FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Utajená komunikace použitím steganografie
Student:	Ondřej Voronecký
Vedoucí:	Ing. Michal Štepanovský, Ph.D.
Studijní program:	Informatika
Studijní obor:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Steganografie je podobor kryptografie, který se zabývá ukrytím tajné zprávy tak, aby si vnější pozorovatel vůbec neuvědomil, že probíhá utajená komunikace. Cílem této práce je vytvořit SW nástroj/aplikaci, která bude schopna ukrýt tajnou zprávu do obrázku (nejlépe ve formátu png) a zároveň tuto zprávu z obrázku extrahovat. Doplňkovým cílem práce je zabezpečit zprávu pomocí samoopravných kódu, které umožní zprávu extrahovat i po mírné editaci obrázku s ukrytou zprávou.

1. Vypracujte rešerši pojednávající o steganografii a o samoopravných kódech.
2. Popište postup ukrytí zprávy do obrázku a vypracujte návrh SW aplikace včetně definice GUI.
3. Implementujte SW aplikaci dle předchozího návrhu.
4. Analyzujte možnosti samoopravných kódů ve vztahu k rozsahu a charakteru poškození obrázku (například poškození souvislé oblasti, poškození náhodně vybraných pixelů, apod.).
5. Ověřte funkčnost SW aplikace.

Jednotlivé body a náplň bakalářské práce konzultujte s vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Pavel Tvrdík, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 29. ledna 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Utajená komunikace použitím steganografie

Ondřej Voronecký

Katedra počítačových systémů

Vedoucí práce: Ing. Michal Štepanovský, Ph.D.

3. června 2020

Poděkování

Chtěl bych velmi poděkovat vedoucímu Ing. Michalu Štepanovskému, Ph.D. za jeho rady, připomínky a vedení. Dále bych chtěl poděkovat své rodině za poskytnutí zázemí pro psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 3. června 2020 ……

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

Voronecký, Ondřej. *Utajená komunikace použitím steganografie*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato práce se zabývá steganografií, konkrétně ukrytím libovolných dat do poskytnutého obrázku, při použití samoopravných kódů. V práci jsou analyzovány způsoby ukrytí dat do obrázku, různé druhy samoopravných kódů a jejich možnosti ve vztahu k rozsahu a typu poškození obrázku. Na základě toho je vytvořena aplikace, včetně uživatelského rozhraní, která ukryje poskytnutá data do zvoleného obrázku za pomoci vybraného samoopravného kódu a metody vkládání. Pomocí aplikace jsou samoopravné kódy a metody vkládání prakticky analyzovány. Aplikace je vytvořena a testována pro operační systém GNU/Linux.

Klíčová slova utajená komunikace, obrazová steganografie, samoopravné kódy, odolnost kódů, odolnost metod vkládání, aplikace pro ukrytí tajné zprávy

Abstract

This thesis focuses on the steganography, specificaly hiding any data into the provided image, using error-correcting codes. The thesis analyzes the methods of hiding data, various types of error-correcting codes and their capabilities to extract the hidden data from the damaged (or modified) image. Based on this, an application for hidding the given data into the image is created, including a user interface. This application allows to select the error-correcting code and embedding method to be used. With the help of created application, the error-correcting codes and embedding methods are practicaly analysed. The application was created and tested for the GNU/Linux operating system.

Keywords secret communication, image steganography, error-correcting codes, code resilience, embedding methods resilience, application for hiding secret data

Obsah

Seznam obrázků

Seznam tabulek

Úvod

První zmínky o steganografii, stejně tak jako o kryptografii, sahají až do dob před naším letopočtem. Jejich markantní rozvoj však nastal až s příchodem počítačů a potřebou data utajit. Steganografie byla poprvé použita panovníkem Histiaeusem ve starověkém Řecku. Používal ji pro komunikaci se svým synovcem, se kterým plánoval vzpouru. Zpráva byla vytetována na oholené hlavy otroků, které se poté nechaly opět zarůst. Otroci byli posláni k synovci, který si mohl zprávu přečíst. Kryptografie byla poprvé cíleně použita Juliusem Caesarem, dnes známá jako Caesarova šifra. Ta byla realizována substitucí písmene za písmeno, které se v abecedě nachází o pevně určený počet pozic před, nebo po. [1]

V dnešní době, kdy jsou v podstatě všechny informace v digitální formě, je více než vhodné data nějakým způsobem utajit a tím zabránit jejich zneužití. Jak je z odstavce výše patrné, tak i když je funkcí steganografie i kryptografie zabezpečit komunikaci, zaujímají odlišné přístupy k provedení. Funkcí kryptografie je data zašifrovat, čímž se stanou čitelná pouze pro subjekty, které mají klíč. V případě Caesarovy šifry se jedná o hodnotu posunu. Steganografie se naopak snaží utajit komunikaci tak, že nezúčastněné strany ani neví, že nějaká komunikace probíhá. Oba přístupy se dají zkombinovat, tím využít výhod obou přístupů a potlačit jejich slabiny.

My se budeme v této práci zabývat ukrytím zvolených dat do obrázku, za použití steganografie a různých samoopravných kódů, které nám ukrytá data ochrání před jejich poškozením.

Nejprve se seznámíme se steganografií a její aplikací na digitální data, s důrazem na různé přístupy a způsoby ukrytí dat do obrázku. Dále se zaměříme na efektivitu těchto metod, vzhledem k velikosti ukládaných dat a velikosti obrázku, a poté vzhledem k míře jejich utajení. Následuje kapitola pojed-

návající o samoopravných kódech, které budou při ukládání dat použity. Je vhodné tyto kódy aplikovat hlavně z toho důvodu, že obrázky mohou být různě upravovány či poškozeny (zmenšení, vyříznutí určité části či částí, komprese, náhodně poškozené pixely a jiné). V některých případech nám mohou samoopravné kódy pomoci, alespoň částečně, data obnovit. Analyzujeme jejich principy, možnosti a efektivitu. V další kapitole vypracujeme návrh GUI aplikace, která bude výše popsané způsoby ukrytí a samoopravné kódy realizovat. Na základě tohoto návrhu aplikaci následně implementujeme.

Pomocí aplikace otestujeme předpokládané možnosti samoopravných kódu, vzhledem k rozsahu a formě poškození/úpravy obrázku a výsledky zhodnotíme.

Kapitola 1

Cíl práce

Cílem rešeršní části práce je popis steganografie a její aplikace na různá média, s důrazem na její použití při ukrývání dat do obrázku. Stručně se seznámíme s její aplikací na texty, aby si čtenář dokázal lépe představit její principy. Následuje popis samoopravných kódů, jejich principů, analýza jejich možností vůči poškození obrázku a efektivita. V rámci této části bude také vypracován návrh GUI aplikace, která bude tyto způsoby ukrytí a samoopravné kódy implementovat.

Cílem praktické části je implementace GUI aplikace v jazyce Python, dle předchozího návrhu a její otestování. Aplikace umožňuje uložit data do poskytnutého obrázku s využitím zvolené metody a samoopravného kódu. Pomocí aplikace otestujeme schopnosti samoopravných kódů a metod vkládání na reálných datech a výsledky zhodnotíme.

Kapitola 2

Steganografie

Tato kapitola se bude zabývat teoretickými základy vědní disciplíny steganografie. Nejdříve budou probrány potřebné pojmy, definice a možné aplikace steganografie na digitální data. Nejdříve si stručně ukážeme aplikaci na texty a poté se budeme především zaměřovat na možné využití při ukrývání dat do obrázku. Probereme různé způsoby, jak toto ukládání realizovat a vyhodnotíme dle 2.3.

2.1 The prisoners' problem

Jak již bylo řečeno, tak důvodem využití steganografie je potřeba ukrytí tajné zprávy takovým způsobem, aby objekty neúčastnící se komunikace nevěděly, že komunikace vůbec probíhá. Tento problém, s názvem *The prisoners' problem*, poprvé demonstroval a formuloval G. J. Simmons. [2] Na tomto příkladu demonstruje, jak mohou objekty tajně komunikovat přes nezabezpečený komunikační kanál. Tento problém je považován za základ celého vědního oboru.

V problému vystupují celkově tři subjekty. Dva z nich jsou pachatelé trestného činu, kteří budou umístěni do vězení. Každý bude v samostatné oddělené cele, bez možnosti přímé komunikace mezi sebou. Jejich jedinou možností pro výměnu informací je posílání zpráv přes dozorce, který je třetím účastníkem komunikace (nezabezpečený kanál). Ten si je může buď jen přečíst, nebo je může upravit takovým způsobem, aby zachoval jejich původní význam. Úkolem vězňů je naplánovat útěk přes nezabezpečený komunikační kanál tak, aby si toho dozorce vůbec nevšiml.

Autor dále popisuje, že nejlepším řešením tohoto problému je ukrytí zpráv do textu či obrázku, který bude navenek vypadat zcela nevinně. Pokud dozorce zjistí, že mezi sebou vězni tajně komunikují, tak se stegosystém 2.2 bere jako nefunkční. To je zásadní rozdíl od kryptosystému 2.3, kdy je potřeba data

částečně nebo zcela dešifrovat. Zároveň příklad poukazuje na problém, kdy dojde k úpravě zprávy dozorcem nebo jejímu podvržení.

Některé principy publikované v této práci se všeobecně ujaly a jsou odborníky na steganografii často využívány pro testování stegosystémů. Při tomto testování se mění vlastnosti dozorce. Buď jedná pasivně, kdy zprávy nijak nemodifikuje a může si je jen prohlížet. Dále může zprávy naschvál modifikovat, za účelem zničení tajné zprávy, a nebo vytvářet vlastní zprávy a tím se vydávat za jednoho z pachatelů. [3] Při vyhodnocování našich stegosystémů budeme uvažovat pasivního, resp. aktivního dozorce, který zprávy modifikuje a nevytváří vlastní.

2.2 Základní pojmy

Definice 2.1 (Steganografie) *Vědní obor zabývající se ukrytím tajných dat do běžného veřejného souboru, dat, textu či média takovým způsobem, aby tajná data nebyla neinformovanými objekty detekována.* [4]

Definice 2.2 (Stegosystém) *Konkrétní algoritmus popisující vložení a následnou extrakci dat.*

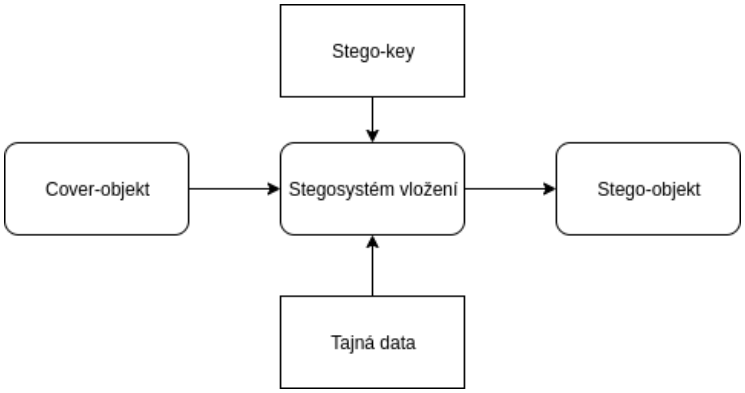
Definice 2.3 (Kryptosystém) *Konkrétní algoritmus popisující zašifrování a dešifrování dat.*

Definice 2.4 (Stegoanalýza) *„Vědní obor zabývající se metodami detekce přítomnosti steganograficky ukrytých informací, jejichž cílem není dekódování obsahu.“* [5]

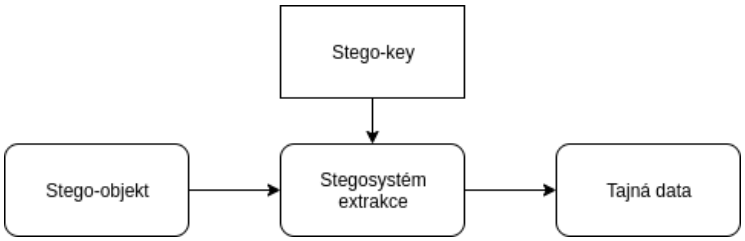
Dle definice 2.1 se tedy steganografie dá aplikovat v různých formách. Její nejvíce běžné využití v digitálním světě je na text, audio, video či obrázek. Důvodů pro výběr těchto nosičů je mnoho. Především je to jejich množství a jejich široké užití mezi uživateli.

Na schématech 2.1 a 2.2 vidíme, jak obecně funguje stegosystém pro vkládání a extrakci dat.

Cover-objekt znázorňuje jakýkoli objekt (obrázek, text, …), do kterého se rozhodneme data vložit. Následně se může zvolit stego-key, což je sdílené tajemství mezi komunikujícími stranami, které zavádí určitou obfuskaci dat při vkládání do cover-objektu. Bez stego-key nejsme schopni zprávu ze stego-objektu



Obrázek 2.1: Vložení dat do cover-objektu



Obrázek 2.2: Extrakce dat ze stego-objektu

správně extrahovat. Po provedení vznikne nový objekt s názvem stego-objekt, který obsahuje tajná data.

