Mendelova univerzita v Brně
Provozně ekonomická fakulta

Aplikace pro ukládání utajených zpráv do formátů obrazových souborů

Bakalářská práce

Vedoucí práce:
Ing. Mgr. Jana Dannhoferová, Ph.D.

Jakub Hlaváč

Brno 2016

Rád bych touto cestou poděkoval vedoucí mé bakalářské práce Ing. Mgr. Janě Dannhoferové, Ph.D. za mnoho cenných rad, ochotu a trpělivost při řešení veškerých problémů, které se v průběhu práce vyskytly.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Aplikace pro ukládání utajených zpráv do formátů obrazových souborů**
vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*
Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.
Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 15. května 2016 .....

Abstract

HLAVAC, J. Application for secret messages storing into image file formats. Bachelor thesis. Brno, 2016.

The bachelor thesis describes creation of the Java application for saving secret messages into image file formats. Existing algorithms for message hiding are described in detail. Typical applications which are used for data hiding are compared and evaluated according to criteria. There are also attached source codes of the application.

Keywords

Data hiding, palette-based image, steganography, security, color quantization, image processing.

Abstrakt

HLAVÁČ, J. Aplikace pro ukládání utajených zpráv do formátů obrazových souborů. Bakalářská práce. Brno, 2016.

Bakalářská práce popisuje tvorbu aplikace v jazyce Java pro ukládání utajených zpráv do formátů obrazových souborů. Jsou zde popsány algoritmy, které se běžně využívají pro ukrytí dat do takových formátů. Dále práce zkoumá typické aplikace, které se využívají pro ukrývání dat do souboru. Tyto aplikace dále srovnává v rámci hodnocených kritérií. Součástí práce jsou i zdrojové kódy aplikace, která provádí ukrytí zprávy.

Klíčová slova

Ukrývání dat, paletový obraz, steganografie, bezpečnost, redukce barevného prostoru, zpracování obrazu.

Obsah

1 Úvod a cíl práce

1.1 Úvod

Komunikace je nedílnou součástí každé společnosti. Umožňuje předávání informací a myšlenek. Rozšíření informačních technologií vytváří způsoby komunikace, které dříve nebyly možné, ale současně poskytuje prostor mnoha novým hrozbám.

Bezpečnost dat se stala jedním ze základních kritérií komunikace ve všech odvětvích informačních technologií. Stále častěji se můžeme setkávat s pokusy o neoprávněný přístup k datům nebo o porušení přenášených informací mnoha různými způsoby. Aby se minimalizovalo riziko porušení dat, kombinuje se často mnoho různých bezpečnostních přístupů současně.

Základním bezpečnostním mechanismem, který se používá pro zabezpečenou komunikaci, je šifrování. Využití šifrování zabraňuje odhalit obsah přenášené zprávy, přestože šifrovanou zprávu máme k dispozici. Za dobu používání šifrování se již ale mnoho šifrovacích algoritmů podařilo překonat a již odhalení průběhu komunikace mezi dvěma subjekty může být nežádoucím únikem informace.

Jedním ze základních přístupů, který minimalizuje riziko odhalení je steganografie, která se snaží ukrýt samotnou probíhající komunikaci a tím i minimalizovat riziko, že se bude snažit neoprávněná osoba k datům přistoupit. Pokud by byla komunikace odhalena, většinou nebývá příliš náročné samotnou zprávu objevit. Proto se přenášená zpráva často zašifruje před ukrytím.

K ukrytí lze použít mnoho různých typů souborů. Často používanými strukturami pro ukrytí dat jsou obrazové soubory. Obrazová data jsou často přenášena přes internet a přenos takových dat tedy nevyvolává nežádoucí pozornost.

1.2 Cíl práce

Cílem této práce je návrh a tvorba aplikace, která bude schopna kódovat informace do palety barev grafických formátů. Aplikace bude umožňovat vybrat si výstupní formát, a to i v případě, že množství barev vstupního souboru bude rozsáhlejší než maximální počet barev souboru výstupního. V takovém případě budou využity algoritmy pro redukci barevného prostoru, aby bylo zachováno co největší množství informace v obraze.

2 Publikační rešerše

2.1 Pojem steganografie

Přestože zabezpečená komunikace většinou využívá pouze šifrování, zvyšující se tlak na bezpečnost vyžaduje využití dalších bezpečnostních principů. Ukrýváním informací do médií se zabývá věda nazývaná steganografie. Steganografie pochází z řeckých slov „stegos“ tedy skrytý a „grafia“ tedy psaní (Subhedar, Mankar, 2014). Obecně se využívá pro ukrývání dat do určitého souboru. Přiřazená data se dále přenášejí uvnitř souboru nejčastěji za účelem doručování utajených informací (Wu a kol., 2004). Steganografie je využívána většinou v digitální podobě, ale existují i metody, kde lze využít technik ukrývání informací i po vytištění na papír (Cheddad a kol., 2010).

Steganografie je často zaměňována s pojmem vodoznak. Pokud jsou ukryté informace uvnitř souboru zaměřeny na identifikaci, ochranu autorských práv nebo sledování transakcí, pak je správným pojmem tvorba vodoznaku (Satir, Isik, 2012).

Pro aplikování přístupů steganografie se běžně využívá mnoho různých metod závislých kromě typu média mnohdy i na samotném formátu a jeho přesné specifikaci. Většina známých technik je tak přizpůsobena nejčastěji používaným formátům na internetu, tedy formátům PNG, GIF a JPEG (Cheddad a kol., 2010).

Digitální steganografie nejčastěji využívá jako média pro ukrytí informace textové, obrazové, audio a video soubory (Roy, Venkateswaran, 2013).

2.2 Grafický souborový formát

Grafický souborový formát popisuje strukturu obrazových dat uložených v grafických souborech. Struktura uložených dat ovlivňuje, jak efektivně lze k datům přistupovat a zpracovávat je. Data uložená uvnitř grafických souborů mohou být rastrová nebo vektorová (Murray, Vanryper, 1997).

- Rastrová data se zobrazují ve formě sady pixelů. Každý pixel má hodnotu barvy, kterou reprezentuje (Murray, Vanryper, 1997). Grafický soubor rastrového obrazu se skládá z informací o velikosti obrazu, kódování barev a informací o kompresi grafického souboru, pokud je kompresní algoritmus formátem podporován (Martišek, 2002).
- Vektorová data obsahují informace o útvarech vytvořených z čar, atributech a pravidlech sloužících pro vykreslení dat (Murray, Vanryper, 1997). Z vektorových dat musí vhodný grafický editor dokázat vyčíst informace o tvaru a barvě objektu a tloušťce čáry, kterou je objekt vykreslen (Martišek, 2002).

2.2.1 Konverze grafických souborových formátů

Konverzí formátů je označována změna struktury souboru odpovídající jednomu formátu do struktury souboru jiného formátu. Podle rozdílnosti vstupního a výstupního

formátu se může jednat o elementární nebo velmi složitý proces (Murray, Vanryper, 1997).

2.2.2 Kompresní algoritmy grafických souborových formátů

Grafické soubory mohou zabírat mnoho místa v paměti počítače, a je proto vhodné, aby grafický souborový formát umožňoval ukládat obraz v komprimované podobě (Žára a kol., 2004). Některé grafické formáty využívají k uložení grafických dat kompresní algoritmy, které zmenšují výslednou velikost grafických souborů. Tyto algoritmy lze rozdělit na ztrátové a bezztrátové. Ztrátové algoritmy při ukládání ztrácí část informace. Bezztrátové algoritmy ukládají veškerá zdrojové informace a dosahují úspory prostoru až 50 % (Harman, 2012).

V rastrové počítačové grafice se lze nejčastěji setkat s následujícími kompresními metodami (Žára a kol., 2004):

- RLE neboli Run length encoding je bezztrátová kompresní metoda, která využívá opakujících se hodnot sousedních pixelů. Využívána bývá například u formátu PCX.
- CCITT neboli Huffmanovo kódování je bezztrátová kompresní metoda, která využívá různé bitové kódy pro reprezentaci symbolů. Délka bitového kódu bývá určena na základě četnosti výskytu symbolu. Využívána bývá například u formátu TIFF.
- LZW neboli Slovníkové kódování je obecná bezztrátová kompresní metoda, která nahrazuje vzory vstupních dat binárními kódy. Pro překlad je využívaný slovník. Délka slovníku může postupně narůstat. Využívána bývá například u formátů GIF, PNG, ZIP a RAR.
- DCT neboli Diskrétní kosinová transformace je ztrátová kompresní metoda, která je schopna efektivně pracovat s obrazy, které mají sousední pixely s podobnými, ale rozdílnými barvami. Nejznámějším případem využití této metody je formát JPEG.

2.3 Reprezentace obrazu v počítačové grafice

V počítačové grafice lze rozlišit různé typy obrazových oblastí. Obraz v odlišné oblasti může usnadnit některé prováděné operace. Mezi oblastmi lze převádět obrazy například pomocí různých typů Fourierových transformací. Základní oblast počítačové grafiky je označována jako prostorová oblast. V této oblasti je obraz představován maticí pixelů. Frekvenční oblast je reprezentací Fourierova obrazu a je představována sadou sinusových funkcí. Jednotlivé sinusové funkce mohou mít rozdílnou amplitudu a mohou být fázově posunuty. Dále bývá rozlišována i oblast časová, kterou lze získat z oblasti frekvenční (Žára a kol., 2004).

2.4 Nejdůležitější metody steganografie v digitálním obraze

2.4.1 Steganografie využívající souborový formát

Steganografie využívající souborový formát obsahuje jedny z nejzákladnějších metod, které lze pro steganografii uplatnit. Typickým případem takové metody je využití příznaku konce souboru ve formátu. Za tento příznak jsou vloženy další informace, které se již při běžném zobrazení grafickým editorem neprojeví. Metoda je přesto snadno odhalitelná již při zobrazení souboru v základním textovém editoru ve formě prostého textu (Cheddad a kol., 2010).

2.4.2 Steganografie využívající obrazovou prostorovou oblast

Metody využívající obrazovou prostorovou oblast ukládají ukrývaná data přímo do pixelů původního obrazu. Lze sem zařadit patrně jednu z nejznámějších metod obrazové steganografie využívající paletu barev, ve které bývá utajovaná zpráva uložena do nejméně významného bitu v bajtu (Chaumont a kol., 2013).

V případě využití metody nejméně významného bitu je nutné zvolit, kolik posledních bitů je možné zkreslit. Při využití vyšších bitů pro uložení ukrývané informace je pravděpodobné, že zkreslení obrazu může napomoci odhalit ukrytá data. Zkreslení obrazu a množství uložitelných dat zde tedy působí proti sobě (Cheddad a kol., 2010). Tento princip ukrytí lze v některých případech odhalit pomocí vhodných programů a algoritmů (Lin, 2014).

