.

Výsledky testu jsou k vidění v tabulce 5.6. Jelikož celá sada obrázků vykazovala stejné výsledky, jsou tyto výsledky vedeny pouze pod souhrnným údajem uvedeným jako text ANO nebo NE v tabulce. Text ANO značí, že po provedení transformací se povedlo odkrytí tajných dat v podobě totožného souboru, a to ve všech případech. Porovnání, zda je odhalený soubor totožný s ukrývaným bylo provedeno pomocí příkazu `diff`. Text NE poté značí, že se nepovedlo daná data získat, a to opět ve všech případech. Selhání získání dat nebylo ani v jednom případě způsobeno chybou programu nebo nedostatečnou kapacitou pro skrytí, ale vždy se jednalo o poškození skryté informace. Z tabulky 5.6 lze usoudit, že selhání testu otočení pouze o 90 stupňů je pochopitelné, jelikož jsou po tomto otočení pixely na jiných pozicích než jsou čteny. Jak ale dokazuje test s otočením zpět, tak v případě BMP a PNG jsou data skrytá do obrázku u všech implementovaných metod pro tyto formáty nepoškozena. Nepoškozena zůstávají i při několikanásobném pootočení. Důležité je, aby obrázek byl při odkrytí ve stejné pozici jako byl při skrývání. Lze tedy říct, že metody implementované pro PNG a BMP jsou odolné vůči otočení. To se však nedá říct o metodách na obrázky ve formátu JPEG. Zde z důvodu opětovného aplikování JPEG komprese dochází i k přepočítání DCT koeficientů, což vede ke zničení skrytých dat.

### Testovací skript

Pro testování byly použity dva skripty. První skript slouží pro vytvoření steganogramů obsahující zadaný soubor. Tyto steganogramy jsou ukládány do zadané cílové složky do podsložek s názvem výstupního formátu a metody, která je aplikována. Druhý skript poté provede již zmíněné 3 testy. Každý test spočívá v provedení požadovaných transformací, které následuje pokus o odkrytí dat. Při neúspěšném odkrytí je započítán neúspěch. V případě úspěšného odkrytí je odkrytý soubor testován na shodnost s původním souborem. Pokud jsou shodné, je výsledek testu započítán jako úspěšný anebo jako neúspěšný v opačném případě. Tyto počty úspěchů a neúspěchů jsou následně zapsány do zadaného výstupního CSV souboru.

První skript se spouští následně:

```
./prepareSteganogramsToFolders.sh složkaObrázky cílováSložka tajnýSoubor
```

Skript přijímá jako argumenty 2 složky a jeden soubor. Argument `složkaObrázky` obsahuje obrázky do kterých budou skryty data. Argument `cílováSložka` je složka, kde budou uloženy vytvořené steganogramy do podsložek s názvem tvořeným výstupním formátem a použitou metodou. Poslední argument je `tajnýSoubor`, tento soubor bude skryt do všech obrázků ve složce složkaObrázky.

Spuštění druhého skriptu je následující:

```
./rotateTest.sh složkaObrázky tajnýSoubor výstupníCSV
```

Skript na vstupu dostává `složkaObrázky`, což je cílová složka přechozího skriptu. Dále `tajnýSoubor`, což je totožný soubor jako v prvním skriptu, který je využit pro testování, zda jsou získaná tajná data totožná s ukrývanými. A jako poslední přijímá cestu (`výstupníCSV`), kde budou zapsány výsledky testu ve formátu CSV.

| **Formát** | **Metoda** | **90 stupňů** | **90 stupňů a zpět** | **360 stupňů po 90 stupních** |
|---|---|---|---|---|
| BMP | LSB 1 BIT | NE | ANO | ANO |
| | LSB 2 BIT | NE | ANO | ANO |
| | LSB 3 BIT | NE | ANO | ANO |
| | LSB 4 BIT | NE | ANO | ANO |
| | Inverted LSB | NE | ANO | ANO |
| | TripleA 2b | NE | ANO | ANO |
| | TripleA 3b | NE | ANO | ANO |
| | TripleA 4b | NE | ANO | ANO |
| | EMD 2 | NE | ANO | ANO |
| | EMD 4 | NE | ANO | ANO |
| | EMD 8 | NE | ANO | ANO |
| | PVD | NE | ANO | ANO |
| PNG | LSB 1 BIT | NE | ANO | ANO |
| | LSB 2 BIT | NE | ANO | ANO |
| | LSB 3 BIT | NE | ANO | ANO |
| | LSB 4 BIT | NE | ANO | ANO |
| | Inverted LSB | NE | ANO | ANO |
| | TripleA 2b | NE | ANO | ANO |
| | TripleA 3b | NE | ANO | ANO |
| | TripleA 4b | NE | ANO | ANO |
| | EMD 2 | NE | ANO | ANO |
| | EMD 4 | NE | ANO | ANO |
| | EMD 8 | NE | ANO | ANO |
| | PVD | NE | ANO | ANO |
| JPEG | JSTEG 1 BIT | NE | NE | NE |
| | JSTEG 2 BIT | NE | NE | NE |
| | JSTEG 3 BIT | NE | NE | NE |
| | JSTEG 4 BIT | NE | NE | NE |
| | F3 | NE | NE | NE |
| | F4 | NE | NE | NE |
| | F5 | NE | NE | NE |

Tabulka 5.6: Tabulka výsledků testů robustnosti proti otočení, kde NE odpovídá nezvládnutí testu a ANO zvládnutí u celé sady.

### 5.5.2 Změna velikosti

Dalším testovaným kritériem je odolnost vůči změně velikosti. Změna velikosti je také jednou ze základních úprav, avšak není tak lehce dostupná ve většině prohlížečů obrázků.

Tento test byl opět prováděn na sadě obrázků z testu kapacity o rozměrech 720x480. Opět byly tyto obrázky předzpracovány skriptem, který ukryje tajná data a rozdělí je do složek, stejně jako v případě testu na odolnost vůči otáčení. Stejná jsou také ukládaná data. Následně jsou tyto obrázky předány do druhého skriptu, který již provádí testování. Toto testování se skládá ze 4 testů. Prvním testem je pouhé zvětšení na dvojnásobnou velikost, druhým pouhé zmenšení na poloviční velikost, třetím je zvětšení na dvojnásobnou velikost a následné zmenšení do původní velikosti a posledním testem je zmenšení na poloviční velikost a následné zvětšení zpět. Po všech těchto testech se skript pokusí získat tajná data a zapíše výsledky jako tomu bylo u skriptu testujícího odolnost vůči otočení stegangoramu.

Výsledek testování je vidět v tabulce 5.7. Výsledky jsou zpracovány stejně jako bylo uvedeno v přechozím testování odolnosti vůči otočení. V zmíněné tabulce výsledků můžeme pozorovat, že všechny implementované metody nejsou odolné vůči změně velikosti steganogramu.