Extrakce ze stego-objektu probíhá obdobně. Pokud známe stego-key, můžeme provést extrakci a získat tajná data.

V této práci budou cover-objekty a stego-objekty výhradně digitální přenosová média. Dá se na ně aplikovat vícero metodik a v dnešní době je většina přenosových médií právě v digitální formě.

2.3 Požadavky na stegosystém

Jak uvádí autoři, tak stegosystém se dá hodnotit na základně čtyř základních kritérií, pomocí kterých určíme jeho kvalitu. [6] Díky tomu můžeme jednotlivé stegosystémy mezi sebou porovnávat a určit výhody a nevýhody každého z nich.

Bezpečnost nám říká, jak moc je obtížné v stego-objektu odhalit ukrytá data. Jedná se o základní a nejdůležitější požadavek na jakýkoli stegosystém a měli bychom se snažit na toto kritérium co nejvíce apelovat. Stegosystém, který se snaží o nedetekovatelnost člověkem, může být stále náchylný na statis-

tické analýzy. [7] Dobrý stegosystém by se měl snažit minimalizovat oba tyto faktory, aby dosáhl co nejlepšího ukrytí dat. Pro jeho hodnocení se porovná cover-objekt oproti stego-objektu a na základě toho se rozhodne o jeho kvalitě.

Odolnost je dalším důležitým kritériem. Určuje, jak moc je pro útočníka obtížné skrytá data extrahovat, odstranit či poškodit. K těmto změnám může dojít záměrně (použití specializovaného SW), nebo zcela nevědomě (změna formátování textu, rotace obrázku, změna intenzity barev obrázku, …). Většina stegosystémů neposkytuje dostatečnou odolnost, a proto bude vhodné tuto slabinu minimalizovat právě samoopranými kódy.

Kapacita nám určuje množství dat, která můžeme do cover-objetu dané velikosti vložit, abychom stále dodrželi požadovanou bezpečnost. Mezi bezpečností a kapacitou platí nepřímá úměra, čím vyšší kapacita, tím nižší bude bezpečnost a naopak. [6] Při našem hodnocení budeme kapacitu porovnávat na základě využitých bitů (ukrývající tajná data) ku všem bitům obrázku.

Spolehlivost je posledním a neméně důležitým kritériem. Dobrý stegosystém by měl být schopen při použití správného stego-klíče data ze stego-objektu v pořádku extrahovat, pokud možno s nulovou ztrátou informací. Tento požadavek by se dal svou důležitostí zařadit vedle bezpečnosti. Pokud bude stegosystém splňovat všechna kritéria uvedena výše, ale nebude spolehlivý, pak jako takový postrádá smysl.

2.4 Textová steganografie

Textová steganografie využívá pro ukrytí dat, jak už název napovídá, text jako médium. Výsledný text obsahující tajná data se nazývá *stego-text* a text vybraný pro ukrytí dat se nazývá *cover-text*. Buď se data ukryjí již do vytvořeného textu (cover-textu), nebo se z dat vygeneruje stego-text, který tato data obsahuje. V neposlední řadě je možné vybrat cover-text takový, který již naši tajnou zprávu obsahuje a je zároveň stego-textem. Tento proces se dá rozdělit na tři kategorie, jak je vidět na obrázku 2.3.

2.4.1 Metody využívající formátování

Tento způsob ukrytí dat do textu využívá cover-text, který upravuje pomocí změny formátování. Buď změnou a přizpůsobením interpunkce, kdy je například bitová hodnota 0 zakódovaná jako tečka za větou a bitová hodnota 1 jako čárka, nebo úpravou bílých a netisknutelných znaků v dokumentu. Dalšími možnostmi je záměrné vytváření pravopisných chyb a úprava fontu textu. Z pravidla volíme takové fonty, které jsou si co nejvíce podobné. Vše závisí na tom, jak se rozhodneme tyto úpravy kódovat a interpretovat.

Obrázek 2.3: Kategorie textové steganografie [8]

Zajímavý zástupce, s názvem *Line Spacing*, využívá mezer mezi řádky. Velikost těchto mezer změní o několik jednotek pixelů a tím se dají do cover-textu zakódovat jednotlivé bity dat. Toho se dá využít k ochraně dokumentu, kdy do něj pomocí této metody zakomponujeme vodoznak a přitom nenarušíme čtenářský komfort.

Tyto metody jsou efektivní z hlediska utajení hlavně z toho důvodu, že si jich čtenář při běžném čtení vůbec nevšimne, nebo ho nenapadne jejich vyšší význam (v případě pravopisných chyb). Z pohledu poměru mezi množstvím ukrytých dat a velikostí textu na tom tyto metody nejsou moc dobře, a proto se většinou využívá jejich vzájemná kombinace, díky čemuž se kapacita navýší.

2.4.2 Metody náhodného a statistického generování

Druhá kategorie volí naprosto odlišný přístup. Místo úpravy daného cover-textu generuje stego-text. Ten se snaží co nejvíce přiblížit vlastnostem reálnému textu pomocí statistických a pravděpodobnostních metod. Stego-text se dá generovat po jednotlivých znacích, bigramech, trigramech, slovech, atd. Často jsou pro generování použity tzv. *Markovovy řetězce*. Můžeme si například zvolit stegosystém, kdy budou data uložena jako druhé písmeno v každém slově. Na základě tohoto pravidla vygenerujeme stego-text.

Jelikož jsou metody založené na statistice a náhodě, tak výsledný text může vzbudit značné podezření. Věty výsledného textu totiž mohou být gramaticky a syntakticky správně, ale dost často nedávají žádný smysl.

2.4.3 Lingvistické metody

Třetí kategorie může využívat jak modifikaci existujícího cover-textu, tak generování stego-textu. Výhodou je ochrana proti změnám formátování textu při použití různých formátovacích programů. [8] Zároveň také oproti předchozí metodě nevzbuzuje na první pohled tak velké podezření, právě kvůli důrazu na lingvistiku, a ne jen na statistiku.

Příkladem je *Sémantická metoda*, která je založena na modifikaci cover-textu nahrazením slov synonymy. [9] Pak už záleží na zvoleném stegosystému, jak data do synonym ukryjeme. Nevýhodou této metody je možnost vytvoření podezřelého až nesmyslného textu:

Odnesl jsem kočku na veterinu. Odnesl jsem dívku na veterinu.

Jak je výše vidět, tak synonymum pro slovo *kočka* je i označení pro hezkou dívku a věta je pak spíše k pobavení, než k ukrytí dat.

2.5 Obrazová steganografie

Jedná se o poslední a nejvíce prozkoumaný cover-objekt. Vzhledem jejich rozšíření a redundanci (různé obrázky stejných objektů) se stávají vhodným kandidátem pro steganografii. Je to způsobeno i tím, že na rozdíl od textové steganografie nenarážíme na problémy typu text nedává smysl, špatná syntax a jiné. Můžeme se však setkat s jinými problémy (zhoršení kvality obrázku, viditelné změny, rozdíly založené na statistice, …).

Metod jak ukrýt data do cover-image je mnoho. Nejprve probereme ty nejjednodušší a nejzákladnější a dopracujeme se až k zajímavějším metodám, které většinou využívají jednoduché metody jako základ. Poté u každé vyhodnotíme základní požadavky na stegosystém a určíme, jak dobře je splňuje.

Problémem většiny obrazových stegosystémů je ten, že z pravidla nepočítají s úskalími jako komprese (formát jpeg), rotace, mírné editace či náhodné poškození pixelu. Dále také ukládají tajná data na předem předdefinovaná místa a tím zjednodušují jejich nalezení útočníkem. Některé z nich se pokusíme vyřešit právě vylepšenými metodami.

2.5.1 Přehled formátů

„*Digitální obrázky se dají rozdělit na čtyři základní formáty – rastrové, paletové, transformované a vektorové.*“ [3] Stručně probereme a shrneme první tři, jelikož mohou být, na rozdíl od vektorových, vhodné jako cover-image. Vektorové formáty obrázek popisují jako soubor geometrických tvarů, kde má každý svůj předpis a následně několik parametrů (šířka, barva, …). Nabízejí tedy velmi malý prostor pro vložení tajných dat a i sebemenší změna by mohla mít velký vliv na celkový výsledný obraz.

Rastrové a *paletové* formáty spadají do *spatial-domain* formátu. Spatial-domain obrázky jsou uloženy jako matice, která popisuje barvy individuálních pixelů. Jedná se tedy o bezeztrátové uložení dat. Rastrový formát obsahuje hlavičku,

kde jsou uloženy základní informace o velikosti, barevnosti a za hlavičkou následuje matice pixelů. Pixely matice jsou popsány jednotlivě, tedy pro každou barvu či odstín máme pevně daný počet bytů, kterými lze popsat. Tento způsob ukládání je paměťově nejnáročnější, což je vhodné pro steganografii z hlediska kapacity. Nejběžnějšími zástupci tohoto typu jsou formáty PNG, BMP a TIFF. Odlišný způsob zaujímají paletové obrázky, které na začátku obsahují kromě hlavičky i paletu. Paleta nám definuje rozsah barev, kterých mohou jednotlivé pixely nabývat. Hodnoty pixelů jsou popsány jako index do palety, která obsahuje nejvýše 256 hodnot, tedy různých barev. Tento model je výhodný hlavně z toho důvodu, že se značně zredukuje velikost obrázku. Vzhledem k tomu, že je velikost palety nejvýše 256, pak nám pro indexaci barev stačí jediný byte pro každý pixel. Zástupcem jsou například formáty GIF a PNG.

Transformované formáty spadají do *transform-domain* formátu. Na rozdíl od spatial-domain formátů se jedná o ztrátové uložení dat. Data obrázku projdou sadou transformací, které sice zredukují jeho výslednou velikost, ale také ztratíme některé informace. Dle [3] tuto změnu lidské oko nepozná, ale pro uložená tajná data to může mít destruktivní následky.

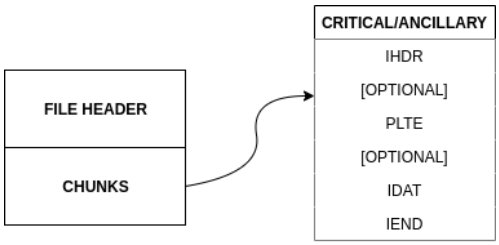
Pro znázornění barev jakéhokoli formátu se používá základní model RGB (red, green, blue) pro barevné obrázky, jednobytová hodnota pro stupně šedi a jednobitová pro černo-bílé obrázky. Složky v RGB modelu mohou mít různou přesnost (počet bitů, které je reprezentují), což se označuje jako bitová hloubka. Na základě toho se určuje, kolik různých barev můžeme vytvořit, jelikož podle intenzity jednotlivých složek modelu se vytvoří jakákoli jiná barva z daného rozsahu.

Pro naše potřeby budeme využívat formáty rastrové. Vzhledem k tomu, že u nich dochází pouze k bezeztrátové kompresi a každý pixel je reprezentován až několika byty dat, nabízejí veliký prostor pro vložení tajné zprávy. Paletové formáty by se daly také využít, ale jen pro ukrytí velmi krátkých zpráv. To je velmi nevýhodné v kombinaci se samoopravnými kódy, díky kterým se tajná zpráva zvětší a paletové formáty by se staly zcela nevhodnými. Vhodným formátem by mohly být i transformované formáty, my se však v této práci soustředíme spíše na jiné typy poškození obrázku, než na ty způsobené ztrátovou kompresí.

2.5.2 Formát PNG

Pro naše použití zvolíme formát PNG, který spadá do spatial-domain obrázků a může být jak rastrový, tak paletový. Jak bylo zmíněno výše, my budeme pracovat s rastrovým formátem. Dalším důvodem pro vybrání tohoto formátu byla jeho popularita mezi uživateli a rozšířenost.

Záladní strukturu formátu souboru si můžeme prohlédnout na schématu 2.4, které zachycuje jeho základní rozdělení na dvě části.



Obrázek 2.4: Struktura PNG formátu

Na začátku každého PNG souboru se nachází file header. Jedná se o jasně definovanou posloupnost 8 byte, která je v každém souboru stejná. Slouží pro kontrolu chyb při přenosu po síti. První byte testuje podporu 8-bitového přenosu a zároveň je používán, aby nedošlo k záměně za textový soubor. Následující tři byty určují formát, jelikož v ASCII kódování reprezentují string „PNG“. Pátý a šestý byte slouží k detekci nevhodného přenosu souboru, který zaměňuje CR-LF konce řádků, používané na platformě Windows. Sedmý byte slouží k zastavení výpisu souboru na platformě DOS. Poslední byte má podobnou funkci jako sekvence CR-LF, akorát pro platformy Unix, která používá jen LF. V hexadecimální soustavě má následující podobu:

89 50 4E 47 0D 0A 1A 0A

Následuje část souboru, které se říká chunks. Typy chunků se dělí na critical (povinné) a ancillary (volitelné). Každý povinný chunk má přesně definovanou strukturu, která se může lišit od ostatních povinných chunků. Naopak všechny volitelné chunky musí splňovat jednotnou strukturu, která se dělí na čtyři části. První o velikosti 4 bytů nám říká samotnou velikost dat uložených v chunku. V dalších 4 bytech je uloženo jméno chunku v ASCII kódování. Následují samotná data, která jsou zakončena 4 byty kontrolního součtu CRC. [10] Povinné chunky musí respektovat předem definované pořadí, které je na 2.4 znázorněné. Na určitá místa mezi těmito chunky, označená nápisem [OPTIONAL], se dají vložit chunky volitelné. Některé z nich jsou definované standardem, ale nic nám nebrání ve vytvoření vlastních. Pro naše potřeby však nebudou třeba a nebudeme s nimi pracovat.