2.4.3 Steganografie využívající obrazovou frekvenční oblast

Steganografie frekvenční oblasti ukrývá utajenou zprávu do koeficientů frekvenčně orientovaných mechanismů. Pro toto ukrytí lze využít například koeficienty diskrétní kosinové transformace (DCT), diskrétní vlnkové transformace (DWT), Z transformace nebo Fresneletovi transformace (FT) (Uma Maheswari, Jude Hemanth, 2015).

Metoda založená na frekvenční oblasti je nejčastěji využívána u formátu JPEG. Je založená na modifikaci kvantizační tabulky a kvantizovaných koeficientů diskrétní kosinové transformace (Wang a kol., 2013). Ukrývané informace jsou tak po bitech vkládány do vhodných frekvenčních koeficientů. Tím nastává modifikace kvantizační tabulky. Pokud je s koeficienty pozorně manipulováno, nevzniká žádná viditelná změna pro identifikaci provedených úprav (Cheddad a kol., 2010).

2.4.4 Adaptivní steganografie

Tento přístup využívá již zmíněné běžné metody steganografie na vhodnou část obrazu, ve které budou provedené změny těžko odhalitelné. Vhodná oblast je určena pomoci statistických metod a na základě výsledků jsou modifikovány DCT nebo LSB koeficienty. Tímto přístupem se metoda vyhýbá například úpravě oblastí s jednotnou barvou (Cheddad a kol., 2010).

2.5 Barva v počítačové grafice

2.5.1 Barevná hloubka

Počet barev, kterých může každý pixel rastrového obrazu nabývat, je dán barevnou hloubkou obrazu. Výsledný počet barev lze dopočítat podle vztahu 2^n, kde n je počet bitů barevné hloubky obrazu. Pokud má obraz například jednobitovou barevnou hloubku, každý pixel může nejčastěji nabývat hodnoty černé nebo bílé barvy. Vzniká tak monochromatický obraz (Martišek, 2002).

2.5.2 Barevný prostor

Barevné prostory definují, kterými složkami se barva využívaná při tvorbě obrazu popisuje. Nejčastěji se rozlišují následující prostory (Žára a kol., 2004):

- RGB: využívá intenzity červené, zelené a modré barvy popsané rozsahem 0 až 1 nebo 0 až 255.
- RGBA: využívá barevný prostor RGB doplněný o kanál průhlednosti.
- CMY, CMYK: je popsán tyrkysovou, fialovou, žlutou a někdy i černou barvou, ale na rozdíl od RGB využívá subtraktivní skládání barev.
- HSV: využívá parametry prostoru: barevný tón, sytost a hodnotu jasu.
- HLS: pro popis barvy využívá barevný tón, světlost a sytost.

2.5.3 Redukce barevného prostoru

Redukce barevného prostoru je proces, při kterém je redukováno množství rozdílných barev v obraze. Současně je žádoucí, aby upravený obraz byl co nejpodobnější zdrojovému obrazu. Výhodou tohoto procesu je snížení množství paměti, které bývá potřeba pro uchování souboru (Yue a kol., 2014). Dalším častým využitím těchto metod jsou případy, kdy je nutné zobrazit obraz na zařízeních, které mají omezené množství zobrazitelných barev (Pérez-Delgado, 2015).

Redukcí barevného prostoru se ztrácí část barev. Takové barevné omezení může být v některých případech příliš patrné. Pro minimalizování takového efektu se v počítačové grafice využívají metody založené na vlastnostech lidského zraku, tedy metody polotónování a rozptylování (Žára a kol., 2004).

2.5.4 Redukce barevného prostoru paletového obrazu

Paletová obrazová data jsou charakterizována paletovou tabulkou a indexovaným obrazem (Niraimathi a kol., 2012). Paletová tabulka slouží pro překlad indexované hodnoty na vhodnou RGB barvu. Indexovaný obraz obsahuje index pro každý pixel, který je následně v paletě dohledáván a překládán na barvy (Yue a kol., 2014).

Proces redukce barevného prostoru paletového obrazu může být rozdělen do dvou fází. V první fázi je třeba sestavit paletu obsahující omezené množství barev. Dále je nezbytné přiřadit každému pixelu obrazu barvu z nové palety (Yu a kol., 2007). Metody sestavení nové palety barev mohou být obrazově závislé a obrazově nezávislé.

- Obrazově nezávislé metody využívají předem definovanou paletu, která není závislá na obrazu samotném. Barvy nacházející se v paletě tak nemusí být zcela vhodné pro konkrétní obraz (Yue a kol., 2014).
- Obrazově závislé metody tvorby palety jsou vhodnější, pokud je třeba dosáhnout co nejmenší odchylky barev u nového obrazu. V takovém případě je nově sestavená paleta specificky generována pro daný obraz. Pro sestavení vhodné palety se nejčastěji využívá preclustering nebo postclustering. Preclustering rozděluje obraz do více podprostorů na základě rozdělení barev. Strategie postclusteringu je založena na iterativním zlepšování počáteční palety. Preclustering není ve srovnání s postclusteringem tolik složitý, ale dosahuje horších výsledků (Yue a kol., 2014).

2.6 Nejběžnější rastrové grafické formáty

2.6.1 GIF

GIF je formát určený pro rastrovou grafiku. Ve své nejnovější verzi podporuje základní možnosti průhlednosti i animace. Pro uchování informací o barvách využívá paletu barev odkazovanou pomocí indexů. Počet barev této metody je maximálně 256 (W3.org, 1990). Pro efektivní kompresi dat využívá metodu LZW (Žára a kol., 2004).

2.6.2 TIFF

TIFF je označení pro formát využívaný v počítačové grafice. Účelem tohoto formátu je popsat a ukládat rastrová obrazová data. Hlavní výhodou tohoto formátu při srovnání s konkurencí je možnost vytvářet vícestránkové soubory. Pro kompresi může být využita bezztrátová metoda LZW (Adobe.com, 1992). Dovoluje také využít dlaždicové uspořádání dat, které umožní práci s rozsáhlými soubory i na menších zobrazovačích a je tedy vhodný pro zpracování profesionálních barevných výstupů (Žára a kol., 2004).

2.6.3 PNG

Portable network graphics neboli PNG je bezztrátový, přenositelný, dobře komprimovaný rastrový grafický formát využívaný v moderní počítačové grafice. Tento formát je považován za náhradu grafického formátu GIF a v mnoha případech i formátu TIFF. PNG využívá bezztrátovou deflate/inflate kompresi s posuvným oknem

(sliding window). Z hlediska uložení informací o barvách lze rozlišit tyto základní typy souboru PNG (W3.org, 2003b):

- Obraz plných barev (true color): každý pixel obsahuje hodnotu červené, modré a zelené barvy.
- Obraz ve stupních šedi: každý pixel obsahuje hodnotu šedé barvy.
- Obraz plných barev (true color) s alfa kanálem: každý pixel obsahuje hodnotu červené, modré a zelené barvy a v kanále alfa i hodnotu průhlednosti.
- Obraz ve stupních šedi s alfa kanálem: každý pixel obsahuje hodnotu šedé barvy a v kanále alfa i hodnotu průhlednosti.
- Obraz s indexovanými barvami: informace o barvě je uložena v oddělené paletě a každý pixel obsahuje index do palety. Množství barev uchovávaných touto metodou je omezeno, protože paleta může dosahovat maximálně 256 položek.

2.6.4 BMP

Formát BMP je obrazový rastrový formát podporující barevnou hloubku 1, 4, 8, 16, 24, 32 bitů. Podporuje využití indexované barevné palety do rozsahu 256 barev. V případě využití 4 nebo 8 bitových barevných obrazů je využívána RLE bezeztrátová komprese (Miano, 1999).

2.6.5 TGA

TGA neboli Targa je formát umožňující práci s obrazy s barevnou hloubkou 32 bitů a podporuje i barevný kanál alfa. Hlavní výhodou formátu je možnost ukládání dat do souboru po jednotlivých pixelech (Žára a kol., 2004).

2.6.6 JPEG JFIF

Obecně se pojmem JPEG označuje kompresní grafický formát. JPEG je ale pouze metoda ztrátové komprese. Skutečným názvem formátu je JPEG JFIF (W3.org, 2003a). JPEG je dnes nejpoužívanější grafický formát mezi běžnými uživateli. Příkladem běžného využití mohou být digitální fotoaparáty (Solomon, Breckon, 2011). Tento formát je ztrátový a využívá kompresní algoritmus DCT (Murray, Vanryper, 1997). Metoda DCT odstraňuje podobné pixely za cenu ztráty informace (Harman, 2012).

2.6.7 PSD

PSD je nativní grafický formát využívaný aplikací Photoshop. Uchovává informace o barvách ve vysoké kvalitě a je tedy vhodný pro zpracovávání kvalitních obrazových dat (Harman, 2012). Ve struktuře formátu jsou uloženy i informace o vrstvách a maskách obrazu (Adobe.com, 2013).

3 Metodika

Pro vývoj aplikace pracující s grafickými formáty bylo třeba zvolit, které formáty má práce zpracovávat. Na vstupu může aplikace zpracovat grafický obrazový formát libovolné struktury, ale výstupní grafický formát musí obsahovat indexovanou paletu barev. Proto byl pro výchozí specifikaci podporovaných formátů využit průzkum provedený v rámci publikační rešerše, který shrnuje základní formáty moderní počítačové grafiky. Základními vstupními formáty byly tedy zvoleny JPEG, GIF, PNG, TIFF, BMP, PSD a jako základní výstupní formáty byly určeny GIF, PNG, BMP, TIFF.

Aplikace musí do palety grafických obrazových formátů ukrýt vybraná data. V publikační rešerši jsou popsány nejběžnější algoritmy, které dokážou data do obrazových formátů ukrýt. Z těchto algoritmů byl tedy zvolen algoritmus LSB pro jednoduché a současně efektivní ukrytí zprávy.

V průběhu analýzy byly vybrány nejběžnější aplikace, které se dnes využívají pro ukrývání informací do souborových formátů. Na základě požadavků na běžné aplikace rozšířené o bezpečnostní vlastnosti se stanovila kritéria a jejich váhy, podle kterých se aplikace srovnávaly. Aplikace byly ohodnoceny a na základě získaných výsledků byl získán přehled podporovaných funkcí a přístupů, které jsou využívány v současných aplikacích. Tento přehled doplnil tvořenou aplikaci o klíčové funkce jako podpora šifrování, multiplatformnost a podpora komunikace v českém jazyce. Tvořená aplikace tak bude schopna lépe konkurovat současným řešením.