#### Testovací skript

Spuštění druhého skriptu je následující:

```
./scaleTest.sh složkaObrázky tajnýSoubor výstupníCSV
```

Skript na vstupu dostává `složkaObrázky`, což je cílová složka přechozího skriptu. Dále `tajnýSoubor`, což je totožný soubor jako v prvním skriptu, který je využit pro testování, zda jsou získaná tajná data totožná s ukrývanými. A jako poslední přijímá cestu (`výstupníCSV`), kde budou zapsány výsledky testu ve formátu CSV.

| Formát | Metoda | Zmenšení | Zvetšení | Zvětšení a zmenšení | Zmenšení a zvětšení |
|---|---|---|---|---|---|
| BMP | LSB 1 BIT | NE | NE | NE | NE |
| | LSB 2 BIT | NE | NE | NE | NE |
| | LSB 3 BIT | NE | NE | NE | NE |
| | LSB 4 BIT | NE | NE | NE | NE |
| | Inverted LSB | NE | NE | NE | NE |
| | TripleA 2b | NE | NE | NE | NE |
| | TripleA 3b | NE | NE | NE | NE |
| | TripleA 4b | NE | NE | NE | NE |
| | EMD 2 | NE | NE | NE | NE |
| | EMD 4 | NE | NE | NE | NE |
| | EMD 8 | NE | NE | NE | NE |
| | PVD | NE | NE | NE | NE |
| PNG | LSB 1 BIT | NE | NE | NE | NE |
| | LSB 2 BIT | NE | NE | NE | NE |
| | LSB 3 BIT | NE | NE | NE | NE |
| | LSB 4 BIT | NE | NE | NE | NE |
| | Inverted LSB | NE | NE | NE | NE |
| | TripleA 2b | NE | NE | NE | NE |
| | TripleA 3b | NE | NE | NE | NE |
| | TripleA 4b | NE | NE | NE | NE |
| | EMD 2 | NE | NE | NE | NE |
| | EMD 4 | NE | NE | NE | NE |
| | EMD 8 | NE | NE | NE | NE |
| | PVD | NE | NE | NE | NE |
| JPEG | JSTEG 1 BIT | NE | NE | NE | NE |
| | JSTEG 2 BIT | NE | NE | NE | NE |
| | JSTEG 3 BIT | NE | NE | NE | NE |
| | JSTEG 4 BIT | NE | NE | NE | NE |
| | F3 | NE | NE | NE | NE |
| | F4 | NE | NE | NE | NE |
| | F5 | NE | NE | NE | NE |

Tabulka 5.7: Tabulka výsledků testů robustnosti proti zvětšení a zmenšení, kde NE odpovídá nezvládnutí testu a ANO zvládnutí u celé sady.

### 5.5.3 Odolnost vůči změně formátu

Někdy může dojít ke změně formátu, ať už účelově nebo například při nahrávání na server. Proto byl vytvořen tento test, který převede steganogram na jiný formát a následně provede převod zpět.

Tento test byl proveden pomocí dvou skriptů. Prvním skriptem byl skript pro uložení dat do steganogramů a jejich rozdělení do podsložek, využitý dříve. Druhým byl samotný testovací skript. Tento skript jednotlivé steganogramy převede na jiný formát, následně je převede zpět na původní formát a pokusí se získat tajná data. Výsledky jsou zapsány do CSV. Spuštění skriptů je následující:

Výsledky tohoto testování jsou v tabulce 5.8. Z výsledků je patrné, že nejhůře obstály steganogramy ve formátu JPEG a celkově převod z jakéhokoliv formátu do JPEG formátu. Zde došlo k poškození nesených dat v důsledku přepočítání ztrátové komprimace. Srovnatelně na tom poté byly steganogramy ve formátu BMP a PNG. Zde převod do BMP a PNG nevedl k poškození dat. Z výsledků tohoto testu je možné určit v jakém případě bude možné získat tajná data v případě změny formátu. Vidíme například, že pokud server, na který chceme nahrát steganogram ve formátu PNG, převede tento steganogram do formátu JPEG, nebude ho již možné obnovit.

#### Testovací skript

Druhý skript se spouští následovně:

```
./formatTest.sh složkaObrázky tajnýSoubor výstupníCSV
```

Jako `složkaObrázky` je očekávána složka s podsložkami vytvořená prvním skriptem. Další je `tajnýSoubor`, který musí být stejný jako při ukrytí, jelikož se vůči němu porovnávají skrytá data. Poslední část (`výstupníCSV`) je cesta k souboru kde budou uloženy výsledky ve formátu CSV.

| Formát | Metoda | BMP | PNG | JPEG |
|---|---|---|---|---|
| BMP | LSB 1 BIT | ANO | ANO | NE |
| | LSB 2 BIT | ANO | ANO | NE |
| | LSB 3 BIT | ANO | ANO | NE |
| | LSB 4 BIT | ANO | ANO | NE |
| | Inverted LSB | ANO | ANO | NE |
| | TripleA 2b | ANO | ANO | NE |
| | TripleA 3b | ANO | ANO | NE |
| | TripleA 4b | ANO | ANO | NE |
| | EMD 2 | ANO | ANO | NE |
| | EMD 4 | ANO | ANO | NE |
| | EMD 8 | ANO | ANO | NE |
| | PVD | ANO | ANO | NE |
| PNG | LSB 1 BIT | ANO | ANO | NE |
| | LSB 2 BIT | ANO | ANO | NE |
| | LSB 3 BIT | ANO | ANO | NE |
| | LSB 4 BIT | ANO | ANO | NE |
| | Inverted LSB | ANO | ANO | NE |
| | TripleA 2b | ANO | ANO | NE |
| | TripleA 3b | ANO | ANO | NE |
| | TripleA 4b | ANO | ANO | NE |
| | EMD 2 | ANO | ANO | NE |
| | EMD 4 | ANO | ANO | NE |
| | EMD 8 | ANO | ANO | NE |
| | PVD | ANO | ANO | NE |
| JPEG | JSTEG 1 BIT | NE | NE | NE |
| | JSTEG 2 BIT | NE | NE | NE |
| | JSTEG 3 BIT | NE | NE | NE |
| | JSTEG 4 BIT | NE | NE | NE |
| | F3 | NE | NE | NE |
| | F4 | NE | NE | NE |
| | F5 | NE | NE | NE |

Tabulka 5.8: Tabulka výsledků testů odolnosti vůči změně formátu, kde NE odpovídá nezvládnutí testu a ANO zvládnutí u celé sady.

### 5.5.4 Odolnost proti komprimaci

Pokud někomu posíláme fotky ve větším množství, je běžné, že je vložíme do archivu a pošleme jako jeden celek. Takto můžeme postupovat i v případě steganogramů. Můžeme například skrýt data do čtyř obrázků, vložit je mezi dalších padesát a všechny je vložit do jednoho archívu. Z tohoto důvodu vznikl nápad na tento test, který má otestovat, zda steganogramy vydrží kompresi a dekompresi bez poškození.

Tento test byl proveden pomocí dvou skriptů. Prvním skriptem byl již zmiňovaný skript pro uložení dat do steganogramů a jejich rozdělení do podsložek. Druhým byl samotný testovací skript. Tento skript jednotlivé steganogramy komprimuje do archivu a následně extrahuje. Tento extrahovaný steganogram je poté předám do aplikace k odkrytí tajných dat. Výsledky jsou zapsány do CSV. Spuštění skriptů je následující:

Výsledky tohoto testování jsou v tabulce 5.9. Z výsledků je patrné, že testovaná komprimace nijak nepoškodila žádný ze steganogramů.

### Testovací skript

Druhý skript je možné spustit takto:

```
./ziptarTest.sh složkaObrázky tajnýSoubor výstupníCSV
```

Jako `složkaObrázky` je očekávána složka s podsložkami vytvořená prvním skriptem. Další je `tajnýSoubor`, který musí být stejný jako v prvním skriptu, jelikož se vůči němu porovnávají skrytá data. Poslední část (`výstupníCSV`) je cesta k souboru kde budou uloženy výsledky ve formátu CSV.

| Formát | Metoda | tar | tar.gz | zip |
|---|---|---|---|---|
| BMP | LSB 1 BIT | ANO | ANO | ANO |
| | LSB 2 BIT | ANO | ANO | ANO |
| | LSB 3 BIT | ANO | ANO | ANO |
| | LSB 4 BIT | ANO | ANO | ANO |
| | Inverted LSB | ANO | ANO | ANO |
| | TripleA 2b | ANO | ANO | ANO |
| | TripleA 3b | ANO | ANO | ANO |
| | TripleA 4b | ANO | ANO | ANO |
| | EMD 2 | ANO | ANO | ANO |
| | EMD 4 | ANO | ANO | ANO |
| | EMD 8 | ANO | ANO | ANO |
| | PVD | ANO | ANO | ANO |
| PNG | LSB 1 BIT | ANO | ANO | ANO |
| | LSB 2 BIT | ANO | ANO | ANO |
| | LSB 3 BIT | ANO | ANO | ANO |
| | LSB 4 BIT | ANO | ANO | ANO |
| | Inverted LSB | ANO | ANO | ANO |
| | TripleA 2b | ANO | ANO | ANO |
| | TripleA 3b | ANO | ANO | ANO |
| | TripleA 4b | ANO | ANO | ANO |
| | EMD 2 | ANO | ANO | ANO |
| | EMD 4 | ANO | ANO | ANO |
| | EMD 8 | ANO | ANO | ANO |
| | PVD | ANO | ANO | ANO |
| JPEG | JSTEG 1 BIT | ANO | ANO | ANO |
| | JSTEG 2 BIT | ANO | ANO | ANO |
| | JSTEG 3 BIT | ANO | ANO | ANO |
| | JSTEG 4 BIT | ANO | ANO | ANO |
| | F3 | ANO | ANO | ANO |
| | F4 | ANO | ANO | ANO |
| | F5 | ANO | ANO | ANO |

Tabulka 5.9: Tabulka výsledků testů odolnosti proti komprimaci, kde NE odpovídá nezvládnutí testu a ANO zvládnutí u celé sady.