IHDR je povinný chunk obsahující hlavičku obrázku, která definuje jeho základní parametry 2.1. Width a height nám určuje šířku a výšku obrázku v pixelech. Bit depth nám říká počet bitů, kterými je reprezentována složka pixelu. V případě barevných obrázků, kde je pixel tvořen složkami RGB, nám bit depth určuje počet bitů každé složky RGB. Čím je tato hodnota větší,

tím více barev může pixel nabývat. Color type určuje způsob interpretace dat obrázku. To znamená, jestli jsou interpretována jako indexy do palety, jestli je využit alpha channel a jiné. Povolené kombinace color type a bit depth jsou uvedeny v tabulce 2.2, včetně jejich významu. Alpha channel určuje průhlednost pixelu, za kterým je připojen. Compression method nám říká, která metoda byla použita pro bezeztrátovou kompresi dat obrázku. Aktuálně se využívá jen komprese Deflate/Inflate, pro kterou je rezervována hodnota 0. Filter method určuje, jakým způsobem byla data před kompresí předzpracována. Stejně jako u předešlého parametru definujeme opět jen jeden typ filtru s rezervovanou hodnotou 0. Poslední parametr interlance method definuje, jakým způsobem jsou data pixelů uložena. Momentálně může nabývat dvou hodnot – 0 (ukládání po řádcích) nebo 1 pro uložení algoritmem Adam7.

Hodnota	**Velikost (byte)**
Width	4
Height	4
Bit Depth	1
Color type	1
Compression method	1
Filter method	1
Interlance method	1

Tabulka 2.1: Struktura IHDR chunku

Color type	**Bit depth**	**Interpretace**
0	1, 2, 4, 8, 16	Pixel v odstínech šedi
2	8, 16	Pixel v RGB
3	1, 2, 4, 8	Pixel je index do palety
4	8, 16	Pixel v odstínech šedi, za kterým následuje alpha channel
6	8, 16	Pixel v RGB, za kterým následuje alpha channel

Tabulka 2.2: Povolené kombinace Color type a Bit depth

PLTE chunk obsahuje posloupnost trojic bytů a každý byte trojice je jedna barevná složka modelu RGB. Počet trojic v chunku může být od 1 až do 256 a jsou indexovány hodnotami z IDAT. Jedná se o povinný chunk pro color type 3 a o volitelný pro color type 2 a 6. Vzhledem k maximálnímu počtu trojic (256), mohou mít obrázky využívající PLTE nejvýše 8 bitovou hloubku. Pokud bychom tedy měli obrázek uložený bez palety s bitovou hloubkou větší než 8, pak dojde při uložení s paletou ke ztrátové kompresi.

IDAT jsou samotná data obrázku, která prošla filtrem a jsou zkomprimovaná algoritmem Deflate/Inflate. Na rozdíl od ostatních povinných chunků se v PNG souboru může vyskytovat vícero IDAT chunků, které však prezentují právě jeden obrázek. Pro práci s těmito daty musíme provést opačný proces, než při ukládání. Tedy nejdříve data dekomprimujeme a poté na ně aplikujeme filtr. V případě vícero chunků se proces aplikuje na každý zvlášť a výsledná data dostaneme jejich spojením. Mezi jednotlivými IDAT chunky se nesmí nacházet žádné nepovinné chunky.

IEND označuje konec souboru. Za tímto chunkem již nesmí být žádná data a pokud by byla, tak budou ignorována.

Pro naše potřeby budeme tedy využívat chunky IDAT, do kterých budeme vkládat tajná data.

2.5.3 Metoda LSB

Jedná se o nejjednodušší a nejvíce názornou metodu, jak ukrýt data do cover-image. Označme bitovou hloubku pixelu b_d a bez újmy na obecnosti předpokládejme obrázek ve stupních šedi s $b_d = 8$. Každému pixelu $p[i]$ pak odpovídá hodnota $x[i] \in A = \{0, ..., 2^{b_d} - 1\}$ pro $i \in \{1, ..., n\}$ kde n je počet pixelů. Jednotlivé bity v pixelu mohou být indexovány jako $b[i, j]$, kde i je index pixelu a j je index bitu v $p[i]$. V případě ukládání bytů formou big-endian bude bit $b[i, b_d - 1]$ ten, který nahradíme bitem zprávy $m[j]$. [3] Pro tento bit budeme využívat ekvivalentní zápis $b[i, b_d - 1] = LSB(p[i])$. V případě barevných obrázků se $p[i]$ skládá z trojice $x_R[i]$, $x_G[i]$ a $x_B[i]$. To nám umožní pro ukládání m využít každou složku z trojice a zvýšit tím kapacitu.

Algoritmus 1 znázorňuje ukrytí zprávy m do obrázku. V případě zadání stego-key se vygeneruje pseudo-náhodný průchod pixely (či jejich složkami), aby se jednotlivé bity zprávy m rovnoměrně pseudonáhodně rozmístili po celém obrázku. Pokud se stego-key rovná 0, pak je průchod lineární po řádcích z levého horního rohu do pravého dolního. Je vhodné stego-key poskytnout, jelikož se jedná o nejznámější metodu a pro útočníka je pak jednoduché data extrahovat. Dále také zajistí rozptýlení dat po celém obrázku a tím se vložená data stanou méně nápadná. Tajná zpráva m je z obrázku extrahována algoritmem 2.

Při vyhodnocování, jak LSB splňuje požadavky na stegosystém, budeme předpokládat využití celkové kapacity cover-image, která je k dispozici. Zároveň budeme uvažovat situaci, kdy byl při ukládání využit stego-key.

Vzhledem k tomu, že se jedná o změny v rozsahu jednoho bitu na jednu složku pixelu, a to ještě pravděpodobností $\frac{1}{2}$, jsou tyto změny lidským okem velmi těžko detekovatelné. Pořízené fotografie, které nebyly posléze retušovány, obsahují šum způsobený různými podmínkami při pořizování. [11] Díky tomu se

Algoritmus 1: Vložení zprávy m do obrázku [3]

Input: Zpráva m, stego-key k a obrázek X
Output: Stego-obrázek Y
`/* Inicializace PRNG klíčem` k`. Pokud se` $k = 0$`, tak se použije defaultní průchod. */`
`/* Vygenerování cesty Path o délce n = počet pixelů. */`
1 Path = Perm(n)
`/* Oříznutí zprávy` m`, pokud se nevejde do obrázku */`
2 m = min(m,n)
3 **for** $i \leftarrow 0$ **to** $m - 1$ **do**
4 ⌊ Y[Path[i]] = X[Path[i]] + m[i] - LSB(X[Path[i]])

Algoritmus 2: Extrakce zprávy m z obrázku [3]

Input: Stego-key k a obrázek Y
Output: Zpráva m
`/* Inicializace PRNG klíčem` k`. Pokud se` $k = 0$`, tak se použije defaultní průchod. */`
`/* Vygenerování cesty Path o délce n = počet pixelů. */`
1 Path = Perm(n)
2 **for** $i \leftarrow 0$ **to** $m - 1$ **do**
3 ⌊ m[i] = LSB(Y[Path[i]])

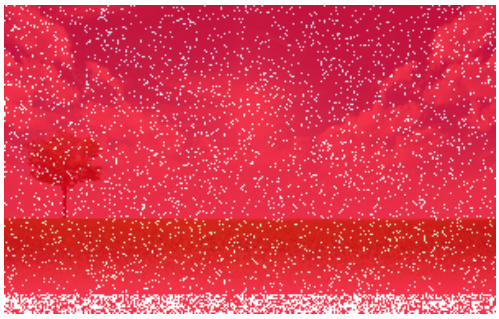
uspořádaná n-tice $(b[0, b_d - 1], ..., b[n - 1, b_d - 1])$ tváří jako zcela náhodná. Pokud tedy dodržíme výše uvedené předpoklady pro ukládání bitů zprávy m do náhodně vybraných pixelů $p[i]$, pak by měla mít tato metoda vysokou bezpečnost. To však není pravda a tato metoda může být odhalena za pomoci histogramu, párové analýzy či jiné statistické metody. [3]

Na obrázku 2.5 můžeme vidět rozdíl obrázku před a po vložení tajné zprávy. Pokud se jedná o souvislou plochu jedné barvy bez šumu, pak jsou patrné barevnostní rozdíly pixelů i přesto, že se jejich hodnota změnila o nejvýše jeden bit na složku RGB. Vzhledem k tomu, že jedna složka pixelu odpovídá jednomu bitu zprávy, jsou změny dost časté a o to více patrné. Na následujícím obrázku 2.6 jsou změny vidět o dost obtížněji, až vůbec, jelikož se nejedná o souvislou plochu jedné barvy. Modifikované pixely jsou znázorněny červenou barvou na obrázku 2.6c. Na obrázku 2.7 si můžeme prohlédnout, jak se s narůstajícím počtem náhodně poškozených bitů zhoršuje celková kvalita fotografie. Toto chování můžeme nejlépe pozorovat na černých hodinkách, na kterých je při poškození tří posledních bitů vidět jasný šum. Pro zřetelnější rozdíly doporučuji digitální verzi této práce.

(a) Cover-image (100x98) (b) Stego-image (100x98)

Obrázek 2.5: Porovnání obrázků před a po vložení zprávy [12]

(a) Cover-image (b) Stego-image



(c) Diference

Obrázek 2.6: Diference obrázků (300 x 188) před a po uložení zprávy [13]

(a) Jeden poškozený bit složky RGB

(b) Dva poškozené bity složky RGB

(c) Tři poškozené bity složky RGB

Obrázek 2.7: Porovnání obrázků s různým počtem poškozených bitů v rámci jednotlivých složek RGB

LSB postrádá jakoukoli odolnost, jelikož i sebemenší úpravy jakéhokoli pixelu mohou způsobit trvalé poškození segmentu tajných dat. Při specifických úpravách, jako je změna intenzity barev celého obrázku, mohou být ztracena všechna data. To je způsobeno také tím, že jsou tajná data ukládána do bitu, který svou hodnotu mění nejčastěji.

Kapacita metody LSB se řadí mezi nejvyšší. Jestliže p je součet všech složek pixelů, pak největší možná délka zprávy m v bitech je rovna právě p. Poměr využitých bitů ku všem bitům obrázku je roven $\frac{p}{p*b_d} = \frac{1}{b_d}$.

Metoda je spolehlivá, pokud nedojde k editaci obrázku a využije se správný stego-key. V tom případě je možné extrahovat vložená data s nulovou ztrátou.

2.5.4 Metoda ±1 embedding

Jedná se o obdobu metody LSB s tím rozdílem, že se při ukládání nemodifikuje pouze bit $b[i, b_d - 1]$. Opět budeme předpokládat obraz ve stupních šedi s $b_d = 8$. Pokud platí, že $b[i, b_d - 1] = m[i]$, pak se LSB pixelu nemodifikuje a pokračuje se dalším bitem zprávy $m[i + 1]$. V opačném případě se místo změny hodnoty $b[i, b_d - 1]$ navýší či sníží hodnota celého pixelu $p[i]$ o ± 1. To, jestli se bude hodnota o jedna snižovat či zvyšovat, se vybírá zcela náhodně. Jediná vyjímka je, pokud jsou všechny bity pixelu $p[i]$ nastaveny na 1 nebo 0. V prvním případě by zvýšení hodnoty pixelu $p[i]$ o 1 způsobilo přetečení a každý bit by se překlopil na hodnotu 0, což se vizuálně zobrazuje jako naprosto odlišná barva. Stejné platí pro druhý případ s tím rozdílem, že by se hodnota 1 odečetla. V těchto situacích se volba odečtení nebo přičtení nevybírá náhodně, ale je na základě hodnoty $p[i]$ určena. V extrémním případě mohou být změněny všechny bity daného pixelu, například po přičtení 1 k pixelu $p[i] = (01111111)$ bude hodnota rovna $p[i] + 1 = (10000000)$. [3]

Proces vložení je popsán algoritmem 3. Extrakce tajných dat funguje na stejném principu, jako u algoritmu 2 metody LSB.

Jak autor dále uvádí, tak bezpečnost tohoto algoritmu je mnohonásobně větší, než u předešlé metody. Sice metody na detekci metody ±1 embedding existují, ale jejich věrohodnost je značně nižší oproti LSB.

Na obrázku 2.8 jsou patrné stejné změny, jako u 2.5. Sice se zvýšila bezpečnost vůči statistickým metodám, ale změny na souvislé oblasti jsou stále rozsáhlé a časté.

Výsledky ostatních kritérií, dle kterých hodnotíme stegosystém, zůstávají shodná s předešlou metodou.