V úvodu návrhu aplikace byl popsán navržený algoritmus ukrytí zprávy. Dále byly pro tvorbu aplikace vzhledem k požadavku multiplatformnosti i zkušenostem autora zvoleny programovací jazyk Java a vývojové prostředí NetBeans. Následně se provedla vzhledem k rozsahu aplikace dekompozice na menší problémy. Pro každý nalezený problém bylo individuálně navrženo vhodné řešení. Toto řešení bylo textově popsáno a demonstrováno částmi kódu ve zvoleném jazyce.

Při implementaci aplikace se vyskytly problémy, které bylo nezbytné vyřešit. Jednotlivé problémy byly popsány a doplněny o řešení, která v práci autor využil.

Závěrem byla vytvořená aplikace otestována. Testovány byly postupně nejběžnější typy grafických souborů, které dnes počítačová grafika využívá a základní bezpečnostní prvky aplikace.

4 Analýza

V rámci práce budou srovnány dostupné steganografické aplikace. Nejprve je třeba určit, které aplikace budou do srovnání zařazeny. Snahou analýzy je zkoumat různé typy aplikací. Toho bude dosaženo pomocí určení základních skupin aplikací zabývající se steganografií. Aplikace tak budou rozděleny do skupin podle typu vstupních a výstupních formátů. Následně budou pro každou skupiny identifikovány typické aplikace, které budou zařazeny do srovnání. V rámci analýzy je věnována největší pozornost aplikacím pracujícími s obrazovými formáty, protože taková aplikace bude i výstupem této práce.

Vybrané aplikace je třeba porovnat. Kritéria, která budou při srovnání pozorována, jsou uvedena v tabulce 1.

Tabulka 1: Vybraná kritéria testovaných aplikací

Číslo	**Kritérium**	**Váha**
1	Vzhled a uživatelská přívětivost	0,1
2	Množství vstupních formátů	0,2
3	Typy výstupů	0,2
4	Podpora šifrování	0,2
5	Multiplatformnost	0,2
6	Ukrývaný typ dat	0,1

Kritéria byla volena podle nejčastěji pozorovaných vlastností u běžných aplikací a následně doplněna o kritéria, která jsou specifická pro bezpečnostní aplikace. Každé kritérium má svoji váhu. Tato váha byla snížena pro srovnání vzhledu aplikací, protože u tohoto typu aplikací nemá příliš propracované uživatelské prostředí velký význam. Většinou bývá vyžadováno pouze pole pro vyplnění vstupního souboru, pole pro ukrývaná a zobrazovaná data a případně konfigurace šifrovacího klíče. Grafické soubory nezvládnou ukrýt příliš velké množství dat, a proto jsou většinou ukrývána pouze textová data. Proto byla snížena i váha kritéria ukrývaný typ dat. Ostatní kritéria byla vyhodnocena jako srovnatelná a mají tedy stejnou váhu.

V rámci analýzy budou přidělovány body v rozsahu od 0 do 5 podle úrovně splnění požadavků. Podmínky pro získání maximálního počtu bodů v daných kategoriích jsou popsány v následujícím seznamu:

- Vzhled a uživatelská přívětivost: hodnocení bude určeno převážně subjektivními dojmy autora práce. Body budou udělovány především za přehlednost a intuitivnost uživatelského prostředí, dále bude možné získat body za informování uživatele o průběhu činnosti aplikace a absenci nutnosti instalace aplikace.
- Množství vstupních formátů: aplikace získá za každý vstupní formát, který bude podporovat, 1 bod.

- Typy výstupů: v této kategorii rozhodne o počtu získaných bodů počet výstupních formátů a možnost uložit výstupní soubor s ukrytými daty v jiném formátu než byl vstupní soubor.
- Podpora šifrování: aplikace se ohodnotí podle možnosti šifrovat data před ukrytím do souboru.
- Multiplatformnost: aplikace získá množství bodů podle počtu podporovaných platforem.
- Typ ukrývaných dat: hodnocení se určí podle různorodosti typů dat, které dokáže aplikace ukrýt. Pokud aplikace umožní uložit pouze textová data, získá 1 bod. Pokud uloží libovolná data, která se převedou na posloupnost bajtů, získá 5 bodů.

Výstupem analýzy bude tabulka s vypočtenými hodnotami pro každou aplikaci. Hodnoty budou získány následujícím postupem:

1. Vynásobení počtu získaných bodů v kategorii s váhou dané kategorie uvedené v tabulce 1.
2. Součet všech takto získaných hodnot pro každou aplikaci zvlášť.

Výsledné hodnoty tak budou nabývat od 0 do 5 bodů. 0 bodů získají aplikace, které nejhůře splňují zadaná kritéria, a 5 bodů znamená dokonalé splnění všech kritérií. Takto ohodnocené aplikace lze snadno seřadit a porovnat mezi sebou ve všech vybraných kategoriích.

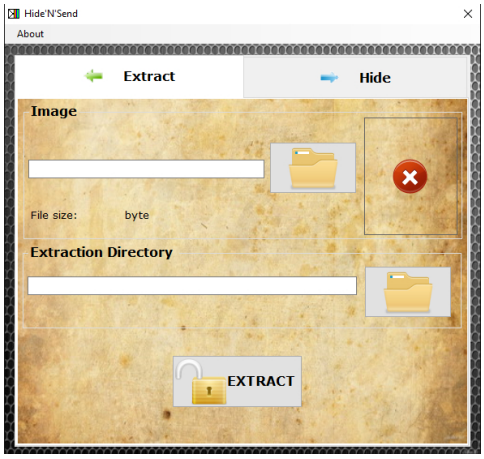
4.1 Steganografické aplikace grafických formátů

Tyto aplikace ukrývají data do obrazových formátů různých typů. Obecně závisí na konkrétním algoritmu a také na struktuře konkrétního formátu, kolik dat dokáže zvolený formát ukrýt. Nekompresní obrazové formáty obsahují velký prostor pro ukrytí zpráv, ale současně mohou být častěji prověřovány algoritmy stegoanalýzy (Wu a kol., 2004). Například soubory PNG se mohou vyskytovat ve verzi s indexovanou barevnou paletou, ale mohou také uchovávat barvy odděleně u pixelů (W3.org, 2003b). Tento rozdíl může vést ke zcela odlišným možnostem ukrytí u souborů se stejným formátem.

4.1.1 Aplikace pracující s jedním obrazovým formátem

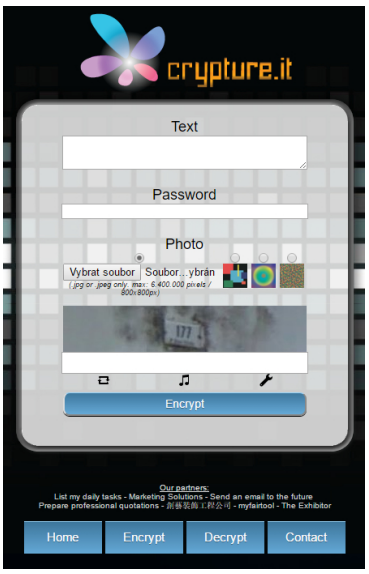
Tato skupina aplikací se specializuje na ukrytí dat do jednoho obrazového formátu. Takové aplikace se snaží maximálně využít potenciál souboru, do kterého data ukrývají. Zásadní nevýhodou je ale nutnost dodat obraz v požadovaném vstupním formátu, případně jej převést jinou dostupnou aplikací.

- Gifshuffle: aplikace, která využívá uspořádání barevné palety barev grafického formátu GIF. Tuto aplikaci se autor rozhodl zmínit především pro unikátní schopnosti práce s paletou barev. Do obrazu s maximálním počtem barev umožňuje uložit až 210 bajtů dat pomocí změny pořadí barev v barevné paletě. Pro kompresi dat může využít Huffmanovo kódování. Software je volně k dispozici včetně zdrojových kódů (Kwan, 2010).
- Hide'N'Send: program pracující s digitálními obrazy ve formátu JPEG. Podporuje využití šifrovacích algoritmů AES, RC4, RC2 a pro ukrytí dat steganografické algoritmy F5 a LSB (Soft32.com, 2012).



Obrázek 1: Prostředí aplikace Hide'N'Send

- Crypture: online aplikace pro uložení textových dat do formátu JPEG. Data mohou být před uložením zašifrována heslem (Crypture.it, 2013). Vzhledem k webové podobě aplikace je vzhled prostředí narušován reklamou. Při každém ukrytí je uživatel nucen přepsat znaky z obrázku nebo zvukový záznam pro ověření uživatele.

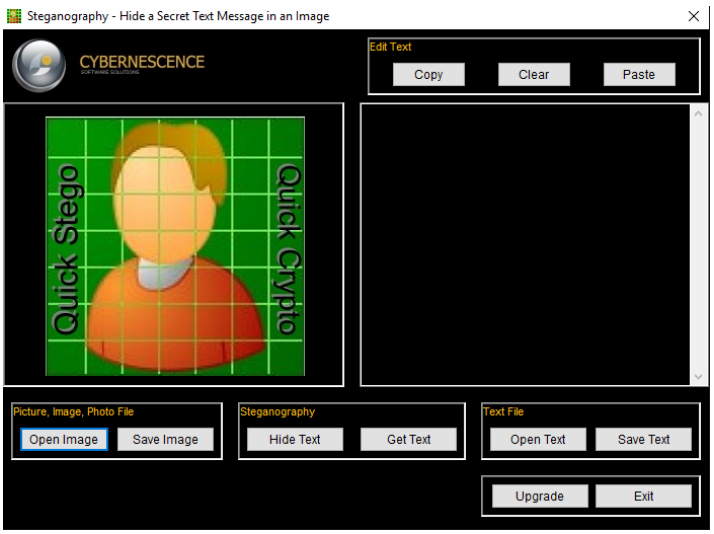


Obrázek 2: Prostředí aplikace Crypture

4.1.2 Aplikace pracující s více obrazovými formáty

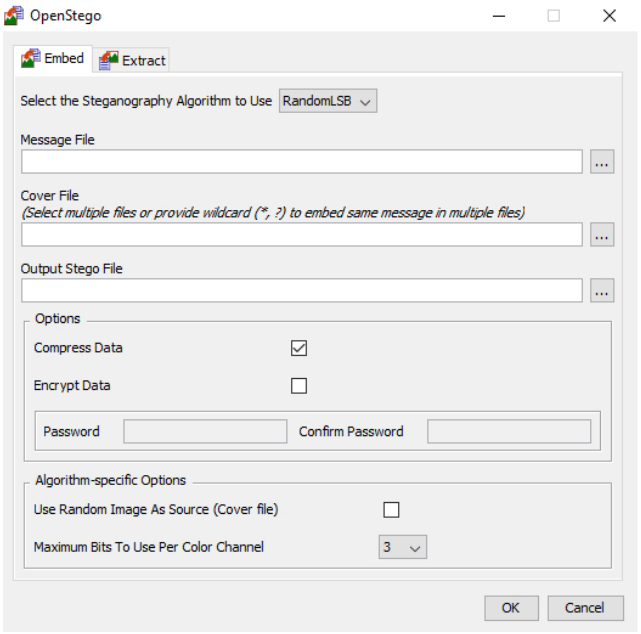
Tato skupina aplikací je schopna zpracovávat více vstupních formátů a není zde tedy tak výrazné omezení na vstupní soubor. Množství dat, které lze ukrýt, se liší podle typu a velikosti vstupního souboru. Některé z tohoto druhu aplikací umožňují zvolit formát výstupního souboru odlišný od vstupního.