### 5.5.5 Přeposlání přes různé komunikátory

Jelikož steganografie je o komunikaci přes běžné komunikační kanály, tak byl do testování přidán tento test. Tento test otestuje, zda steganogramy vytvořené jednotlivými metodami přežijí přenos přes některé populární komunikátory.

Toto testování bylo prováděno ve čtyřech částech a většina tohoto testování kromě skrývání dat do obrázku a pokusu o odhalení skrytých dat probíhala ručně. Ruční testování neulehčovaly ani zvolené komunikátory. Z tohoto důvodu by bylo testování na plné sadě použité v předchozích testech časově náročné a tak bylo testování omezeno na menší počet steganogramů. První část spočívala ve skrytí tajných dat do souboru, ovšem v tomto testu nebyl zvolen přístup, kde jsou steganogramy ukládány do podsložek, ale název metody a výstupního formátu je obsažen do názvu souboru. Druhá fáze testování spočívala v přeposlání jednoho steganogramu pro každý testovaný formát. Toto testování mělo za cíl ověřit chování komunikátoru k danému obrazovému formátu, jelikož některé komunikátory formáty převádějí, jak bude popsáno vždy u daného komunikátoru. Třetí fází poté bylo otestování všech metod všech formátů, které prošly druhou fází. Zde bylo pro otestování jednotlivých metod zvoleno 10 co nejrůznorodějších obrázků. Tyto obrázky byly přesunuty skrze komunikátor a na druhé straně bylo vyzkoušeno odtajnění tajné zprávy, což je poslední fází procesu.

Všechny výsledky tohoto testování jsou uvedeny v tabulce 5.10 a níže jsou popsány jednotlivě testované komunikátory.

#### Testovací skript

První skript se spouští následovně:

```
./prepareSteganogramsIntoOneFolder.sh složkaObr cílováSložka tajnýSoubor
```

Skript přijímá jako argumenty dvě složky a jeden soubor. Argument `složkaObr` obsahuje obrázky, do kterých budou skryty data. Argument `cílováSložka` je složka, kde budou uloženy vytvořené steganogramy do podsložek s názvem tvořeným výstupním formátem a použitou metodou. Poslední argument je `tajnýSoubor`, tento soubor bude skryt do všech obrázků ve složce `složkaObr`.

Spuštění druhého skriptu je následující:

```
./socialTest.sh složkaObrázky tajnýSoubor výstupníCSV
```

Skript na vstupu dostává `složkaObrázky`, což je složka s obrázky po přenosu přes komunikátory. Důležité je, aby soubory měli stejné pojmenování jako měly po vytvoření prvním skriptem. Dále `tajnýSoubor`, což je totožný soubor jako v prvním skriptu, který je využit pro testování, zda jsou získaná tajná data totožná s ukrývanými. A jako poslední přijímá cestu (`výstupníCSV`), kde budou zapsány výsledky testu ve formátu CSV.

#### Testované komunikátory

**Facebook Messenger** Messenger je jeden ze zmíněných komunikátorů, které při přeposílání konvertují jeden formát na jiný. Toto se děje u formátu BMP, který je převeden na JPEG. Toto zjištění plyne z přijímaného formátu. Pokud tedy pošleme soubor s příponou bmp na druhé straně už stáhneme pouze soubor s příponou jpeg. Při přeposlání souborů s příponou png nebo jpeg se změna přípony, tedy formátu nekoná. U JPEG se však při výchozím nastavení skrývání nepovede data obnovit, jelikož Messenger znovu provede kompresi JPEG. Po analýze přenesených souborů, bylo zjištěno, že Facebook používá nastavení

kvality komprese na 71%. Po tomto zjištění bylo provedeno nové testování, kde vytvořené steganogramy pomocí metod pro formát JPEG byly nastaveny na kompresi s kvalitou 71%. V tomto případě prošly všechny testované soubory. Dalším chováním, které komplikuje hromadné testování, avšak při normální tajné komunikaci ničemu nevadí, je přejmenování souboru při přenosu. Pokud nahrajeme na jedné straně soubor, který nese v názvu metodu a formát, tak na druhé straně tento soubor stahujeme přejmenovaný, dle nějaké logiky aplikace. Proto bylo při testování nutné posílat steganogramy po jednotlivých metodách, stahovat je v pořadí a přejmenovávat.

**WhatsApp** WhatsApp je druhým komunikátorem, který provádí konverzi mezi formáty. U tohoto komunikátoru jsou všechny formáty převedeny do formátu JPEG. Ovšem pokud je původní steganogram již v tomto formátu, není pravděpodobně znovu podroben kompresi nebo používá stejné nastavení pro kompresi jako implementovaná aplikace v rámci této práce, jelikož přeposlané JPEG obrázky jsou nepoškozeny. Opět se zde jako v případě Messengeru přejmenovává přeposílaný soubor, což s sebou nese zmíněné nevýhody při testování. V případě tohoto testu bylo zjištěno, že v důsledku konverze mezi formáty dojde k poškození všech steganogramů ve formátu PNG a BMP. Steganogramy ve formátu JPEG jsou přeneseny a obnoveny bez poškození a získaná tajná data se shodují.

**Email** V případě emailu byly steganogramy vloženy jako příloha emailu. Avšak mnou používané emaily mají omezení na vložení přílohy o maximální velikosti 25MB. Proto bylo nutné tyto steganogramy poslat po částech. Výhodou však bylo zachování názvů steganogramů. V případě emailu nedošlo ani k jednomu poškození steganogramu.

**ZIP** Jak bylo ukázáno v minulém testu při komprimaci steganogramů nedochází k jejich poškození. Tohoto faktu lze využít pří přeposílání přes různé komunikátory, jelikož například v případě Messengeru a WhatAppu dochází u některých formátů k jejich poškození. Pokud z nějakého důvodu chceme tento formát přesto použít, můžeme tak učinit oklikou a to pomocí komprimace. Pokud steganogram komprimujeme, pak jej lze nepoškozený přenést i přes tyto komunikátory. Výsledek testu s názvem ZIP, pak znamená výsledek přenesení archivu obsahující steganogramy skrze všechny testované komunikátory.