Algoritmus 3: Vložení zprávy m do obrázku algoritmem ± 1 embedding

Input: Zpráva m, stego-key k a obrázek X
Output: Stego-obrázek Y

```
/* Inicializace PRNG klíčem k. Pokud se k = 0, tak se
   použije defaultní průchod.                          */
/* Vygenerování cesty Path o délce n = počet pixelů.   */
```
1 Path = Perm(n)
```
/* Oříznutí zprávy m, pokud se nevejde do obrázku      */
```
2 m = min(m,n)
3 **for** $i \leftarrow 0$ **to** $m - 1$ **do**
4 **if** *LSB(X[Path[i]]) == m[i]* **then**
5 **continue**
```
   /* Decision je PRNG, který vrací hodnoty e ∈ {1,0}.
      Hodnota MAX reprezentuje hodnotu, kdy mají všechny
      bity hodnotu 1. Opakem je MIN.                   */
```
6 n = Decision()
7 **if** *X[Path[i]] == MAX* **then**
8 n = 1
9 **else if** *X[Path[i]] == MIN* **then**
10 n = 0
11 Y[Path[i]] = $X[Path[i]] + (-1)^n$

(a) Cover-image (b) Stego-image

Obrázek 2.8: Porovnání obrázků (100 x 98) před a po uložení zprávy [12]

2.5.5 Metoda Matrix embedding

Jedná se o zajímavou metodu, která minimalizuje počet změn nutných k uložení tajné zprávy do obrázku a tím se zvyšuje bezpečnost stegosystému. [3] Nevýhodou však je, že zúčastněné strany (odesílatel i příjemce) se musí předem dohodnout na sdíleném tajemství. Jak se dočteme níže, sdílené tajemství je matice využívaná pro vkládání i extrakci zprávy m. Bez ní není možné metodu realizovat. Této nevýhodě se vyhneme tak, že bude matice zvolena předem a nebude ji možné změnit.

Bez újmy na obecnosti se opět omezíme na 8-bitový obrázek ve stupních šedi (jeden byte na pixel). Na rozdíl od metod výše, nevyužívá matrix embedding pro každý bit zprávy m jeden pixel, ale pro segment zprávy m o p bitech využije k pixelů, kdy $p < k$. Pro představu si uvedeme příklad, na kterém si názorně vysvětlíme princip celé metody [3].

Uvažujme cover-image, který má celkově n pixelů a zprávu m, jejíž relativní délka je $\frac{2}{3}$. To znamená, že délka zprávy bude $n\frac{2}{3}$ bitů. Budeme tedy ukládat 2 bity zprávy do skupiny 3 pixelů cover-image. Pro vkládání zvolíme metodu LSB s tím rozdílem, že nebudeme upravovat dva ze tří pixelů, aby vyhovovali hodnotám bitů zprávy m. Místo toho hodnoty pixelů p_1, p_2 a p_3 upravíme tak, aby vyhovovaly následujícím rovnicím:

$$m[i] = LSB(p_1) \oplus LSB(p_2) \tag{2.1}$$
$$m[i+1] = LSB(p_2) \oplus LSB(p_3) \tag{2.2}$$

Funkce $LSB(p[i])$ nám vrací nejméně vyznaný bit pixelu $p[i]$ a $\oplus$ značí operaci xor. Při hledání řešení rovností výše, mohou nastat čtyři možnosti. V případě, že hodnoty pixelů již vyhovují zadaným rovnicím, jejich hodnoty měnit nebudeme. Pokud nevyhovuje rovnice 2.1, pak překlopíme bit $LSB(p_1)$ a tím rovnost splníme. V případě nerovnosti 2.2, překlopíme obdobně bit $LSB(p_3)$. Pokud nevyhovují obě rovnosti zároveň, překlopíme bit $LSB(p_2)$. [3] Můžeme si povšimnout, že pro uložení 2 bitů zprávy m změníme hodnotu nejvýše jednoho pixelu ze tří.

V případě extrakce zprávy ze stego-image již nemůžeme použít známý algoritmus 2, jelikož bity zprávy nejsou zakódovány do jednotlivých pixelů, ale do skupiny pixelů. V našem případě do trojic. Pro extrakci zprávy m využijeme následující rovnost:

$$m = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} y \tag{2.3}$$

Kde y představuje vektor délky 3, který je tvořen upravenými hodnotami pixelů ze stego-image. Platí tedy $y = (LSB(p_1), LSB(p_2), LSB(p_3))$.

I když se dá výše uvedená metoda využít v takovém stavu, v jakém byla uvedena, je vhodné ji zobecnit a rozšířit. Pro vkládání i extrakci se dají využít lineární kódy dle výběru. My se zaměříme na Hammingovy kódy, kdy pro ukládání dvou bitů zprávy m nebudeme omezeni na skupinu pixelů o velikosti tři.

Hammingovy kódy se řadí mezi nejzákladnější samoopravné kódy a jsou široce užívány. Dají se aplikovat na kompresi dat, opravu chyb při přenosu dat a jiné. V našem případě je využíváme na redukci počtu změn provedených v cover-image k vyprodukování stego image. Při tomto využití nám však nezabezpečí data před poškozením a nevyhneme se tedy potřebě data samoopravnými kódy zabezpečit. Toto téma bude probráno více podrobně v kapitole 3. Pro naše potřeby si však určité základy a principy musíme vysvětlit již teď. Nejdříve si zadefinujeme sérii pojmů [14, 15], které jsou třeba k pochopení této problematiky.

Definice 2.5 (Abeceda) *Abeceda je jakákoli neprázdná a konečná množina.*

Pro naše potřeby budeme pracovat s abecedou A definovanou jako $A = \{0, 1\}$.

Definice 2.6 (Kód) *Kód C je podmnožinou A^n pro nějakou množinu nebo abecedu A.*

Definice 2.7 (Slovo a Kódové slovo) *Slovo je označení pro prvek množiny A^n s délkou n. Kódovým slovem se myslí prvek z množiny C.*

Definice 2.8 (Lineární kód) *Kód C délky n nad $\mathbb{Z}_p$ je lineární kód, pokud tvoří podprostor v lienárním prostoru $\mathbb{Z}_p^n$. Jestliže $\dim C = k$, pak se jedná o lineární (n, k)-kód nad $\mathbb{Z}_p$.*

Zápis $\mathbb{Z}_p$ představuje konečné těleso celých čísel modulo prvočíslo p. Můžeme si povšimnout, že celkový počet kódových slov v C je roven p^k. Také platí, že počet informačních znaků slova $x \in C$ je k a kontrolních právě $n - k$.

Definice 2.9 (Generující matice) *Buď $B = \{\vec{b}_1, ..., \vec{b}_k\}$ báze lineárního (n, k)-kódu C. Pak se matice $G = \begin{pmatrix} \vec{b}_1 \\ \vdots \\ \vec{b}_k \end{pmatrix}$ nazývá generující matice kódu C.*

Definice 2.10 (Kontrolní matice) *Nechť C je lineární (n,k)-kód nad $\mathbb{Z}_p$. Buď $B = \{\vec{b}_1, ..., \vec{b}_{n-k}\}$ báze podprostoru $C^\perp$. Pak se matice $H = \begin{pmatrix} \vec{c}_1 \\ \vdots \\ \vec{c}_{n-k} \end{pmatrix}$ nazývá kontrolní matice kódu C.*

Zápis vlastně znamená, že jestliže $x \in C$, pak musí platit rovnost $Hx = 0$. To je způsobeno tím, že vektory matice H jsou kolmé na vektory matice G. Z toho plyne, že skalární součin vektorů $a \in G$ a $b \in H$ se musí rovnat $a \cdot b = 0$. Tím se dá jednoduše ověřit, zda přijaté slovo patří do kódu C, či nikoli.

Hammingovy kódy jsou označovány jako lineární (n,k) q-ární kódy, kde q označuje velikost $\mathbb{Z}_q$. [15] V našem případě budeme pracovat vždy nad konečným tělesem $\mathbb{Z}_2^n$, tedy nad abecedou $A = \{0,1\}$ se slovy o délce n.

Dle autora je relativní velikost uložené zprávy, za pomoci Hammingových kódů, rovna $\alpha_b = \frac{b}{2^b-1}$. [3] To znamená, že b bitů zprávy jsme schopni uložit do $2^b - 1$ pixelů obrázku. Celý princip metody je vlastně založen na „opravě“ hodnot bitů pixelů tak, aby odpovídaly naší požadované zprávě m. Generující matici G tedy nepotřebujeme a vystačíme si s kontrolní maticí H, která má následující podobu:

$$H = \begin{pmatrix} a_{0,0} & \cdots & a_{0,2^b-1} \\ \vdots & \ddots & \vdots \\ a_{b,0} & \cdots & a_{b,2^b-1} \end{pmatrix} \tag{2.4}$$

Sloupce jsou všechny nenulové vektory délky b a řádky představují bázi podprostoru $C^\perp$. Jednotlivé sloupce můžeme tedy vygenerovat jako hodnoty od 1 do $2^b - 1$ a zapsat v binární soustavě. Jako v příkladu výše, bude vektor x představovat hodnoty pixelů $x = (LSB(p_0), \ldots, LSB(p_{2^p-1}))$. Naším cílem při ukládání bitů do pixelů je to, abychom splnili následující rovnici:

$$Hy = m \iff Hy - m = 0 \tag{2.5}$$

Při ukládání zprávy můžeme narazit na dva případy. Prvním je, že rovnice $Hx = m$ platí, bez změny hodnot vektoru x. V tom případě žádné změny neprovádíme a přesuneme se na následující sekvenci bitů zprávy a pixelů obrázku. V opačném případě musí být bity vektoru x vhodně upraveny. To se dá jednoduše zjistit po vypočtení rozdílu $Hx - m$ a díky tomu víme, jakou

hodnotu je třeba k x přičíst, aby byla rovnice splněna. Hodnotu nalezneme v matici H a změníme korespondující bit ve vektoru x. Tedy pokud jsme nalezli hodnotu v j-tém sloupci, pak překlopíme j-tý bit vektoru x. Po těchto úpravách je rovnice 2.5 splněna a přesouváme se na následující bity zprávy a pixely obrázku.

Tento způsob nám umožnil uložit b bitů zprávy m do $2^b - 1$ pixelů obrázku, při změně nejvýše jednoho bitu vektoru x. Při porovnání s předchozími metodami se značně zvýšila bezpečnost, díky minimalizaci počtu změn. Pokud využijeme princip metody ± 1 embedding, kdy změníme hodnotu celého pixelu, a ne pouze hodnotu $LSB(p[i])$, můžeme bezpečnost ještě značně navýšit.

Můžeme si povšimnout, že na rozdíl od předešlých obrázků jsou změny v 2.9 méně rozsáhlé, a ne tak dobře patrné, i když se jedná o souvislou plochu jedné barvy.

(a) Cover-image (b) Stego-image

Obrázek 2.9: Porovnání obrázků (100 x 98) před a po uložení zprávy [12]

Odolnost této metody zůstává v podstatě stejná, jako u předešlých metod 2.5.3 a 2.5.4, s tím rozdílem, že pokud poškodíme LSB bit jednoho či více z $2^b - 1$ pixelů, můžeme ztratit b bitů tajné zprávy. Na rozdíl od předešlých metod, mohou při extrakci zprávy nastat situace, kdy i přes poškození bitů pixelů extrahujeme správnou b-tici bitů tajné zprávy. Pro ukázku budeme uvažovat situaci, kdy $b = 3$ a vektory x, y budou představovat hodnoty LSB bitů pixelů. Jak je v příkladu 2.5.5 vidět, pro dva různé vektory x a y jsme dostali stejný výsledek. Tato vlastnost však není úmyslná a je čistě náhodné, zda se poškodí či nepoškodí správné bity pixelů. Z toho důvodu metoda stále pozbývá jakoukoli odolnost vůči poškození.

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$x = (0, 0, 1, 0, 0, 0, 0), \quad y = (1, 1, 0, 0, 0, 0, 0)$$

$$Hx = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \quad Hy = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

Celková kapacita se oproti předchozím metodám snížila. Relativní kapacita totiž závisí na vzorci $\frac{b}{2^b-1}$ a s narůstajícím b začíná být silně neefektivní. Poměr využitých bitů ku všem bitům obrázku zůstává $\frac{1}{b_d}$, kde b_d značí bitovou hloubku pixelu.

Metoda je opět spolehlivá s nulovou ztrátou, pokud nedošlo k editačním změnám a byl využit správný stego-key i matice H.

Pro naše potřeby bude vhodný kód s parametrem $b = 3, 4, 5$, který stále zachovává poměrně vysokou kapacitu. V nejhorším případě, kdy $b = 5$, bude relativní kapacita rovna $\frac{5}{2^5-1} = 0.161$. Vyšší hodnoty b jsou nevhodné vzhledem k tomu, že budeme chtít na probrané metody aplikovat samoopravné kódy se zaměřením na opravu chyb, čímž dále zredukujeme místo pro ukládaná data.