- QuickStego: volně dostupný software určený pro obrazovou steganografii. Umožňuje načíst data ze souborů typu BMP, GIF nebo JPG/JPEG. Následně ukryje do načtených dat textovou zprávu a umožní uživateli vytvořit výsledný soubor typu BMP. QuickStego nepodporuje šifrování ukrývané zprávy (Quickcrypto, 2015).



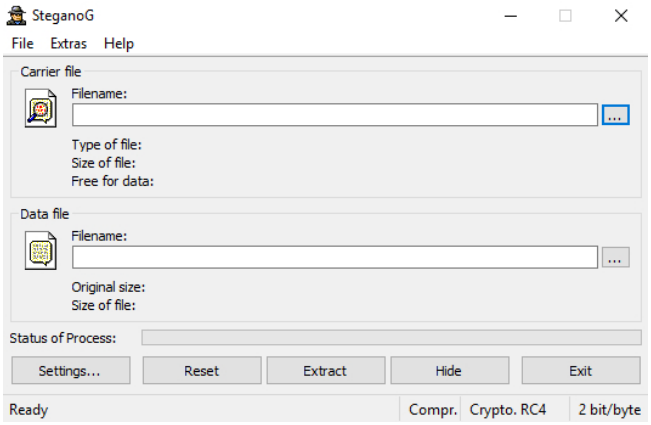
Obrázek 3: Prostředí aplikace QuickStego

- OpenStego: multiplatformní steganografický software napsaný v jazyce Java pro ukrývání dat a tvorbu vodoznaků. Umožňuje šifrovat informace na základě algoritmu DES. Pro ukrývání informací využívá algoritmus RandomLSB a pro tvorbu vodoznaku Dugadův algoritmus. Aplikace je dostupná pod licencí GNU General Public License 2.0 (GPL) (Vaidya, 2015). Příkladem podporovaných vstupních souborů jsou JPEG, GIF, BMP nebo PNG. Výstupní soubor je vždy ve formátu PNG.



Obrázek 4: Prostředí aplikace OpenStego

- Portable SteganoG: aplikace pro zašifrování souborů heslem a následné ukrytí dat do obrazu ve formátu BMP (Softpedia.com, 2013). V nastavení aplikace lze vybrat 1 z 5 šifrovacích algoritmů, zvolit ze 3 jazyků (čeština není podporována) a případně vypnout kompresi dat. Při testování aplikace byla nalezena i podpora formátu RAW, ale testovací obrazový soubor byl ukrytím dat upraven nežádoucím způsobem. Ukrytá data se povedlo získat, ale nosný obraz nešel zobrazit v běžných prohlížecích nástrojích systému Windows, ve kterých původně zobrazit šel.

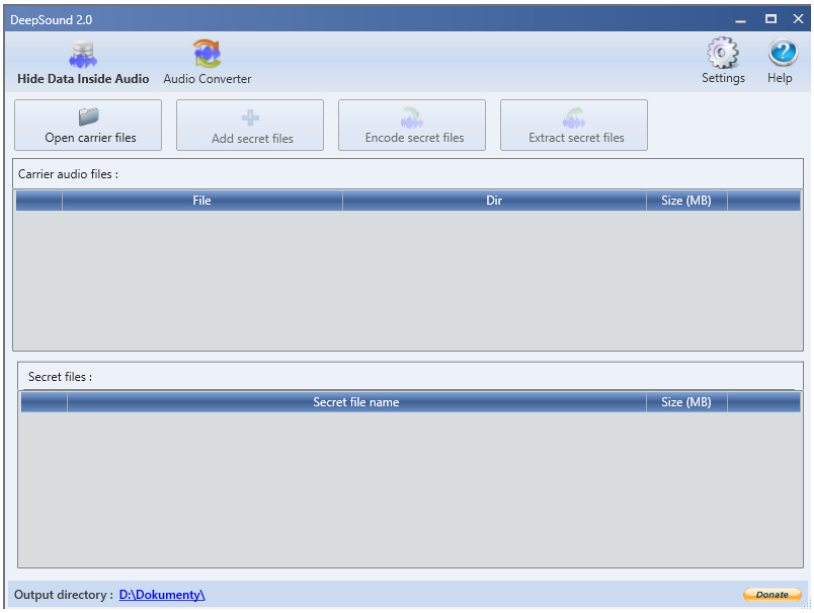


Obrázek 5: Prostředí aplikace SteganoG

4.2 Steganografické aplikace audio formátů

Aplikace ukrývající data do audio formátů mohou využívat mnoho přístupů. Data mohou být ukryta například do frekvencí nepostřehnutelných lidským sluchem (Subhedar, Mankar, 2014).

- DeepSound: software určený pro ukrývání utajených souborů do audio formátů. Umožňuje na vstupu zpracovat formáty FLAC, MP3, WMA, WAV a APE. Následně uschová do audio souboru libovolná data uložená v souboru a vytvoří výstupní soubor ve formátu FLAC, MP3, WAV, APE (Bátora, 2016).

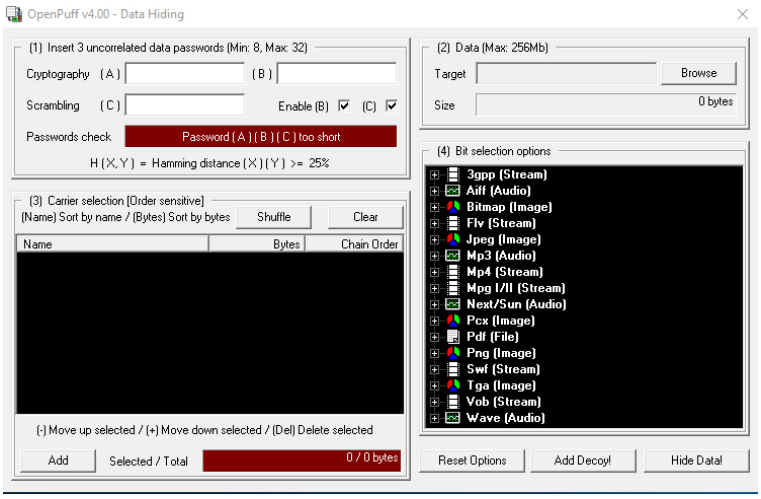


Obrázek 6: Prostředí aplikace DeepSound

4.3 Univerzální steganografické aplikace

Aplikace obsažené v této sekci umožňují zpracovávat různé typy formátů. Pro ukrývání dat digitální steganografií se většinou využívají grafické, audio a video formáty (Satir, Isik, 2012). Tento typ aplikací je pro uživatele patrně nejsnáze použitelný. Střídáním různých typů souborů se také může snížit riziko odhalení.

- OpenPuff: OpenPuff je steganografický freeware software určený pro operační systém Windows. Umožňuje šifrovat i ukrývat libovolná data do obrazových (BMP, JPG, PCX, PNG, TGA), audio (AIFF, MP3, NEXT/SUN, WAV), video (3GP, MP4, MPG, VOB) a Flash-Adobe formátů (FLV, SWF, PDF). Do výstupního souboru lze ukrýt i více vstupních souborů omezených celkovou velikostí a kapacitou výstupního souboru (Embeddedsw.net, 2015).



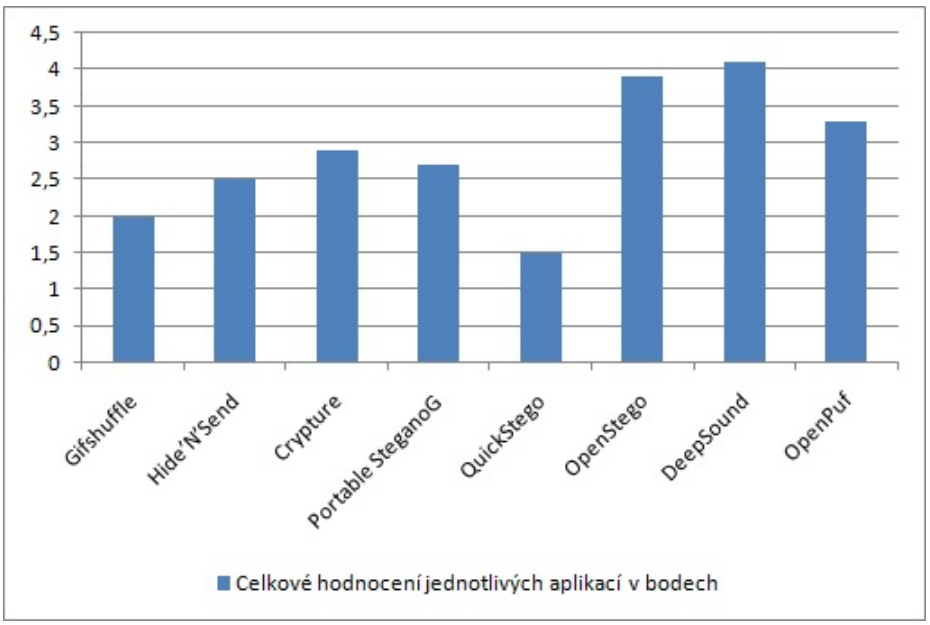
Obrázek 7: Prostředí aplikace OpenPuff

4.4 Výsledky analýzy

V tabulce 2 jsou zobrazeny výsledky jednotlivých aplikací srovnávaných v rámci této práce v bodech v rozsahu 0 až 5 bodů. Čísly v záhlaví jsou označeny jednotlivé kategorie uvedené v tabulce 1.