| Formát | Metoda | Messenger | WhatsApp | Email | ZIP |
|---|---|---|---|---|---|
| BMP | LSB 1 BIT | NE | NE | ANO | ANO |
| | LSB 2 BIT | NE | NE | ANO | ANO |
| | LSB 3 BIT | NE | NE | ANO | ANO |
| | LSB 4 BIT | NE | NE | ANO | ANO |
| | Inverted LSB | NE | NE | ANO | ANO |
| | TripleA 2b | NE | NE | ANO | ANO |
| | TripleA 3b | NE | NE | ANO | ANO |
| | TripleA 4b | NE | NE | ANO | ANO |
| | EMD 2 | NE | NE | ANO | ANO |
| | EMD 4 | NE | NE | ANO | ANO |
| | EMD 8 | NE | NE | ANO | ANO |
| | PVD | NE | NE | ANO | ANO |
| PNG | LSB 1 BIT | ANO | NE | ANO | ANO |
| | LSB 2 BIT | ANO | NE | ANO | ANO |
| | LSB 3 BIT | ANO | NE | ANO | ANO |
| | LSB 4 BIT | ANO | NE | ANO | ANO |
| | Inverted LSB | ANO | NE | ANO | ANO |
| | TripleA 2b | ANO | NE | ANO | ANO |
| | TripleA 3b | ANO | NE | ANO | ANO |
| | TripleA 4b | ANO | NE | ANO | ANO |
| | EMD 2 | ANO | NE | ANO | ANO |
| | EMD 4 | ANO | NE | ANO | ANO |
| | EMD 8 | ANO | NE | ANO | ANO |
| | PVD | ANO | NE | ANO | ANO |
| JPEG | JSTEG 1 BIT | ANO | ANO | ANO | ANO |
| | JSTEG 2 BIT | ANO | ANO | ANO | ANO |
| | JSTEG 3 BIT | ANO | ANO | ANO | ANO |
| | JSTEG 4 BIT | ANO | ANO | ANO | ANO |
| | F3 | ANO | ANO | ANO | ANO |
| | F4 | ANO | ANO | ANO | ANO |
| | F5 | ANO | ANO | ANO | ANO |

Tabulka 5.10: Tabulka výsledků testů robustnosti proti poškození přeposláním, kde NE odpovídá nezvládnutí testu a ANO zvládnutí u celé sady.

# Kapitola 6

# Závěr

V této práci byl čtenář seznámen se steganografií, jak na obecné úrovni, tak i konkrétně v oblasti digitální steganografie pro skrývání informace v obrazových datech. V rámci obecné steganografie bylo uvedeno o co se jedná, zmíněny byly známé výskyty steganografie v historii a také modernější výskyty. Po této části následovalo vysvětlení dělení digitální steganografie, dle typu nosiče, typu metod a dále bylo vysvětleno jaký vliv má volba skrývané informace. Jelikož je tato práce zaměřena primárně na obrazovou steganografii, byla pro toto téma vymezená samostatná kapitola. V této kapitole bylo seznámení s obrázkem a jednotlivými použitými obrazovými formáty. Následně byla vysvětlena hlavní část věnující se vlastnostem steganografických metod a samotným metodám.

Požadavkem v zadání bylo implementovat několik existujících metod. Proto byla v rámci této práce vytvořena i aplikace pro aplikaci steganografických metod pro obrazovou steganografii. Přesněji se jedná o dvě aplikace se stejným logickým jádrem. Jedna aplikace využívá grafické uživatelské rozhraní a druhá konzolové z důvodu použití ve skriptech. Mým hlavním cílem bylo udělat aplikaci, co nejvíce použitelnou s co nejmenším omezením na typ skrývané informace. Z tohoto důvodu byl zvolen postup implementace, kde jako skrývaná data může být využit jakýkoliv soubor a uživatel tak není omezen například jen na textové zprávy a může tajně posílat například i archívy a dokumenty. Dalším mým stanoveným požadavkem bylo vytvořit aplikaci, která bude podporovat více formátů, jelikož ne vždy chceme použít jeden daný formát anebo to není vždy možné, jak se ukázalo v testech na přenesení steganogramu skrze různé komunikátory. Navíc bylo přidáno dostatečné zabezpečení skrývaných dat, a to za pomoci kontrolního součtu a šifrování.

Implementované metody byly následně podrobeny testování. Testována byla kapacita, nepostřehnutelnost, robustnost a vliv vstupního formátu. Výsledky většiny těchto testů byly shrnuty do tabulky v příslušné části.

Vytvořená aplikace poskytuje 3 výstupní formáty a k nim dostupné steganografické metody. Pro BMP 12, pro PNG 12 a pro JPEG 7 metod. U některých metod je dále možné zapnout náhodné rozložení po celém obrázku. Avšak vždy je co zlepšovat, a tak pro tyto případy jsem aplikaci navrhoval tak, aby byla do budoucna snadno rozšířitelná o další formáty, a hlavně o další steganografické metody. V tomto směru vidím možný vývoj představené aplikace.

Bylo by tedy možné doimplementovat další metody a formáty. Dokonce by bylo možné mezi tyto steganografické metody zařadit i metody využívané pro vodoznaky. Dalším možným rozšířením by bylo implementování vlastní třídy pro kompresi a práci s JPEG formátem. Co se tyká samotné práce, tak by bylo možné hlouběji se věnovat testování steganografických metod, a to zejména z druhé strany steganografie, tedy pomocí steganoanalýzy.

Implementovat steganoanalytický nástroj anebo využít některý z dostupných nástrojů. Steganoanalýza je ovšem rozsáhlé téma, kde existují různé specifické i obecné metody a vydala by na celou další práci a není zaměřením této práce.

Dle mého názoru je steganografie a steganoanalýza zajímavá, i když obtížná a má dle mého názoru budoucnost. Zejména v oblasti obrany proti novým metodám steganografii, která se poslední dobou vyskytují ve škodlivém softwaru, jako nepozorovaná forma komunikace.

# Literatura


[1] *Digital watermarking and steganography*. Burlington: Morgan Kaufmann, druhé vydání, 2008, ISBN 978-0-12-372585-1.

[2] Usage of image file formats for websites. https://w3techs.com/technologies/overview/image_format/all, 2019, [Online; accessed 14-February-2019].

[3] Agarwal, M.: Text Steganographic Approaches: A Comparison. *arXiv.org*, ročník 5, č. 1, 2013, ISSN 0975-2307.
URL http://search.proquest.com/docview/2085278910/

[4] Akhtar, N.; Khan, S.; Johri, P.: An improved inverted LSB image steganography. IEEE, 2014, ISBN 978-1-4799-2900-9, s. 749–755.

[5] Buck, P.: Reverse Engineering the Machine Identification Code. https://www.researchgate.net/publication/325976319_Reverse_Engineering_the_Machine_Identification_Code, Jun 2018.

[6] Dhruw, T.; Tiwari, N.: Different method used in pixel value differencing algorithm. *IOSR J. Comput. Eng*, ročník 18, č. 3, 2016: s. 102–109, doi:10.9790/0661-180304102109.

[7] Fkirin, A.; Attiya, G.; El-Sayed, A.: Steganography Literature Survey, Classification and Comparative Study. *Communications on Applied Electronics*, ročník 5, č. 10, 2016: s. 13–22, ISSN 23944714.

[8] Gardner, F.: How do terrorists communicate? http://www.bbc.com/news/world-24784756, Nov 2013.

[9] Gery, R.: DIBs and Their Use. *Microsoft Developer Network Technology Group*, Mar 1992.
URL https://msdn.microsoft.com/en-us/library/ms969901.aspx

[10] Gonzalez, R. C.: *Digital image processing*. Upper Saddle River: Pearson ; Prentice Hall, třetí vydání, 2008, ISBN 0-13-168728-X.

[11] Gutub, A.; Al-Qahtani, A.; Tabakh, A.: Triple-A: Secure RGB image steganography based on randomization. IEEE Publishing, 2009, ISBN 9781424438075, s. 400–403.

[12] Hamilton, E.: JPEG File Interchange Format. 1992.
URL https://www.repository.cam.ac.uk/bitstream/handle/1810/54/jfif.pdf?sequence=1&isAllowed=y

[13] J. Simmons, G.: The Prisoners' Problem and the Subliminal Channel. 01 1983, s. 51–67, doi:10.1007/978-1-4684-4730-9_5.