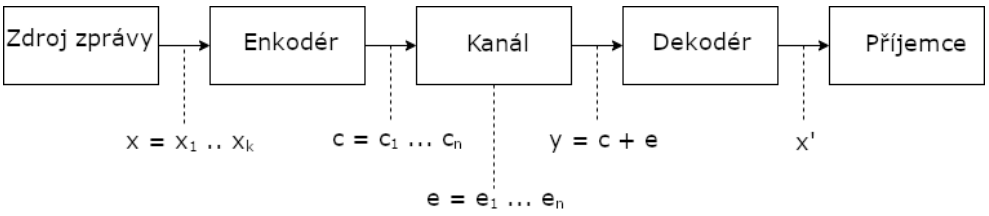
Kapitola 3

Samoopravné kódy

Jak již bylo nastíněno dříve, při přenosu dat přes komunikační kanál může docházet k šumu, který přenášená data poškodí. K tomu většinou dochází vlivem různých fyzikálních jevů a samoopravné kódy se snaží tento šum potlačit a doručit příjemci data v takovém stavu jako byla odeslána.

V této kapitole nebudeme uvažovat poškození tohoto charakteru. Naším cílem není ochránit data uložená do obrázku proti šumu při přenosu, ale při cíleném nebo nevědomé editaci uživatelem. Konkrétně se bude jednat o mírné editace obrázku a o editace, které uživatelé běžně dělají. Budeme uvažovat následující typy poškození: vyříznutí souvislé části obrázku, vyříznutí několika menších částí, vložení textu do obrázku a poškození náhodně vybraných pixelů. Typy poškození jako je rotace, zvětšení, zmenšení a jiné, neuvažujeme kvůli jejich vlivu na všechny pixely obrázku. Samoopravné kódy mají své limity a poškození takového rozsahu by byla mimo jejich možnosti.

Všechny samoopravné kódy jsou založené na stejném základním principu. Do vstupních dat se přidají dodatečné informace, pomocí kterých jsme schopni detekovat a případně rekonstruovat zprávu, která byla poškozena vlivem šumu či v našem případě vlivem editace obrázku. Obrázek 3.1 názorně ilustruje aplikaci samoopravných kódů na posílanou zprávu. [16] Do zprávy x o délce k jsou přidány kontrolní bity a vznikne kódové slovo, viz 2.7, c o délce n. Toto slovo je posléze posláno přes komunikační kanál. Při tomto přenosu může dojít k poškození slova, což nám znázorňuje vektor e, který představuje šum či v našem případě poškození uživatelem. Po poškození kódového slova x vektorem e je vyprodukován vektor $y = c + e$. Vektor y je poté zpracován dekodérem, který se pokusí opravit chyby pomocí kontrolních bitů a kontrolní bity ze zprávy odstraní. Výsledkem dekodéru je odhad původního vektoru x, označený jako x'. Jedná se o odhad z toho důvodu, že rozsah detekce a oprav chyb závisí na typu kódu a jeho parametrech.



Obrázek 3.1: Komunikační kanál se samoopravnými kódy [16]

Samoopravné kódy se dají rozdělit do dvou základních kategorií, na blokové kódy probrané v 3.1 a na konvoluční kódy uvedené v 3.2. Jelikož se budeme v práci dále věnovat konvolučním kódům, nebudeme popis blokových kódů rozebírat do detailů. V sekci 3.3 uvádíme konkrétní parametry zvoleného konvolučního kódu, který v praktické části naimplementujeme. V každé kategorii budeme vždy uvažovat kódy definované nad konečným tělesem $\mathbb{Z}_2$.

3.1 Blokové kódy

V této sekci probereme pouze základní informace související s blokovými kódy. Podrobnější náhled do problematiky nalezneme v [16].

Jak už název napovídá, pracují v rámci jednoho bloku pevné velikosti a označují se jako lineární (n, k) kódy. To znamená, že pro vstupní blok o délce k vygenerují kódové slovo c o délce n, které obsahuje kontrolní bity. Ty se mohou nacházet na libovolně zvoleném místě výstupního bloku, ale většinou se umisťují za konec k informačních bitů a jejich počet je roven $n - k$. Pro tuto kategorii je typické, že se pracuje s velikými hodnotami k, n a hodí se při opravě náhodných chyb v rámci dekódovaného bloku. Chybu považujeme za náhodnou, jestliže je bit zprávy c poškozen šumem e nezávisle na ostatních. [17] Vzhledem k tomu, že velikost bloku je konečná, tak se pro generování těchto kódu jako enkodér využívá generující matice 2.9 a pro dekodér matice kontrolní 2.10.

Jako příklad mohou být uvedeny Hammingovy kódy, které jsme využili v kapitole 2.5.5 pro ukládání zprávy do obrázku. Při tomto využití Hammingových kódů jsme přeskočili fázi samotného kódování enkodérem (generující maticí) a místo toho jsme LSB bity pixelů brali jako vektor y, který je výstupem komunikačního kanálu. Využili jsme pouze dekodér (kontrolní matici) a s její pomocí opravili případné náhodné chyby.

3.2 Konvoluční kódy

Konvoluční kódy, na rozdíl od blokových, dostanou na vstup celou posloupnost bitů, namísto bloku pevně dané velikosti. Nejzásadnějším rozdílem konvolučních kódů oproti blokovým, je závislost výstupu c na L předchozích vstupních bitech. Konvoluční kódy značíme jako lineární (n, k, m) kódy, kde k je počet vstupních bitů, n označuje počet výstupních bitů z enkodéru a m určuje velikost lineárně posuvného registru[1]. [18] Z toho plyne, že konvoluční kódy pro své správné fungování potřebují paměť navíc o velikosti m, kde se budou držet hodnoty předešle zpracovaných bitů.

Parametr L, nazývaný *délka omezení*, nám udává počet pozic v registru, přes které může jeden vstup ovlivnit celý systém. [17] Jinými slovy se jedná o počet předchozích bitů, které nám ovlivňují hodnotu bitu následujícího a je definována jako $L = m + 1$.

Díky přidaným vlastnostem, v porovnání s kódy blokovými, jsou schopné opravit i velké množství chyb. Kromě náhodných chyb jsou konvoluční kódy schopné opravit i tzv. *burst errors*, což je důvod proč budeme v praktické části implementovat právě kód z této kategorie. Jedná se o delší sekvenci poškozených bitů, přičemž poškození tohoto charakteru není zpravidla náhodné. V našem případě můžeme jako burst error chápat například vyříznutí souvislé části obrázku.

Konvoluční kódy obvykle využívají malé hodnoty (n, k), které se zpravidla nemění, a namísto toho pracují s různými hodnotami L, resp. m. [16] To nám umožní udržet stejný informační poměr $R = \frac{k}{n}$[2] a přitom navýšit počet chyb, který je kód schopen opravit. Počet chyb t, které je daný kód schopen opravit, je závislý na tzv. *volné vzdálenosti* d_{free} a platí nerovnost $d_{free} \geq 2t + 1$, která se dá přepsat jako $t = \lfloor \frac{d-1}{2} \rfloor$. [19] Volná vzdálenost je definována jako nejmenší Hammingova vzdálenost mezi všemi dvojicemi kódových slov. [19]

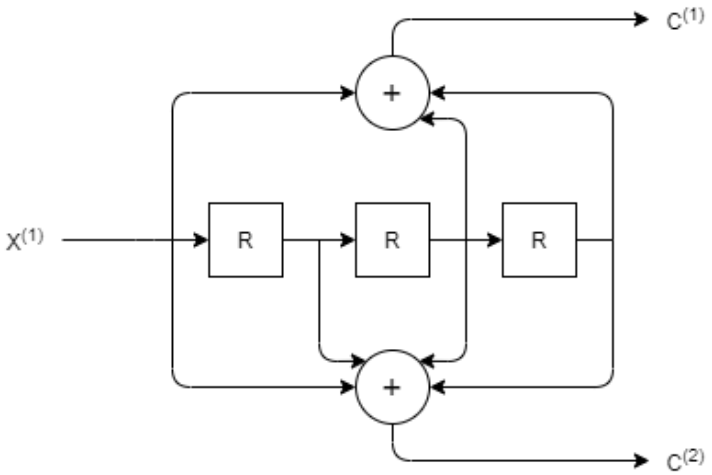
Na schématu 3.2 si můžeme prohlédnout jednoduchý konvoluční kód s velikostí registru $m = 3$, který bere jako vstup jeden bit a na výstup dává bity dva. Na tomto $(2, 1, 3)$ kódu si uvedeme různé způsoby, jakými se dá reprezentovat enkodér.

3.2.1 Reprezentace maticí

Jedná se o nejvíce kompaktní reprezentaci konvolučních kódů, kterou jsou zapsány vhodné kódy v 3.1. Abychom byli schopni vybraný kód v praktické části naimplementovat, vysvětlíme si význam takového zápisu.

[1] Velikost registru nám říká, kolik předchozích bitů jsme schopni si zapamatovat.

[2] *Informační poměr* nám udává velikost zprávy obsahující pouze informační bity ku velikosti zprávy obsahující i bity kontrolní.



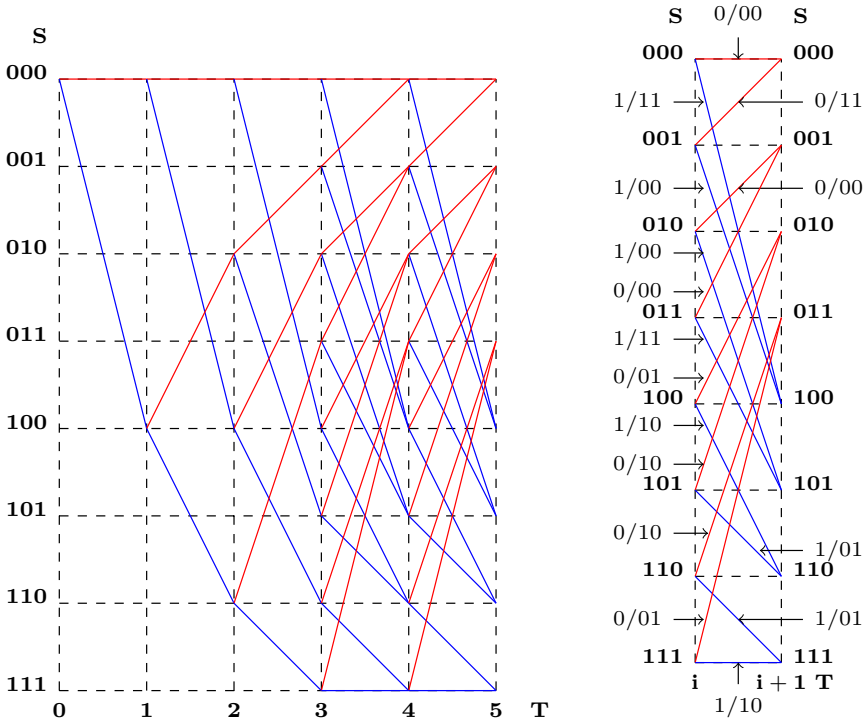
Obrázek 3.2: Struktura konvolučního kódu s parametry $k = 1$, $n = 2$, $R = \frac{1}{2}$, $L = 4$

Matice reprezentuje připojení jednotlivých vstupů/výstupů buněk lineárně posuvného registru do modulo-2 sčítačky. [20] Obsahuje oktalové zápisy schémat zapojení, po jehož převedení na binární soustavu nám 1 říká, že daný spoj bude přiveden do modulo-2 sčítačky, a 0 znamená, že nikoliv. Například schéma 3.2 odpovídá oktalovému zápisu $(\,13\ 17\,)$, který je po převedení do binární soustavy roven $(\,1011\ 1111\,) = (\,c^{(1)}\ c^{(2)}\,)$.

3.2.2 Reprezentace mřížkou

Tato reprezentace je v podstatě jiný způsob zakreslení konečného automatu daného konvolučního kódu a budeme ji využívat právě v sekci 3.2.3 kvůli jejímu přehlednému a zhuštěnému zápisu. [21] Zápis $\mathbf{x}/\mathbf{c}$, kterým jsou označeny jednotlivé přechody v 3.3b, označuje vstupní bit x a výstupní bity c.

Pro názorné vysvětlení opět využijeme schéma 3.2, které jsme znázornili v podobě mřížky 3.3a. Osa S zobrazuje všechny možné stavy, do kterých se může lineárně posuvný registr dostat a počet stavů je roven 2^m, tedy v našem případě 8. Osa T zobrazuje jednotky času. V čase $t = 0$ se nacházíme ve stavu 000 a při přechodu z $t = 0$ na $t = 1$ přijmeme jeden vstupní bit, který může nabývat hodnoty 0 (znázorněno červeně) nebo 1 (znázorněno modře). Pro vstupní hodnotu $x = 0$ stav zůstává roven 000 a výstupem bude $c = 00$. V opačném případě se dostaneme do stavu 100 a výstup bude $c = 11$. Tímto způsobem budeme pokračovat tak dlouho, dokud nenačteme všechny vstupní bity. Následně je nutné ke vstupu připojit m nulových bitů, abychom registr dostali do výchozího nulového stavu. Na obrázku 3.4 si můžeme prohlédnou průchod mřížkou pro konkrétní vstup $x = (0, 1, 0, 1, 1)$, po jehož zpracování bude výstup roven $c = (00, 11, 01, 00, 01)$.