Tabulka 2: Výsledky analýzy aplikací

Vybraná aplikace	**1**	**2**	**3**	**4**	**5**	**6**	**Hodnocení aplikace**
Gifshuffle	1	1	1	5	2	1	2,0
Hide'N'Send	4	1	1	5	1	5	2,5
Crypture	4	1	1	5	5	1	2,9
Portable SteganoG	4	2	1	5	1	5	2,7
QuickStego	4	3	1	0	1	1	1,5
OpenStego	4	5	1	5	4	5	3,9
DeepSound	4	5	5	5	1	5	4,1
OpenPuff	4	5	1	5	1	5	3,3



Obrázek 8: Výsledky analýzy aplikací v grafické podobě

Z výsledků analýzy aplikací je patrné, že dnešní steganografické aplikace již mají přehledné uživatelské rozhraní. Prostředí je často intuitivní, ale problémem může být slabá jazyková podpora. Většina aplikací komunikuje pouze anglicky. Čeština nebyla podporovaná u žádné z testovaných aplikací. O průběhu zpracování dat není uživatel většinou informován a dostupná tak bývá pouze informace o úspěšném vytvoření souboru.

Novější aplikace již umožňují zpracovávat mnoho vstupních formátů různých typů. Problémy s kompatibilitou formátů během testování nenastaly a aplikace zpracovaly všechny vstupní formáty, které měly uvedeny na svých oficiálních stránkách. Jediným případem komplikací byla aplikace Portable SteganoG, která dokázala načíst a zpracovat soubor ve formátu RAW, ale při pokusu o otevření upraveného souboru původně funkční prohlížeč obrazů nový obraz neotevřel.

Rozšířeným nedostatkem aplikací je absence schopnosti převádět formát vstupního obrazu na odlišný výstupní formát. Pokud tato možnost již je, bývá většinou pouze 1 výstupní formát pro všechny typy vstupních formátů a uživatel tedy nemůže zvolit vyhovující. Pokud potřebuje uživatel získat obraz v určitém formátu, je nucen použít jinou aplikaci pro převod formátu obrazu a až následně zpracovat obraz steganografickou aplikací. Aplikace DeepSound dokázala během analýzy zvolit formátem výstupního souboru formát odlišný od vstupního souboru a uživatel si mohl vybrat vyhovující formát.

Všechny testované aplikace kromě aplikace QuickStego podporují šifrování nejméně jedním algoritmem. U aplikace QuickStego výrobce doporučuje v případě potřeby šifrovat zakoupit alternativní placený produkt QuickCrypto, který již tuto funkcionalitu umožňuje (Quickcrypto, 2015).

Všechny testované aplikace jsou dostupné pro operační systém Microsoft Windows. Pokud uživatel potřebuje pracovat na jiné platformě, tak je nabídka aplikací omezenější. Vhodné řešení v tomto případě představuje aplikace Crypture, která je k dispozici ve formě webové aplikace.

Dle zaměření použití umožňují aplikace uložit pouze textová data nebo libovolný soubor, který je následně převeden do posloupnosti bajtů a ukryt.

5 Návrh řešení

5.1 Programovací jazyk

Pro implementaci zadané aplikace byla zvolena Java. Java je platforma a objektově orientovaný programovací jazyk určený pro tvorbu multiplatformních aplikací. Podle cíle využití lze platformu Javu rozdělit na tyto skupiny (Pecinovský, 2012):

- Java 2 Standard Edition: platforma určená pro tvorbu desktopových aplikací,
- Java 2 Enterprise Edition: platforma určená pro tvorbu distribuovaných aplikací,
- Java 2 Micro Edition: platforma určená pro vývoj na mobilní zařízení,
- Java Card: platforma určená pro vývoj na čipové karty.

Java obsahuje vhodné třídy, které umožňují snadnou manipulaci s paletou obrazu. Třída ImageIO umožňuje zpracovávat na vstupu grafické soubory různých formátů, následně provést nezbytné úpravy a snadno vytvořit výstupní soubory jiného typu. Pro optimální vyřešení problému existuje mnoho externích knihoven, které mohou značně zjednodušit výslednou implementaci. Vzhledem k přirozeným vlastnostem Javy bude výsledná aplikace multiplatformní. Pro snadnou práci s externími knihovnami bude využit správce závislostí Maven.

5.1.1 Vhodné externí knihovny a zdroje kódu

- Jasypt, Bouncy Castle: Java knihovna Jasypt umožňuje jednoduše přidat do projektů Java široké možnosti šifrování bez nutnosti hlubokých znalostí kryptografie. Poskytuje uživateli možnost šifrovat například text, hesla, čísla nebo pole bajtů. Knihovna je v základním nastavení jednoduchá na použití, ale současně i vysoce konfigurovatelná. Tato knihovna je k dispozici pod licencí The Apache Software License, Version 2.0. Pro podporu moderních šifrovacích algoritmů je přidána knihovna Bouncy Castle, která je následně využita knihovnou Jasypt pro nastavení vhodného algoritmu (Fernández, 2014).
- TwelveMonkeys ImageIO: knihovna TwelveMonkeys je kolekce modulů a rozšíření pro Java třídu ImageIO. Účelem této knihovny je poskytnout podporu pro obrazové formáty, které JRE dosud nepodporovalo. Umožňuje tak pracovat s formáty jako PSD, TIFF, TGA (Kuhr, 2015).
- ImageJ: aplikace ImageJ slouží pro analýzu a úpravu obrazových dat. Podporuje mnoho obrazových formátů a algoritmů. Tato aplikace včetně zdrojových kódů je k dispozici v rámci licence public domain (Nih.gov, 2016).

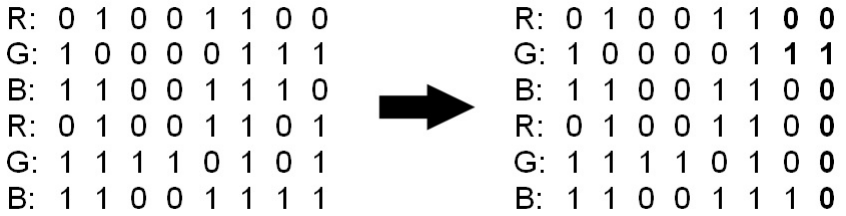
5.2 Vývojové prostředí

Aplikace bude vytvářena ve vývojovém prostředí NetBeans. NetBeans je volně dostupné vývojové prostředí podporující jazyky Java, HTML5, JavaScript, PHP, C/C++ a Groovy. Umožňuje intuitivní tvorbu základního grafického prostředí vyvíjené aplikace v grafickém průvodci a snadnou správu rozsáhlých projektů (Netbeans.org, 2016).

5.3 Algoritmus ukrytí zprávy

Pro ukrytí zprávy do grafických obrazových formátů bude využita metoda LSB. Tato metoda obsazuje nejméně významné bity, ale tím i postupně ztrácí původní informace o barvách obrazu. V případě nahrazení nejnižších bitů nenastává žádná patrná změna. Hodnota každé ze 3 barev, která je vyjádřená v rozsahu od 0 do 255, se změní maximálně o hodnotu 1.

V této práci bude využito i zkreslení vyšších bitů. Tyto změny mohou způsobit znatelnější změny obrazu, ale současně s ukrýváním zprávy vznikají v paletě i nově dostupné barvy. Vzhledem k využití indexované palety barev lze tyto nové barvy využít pro sestavení nového obrazu na základě původního vzoru. Toto řešení je nevhodné pro fotografie s větším počtem barev, ale v případě využití abstraktních obrazů nebo obrazů vyjadřujících kreslené scény lze získat relativně velký prostor pro ukrytí zprávy i v malém obrazovém souboru.



Obrázek 9: Princip ukrytí znaku A

K posloupnosti znaků ukryté podle postupu znázorněného na obrázku 9 je dále přidána za konec ukrývané zprávy posloupnost 8 bitových 0, která určuje konec zprávy. Prvních 8 bitů se dále obsadí hodnotou informující o počtu barev, které jsou využity pro ukrytí zprávy.

5.4 Struktura aplikace

Při řešení problému je možné rozdělit implementaci do následujících oddělených fází:

1. načtení a zpracování vstupního souboru;
2. tvorba nové palety při překročení maximálního počtu barev;
3. načtení a zpracování ukrývaných dat;

4. ukrytí dat do palety;
5. odkrytí dat z palety;
6. obnova původní grafiky s využitím nové palety;
7. uložení upraveného souboru;
8. návrh grafického uživatelského rozhraní.

Ke každé z těchto fází lze přistupovat samostatně a řešení jednotlivých problémů navrhnout odděleně.

5.4.1 Načtení a zpracování vstupního souboru

Java má implementované nástroje pro načtení různých typů obrazových souborů. Tyto nástroje lze nalézt ve třídě ImageIO. Metoda read načte požadovaný vstup a vytvoří instanci třídy BufferedImage. Instance třídy BufferedImage tak představuje načtený obraz. Načtení je provedeno následujícím příkazem:

```
ImageIO.read(new File(sourcePath));
```

V případě načtení vstupních souborů, které by obsahovaly grafickou paletu, by bylo možné přistoupit přímo ke složkám palety pomocí příkazu:

```
readedImage.getColorModel();
```

Aplikace ale musí zpracovat i soubory, které indexovanou grafickou paletu neobsahují. Následující metoda tedy projde všechny pixely obrazu a jejich barvu jednotlivě uloží do množiny barev. Množina je zde vybrána, aby se nevyskytovaly nadbytečné duplicitní hodnoty.

V případě načítání obrazů ve vysokém rozlišení může tato operace vyžadovat delší dobu na provedení. Pokud je například načten obraz v rozlišení FullHD, může počet kroků přidání barvy do množiny nabývat hodnoty 2 073 600.

```
LinkedHashSet colors = new LinkedHashSet<>();
for(int y = 0; y < readedImage.getHeight(); y++) {
    for(int x = 0; x < readedImage.getWidth(); x++) {
        int pixel = readedImage.getRGB(x, y);
        colors.add(pixel);
    }
}
```

Pro snadnější práci jsou takto získané složky rozděleny podle barvy do polí červených (red), zelených (green) a modrých (blue) barev. Hodnoty barev získané předchozí metodou jsou v datovém typu integer. Takto uložená hodnota uchovává údaje o 3 složkách barvy a případně i hodnotu průhlednosti. Z hodnoty barvy uložené v datovém typu integer lze přečíst jednotlivé složky pomocí bitových posunů. Každá složka barvy je uložena na prostoru 8 bitů, tedy 1 bajtu.

```
red = new byte[colorSet.size()];
green = new byte[colorSet.size()];
blue = new byte[colorSet.size()];

for (int i: colorSet) {
    this.red[j] = (byte) (i >> 16 & 0xff);
    this.green[j] = (byte) (i >> 8 & 0xff);
    this.blue[j] = (byte) (i & 0xff);
    j++;
}
```

Veškeré metody popisované v této části jsou implementovány ve třídě ColorImageReader.