[14] Jayaram, P.; Ranganatha, H.; Anupama, H.: Information hiding using audio steganography–a survey. *The International Journal of Multimedia & Its Applications (IJMA)*, ročník 3, 2011, ISSN 0975-5934.
URL https://pdfs.semanticscholar.org/28fe/8e2480bc957c4aa8172214d466ba9613e8a9.pdf

[15] Johnson, N.; Duric, Z.; Jajodia, S.: *Information Hiding: Steganography and Watermarking-Attacks and Countermeasures: Steganography and Watermarking - Attacks and Countermeasures*. Advances in Information Security, Springer US, 2012, ISBN 978-1-4615-4375-6.
URL https://books.google.com/books?id=29XgBwAAQBAJ

[16] Judge, J. C.: Steganography: Past, Present, Future - UCRL-ID-151879. Technická zpráva, 2001.

[17] Kipper, G.: *Investigator's guide to steganography*. Florida: CRC Press Company, vyd. 1. vydání, 2004, ISBN 0-8493-2433-5.

[18] Koníček, T.: Ministerstvo vnitra České republiky.
https://www.mvcr.cz/clanek/oznacovani-jizdnich-kol-pomoci-forenzniho-znaceni-je-dobrou-prevenci-odcizeni.aspx.

[19] Lau, H.: The truth behind the Shady RAT. *McAffe report*, 2011.

[20] Lomas, N.: Encryption under fire in Europe as France and Germany call for decrypt law. https://techcrunch.com/2016/08/24/encryption-under-fire-in-europe-as-france-and-germany-call-for-decrypt-law/?guccounter=1, Aug 2016.

[21] Mielikainen, J.: LSB matching revisited. *IEEE Signal Processing Letters*, ročník 13, č. 5, 2006: s. 285–287, ISSN 1070-9908.
URL http://search.proquest.com/docview/28011668/

[22] Morkel, T.; Eloff, J. H.; Olivier, M. S.: An overview of image steganography. In *ISSA*, 2005, s. 1–11.

[23] Owens, M.: A discussion of covert channels and steganography. *SANS institute*, ročník 1, 2002: s. 1–18.

[24] Randers-Pehrson, G.; Boutell, T.; aj.: PNG (Portable Network Graphics) Specification, Version 1.2. 1999.

[25] Roelofs, G.: A Basic Introduction to PNG Features.
http://www.libpng.org/pub/png/pngintro.html.

[26] Singh, S.: *Kniha kódů a šifer : Tajná komunikace od starého Egypta po kvantovou kryptografii*. Aliter, Praha: Argo, první vydání, 2003, ISBN 80-7203-499-5.

[27] Taouil, Y.; Ameur, E.; Souhar, A.; aj.: High Imperceptibility Image Steganography Methods based on HAAR DWT. *International Journal of Computer Applications*, ročník 138, č. 10, 2016: s. 38–43, ISSN 09758887.

[28] Thung, K.-H.; Raveendran, P.: A survey of image quality measures. In *Technical Postgraduates (TECHPOS), 2009 International Conference for*, IEEE, 2009, ISBN 9781424452231, s. 1–4.

[29] Tišnovský, P.: Anatomie grafického formátu PNG. https://www.root.cz/clanky/anatomie-grafickeho-formatu-png/, Sep 2006.

[30] Tišnovský, P.: Grafický formát BMP - používaný a přitom neoblíbený. https://www.root.cz/clanky/graficky-format-bmp-pouzivany-a-pritom-neoblibeny/, Oct 2006.

[31] Tišnovský, P.: PNG is Not GIF. https://www.root.cz/clanky/png-is-not-gif/, Sep 2006.

[32] Tišnovský, P.: Programujeme JPEG: Huffmanovo kódování kvantovaných DCT složek. https://www.root.cz/clanky/programujeme-jpeg-huffmanovo-kodovani-kvantovanych-dct-slozek/, Jan 2007.

[33] Tišnovský, P.: Programujeme JPEG: Interní struktura souborů typu JFIF/JPEG. https://www.root.cz/clanky/programujeme-jpeg-interni-struktura-souboru-typu-jfifjpeg/, Jan 2007.

[34] Wang, Z.; Bovik, A.; Sheikh, H.; aj.: Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, ročník 13, č. 4, 2004: s. 600–612, ISSN 1057-7149.

[35] Westfeld, A.: F5—a steganographic algorithm. In *International workshop on information hiding*, Springer, 2001, s. 289–302.
URL http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=76AF73ECFABB5C17D53719DF1CCBE379?doi=10.1.1.115.3651&rep=rep1&type=pdf

[36] Wikipedia: Cryptography — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Cryptography, 2018, [Online; accessed 04-March-2018].

[37] Wikipedia: Steganography — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Steganography, 2018, [Online; accessed 04-March-2018].

[38] Wikipedia: JPEG — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=JPEG&oldid=882863803, 2019, [Online; accessed 14-February-2019].

[39] Wu, D.-C.; Tsai, W.-H.: A steganographic method for images by pixel-value differencing. *Pattern Recognition Letters*, ročník 24, č. 9-10, 2003: s. 1613–1626, ISSN 0167-8655.

[40] Zhang, X.; Wang, S.: Dynamical running coding in digital steganography. *Signal Processing Letters, IEEE*, ročník 13, č. 3, 2006: s. 165–168, ISSN 1070-9908.

[41] Zhang, X.; Wang, S.: Efficient Steganographic Embedding by Exploiting Modification Direction. *Communications Letters, IEEE*, ročník 10, č. 11, 2006, ISSN 1089-7798.

# Přílohy

# Příloha A

# Steganografie v historii

Tato část je jen základní ukázkou několika vybraných metod steganografie, které byly v historii použity, další metody mohou být nalezeny v čerpané literatuře nebo poměrně přehledně jsou uvedeny metody vyskytující se v historii v knize [17]. Tato kapitola si neklade za cíl tyto metody nikterak porovnávat, ale spíše informovat o jejich existenci, jelikož jsou zajímavé. Z této kapitoly je zřejmé, že steganografie si našla své nesporné využití i v přítomnosti kryptografie a někdy dokonce důmyslné utajení zprávy ovlivnilo vývoj válek. Velká část zmiňovaných metod použitých v historii je spojována právě s těžkými časy, jako byly války a povstání. Tyto časy totiž vedly ke kontrolám komunikace. Zde nachází své uplatnění steganografie, která při správném využití, působí nenápadně a umožňuje tyto kontroly podstoupit bez odhalení.

## Voskem pokryté varování

Jednou z nejstarších zmínek o steganografii nalezneme ve spisech s názvem Dějiny věnující se převážně řecko-perským válkám, a to od slavného historika Hérodota, jinak také známého jako "otce dějepisu", jak ho nazval Marcus Tullius Cicero. Ta praví o Řeku Démaratosovi, který byl Spartským králem, avšak poté co byl sesazen a vyhnán ze Sparty, našel útočiště v Persii. V Persii Xerxes začal stavět Persepolis a z okolí sbíral poplatky a dary, výjímku tvořily Athény a Sparta, kteří odmítali. To se Xerxesovi nelíbilo a hodlal je potrestat. S rozmachem své říše vybudoval největší armádu v tehdejší historii a chystal se na úder. To vše viděl Démaratos, ten přestože byl vyhnán, nepřestal být loajální k Řecku a jeho lidu. Chtěl tedy Řeky informovat, problémem bylo, jak dopravit zprávu nezachycenou. Nakonec přišel s nápadem. Tímto nápadem bylo skrytí varování do voskové destičky, tehdy populárního psacího nástroje. A to tak, že oškrábal všechen vosk, až na podkladové dřevo a do tohoto dřeva vyryl onu zprávu s varováním. Poté destičku obnovil zalitím novou vrstvou vosku. Takto zalitá zpráva nebyla vůbec viditelná a destička vypadala jako nová a tudíž nebudila podezření. Zpráva byla takto přenesena do Řecka. To dalo Řekům taktickou výhodu. Když dorazila perská armáda, čekali Řecko nepřipravené, avšak opak byl pravdou a Řecko se ubránilo.[26]

## Tetovaná hlava jako pokyn k povstání

Další zmínku opět najdeme v Hérodotových Dějinách. Původce této metody byl opět Řek, tentokrát však Histiaeus, který chtěl započít povstání proti Perskému králi. Pro vyjednání

však musel doručit tajnou zprávu. Metoda, kterou vymyslel byla docela časově náročná. Jednalo se o vytetování zprávy na poslovu hlavu. K tomu bylo zapotřebí oholit otrokovi hlavu, vytetovat zprávu a pak čekat, až poslovi dorostou vlasy a tím se zpráva zakryje. Posel pak mohl nerušeně dojít na místo předání zprávy a tam mu byla hlava oholena.[26] Jak již bylo řečeno, tato metoda byla poměrně zdlouhavým procesem a navíc tetování je na stálo, takže tato skutečnost se jistě musela nějak vyřešit.