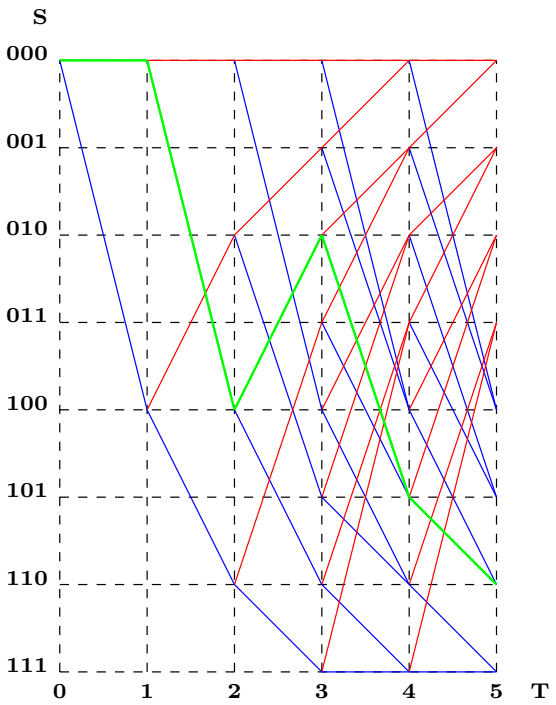
(a) Mřížka (b) Legenda

Obrázek 3.3: Reprezentace mřížkou kódu 3.2

3.2.3 Viterbiho dekódovací algoritmus

Pro dekódování volíme Viterbiho algoritmus (vhodný pro menší m), který spočívá ve výběru nejpravděpodobnější cesty v mřížce, čímž se stanoví odhad x' původního vektoru x. [21] Jako stěžejní metrika výběru cesty se využívá *Hammingova minimální vzdálenost*, která nám určuje vzdálenost přijatého kódového slova od kódového slova, které může být z aktuálního stavu $S_{k,t}$ vytvořeno v čase $t + 1$. [21] Při vytváření cesty se nám může stát, že by Hammingovy vzdálenosti do následujících stavů měly stejnou hodnotu. V tom případě se nám dosavadní cesta P_t rozdělí a algoritmus bude pokračovat na obou z nich. Všechny cesty, které algoritmus dále zpracovává se označují jako *přeživší*. V případě, že některé z přeživších cest skončí ve stejném stavu $S_{k,t}$, vybereme přeživší cestu náhodně a ostatní nadále nezpracováváme. Pokud jsme zpracovali celý vstup y a přeživších cest se stejnou váhou[3] je více než jedna, pak výslednou cestu vybereme také náhodně. [21] Výsledkem je unikátní cesta mřížkou, která nám určuje odhad x'. Podrobnější vysvětlení algoritmu lze dohledat v [16], nebo [21].

[3] Váha cesty je rovna součtu všech vah stavů dané cesty.



Obrázek 3.4: Znázornění cesty pro vstup $x = (0, 1, 0, 1, 1)$ a výstup $c = (00, 11, 01, 00, 01)$

3.3 Výběr kódu

Vzhledem k výše uvedeným vlastnostem bude vhodné pro naši aplikaci zvolit právě konvoluční kódy namísto blokových. Pro naše potřeby bude vhodný informační poměr $R = \frac{1}{2}$, abychom stále udrželi co nejvyšší kapacitu i při výběru malých obrázků nebo vkládání pomocí Hammingových kódů.

V tabulce 3.1 vidíme vhodná schémata konvolučních kódů s námi vybraným poměrem. Výběr z předdefinovaných schémat je výhodný z toho důvodu, že se vyhneme nepříjemnostem, jako jsou *katastrofické kódy*[4] a zároveň máme ověřenou jejich korektní funkčnost.

Jako nejvhodnější schéma pro implementaci ochrany utajené zprávy volíme (2335 3661), které dokáže opravit až $t = 6$ chyb.

[4]Jedná se o kategorii konvoluční kódů, jejichž dekódování v případě přijmutí malého množství chyb způsobí propagaci chyb do velkého množství následujících bitů. [20]

m	d	g
1	3	(1 3)
2	5	(7 5)
3	6	(13 17)
4	7	(23 35)
5	8	(53 75)
6	10	(133 171)
8	12	(561 753)
10	14	(2335 3661)

Tabulka 3.1: Vhodná schémata $(2, 1, m)$ kódu [21]

Kapitola 4

Návrh a implementace aplikace

Abychom mohli prakticky ověřit vlastnosti různých metod vkládání, společně s vlastnostmi samoopravných kódů, tak navrhneme a implementujeme GUI aplikaci. Ta bude implementovat všechny metody ukládání do obrázku, které byly uvedeny v sekci 2.5 a konvoluční kód probraný v kapitole výše. Uživatelský manuál aplikace si můžeme přečíst v příloze A.

Pro tvorbu aplikace byly využity následující technologie:

Python Skriptovací programovací jazyk, konkrétně jeho verze 3.8.2. Byl vybrán především z toho důvodu, že se v něm dají psát přenositelné aplikace a nabízí rozsáhlé knihovny. Oproti ostatním jazykům je přehledný a v napsaném kódu se dá rychle zorientovat. Vzhledem k tomu, že se jedná o vyšší programovací jazyk, tak se vyhneme případným bezpečnostním rizikům jako buffer overflow (typicky v jazyku C) a ostatním častým chybám.

Qt5 Multiplatformní framework, který slouží pro tvorbu intuitivního a graficky přívětivého GUI. Je napsán v jazyce C++ kvůli rychlosti a v Pythonu je k dispozici ve formě knihovny *PyQt*, která vytváří binding na původní konstrukty jazyka C++.

QtDesigner Jedná se o nástroj pro návrh a tvorbu uživatelských rozhraní. Sice je možné nadefinovat uživatelské rozhraní přímo v jazyce Python programově, ale je vhodnější a praktičtější využít značkovací jazyk XML. Ten umožňuje přehledně a jasně popsat všechny prvky GUI. QtDesigner dokáže soubor XML načíst/vytvořit a zobrazit rozhraní v grafické podobě. Dále umožňuje vytvářet rozhraní pomocí modeláře, díky čemuž rovnou vidíme jeho grafickou podobu a ve vygenerovaném XML souboru posléze doladíme případné detaily, jako je například výchozí šířka a výška okna.

PyCharm IDE pro programování v jazyce Python, které bylo využito jak při testování, tak programování na platformě Linux.

V samotném programu jsme využili následující knihovny, které nám ulehčily práci s formátem obrázku PNG, maticemi, binárními daty a samoopravnými kódy. Aplikace využívá i další knihovny, ale vybrali jsme pouze nejzajímavější zástupce. Knihovnu na samoopravné kódy bylo poměrně problémové najít. Sice existuji implementace v samotném Pythonu (CommPy, …), ale z pravidla jsou velice pomalé. Z toho důvodu jsme nuceni využít knihovnu napsanou v jazyce C a jednotlivé funkce volat z Pythonu.

PyQt5 Knihovna zpřístupňující framework Qt5 pro jazyk Python, dostupná z [22].

NumPy Nabízí širokou škálu matematických funkcí, ale především umožňuje efektivní a pohodlnou práci s maticemi a n rozměrnými poli, které budeme v programu hojně využívat [23].

Bitarray Umožňuje zadefinovat pole bitů, které zjednodušuje a zpřehledňuje práci s binárními daty. Knihovna je dostupná na [24]

Pillow Komplexní knihovna, dostupná z [25], poskytuje základní operace s obrázky. V našem případě byla využita na načtení/uložení obrázku a pro konverzi na NumPy n rozměrné pole, se kterým posléze pracujeme. Knihovna si načte hlavičku IHDR a dle ní zpracuje (dekomprimuje a aplikuje filtr) a poskytne samotná obrazová data uložená v IDAT. Ta se knihovní funkcí dají převést na výše uvedené n rozměrné pole a po námi provedených úpravách umožní výsledek převést zpět na formát obrázku a uložit.

Libcorrect Knihovna napsaná v jazyce C, dostupná z [26], která je pomocí knihovny *ctypes*, umožňující volání funkcí jazyka C, využita v Pythonu. Poskytuje volbu schématu konvolučního kódu (polynom, velikost registru), kterým budou data zakódována. K dispozici je také implementace Viterbiho algoritmu 3.2.3, kterým jsou data následně dekódována a případné chyby opraveny.

4.1 Návrh GUI

Aplikace napsaná v Qt5 uživateli poskytuje následující stěžejní funkcionality:

- Výběr metody ukládání dat do obrázku
- Výběr samoopravného kódu

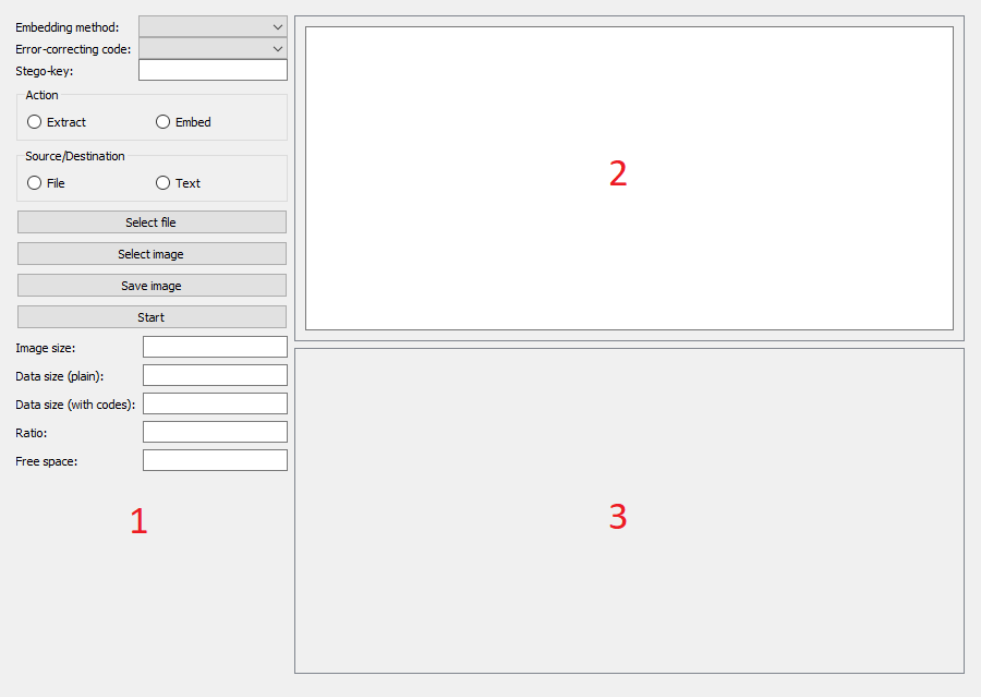
- Načtení/uložení obrázku
- Extrakce/vložení dat

Na obrázku 4.1 si můžeme prohlédnou výsledné GUI, které poskytuje výše uvedené funkcionality. Celé okno se dá rozdělit na tři logické části.

První poskytuje ovládací prvky aplikace a zobrazuje důležité informace o obrázku, zvoleném ukládání a kódování. Vzhledem k tomu, že chceme umožnit extrakci/vložení jakýchkoli dat, a ne jen tisknutelných znaků, přibyla možnost využít soubor jako vstup nebo výstup. Tato možnost je také vhodná s ohledem k tomu, že data mohou být poškozena a z tisknutelných znaků se mohou stát netisknutelné, které by nebylo možné zobrazit v části 2.

Druhá část slouží pro textový vstup a výstup. V případě, že by uživatel při extrakci zvolil textový výstup 2 a obrázek obsahoval binární data (netisknutelné znaky), bude vyzván k uložení dat do souboru.

Třetí část slouží jako náhled obrázku po vložení dat, kde se můžeme podívat, k jak moc viditelným změnám došlo.



Obrázek 4.1: Návrh GUI

4.2 Implementace

Aplikace byla implementována s ohledem na přehlednost, bezpečnost a budoucí rozšiřitelnost. Jedná se o dvouvrstvou architekturu a kód je rozdělen do tří logických částí.

První část se skládá ze souboru *gui.ui* obsahujícího definici uživatelského rozhraní ve formátu XML. Ten je posléze načten v souboru *stego_app.py*, který vytvoří vazbu na jednotlivé grafické prvky a stará se o interakci s uživatelem.

Do druhé části patří soubor *image_methods.py*, který se stará o načítání a ukládání obrázku. Data načtená z IDAT chunku PNG formátu jsou převedena na NumPy n rozměrné pole, kvůli zvýšení rychlosti a přehlednosti práce s jednotlivými pixely. Následně je vygenerována permutace, která obsahuje hodnoty od 0 do k, kde k je počet všech prvků v n rozměrném poli. Výsledná permutace nám tedy určuje pořadí pixelů, v jakém budou data do obrázku uložena. Permutace je vygenerována v závislosti na stego-key, který nám poskytl uživatel. Ten je posléze převeden na SHA256 a využit jako seed pro pseudonáhodný generátor.