5.4.2 Tvorba nové palety při překročení maximálního počtu barev

Pokud má zdrojový obraz maximálně 256 barev, jsou pro tvorbu nové palety využity veškeré barvy zdrojového obrazu. V případě vyššího počtu je nezbytné snížit počet barev. Pro redukci barev je možné využít obrazově nezávislé a obrazově závislé metody tvorby palety. V aplikaci je implementována jedna statická paleta. Tato paleta obsahuje 256 barev, které zastupují přibližně rovnoměrně různé odstíny základních barev. Vzhledem k využitému rozhraní lze jednoduše do aplikace přidat i další barevné kombinace.

Obrazově závislá metoda tvorby palety využívaná v této aplikaci vychází z implementace algoritmu Heckbert's median-cut dostupného pod licencí public domain na stránkách aplikace ImageJ (Nih.gov, 2016).

5.4.3 Načtení a zpracování ukrývaných dat

Data ukrývaná v rámci této aplikace mohou mít pouze textovou podobu. Tato varianta byla zvolena z důvodu využívání grafické palety, která není svým rozsahem vhodná pro ukrytí větších souborů a aplikace je tedy orientována na přenos kratších textových zpráv.

V případě vypnutí šifrování je využit pouze převod zprávy do bajtů na základě předdefinovaného kódování UTF-8. Aplikace proto dokáže uložit i veškeré české znaky. Pomoci přepsání konstanty v aplikaci je možné kódování v případě potřeby jednoduše změnit.

```
private static final String CHARACTER_ENCODING = "UTF-8";
byte[] savingTextBytes = null;
try {
    savingTextBytes = savingText.getBytes(CHARACTER_ENCODING);
} catch (UnsupportedEncodingException e) {
    return "Nepodporované kódování textu";
}
```

Pokud uživatel potřebuje zprávu zašifrovat, je v aplikaci k dispozici metoda využívající knihovnu Jasypt. Výstupní kódování šifrovaného textu je zvoleno BASE64. Toto kódování zajistí kompatibilitu s převodem textu do bajtů. Povolení šifrování je volitelné a v případě použití se prodlouží délka ukládané zprávy.

Pro zašifrování zprávy je v aplikaci k dispozici šifrovací algoritmus AES. Tento algoritmus je dnes používán mnoha šifrovacími aplikacemi. Je snadno implementovatelný a poskytuje vysokou bezpečnost. AES využívá šifrování symetrickým klíčem (Farooq, Aslam, 2016). V aplikaci je tedy využit stejný klíč pro zašifrování i dešifrování zprávy. Pokročilé šifrovací algoritmy jsou v rámci aplikace získány pomocí knihovny Bouncy Castle. Pro oficiální spuštění v rámci služby Oracle je nezbytné, aby byl JAR soubor podepsán specifickým certifikátem vydávaným firmou Oracle. Pokud tedy uživatel nevyužívá OpenJDK, je třeba spoléhat na zjednodušenou konfiguraci zabezpečení poskytovanou knihovnou Jasypt ve výchozím stavu. Toto omezení lze vyřešit podáním žádosti firmě Oracle.

Zašifrování zprávy je vzhledem k využití externích knihoven provedeno následujícími příkazy:

```
org.jasypt.encryption.pbe.StandardPBEStringEncryptor enc
    = new StandardPBEStringEncryptor();
enc.setProvider(new BouncyCastleProvider());

enc.setStringOutputType("BASE64");
enc.setAlgorithm("PBEWITHSHA256AND128BITAES-CBC-BC");
enc.setPassword(pass);
String outputData = enc.encrypt(sourceData);
```

Pro dešifrování textu lze použít upravený algoritmus. Pokud uživatel nezná heslo nebo zadá neplatné, dešifrování není možné provést.

```
org.jasypt.encryption.pbe.StandardPBEStringEncryptor enc
    = new StandardPBEStringEncryptor();
enc.setProvider(new BouncyCastleProvider());

enc.setStringOutputType("BASE64");
enc.setAlgorithm("PBEWITHSHA256AND128BITAES-CBC-BC");
enc.setPassword(pass);
String outputData = enc.decrypt(sourceData);
```

Pro implementaci dalších algoritmů šifrování textu jsou v aplikaci připravena rozhraní CodingAlgorithm a DecodingAlgorithm.

5.4.4 Ukrytí dat do palety

Aplikace využívá pro ukrytí dat upravenou verzi známého algoritmu, který obsazuje nejnižší bity barevných složek. Tento algoritmus je obecně označován LSB. Imple-

mentovaná verze umožňuje postupně obsadit veškeré bity původních barev. Přestože není doporučováno využít všechny bity pro ukrytí zprávy, je tato možnost povolena.

Obsazovány jsou hodnoty od nejnižšího bitu a následně je úroveň zvyšována až po nejvyšší 8. bit. Hodnoty barev jsou tedy zkresleny podle množství ukrývaných dat. V případě úplného přepsání barev sestaví aplikace obraz znovu z barev, které ukrytím zprávy vznikly.

Implementace ukrytí zprávy je dosaženo v několika následujících krocích:

1. Jednotlivé barevné složky jsou spojeny z důvodu snazší manipulace do jedné velké palety barev. Výsledné pole barev má tedy barvy uloženy v opakující se posloupnosti červených, modrých a zelených složek jednotlivých barev.

```
byte[] result = new byte[r.length * 3];
for(int i= 0; i < r.length; i++) {
    result[i * 3] = (byte) r[i];
    result[(i * 3) + 1] = (byte) g[i];
    result[(i * 3) + 2] = (byte) b[i];
}
```

2. Pro zjištění počtu barev, které jsou v paletě obsazeny ukrytými hodnotami, se do prvního bajtu přiřadí velikost palety, která se bude upravovat.

```
byte[] longerArray = new byte[savingTextInBytes.length + 1];
longerArray[0] = (byte) (colorCount - 1);
System.arraycopy(savingTextInBytes, 0, longerArray, 1,
                 savingTextInBytes.length);
savingTextInBytes = longerArray;
```

3. Dále je zpráva ukryta do sloučené palety barev. Algoritmus ukrytí prochází všechny znaky, které je nutné uložit. Následně umístí každý znak po bitu do vhodné pozice. Pokud již na současné úrovni bitů není volný prostor, přejde se na další bitovou úroveň. Metoda setBit je volaná uvnitř ukrývajícího algoritmu a upravuje vybraný bit v bajtu na požadovanou hodnotu. K tomuto převodu využívá bitové operace AND a negace. Přechod a následné úpravy na každé vyšší úrovni budou způsobovat stále větší zkreslení. Zvyšující se zkreslení je vyvoláno způsobem uložení hodnoty barvy. Každá barva je reprezentována 3 bajty. Každý z bajtů je složen z 8 bitů. Změnou každého vyššího bitu jsou postupně vyvolávány zvedající se změny barvy o hodnoty 1, 2, 4, 8, 16, 32, 64 a 128.

```
for (int charI = 0; charI < savingTextInBytes.length; charI++) {
    for (int bitIndex = 0; bitIndex <= 7; bitIndex++) {
        int byteIndex = (charI * 8) + bitIndex;
        result[byteIndex % (colorCount * 3)]
            = setBit(result[byteIndex % (colorCount * 3)],
```

```
                    ((1 << bitIndex) &
                      (savingTextInBytes[charI])) > 0, level);

            if (byteIndex % (colorCount * 3) == (colorCount * 3) - 1) {
                level++;
            }
        }
    }

    private byte setBit(byte input, boolean bitValue, byte bitNumber) {
        return (byte) (bitValue ?
            (input | (1<<bitNumber)) : (input & ~(1<<bitNumber)));
    }
```

4. Na poslední pozici je uloženo 8 bitových 0, které značí konec ukládané zprávy. V případě, že v minulém průběhu cyklu nastal přechod na další bitovou úroveň a nic zde nebylo zapsáno, je třeba se vrátit o jednu úroveň zpět.

```
if ((savingTextInBytes.length * 8) % (colorCount * 3) == 0) {
    level--;
}

for (int i = 0; i <= 7; i++) {
    int bitPosition = ((savingTextInBytes.length * 8) + i);
    if (bitPosition % (colorCount * 3) == 0) {
        level++;
    }
    result[bitPosition % ((colorCount * 3))]
    = setBit(result[bitPosition
        % ((colorCount * 3))], false, level);
}
```

5. Po ukrytí celé zprávy jsou hodnoty výsledného pole s ukrytou zprávou rozděleny do jednotlivých složek červené, zelené a modré barvy.

```
for (int i = 0; i < colorCount && i < 256; i++) {
    r[i] = result[i * 3];
    g[i] = result[(i * 3) + 1];
    b[i] = result[(i * 3) + 2];
}
```

5.4.5 Odkrytí dat z palety

Pro odkrytí zprávy je využit podobný postup jako při ukrytí. Po spojení barevných složek, ve kterých je ukryta zpráva, vzniká proměnná sumPalete se spojenou paletou.

Následně je tedy využit algoritmus, který z bitů obsazených zprávou složí původní posloupnost bajtů.

```
for (int charI = 0; charI < paletteSize * 3; charI++) {
    codedSumPalete[charI] = 0;
    for (int bitIndex = 0; bitIndex <= 7; bitIndex++) {
        codedSumPalete[charI] = (byte) (codedSumPalete[charI]
            | (readBit(sumPalete[((charI * 8) + bitIndex)
            % (paletteSize * 3)], bitIndex, level)));

        if ((charI * 8 + bitIndex) - (level * paletteSize * 3)
            == (paletteSize * 3) - 1) {
            level++;
        }
    }
    if (codedSumPalete[charI] == 0) {
        return Arrays.copyOfRange(codedSumPalete, 1, charI);
    }
}

private byte readBit(byte value, int position, byte level) {
    return (byte) ((value & (1 << level)) > 0 ?
        (1 << position) : 0);
}
```

5.4.6 Obnova původní grafiky s využitím nové palety

Data v této fázi jsou již ukryta v nových polích barev a aplikace tedy může vytvořit a uložit nový obrazový soubor. Z jednotlivých složek barev je složena barevná paleta. Pro vytvoření palety je zde využita instance třídy IndexColorModel, která reprezentuje paletu indexovaných barev v jazyce Java.

```
IndexColorModel palette
    = new IndexColorModel(8, itemCount, r, g, b, pixel);
```