## Čínský styl pašování zpráv

Starověcí Číňané měli také svou techniku steganografie. Využívali hedvábí, na které byla napsána zpráva. Tento kousek jemného hedvábí byl poskládán či pomačkán a následně zalit ve vosku do tvaru kuličky. Tato kulička, pak byla dána poslovi a ten jí spolknul.[26] Přestože jsem zmínku o této metodě nalezl v mnoha zdrojích, nebyla zde uvedena metoda extrakce zprávy, kterou si můžeme jen domýšlet. Tato metoda byla stejně jako předchozí postavena na důvěryhodnosti nositele, který tvořil i její slabinu.

## Na pořadí záleží

V historii se objevuje nulová šifra (Null cipher), ačkoliv nese název šifra, tak by měla spadat spíše mezi steganografické metody. Tato metoda sice utajuje znění zprávy, ale tak, že jej skryje do nevinně vypadajícího textu. A to tak, že zprávu po jednotlivých písmenech ukrývá do slov na určitou pozici, která tato písmena obsahuje a dohromady dávají smysluplné sdělení.[15] Malou ukázkou může být, pokud přečteme každé první písmeno u všech slov ve větě "Brzy ráno na obloze." to nám dá "Brno". O kterou pozici ve slově se jedná musíme vědět dopředu.

Obměnou této metody je do jisté míry, metoda s mřížkou (Cardano grille), která namísto dané pozice využívá vzor. Tento vzor je přenesen do destičky v podobě děr. Do těchto děr se následně vpisuje zpráva opět po písmenech a následně jsou písmena doplněna slovy, které opět dávají smysluplný text a zaplňují prostor ponechaný mřížkou. K čtení takovéto zprávy nám nestačí jen pozice písmene ve slově, ale potřebujeme kopii mřížky.[1] Tato mřížka dělá tuto metodu silnější co se týče bezpečnosti proti prohlédnutí, jelikož se písmena nenacházejí na stejných pozicích, ale také jí znehodnotí v případě že adresát přijde o svou kopii mřížky.

## Neviďitelný inkoust

Poměrně známým nástrojem je neviditelný inkoust. Dnes jde zakoupit i jako hračka pro děti a to ve své pokročilejší formě zobrazitelné pod UV zářením. Jeho začátky byly mnohem prostší.

První neviditelný inkoust byl tvořen mlékem z rostliny zvané pryšec. Při psaní tímto mlékem, napsaný text po chvíli vyschnul a nezanechal po sobě žádné stopy. Avšak pokud jsme místo popsané mlékem nahřáli, například nad plamenem svíčky, tak mléko začalo hnědnout z důvodu uhelnatění obsaženého uhlíku a zpráva se objevila.[26]

Jiná forma neviditelného psaní byla vyvinuta vědcem v šestnáctém století v Itálii, jež se jmenoval Giovanni Battista della Porta. Tento vědec využil pórovité struktury skořápky vajec. Na uvařené vejce bylo pomocí roztoku napsáno sdělení a toto vejce po vstřebání tekutiny, nejevilo žádné známky manipulace. Zpráva byla viditelná až po oloupání, protože

inkoust působil právě na bílek vejce. Roztok byl vyroben z jedné unce (cca 28 gramů) kamence a jednoho pintu (cca 4.7 dl) octa.[26]

Nejčastěji byla jako neviditelný inkoust na jednoduší bázi využívána: šťáva z ovoce, ocet, mléko nebo dokonce moč.[15] Všechna mají společné to, že při vyschnutí a následném zahřátí tmavnou. Avšak s postupem času je nahradily sofistikovanější a hůře odhalitelné inkousty na chemické bázi a tyto starší metody zůstaly spíše jako nouzové.

## Mikrotečky

Technologie mikroteček byly vyvinuty Němci na skrývání zpráv před spojeneckými silami. Tehdejší ředitel FBI, J. Edgar Hoover, dokonce označil technologii mikroteček za nepřátelské mistrovské dílo špionáže, jak je zmíněno v knize [15]. Jednalo se o vysoce detailní fotografie zmenšené na velikost tečky v textu. Tato metoda dokonce dokázala do jedné takovéto tečky ukrýt celou stránku textu nebo celou fotografii, kterou mohla být například letecká fotografie území či mapa vojenské základny. Zobrazení takto zakódované zprávy se provádělo zvětšením za pomoci mikroskopu. K odhalení této metody vedl fakt, že tyto mikrotečky byly na filmovém podkladu, a proto se jejich podklad lišil od nosiče. Filmový podklad byl totiž o dost lesklejší v porovnání s klasickým matným papírem, použitého pro korespondenci. Vývoj mikroteček pokračoval i po válce.

## Mnoho dalších

Mezi další známé metody se řadí schování zprávy na zadní straně poštovní známky nebo děrování novin k označení písmen pro poskládání zprávy. Poměrně zajímavý je i příběh o "Doll Woman"(Panenkové ženě), Velvalee Dickinson, která pomocí steganografie sdělovala pohyby lodí. V knize [1] je jako steganografie uvedena i událost, ve které velitel Jeremiah Denton byl zajat v Severním Vietnamu a při výslechu mrká tajnou zprávu sdělující T-O-R-T-U-R-E (mučení). Tento záznam měl být odvysílán v rámci propagandy.

# Příloha B

# Manuál

## B.1 Instalace

Pro spuštění obou verzí programu je nutné mít nainstalovanou Javu. Javu je možné stáhnout z oficiální stránky https://www.java.com/en/download/. V příkazové řádce je poté možné provést kontrolu pomocí `java -version`.

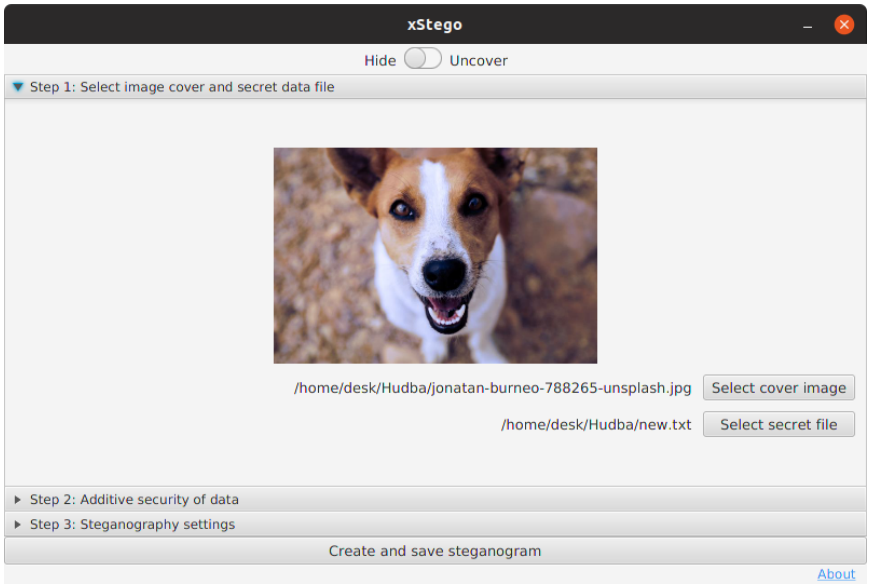
Nic víc není potřeba instalovat. Pouze je nutné aby aplikace (.jar) byla ve stejné složce jako složka s knihovnami (složka s názvem `libs`). Pokud z nějakého důvodu chceme aplikaci v jiné složce než jsou složky s knihovnami, tak je to možné. V tomto případě je nutné při spuštění aplikace dodat parametr programu `java` a to parametr `-Djava.library.path=`, kde za `=` zadáme cestu ke složce s knihovnou pro naší danou platformu, kterou je jedna ze složek ve složce `libs`. Popřípadě je možné tuto cestu dodat do systémové cesty ke knihovnám.