Poslední část obsahuje soubory *embedding_methods.py* a *error_correcting_codes.py* implementující jednotlivé metody vkládání a samoopravné kódy, které byly probrány v předchozích kapitolách.

Kapitola 5

Analýza odolnosti metod a samoopravných kódů

V této kapitole analyzujeme možnosti samoopravných kódů v kombinaci s různými metodami vkládání do obrázku, ve vztahu k rozsahu a charakteru jeho poškození. Jednotlivé sekce kapitoly budou zaměřeny na konkrétní typy poškození obrázku, které byly vybrány s ohledem na jejich zajímavost a pokrytí různých forem editace uživatelem. Ve všech sekcích budeme pracovat se stejným obrázkem, u kterého bude využita jeho maximální kapacita pro danou metodu vkládání. Postupně budeme navyšovat rozsah daného poškození a následně vyhodnotíme, jak samoopravný kód a metoda vkládání obstála. Vzhledem k identickým vlastnostem metod LSB a ±1 embedding z hlediska odolnosti, budeme v analýze pracovat pouze s druhou metodou a výsledky budou platné i pro první.

Pro analýzu využijeme již jednou uvedený obrázek 2.6a, který má rozumné rozměry (300 x 188) a zároveň nabízí dostatečnou kapacitu z hlediska analýzy. Co obrázek zobrazuje a jak vypadá je pro nás irelevantní.

Pro výše uvedený obrázek mají metody 2.5.3 a 2.5.4 shodnou kapacitu jež je rovna $\lfloor \frac{188*300*4}{8} \rfloor = 28200$ byte. Rozměry jsou vynásobeny koeficientem 4, jelikož je u tohoto obrázku využit tzv. alpha channel, probraný v 2.5.2. Naopak u metody 2.5.5 je celková kapacita rovna $\lfloor 28200 * \frac{3}{7} \rfloor = 12085$ byte. Výsledné kapacity se však ještě sníží o několik bytů, které reprezentují délku uložené zprávy a jsou vloženy společně s tajnou zprávou. Následně se kapacita poníží koeficientem informačního poměru zvoleného samoopravného kódu, který je v našem případě vždy roven $R = \frac{1}{2}$. Výsledná kapacita je rovna 14080 pro metodu 2.5.4 a 6021 pro metodu 2.5.5.

Metriku pro hodnocení definujeme jako $\frac{B_{ok}}{B_{all}}$, kde B_{ok} představuje úspěšně

extrahované byty z obrázku a B_{all} označuje celkový počet uložených bytů. Výsledek následně převedeme na procenta vyjadřující úspěšnost extrakce. Posléze spočteme aritmetický průměr úspěšností při všech procentech poškození. Pomocí toho budeme schopni porovnávat výsledky mezi sebou a zhodnotit metodu jako takovou.

V každém typu poškození budeme pracovat s tabulkou, která bude mít jednotnou a jasně definovanou strukturu. První sloupec tabulky nám říká, kolik procent plochy obrázku bylo poškozeno daným typem editace. Následující dva sloupce vyjadřují, zda se dala data z obrázku vůbec extrahovat a zda se délka extrahovaných dat rovnala délce dat vložených. Jestliže délka vložených dat nebyla zachována, pak budeme neextrahované byty považovat za poškozené. Čtvrtý sloupec udává počet poškozených bytů zprávy po provedení opravy, respektive dekódování Viterbiho algoritmem. Může se stát, že počet poškozených bytů nebude vždy růst společně se zvyšováním velikosti poškozené plochy obrázku. To nastává z toho důvodu, že se společně s informačními bity zprávy do obrázku ukládá i délka zprávy jako takové. Pokud se informace o délce uložené zprávy poškodí, může ztráta této informace způsobit nemožnost extrakce (dekódovaná délka zprávy je větší než kapacita obrázku) nebo zavinit výše uvedený rozpor. Poslední údaj v tabulce obsahuje nejdůležitější informaci, což je výsledek námi definované metriky $\frac{B_{ok}}{B_{all}}$.

Ve všech případech je jako samoopravný kód využit konvoluční kód, který jsme zvolili v 3.3, a proto jsou sekce členěny dle metody vkládání.

Veškeré upravené obrázky, které jsme využili k analýze, nalezneme na přiloženém CD v digitální formě.

5.1 Vyříznutí souvislé oblasti

Jedná se o velice běžnou editaci obrázku uživatelem, a proto ji uvádíme jako první. Zvolíme oblast, která zaujímá zvolený počet procent z celkové plochy obrázku a tu vyřízneme, respektive nastavíme všechny složky RGB, včetně alpha channel na hodnotu 0. Vyříznutá plocha zaujímá z počátku pouhé jedno procento a postupně ji rozšiřujeme až na 60 % z celkové plochy obrázku.

5.1.1 Metoda ± 1

V tabulce 5.1 vidíme, že metoda dosahuje 100% úspěšnosti, pokud bude obrázek poškozen nejvýše z 20 %. Při vyšším poškození úspěšnost prudce klesá a při vyříznutí 60 % celkové plochy obrázku, jsme schopni úspěšně extrahovat pouze $20,49$ % vložených bytů. U takto rozsáhlého poškození se jako u jediného poničila i délka uložené zprávy.

Průměrně jsme byli schopni extrahovat 68, 77 % dat, což je velmi dobrý výsledek.

Procentuální poškození obrázku (%)	Extrakce úspěšná	Délka zprávy zachována	Poškozené byty po opravě (B)	Úspěšnost obnovení zprávy (%)
1	✓	✓	0	100
5	✓	✓	0	100
10	✓	✓	0	100
20	✓	✓	0	100
30	✓	✓	9133	35, 13
40	✓	✓	6920	50, 85
50	✓	✓	7921	43, 74
60	✓	✗	11195	20, 49

Tabulka 5.1: Výsledky extrakcí dat z obrázku, po vyříznutí souvislé části

5.1.2 Metoda Matrix embedding

Dle posledního sloupce tabulky 5.2 rovnou vidíme, že je metoda Matrix embedding mnohem více náchylnější na poškození tohoto charakteru, nežli Metoda ±1. Úspěšnosti 100 % bylo dosaženo pouze v případě poškození jednoho procenta a zatímco předchozí metoda nabyla úspěšnosti okolo 20 % až při poničení 60 % plochy, Matrix embedding jí dosáhla již při 20% poškození. Pokud je poškození vyšší než 20 %, nedají se data z obrázku vůbec extrahovat.

Průměrná úspěšnost metody je rovna 27, 51 %.

Procentuální poškození obrázku (%)	Extrakce úspěšná	Délka zprávy zachována	Poškozené byty po opravě (B)	Úspěšnost obnovení zprávy (%)
1	✓	✓	0	100
5	✓	✓	2508	58, 34
10	✓	✓	3596	40, 27
20	✓	✓	4725	21, 52
30	✗	✗	6021	0
40	✗	✗	6021	0
50	✗	✗	6021	0
60	✗	✗	6021	0

Tabulka 5.2: Výsledky extrakcí dat z obrázku, po vyříznutí souvislé části

5.2 Vyříznutí několika nesouvislých oblastí

Tento typ poškození se od předchozího liší tím, že je poškozená plocha obrázku rozdělena na čtyři nesouvislé části, a ty jsou vyříznuty stejným způsobem, jako tomu je v 5.1. Procentuální rozsah poškození zůstává zachován.

5.2.1 Metoda ±1

Tento typ v žádném uvedeném rozsahu poškození nezpůsobil porušení délky zprávy a ve všech případech se dala ukrytá data extrahovat celá jak je vidět v tabulce 5.3. Až do rozsahu poškození 20 % dosahovala metoda 100% úspěšnosti a i při poničení 60 % plochy obrázku jsme byli schopni extrahovat přes čtyřicet procent dat.

Celková úspěšnost dosáhla 75, 78 %.

Procentuální poškození obrázku (%)	Extrakce úspěšná	Délka zprávy zachována	Poškozené byty po opravě (B)	Úspěšnost obnovení zprávy (%)
1	✓	✓	0	100
5	✓	✓	0	100
10	✓	✓	0	100
20	✓	✓	0	100
30	✓	✓	2972	78, 89
40	✓	✓	7645	45, 70
50	✓	✓	8514	39, 53
60	✓	✓	8143	42, 16

Tabulka 5.3: Výsledky extrakcí dat z obrázku, po vyříznutí čtyř nesouvislých části

5.2.2 Metoda Matrix embedding

Výsledky v tabulce 5.4 se moc neliší od výsledků předchozí tabulky 5.2 a v případě poškození vyšších než 20 % nejsme schopni data extrahovat.

Úspěšnost metody je 29, 40 %.

5.3 Vložení textu

Jedná se o další velmi častou editaci. Ať už uživatel vkládá vodoznak, či jiné údaje jako je datum pořízení obrázku či místo.

Abychom mohli tento typ poškození porovnávat s ostatními, je pro každý rozsah poškození vybrána plocha, která zaujímá stejný procentuální obsah,

Procentuální poškození obrázku (%)	Extrakce úspěšná	Délka zprávy zachována	Poškozené byty po opravě (B)	Úspěšnost obnovení zprávy (%)
1	✓	✓	0	100
5	✓	✓	2519	58, 16
10	✓	✓	3156	47, 58
20	✓	✓	4243	29, 52
30	✗	✗	6021	0
40	✗	✗	6021	0
50	✗	✗	6021	0
60	✗	✗	6021	0

Tabulka 5.4: Výsledky extrakcí dat z obrázku, po vyříznutí čtyř nesouvislých částí

jako metody předchozí. Následně je do této plochy zapsán text tak, aby ji pokrýval co nejvíce.

5.3.1 Metoda ±1

V případě vkládání textu jsme 100% úspěšnosti dosáhli na rozdíl od předchozích metod i v případě poškození 30 %. Naopak v případě poškození 40 % úspěšnost prudce klesla na pouhých 6, 92 %. Viz tabulka 5.5.

Průměrná úspěšnost vychází na 71, 47 %.

Procentuální poškození obrázku (%)	Extrakce úspěšná	Délka zprávy zachována	Poškozené byty po opravě (B)	Úspěšnost obnovení zprávy (%)
1	✓	✓	0	100
5	✓	✓	0	100
10	✓	✓	0	100
20	✓	✓	0	100
30	✓	✓	0	100
40	✓	✓	13105	6, 92
50	✓	✓	10893	22, 63
60	✓	✓	8129	42, 26

Tabulka 5.5: Výsledky extrakcí dat z obrázku, po vložení textu

5.3.2 Metoda Matrix embedding

Výsledky v tabulce 5.6, stejně jako předchozí výsledky pro metodu Matrix embedding, ukazují velké problémy už se samotnou extrakcí dat. Od pro-

centuálního poškození 40 % se data z obrázku vůbec nedají extrahovat. To způsobuje značné snížení celkové úspěšnosti.

Výsledná průměrná úspěšnost metody je rovna $36,39$ %.

Procentuální poškození obrázku (%)	Extrakce úspěšná	Délka zprávy zachována	Poškozené byty po opravě (B)	Úspěšnost obnovení zprávy (%)
1	✓	✓	0	100
5	✓	✓	0	100
10	✓	✓	3857	$35,94$
20	✗	✗	6021	0
30	✓	✓	2694	$55,25$
40	✗	✗	6021	0
50	✗	✗	6021	0
60	✗	✗	6021	0

Tabulka 5.6: Výsledky extrakcí dat z obrázku, po vložení textu

5.4 Poškození náhodně vybraných pixelů

Tento typ poškození sice nespadá do běžných editací prováděných uživatelem, ale může se projevit při využití chybového komunikačního kanálu nebo pokud by třetí strana měla podezření a chtěla by do obrázku zanést šum, za cílem zničení tajné zprávy.

Při zanesení poškození tohoto charakteru postupujeme následujícím způsobem. Nejprve vygenerujeme náhodnou permutaci, která nám vybere pixely určené k poškození. Následně z této permutace vybereme prvních n pixelů, kde n je určeno v závislosti na požadovaném rozsahu poškození obrázku. Vybrané pixely jsou shodné u obou metod vkládání v rámci rozsahu poškození. Pokud tedy budeme uvažovat poškození v rámci jednoho procenta, pak jsou vybrány identické pixely, jak u obrázku využívajícího metodu ± 1, tak i u obrázku využívajícího metodu Matrix mebedding. Pro poškození jiného rozsahu je vygenerována odlišná permutace. U takto vybraných pixelů invertujeme hodnoty LSB u všech složek RBG, včetně alpha channelu.

5.4.1 Metoda ± 1

Na rozdíl od předchozích typů poškození jsme schopni extrahovat 100 % uložených dat jen v případě poškození jednoho procenta, jak je vidět v tabulce5.7. Pokud je poškození v rozsahu 20 % a vyšší, nejsme data schopni vůbec extrahovat.

Průměrná úspěšnost metody dosahuje 23, 15 %.