Proces ukrytí vždy předpokládá, že s paletou bylo manipulováno. Nová paleta je tedy použita k nakreslení nového obrazu podle vzoru. Při absenci dané barvy se volí barva z palety, která je dané barvě nejblíže. Obraz je sestaven vytvořením nového objektu třídy BufferedImage. Při vytvoření je nezbytné specifikovat typ obrazu jako obraz indexovaných barev. V aplikaci je tento typ definován konstantou. Proces sestavení nových pixelů je proveden následujícím algoritmem:

```
final int IMAGE_TYPE = BufferedImage.TYPE_BYTE_INDEXED;

BufferedImage convertedImage
```

```
    = new BufferedImage(sourceBufferedImage.getWidth(),
    sourceBufferedImage.getHeight(), IMAGE_TYPE, palette);

for (int x = 0; x < sourceBufferedImage.getWidth(); x++) {
    for (int y = 0; y < sourceBufferedImage.getHeight(); y++) {
        convertedImage.setRGB(x, y, sourceBufferedImage.getRGB(x, y));
    }
}
```

5.4.7 Uložení upraveného souboru

Vytvořený BufferedImage představuje předlohu obrazu, kterou je třeba uložit do výstupního souboru aplikace. K uložení výsledného obrazu je využita třída ImageIO. Po zavolání metody write již Java podle požadavku na typ souboru a umístění vytvoří soubor v paměti počítače.

```
File out = new File(destPath + "." + fileTypeOut);
ImageIO.write(outputBufferedImage, fileTypeOut, out);
```

5.4.8 Návrh grafického uživatelského rozhraní

Aplikace bude využívat pro snadné použití přehledné grafické rozhraní. Toto rozhraní je vytvářeno v grafickém průvodci vývojového prostředí NetBeans. Pro přehledné oddělení jednotlivých funkcí aplikace budou jednotlivé funkční celky rozděleny do sekcí, které budou umístěny v jednotlivých kartách grafického rozhraní. Při tvorbě rozhraní je především zohledněna jednoduchost, přehlednost, intuitivnost a uživatelská přívětivost. Vzhledem ke zohlednění existujících vad zraku budou využity dostatečně kontrastní barvy. Ukrývaná a odkrývaná zpráva proto bude zobrazena černým textem na bílém pozadí.

Během analýzy steganografických aplikací nebyla objevena aplikace komunikující v českém jazyce. Aplikace vytvářená v této práci tedy bude obsahovat české uživatelské prostředí, které umožní snadnou manipulaci česky komunikujícímu uživateli. Pro uživatele, kteří nedisponují znalostí anglického jazyka, může být aplikace vhodnou alternativou mnoha současných aplikací.

Jednotlivé sekce, které představují funkční celky aplikace, jsou rozděleny do karet následujícím způsobem:

- Ukrýt data: tato karta obsahuje řízení vytváření nového souboru. Do pole vstupní soubor je zadána cesta k souboru pro ukrytí dat, do pole výstupní soubor název vytvářeného souboru. Dále je umožněn výběr algoritmu pro generování palety. Pole text pro ukrytí je určeno pro zprávu, která má být vložena do souboru s obrazem. Vytvoření souboru je provedeno tlačítkem uložit.
- Odkrýt data: tato karta obsahuje pole pro vložení cesty k souboru s ukrytými daty, pole pro zobrazení odkrytých dat a tlačítko pro zahájení procesu odkrývání.

- Nastavit šifrování: karta obsahuje povolení nebo zakázání šifrování celé aplikace. Nastavení šifrování se tedy vztahuje pro vytvoření i čtení souboru. Textové pole na kartě slouží pro vložení šifrovacího hesla.
- Log: karta po stisku tlačítka obnovit zobrazí informace o průběhu vytvoření nebo čtení souboru. Informace zobrazené v logu nejsou ukládány z důvodu bezpečnosti aplikace na disk a po zavření aplikace zaniknou.
- O aplikaci: tato karta obsahuje informace o aplikaci, použitých knihovnách třetích stran a licencích, které se k nim vztahují.

6 Implementace a testování

Při implementaci aplikace se vyskytlo několik neočekávaných problémů, které bylo nutné vyřešit. Tyto problémy a následné vysvětlení využitých řešení jsou popsány v následující kapitole. Po vyřešení implementačních problémů autor práce přešel k testování aplikace a hledání dalších případných nedostatků.

6.1 Problémy při implementaci

V této části jsou popsány problémy, které byly v průběhu implementace řešeny.

6.1.1 Množství podporovaných grafických formátů třídou ImageIO

Během tvorby aplikace nastal problém s kompatibilitou třídy ImageIO s více typy formátů. Třída dokáže podle dokumentace zpracovat 5 základních formátů, a sice formáty JPEG, BMP, WBMP, PNG, GIF. Tento seznam zastupuje základní formáty obrazové počítačové grafiky, ale stále existují formáty, které jsou vhodným rozšířením a v seznamu obsaženy nejsou.

Typickým formátem vhodným pro rozšíření aplikace byl určen formát TIFF. Tento formát není dnes příliš využívaný, ale stále existují firemní oblasti, kde bývá vyžadován. Pro přidání podpory formátu byla nalezena knihovna TwelveMonkeys, která dokáže upravit třídu ImageIO do tvaru rozšiřujícího základní formáty. Za pomoci knihoven skupiny TwelveMonkeys tak bylo možné přidat i podporu dalších vstupních formátů IFF, PSD, ICNS, SGI, TGA, PICT, PCX, ICO. Podpora těchto formátů je u aplikací podobného typu výjimečná.

6.1.2 Výsledky třídy Graphics2D

Aplikace při své práci sestaví novou paletu, kterou je následně třeba využít k sestavení nového obrazu na základě původních obrazových dat. K tomuto účelu měla původně sloužit třída Graphics2D. Tato třída podporuje nastavení ditheringu, antialiasingu i dalších parametrů tvorby obrazu. Výstupy získané zavoláním metody drawImage ovšem nebyly vyhovující a i v případě, že byla použita originální paleta, byly v obrazu rozeznatelné změny již na první pohled.

V úvodu problému se autor pokusil změnit nastavení parametrů třídy. V obrazu se objevovaly nevhodné pixely v pravidelných vzorech, a tak byla vypnuta možnost ditheringu, která mohla podobné změny způsobovat. Výsledný výstup se ale neměnil při žádných změnách konfigurace. Bylo tedy třeba vytvořit alternativní algoritmus, který by dokázal znovu sestavit původní obraz z nové palety. Ve výsledném řešení je tedy využit algoritmus, který po pixelu přiřazuje hodnoty ze vstupního obrazu a v případě, že hodnota v paletě není, zvolí hodnotu nejbližší.

```
final int IMAGE_TYPE = BufferedImage.TYPE_BYTE_INDEXED;
```

```
BufferedImage convertedImage
    = new BufferedImage(sourceBufferedImage.getWidth(),
    sourceBufferedImage.getHeight(), IMAGE_TYPE, palette);

for (int x = 0; x < sourceBufferedImage.getWidth(); x++) {
    for (int y = 0; y < sourceBufferedImage.getHeight(); y++) {
        convertedImage.setRGB(x, y, sourceBufferedImage.getRGB(x, y));
    }
}
```

6.1.3 Zobrazení vytvořeného souboru ve formátu BMP

V aplikaci se povedlo sestavit společný algoritmus, který jednotným způsobem zpracoval tvorbu veškerých výstupních souborů. Výstupní soubory bylo možné uložit do libovolného souboru, ale při kontrole těchto výstupů nebylo možné zobrazit obsah obrazu ve formátu BMP. Následným zkoumáním byl odhalen výskyt problému u obrazů BMP, které nemají zcela zaplněnou paletu 256 hodnotami. Bylo tedy nutné vytvořit algoritmus, který doplní zbytek obrazu libovolnými barvami. Pro účely aplikace tedy byly doplněny složky opakující se barvou.

```
if (fileTypeOut.equalsIgnoreCase("BMP")) {
    byte[] largeR = new byte[256];
    byte[] largeG = new byte[256];
    byte[] largeB = new byte[256];

    System.arraycopy(r, 0, largeR, 0, r.length);
    System.arraycopy(g, 0, largeG, 0, g.length);
    System.arraycopy(b, 0, largeB, 0, b.length);

   palette = new IndexColorModel(8, 256, largeR, largeG, largeB, pixel);
}
```

6.1.4 Zobrazení vytvořeného souboru ve formátu TIFF

Při prohlížení výstupních souborů ve formátu TIFF nebylo možné v běžných prohlížečích fotografií operačního systému Windows 10 otevřít výstup vytvořený aplikací. Vytvořený soubor byl následně otestován v operačním systému Fedora 23. Běžně využívaný prohlížeč obrazů tohoto systému již výstupní soubor otevřít dokázal. Grafický editor Photoshop vytvořený TIFF dokázal otevřít také. Předpokládaným důvodem tohoto chování je mnoho rozdílných verzí formátu TIFF a možná nekompatibilita využité verze s prohlížečem fotografií systému Windows. Ukrytý text byl vždy obnoven ze souboru bez komplikací.

Knihovna TwelveMonkeys, která umožňuje zpracovávat tento formát v aplikaci, je ale stále ve vývoji a nemusí tedy vždy vykazovat optimální výsledky. Proto je doporučeno po vydání nové verze knihovny tuto verzi využít i v této aplikaci.

6.1.5 Algoritmus pro redukci barevného prostoru

Aby výsledné soubory aplikace byly co možná nejpřesnější kopií původního obrazu, byla pro sestavení nové palety vybrána i metoda pro obrazově závislou tvorbu palety. Metoda zvolená pro tuto aplikaci se nazývá Median Cut.

Autor se pokusil metodu implementovat sám v rámci této práce, ale bylo zjištěno, že tato metoda vyžaduje přesnou a dobře otestovanou implementaci, aby se předešlo dlouhým propočtům, které by mohly celou aplikaci velmi zpomalit. Tvorba a testování této metody by byla příliš náročná, a proto bylo třeba najít alternativní řešení. Z volně dostupných knihoven nebyla nalezena žádná, která by tuto funkcionalitu poskytovala. Vhodným řešením byla až třída projektu ImageJ, která daný algoritmus implementovala. Vzhledem k licenci typu public domain bylo možné tuto knihovnu upravit a použít její vhodnou podobu v této aplikaci.

6.1.6 Vytvoření spustitelného souboru

Funkčnost aplikace v průběhu implementace byla průběžně ověřována spouštěním přes vývojové prostředí NetBeans. Následně bylo nutné vytvořit a spustit soubor ve formátu JAR, který představuje vytvořenou aplikaci. Tento soubor ovšem nešel otevřít. Spuštěním v příkazovém řádku bylo zjištěno, že neobsahuje odkaz na hlavní soubor aplikace, který ji spouští. Také uvnitř souboru nebyly obsaženy externí knihovny využité v aplikaci, a proto ani úplná funkčnost nebyla možná.