## B.2 GUI

Aplikace nabízí dva řežimy skrytí a odkrytí tyto režimy se přepínají přepínačem v horní části okna. Níže bude zmíněn postup pro vytvoření steganogramu a odkrytí tajných dat ze steganogramu.

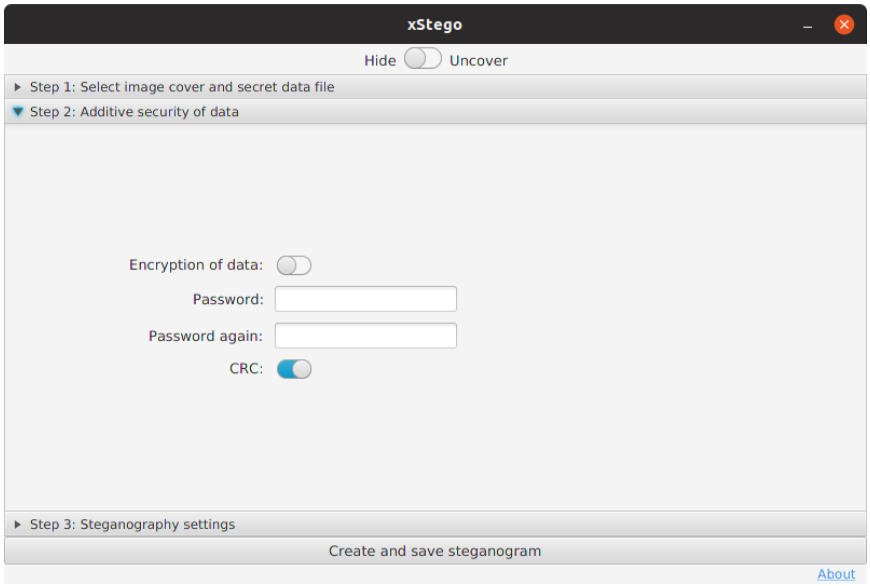
### B.2.1 Skrytí tajných dat

První krok pro skrytí tajných dat můžete vidět na obrázku B.1. Zde je potřeba vybrat obrázek a tajný soubor, který chceme ukrýt. Výběr provedeme pomocí tlačítek "Select cover image" respektive "Select secret file". Po stisknutí bude zobrazeno okno pro výběr souboru. V případě výběru obrázku je výběr omezen na obrazové formáty. Po zvolení souborů jsou zobrazeny cesty k souboru a zvolený obrázek je zobrazen v náhledu.




Obrázek B.1: Skrytí krok 1

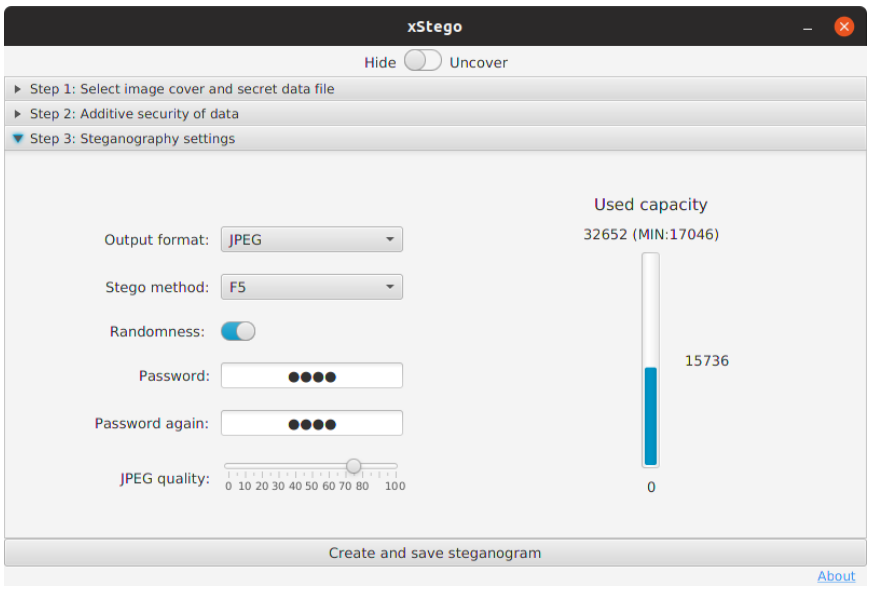
V druhém kroku zachyceném na obrázku B.2 se nachází volitelné dodatečné zabezpečení dat. Můžeme zapnout šifrování dat a zvolit vlastní heslo, které musíme vyplnit shodně do obou polí pro kontrolu překlepů. Dále je v nabídce zabezpečení kontrolním součtem (CRC), což nám umožní detekovat porušení dat.




Obrázek B.2: Skrytí krok 2

Posledním krokem (obrázek B.3) je nastavení steganografie. Zde máme možnost zvolit si výstupní formát souboru a k němu příslušnou metodu. Pokud to zvolená metoda podporuje, můžeme zvolit náhodné rozložení dat po celém obrázku a zvolit si heslo na základě kterého bude náhodné rozdělení počítáno. V případě JPEG formátu máme navíc možnost ovlivnit nastavení kvality (míru komprese). V pravo je zobrazena kapacita a její zaplnění. Posledním

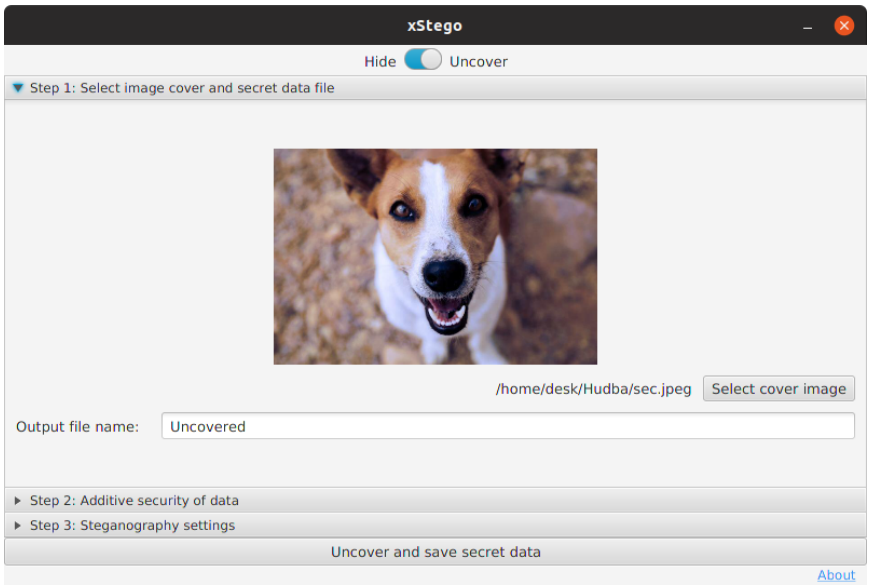
krokem pro vytvoření steganogramu je zmáčknout tlačítko ve spodní části okna a zvolit umístění souboru.