Procentuální poškození obrázku (%)	Extrakce úspěšná	Délka zprávy zachována	Poškozené byty po opravě (B)	Úspěšnost obnovení zprávy (%)
1	✓	✓	0	100
5	✓	✓	7728	45, 11
10	✓	✓	8426	40, 15
20	✗	✗	14080	0
30	✗	✗	14080	0
40	✗	✗	14080	0
50	✗	✗	14080	0
60	✗	✗	14080	0

Tabulka 5.7: Výsledky extrakcí dat z obrázku, po poškození náhodných pixelů

5.4.2 Metoda Matrix embedding

Úspěšnosti 100 % nebylo dosaženo ani v jednom případě a data se dala částečně opravit pouze v případě poškození 1 % s úspěšností 58, 32 %.

Průměrná úspěšnost metody při poškození náhodných pixelů je pouhých 7, 29 %.

Procentuální poškození obrázku (%)	Extrakce úspěšná	Délka zprávy zachována	Poškozené byty po opravě (B)	Úspěšnost obnovení zprávy (%)
1	✓	✓	2509	58, 32
5	✗	✗	6021	0
10	✗	✗	6021	0
20	✗	✗	6021	0
30	✗	✗	6021	0
40	✗	✗	6021	0
50	✗	✗	6021	0
60	✗	✗	6021	0

Tabulka 5.8: Výsledky extrakcí dat z obrázku, po poškození náhodných pixelů

5.5 Vyhodnocení výsledků

Z výsledků získaných výše jsme vytvořili tabulku 5.9, která shrnuje výsledky analýzy. Z tabulky je jasně patrné, že Metoda ±1 dosahuje celkově výší úspěšnosti napříč všemi typy poškození, průměrně 59, 79 %, zatímco Matrix embedding dosahuje průměrně 25, 14 %. Obě metody jsou nejméně odolné vůči poškození náhodných bitů, které zpravidla znemožní extrakci dat.

Pokud nám jde o zachování vyšší bezpečnosti na úkor odolnosti, vychází jako vhodnější metoda Matrix embedding. V případě, kdy preferujeme odolnost na úkor bezpečnosti, zvolíme Metodu ±1, respektive metodu LSB, která odolává s vyšší úspěšností většímu spektru typů poškození. Navíc získáme i vyšší kapacitu.

Typ poškození	Metoda ±1 (%)	Metoda Matrix embedding (%)
Souvislá oblast	68, 77	27, 51
Nesouvislé oblasti	75, 78	29, 40
Vložení textu	71, 47	36, 39
Náhodné pixely	23, 15	7, 29

Tabulka 5.9: Průměrné procentuální úspěšnosti metod

Závěr

V práci jsme se zabývali utajenou komunikací využitím obrazové steganografie. Pro vložení zprávy jsme využili metody LSB, ±1 embedding a Matrix embedding a u všech metod jsme vyhodnotili jejich vlastnosti dle požadavků na stegosystém, které zahrnují bezpečnost, odolnost, kapacitu a spolehlivost. V další části rešerše jsme se zabývali samoopravnými kódy, kde bylo probráno jejich základní rozdělení na blokové a konvoluční. Na základě zjištěných vlastností jsme zvolili kód konvoluční s vhodnými parametry.

V praktické části jsme navrhli a vytvořili aplikaci, implementující probrané metody a samoopravný kód, s možností využít vlastní stegokey. V poslední části jsme analyzovali možnosti samoopravného kódu v kombinaci s metodou vkládání ve vztahu k rozsahu a charakteru poškození obrázku.

Literatura

[1] Siper, A.; Farley, R.; Lombardo, C.: The Rise of Steganography. [online], 2005, [cit. 2020-03-22]. Dostupné z: `http://csis.pace.edu/~ctappert/srd2005/d1.pdf?forcedefault=true`

[2] Simmons, G. J.: *Advances in Cryptology*. Santa Barbara, California: Cambridge University Press, New York, 1983, ISBN 978-1-4684-4730-9.

[3] Fridrich, J.: *Steganography in Digital Media*. New York: Cambridge University Press, New York, 2010, ISBN 978-1-13-919290-3.

[4] Rouse, M.: Steganography. [online], prosinec 2018, [cit. 2020-03-24]. Dostupné z: `https://searchsecurity.techtarget.com/definition/steganography`

[5] Žilka, B. R.: *Steganografie a stegoanalýza*. Diplomová práce, Masarykova univerzita, Fakulta informatiky, Brno, 2008, vedoucí práce: Ing. Mgr. Zdeněk Říha, Ph.D.

[6] Cox, I. J.; Miller, M. L.; Bloom, J. A.; aj.: *Digital Watermarking and Steganography*. USA: Morgan Kaufmann Publishers, druhé vydání, 2008, ISBN 978-0-12-372585-1.

[7] Yahya, A.: *Steganography Techniques for Digital Images*. Palapye, Botswana: Springer, 2019, ISBN 978-3-319-78535-6.

[8] Nodžák, P.: *Digitální textová steganografie*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, 2018, vedoucí práce: Ing. Josef Strnadel, Ph.D.

[9] Lockwood, R.; Curran, K.: Text based steganography. *International Journal of Information Privacy, Security and Integrity*, ročník 3, 1 2017, doi: 10.1504/IJIPSI.2017.10009581.

[10] MIT: PNG (Portable Network Graphics) Specification, Version 1.2. [online], 1999, [cit. 2020-04-02]. Dostupné z: http://png.cybermirror.org/spec/1.2/PNG-Contents.html

[11] StackExchange: How is noise added in an image during image acquisition or image transmission? [online], 2013, [cit. 2020-03-26]. Dostupné z: https://dsp.stackexchange.com/questions/10971/how-is-noise-added-in-an-image-during-image-acquisition-or-image-transmission

[12] Eddins, S.: How to Display Color Swatches. [online], Mar 2020, [cit. 2020-02-06]. Dostupné z: https://blogs.mathworks.com/steve/2020/03/10/how-to-display-color-swatches/

[13] Wide, W.: Green Nature. [online], 2010, [cit. 2020-02-05]. Dostupné z: http://wallpaperswide.com/green_nature-wallpapers.html

[14] Gollová, A.: Lineární kódy. [online], [cit. 2020-04-08]. Dostupné z: https://math.feld.cvut.cz/gollova/tik/tik_h2.pdf

[15] Fiedler, J.: Hamming Codes. [online], 2004, [cit. 2020-04-08]. Dostupné z: https://orion.math.iastate.edu/linglong/Math690F04/HammingCodes.pdf

[16] Huffman, W. C.; Pless, V.: *Fundamentals of Error Correcting Codes*. USA: Cambridge University Press, 2003, ISBN 978-0-511-07779-1.

[17] Dholakiaí, A.: *Introduction to convolutional codes with applications*. New York: Springer Science+Business Media, LLC, 1994, ISBN 978-1-4613-6168-8.

[18] Skalický, J.: *Konvoluční kódy*. Bakalářská práce, Univerzita Karlova v Praze, Matematicko-fyzikální fakulta, Praha, 2009, vedoucí práce: Mgr. Libor Barto, PhD.

[19] ScienceDirect: Convolutional Code. [online], 1990, [cit. 2020-04-13]. Dostupné z: https://www.sciencedirect.com/topics/engineering/convolutional-code

[20] Huang, F. H.: *Evaluation of Soft Output Decoding for Turbo Codes*. Diplomová práce, Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1997, vedoucí práce: Dr. F. Gail Gray.

[21] Imai, H.: *Essentials of Error-Control Coding Techniques*. Kanagawa, Japan: Academic Press, Inc., 1990, ISBN 0-12-370720-X.

[22] Limited, R. C.: PyQt5. [software], 2020, [cit. 2020-03-02]. Dostupné z: https://www.riverbankcomputing.com/software/pyqt/download5

[23] Oliphant, T. E.: A guide to NumPy. [software], 2006, [cit. 2020-03-02]. Dostupné z: `https://github.com/numpy/numpy`

[24] Schnell, I.: bitarray. [software], 2020, [cit. 2020-03-28]. Dostupné z: `https://github.com/ilanschnell/bitarray`

[25] Clark, A.: Pillow. [software], 2020, [cit. 2020-04-12]. Dostupné z: `https://github.com/python-pillow/Pillow`

[26] Armstrong, B.: libcorrect. [software], 2020, [cit. 2020-04-18]. Dostupné z: `https://github.com/quiet/libcorrect`

Příloha A

Uživatelský manuál

Aplikace byla vytvořena pro operační systém GNU/Linux. Pro její spuštění je potřeba nainstalovat program Miniconda3, který nám umožní vytvořit virtual environment, do kterého budeme instalovat potřebné knihovny. V průběhu instalace budeme vyzváni, zda chceme Minicondu3 inicializovat. Tuto výzvu potvrdíme a restartujeme terminál. Po dokončení instalace vytvoříme nový virtual environment v adresáři, kde se nachází zdrojové kódy aplikace, a aktivujeme ho:

```
conda create -n OV_BP python=3.8.2
conda activate OV_BP
```

Poté nainstalujeme všechny potřebné knihovny aplikace příkazem:

```
pip install -r requirements.txt
```

Nyní máme vše připraveno a můžeme aplikaci spustit:

```
python stego_app.py
```

Po spuštění se zobrazí uživatelské rozhraní, zachycené na obrázku A.1.

Embedding method Slouží k vybrání metody vkládání, respektive extrakci. Tato možnost je povinná a jako výchozí hodnota je zvolena metoda LSB.

Error-correcting code Uživatel si může vybrat samoopravný kód, který bude aplikován na poskytnutá data. Výchozí hodnotou je *None*, což znamená, že nebude využit žádný samoopravný kód.

Stego-key Uživatelsky definovaný stego-key, který se může skládat z jakýchkoli tisknutelných znaků. Slouží k vygenerování permutace, která určuje

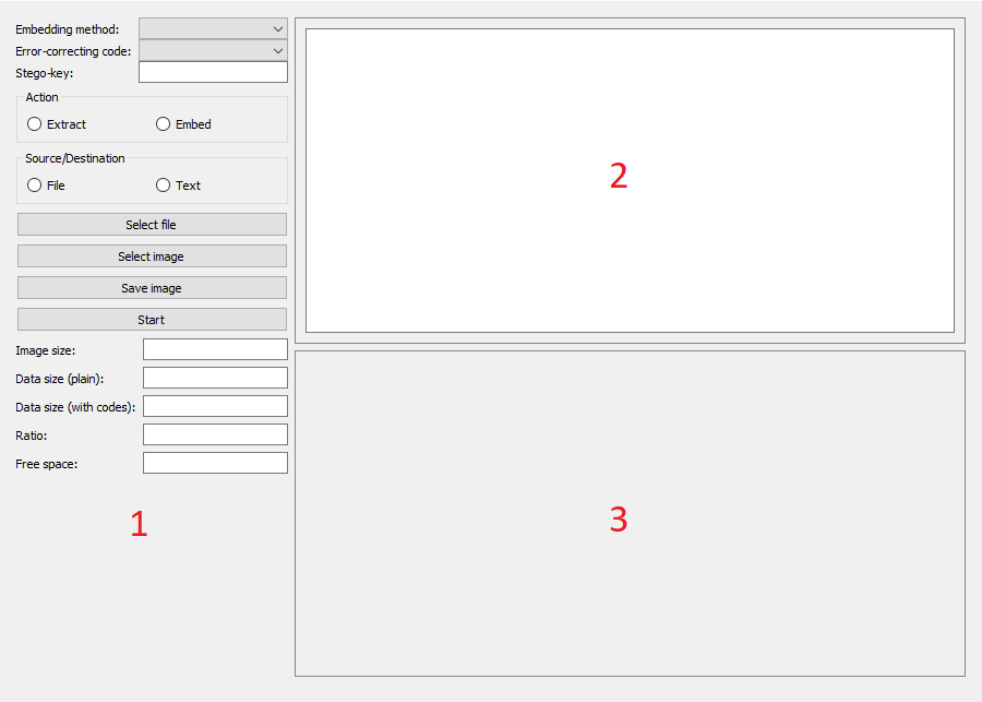
rozmístění bitů po obrázku. V případě ponechání prázdného pole bereme jako výchozí hodnotu 0.

Action Umožňuje vybrat, zda se budou data do obrázku vkládat, nebo zda se budou extrahovat. Tento údaj je povinný.

Source/Destination V případě, že bude v action vybráno *extract*, pak tento údaj určuje, zda se budou extrahovaná data z obrázku ukládat do vybraného souboru, nebo do textového pole 2. V případě *embed* tento údaj určuje, zda se budou data určená pro vložení načítat ze souboru, nebo textového pole 2.

Textové pole (2) Uživatelský textový vstup v případě vkládání dat do obrázku, nebo výstup v případě extrakce z obrázku.

Obrazové pole (3) Náhled obrázku po vložení.



Obrázek A.1: Návod k použití aplikace

Příloha B

Seznam použitých zkratek

SW Software

PNG Portable Network Graphics

GIF Graphics Interchange Format

BMP Windows Bitmap

TIFF Tagged Image File Format

DOS Disk Operating System

ASCII American Standard Code for Information Interchange

GUI Graphical user interface

XML Extensible markup language

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `images` ..... obrázky práce
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `attatchement` ..... přílohy práce