Obrázek B.3: Skrytí krok 3

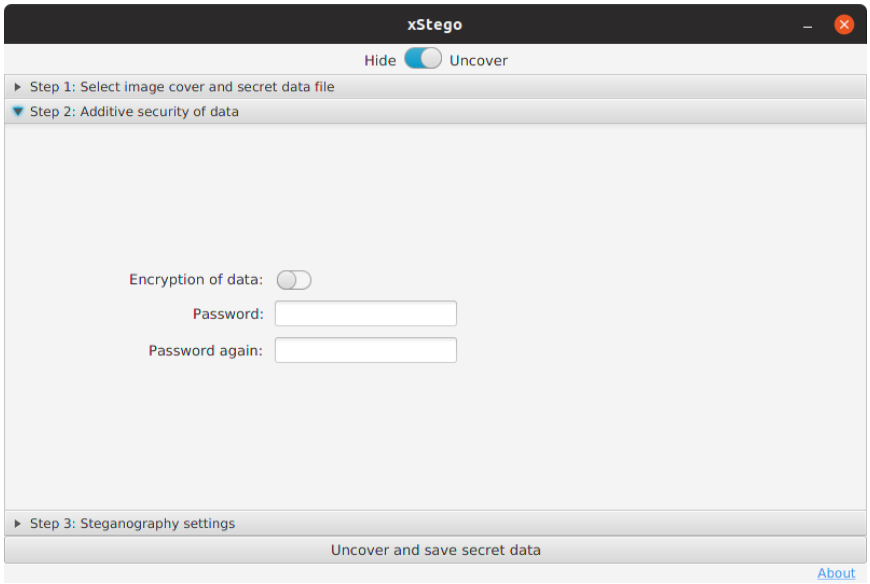
### B.2.2 Odkrytí tajných dat

První krok pro odtajnění dat je na obrázku B.4. Zde je potřeba vybrat obrázek, který nese tajná data. Výběr provedeme pomocí tlačítka "Select cover image". Po stisknutí bude zobrazeno okno pro výběr obrázku. Po zvolení souborů je zobrazena cesta k souboru a zvolený obrázek je zobrazen v náhledu. Níže nastavíme název tajného souboru po odkrytí.



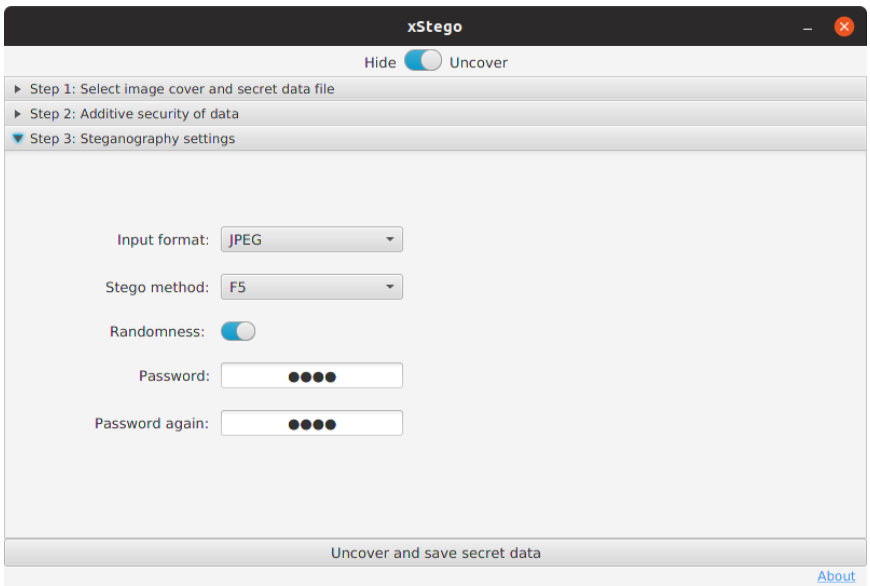

Obrázek B.4: Odtajnění krok 1

V druhém kroku (obrázek B.5) se nachází volitelné dodatečné zabezpečení dat, která bylo zadáno při skrývání. Pokud jej nenastavíme stejně, nedojde ke správnému odhalení dat. Informace o kontrolním součtu, jehož volba byla možná při skrývání je nesena přímo v obrázku a je automaticky ověřena.




Obrázek B.5: Odtajnění krok 2

Posledním krokem (obrázek B.6) je nastavení steganografie. Zde zvolíme vstupní formát souboru a k němu příslušnou metodu, která byla použita při skrývání. Pokud to zvolená metoda podporuje, můžeme zvolit náhodné rozložení dat po celém obrázku a zvolit si heslo na základě kterého bylo náhodné rozdělení počítáno. Posledním krokem pro odkrytí steganogramu je zmáčknout tlačítko ve spodní části okna a zvolit složku, kde bude umístěn soubor s daty.




Obrázek B.6: Odtajnění krok 3

## B.3 CLI

Spuštění této verze programu se provádí pomocí příkazové řádky. Pokud je zadán přepínač -h nebo --help je vypsána nápověda. V příkazové řádce je spuštění následující:

```
java -jar xStegoC.jar <režim> <volby>
<režim> ::= hide | uncover | capacity
<volby> ::= (kombinace voleb dostupných pro daný režim uvedené níže)
```

Podporované názvy formáty a metod, které lze aplikovat:

- JPEG - JSTEG 1b, JSTEG 2b, JSTEG 3b, JSTEG 4b, F3, F4, F5
- PNG - LSB 1b, LSB 2b, LSB 3b, LSB 4b, PVD, Inverted LSB, TripleA 2b, TripleA 3b, TripleA 4b, EMD 2, EMD 4, EMD 8
- BMP - LSB 1b, LSB 2b, LSB 3b, LSB 4b, PVD, Inverted LSB, TripleA 2b, TripleA 3b, TripleA 4b, EMD 2, EMD 4, EMD 8

### Režim "hide"

Tento režim slouží pro skrytí dat do obrázku a má následující volby:

- -c,--CRC povoluje CRC
- -d,--data <arg> cesta k tajný datům (povinný)
- -f,--format <arg> výstupní formát souboru (povinný)
- -i,--input <arg> cesta k vstupnímu obrázku (povinný)
- -k,--key <arg> vlastní heslo pro náhodnost
- -m,--method <arg> steganografická metoda dostupná pro zvolený výstupní formát (povinný)
- -o,--output <arg> cesta do složky kde se má uložit výsledný obrázek (povinná)
- -q,--quality <arg> tato možnost je dostupná pouze pro výstupní formát JPEG. Využívá se k nastavení kvality výstupního obrázku, tedy mírky komprese. Kvalita je zadána jako procentuální hodnota číselně v rozsahu <1,100>
- -r,--random povoluje náhodnost
- -e,--encrypt povoluje šifrování
- -p,--password <arg> vlastní heslo pro šifrování

### Režim "uncover"

Tento režim slouží k získání skrytých dat z obrázku a má následující volby:

- -f,--format <arg> výstupní formát souboru (povinný)
- -i,--input <arg> cesta k vstupnímu obrázku (povinný)
- -k,--key <arg> vlastní heslo pro náhodnost
- -m,--method <arg> steganografická metoda dostupná pro zvolený výstupní formát (povinný)
- -n,--name <arg> název pro odkrytý soubor
- -o,--output <arg> cesta do složky, kde se má uložit výsledný obrázek (povinná)
- -r,--random povoluje náhodnost
- -e,--encrypt povoluje šifrování
- -p,--password <arg> vlastní heslo pro šifrování

### Režim "capacity"

Tento režim slouží k získání dostupné kapacity obrázku. Ve výchozím stavu vrací jak maximální, tak minimální kapacitu. Tento režim má následující volby:

- -f,--format <arg> výstupní formát souboru (povinný)
- -i,--input <arg> cesta k vstupnímu obrázku (povinný)
- -k,--key <arg> vlastní heslo pro náhodnost
- -m,--method <arg> steganografická metoda dostupná pro zvolený výstupní formát (povinný)
- -q,--quality <arg> tato možnost je dostupná pouze pro výstupní formát JPEG, využívá se k nastavení kvality výstupního obrázku, tedy mírky komprese, kvalita je zadána jako procentuální hodnota číselně v rozsahu <1,100>
- -r,--random povoluje náhodnost
- -l,--low vrátí pouze minimální kapacitu (kapacita v nejhorším případě)
- -b,--big vrátí pouze maximální kapacitu (kapacita v ideálním případě)