Většina projektů, které využívají Maven, je sestavena s využitím maven-assembly-plugin. Tento plugin také fungoval na tuto aplikaci, ale vzhledem k implementaci třídy ImageIO bylo potřeba vytvořit v souborech javax.imageio.spi.ImageReaderSpi a javax.imageio.spi.ImageWriterSpi odkazy na všechny nové formáty, které byly doplněny pomocí knihoven. Tyto soubory jsou umístěny ve adresáři /META-INF/services/ výsledného souboru ve formátu JAR. Aby nemusely být odkazy ručně dopisovány do souborů, byl nalezen alternativní plugin maven-shade-plugin. Při následném sestavení projektu ve vývojovém prostředí se vytvořil i druhý soubor ve formátu JAR, který obsahoval správně umístěný odkaz na hlavní soubor, všechny knihovny využité aplikací a také odkazy na formáty poskytnuté knihovnami. Konfigurace pluginu maven-shade-plugin byla vložena na vhodné místo v souboru pom.xml. Pro tuto aplikaci byl plugin nastaven následovně:

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-shade-plugin</artifactId>
<version>2.4.3</version>
<executions>
    <execution>
        <phase>package</phase>
        <goals>
          <goal>shade</goal>
        </goals>
        <configuration>
            <filters>
                <filter>
                    <artifact>*:*</artifact>
                    <excludes>
                        <exclude>META-INF/*.SF</exclude>
                        <exclude>META-INF/*.DSA</exclude>
                        <exclude>META-INF/*.RSA</exclude>
                    </excludes>
                </filter>
            </filters>
            <transformers>
    <transformer implementation="ServicesResourceTransformer"/>
    <transformer implementation="ManifestResourceTransformer">
    <manifestEntries>
        <Main-Class>
            cz.mendelu.thesis.steganography.SteganopraphyGUI
        </Main-Class>
            <X-Compile-Source-JDK>1.8</X-Compile-Source-JDK>
            <X-Compile-Target-JDK>1.8</X-Compile-Target-JDK>
    </manifestEntries>
    </transformer>
            </transformers>
        </configuration>
    </execution>
</executions>
</plugin>
```

6.2 Testování aplikace

V rámci testování aplikace jsou postupně vyzkoušeny jednotlivé typy vstupních souborů. Otestována je i základní funkčnost šifrování ukrývané zprávy.

6.2.1 Testování fotografií

Tato aplikace umožní zpracování fotografií v nejběžnějších formátech, ale pro tento typ dat není příliš vhodná. Pracuje pouze s omezeným počtem barev, který nemusí stačit pro věrné vyjádření původní fotografie.

Na obrázku 10 je původní fotografie, která byla využita pro ukrytí zprávy. Tato fotografie byla pořízena fotoaparátem s rozlišením 16 Mpx a vytvořená fotografie byla uložena do formátu JPEG. Do obrazu byla ukryta zpráva a následně se vytvořil výstup ve formátu PNG. Obraz PNG vytvořený obrazově závislou metodou tvorby palety je zobrazen na obrázku 11 a obraz PNG vytvořený obrazově nezávislou metodou tvorby palety na obrázku 12. Při srovnání výsledků obou metod tvorby nové palety je patrné, že obrazově závislá metoda vykazuje znatelně lepší výsledky.

V testovaném případě bylo 144 169 unikátních barev zdrojového obrazu zredukováno na 256 barev v nové paletě. Pozorovatelným důsledkem jsou pruhy, které vznikly na obloze v testované fotografii.

Ukázky chodu aplikace a postupu ukrytí zprávy jsou přiloženy v přílohách.

Obrázek 10: Původní fotografie

Obrázek 11: Fotografie s ukrytou zprávou s paletou vytvořenou pomocí Median Cut

Obrázek 12: Fotografie s ukrytou zprávou s obrazově nezávislou paletou

6.2.2 Testování obrazů s průhledností

Pro testování průhlednosti byl zvolen obraz v rozlišení 16 Mpx, u kterého byla část (obloha) odstraněna a místo ní zůstal průhledný prostor, což je znázorněno na obrázku 13. Při generování do souborů, které průhledný kanál podporují, byla průhlednost zachována. Vytvořený soubor ve formátu PNG znázorňuje obrázek 14. Při generování do souborů, které průhledný kanál nepodporují, byla průhlednost nahrazena černou barvou. To lze pozorovat například na souboru ve formátu BMP znázorněném na obrázku 15.

V případě částečně průhledného obrazu aplikace rozhodne pro každý pixel, jestli bude zcela průhledný nebo nebude mít žádnou úroveň průhlednosti.

Obrázek 13: Původní fotografie bez oblohy

Obrázek 14: Fotografie s ukrytou zprávou do formátu PNG

Obrázek 15: Fotografie s ukrytou zprávou do formátu BMP

6.2.3 Testování animovaného souboru formátu GIF

Jednou z možností formátu GIF je i animace (W3.org, 1990). Animace umožňuje střídání více obrazů po sobě. Vlastnost animace je u obrazového formátu spíše výjimečná.

Aplikace dokázala zpracovat i soubor formátu GIF, který obsahoval animaci. Z tohoto souboru byl vybrán první snímek a do výstupního souboru využit pouze tento obraz. Animace tedy zachována není, ale jako zdroj obrazu animovaný GIF použít lze.

6.2.4 Testování formátu PSD

Formát PSD umožňuje pracovat ve vrstvách. V případě využití tohoto formátu jsou tedy viditelné vrstvy PSD sloučeny a podobně jako při exportu souboru z aplikace Photoshop zpracovány na výstup. Aplikace dokázala tedy formát PSD využít jako předlohu pro vytvoření libovolného výstupního formátu podporovaného aplikací.

6.2.5 Odkrytí zašifrované zprávy

Aplikace podporuje šifrovací algoritmus, který se zapíná pro celou aplikaci v jedné kartě. Vyplněné heslo je tedy využito pro ukrytí i odkrytí. V rámci testování byla následující zpráva ukryta do obrazu.

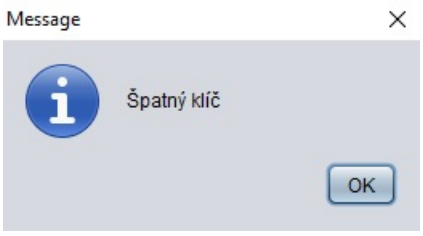
```
Zpráva: Ahoj světe!
Heslo: heslo
```

Zpráva byla zašifrována, ale pro účely testování nebylo zapnuto šifrování při procesu dešifrování. Při odkrývání zprávy tedy aplikace zobrazila následující posloupnost znaků:

```
G+3HZA3/FThT2Txars3tFVRZui680P5DmON6w5y8reQ=
```

Navrácená zpráva byla stále zašifrována a pro potencionálního útočníka tedy neměla podstatný význam.

Následně bylo šifrování zapnuto, ale záměrně bylo vyplněno špatné heslo. Takto odkrývaná zpráva nebyla vůbec navrácena a uživatel byl informován následujícím oknem:



Obrázek 16: Okno informující o špatně vloženém klíči šifrované zprávy

Zpráva je tedy získána až po vyplnění původního hesla, které bylo zadáno pro zašifrování.

6.3 Závěr testování

Aplikace dokáže zpracovat mnoho formátů. V průběhu testování byly popisovány pouze obrazy a formáty, u kterých byl očekávaný výskyt problémů.

Pokud má obraz velké množství barev, je provedená redukce v některých případech patrná pouhým okem. To ovšem vede pouze k odhalení, že v obraze byl snížen počet barev a nijak nepomáhá odkrytí uschované zprávy.

Testování provedené v rámci této práce je zaměřeno převážně na funkčnost ukrytí zprávy. Vzhledem k zaměření aplikace by bylo vhodné provést před používáním aplikace i testování odolnosti aplikace proti algoritmům stegoanalýzy a odolnosti prolomení zvoleného šifrovacího algoritmu. Provedení takového testování by ale vzhledem k přílišné náročnosti překračovalo rozsah této práce.

7 Ekonomické zhodnocení

Poptávka po zabezpečení se neustále zvyšuje a uživatelé tak využívají stále více bezpečnostních principů (Subhedar, Mankar, 2014). To může tedy vést i k růstu nákladů, které firmy musí na bezpečnost vynaložit. Aplikace vytvořená v této práci umožňuje zdarma ukrýt uskutečňující se komunikaci a uživatelům tak poskytuje konkurenční výhodu.

Vytvořená aplikace je vzhledem k využitému programovacímu jazyku dostupná pro více platforem. Může tak být využívána více uživateli bez vynakládání dalších finančních prostředků.

7.1 Náklady na tvorbu aplikace

Náklady na tvorbu aplikace, pokud by byla vyvíjena ve firmě, lze patrně nejlépe odhadnout na základě vyplacené mzdy.

Běžný zaměstnanec v odvětví informačních a telekomunikačních činností dosahuje průměrného hrubého měsíčního příjmu ve výši 49 678 Kč (Český statistický úřad, 2016). Pokud tedy zaměstnanec pracuje 8 hodin denně 20 dnů v měsíci, lze jeho hrubou hodinovou mzdu odhadnout na 310 Kč. Doba, která je potřebná pro tvorbu této aplikace jednou osobou, je odhadnuta na základě zkušeností autora na 60 hodin. Hrubou mzdu, kterou by bylo nutné vyplatit zaměstnanci, lze tedy určit vynásobením pracovní doby hrubou hodinovou mzdou.

Tabulka 3: Hrubá mzda zaměstnance

Práce v hodinách	60
Hrubá hodinová mzda v Kč	310
Hrubá mzda za provedenou práci v Kč	18 600

Hrubá mzda zaměstnance, který by vytvořil danou aplikaci, je tedy odhadnuta na 18 600 Kč. Zaměstnavatel by musel dále započítat náklady zaměstnavatele na sociální a zdravotní pojištění, náklady na důkladné testování aplikace a náklady na vybavení pracovníka vhodnými nástroji.

8 Diskuse

V této práci bylo srovnáno 8 nejznámějších steganografických aplikací. Tyto aplikace poskytují podobnou funkcionalitu a jejich základním principem je převést vstupní soubor na výstupní soubor stejného nebo podobného formátu bez větší změny struktury souboru. Ve výsledném souboru tak nastávají pouze nezbytné změny. Pokud by byl tedy vstupní a výstupní soubor srovnán, byl by nalezen pouze minimální rozdíl.

V rámci této práce se k tomuto problému přistupuje odlišným způsobem. Vstupní data jsou brána pouze jako určitá předloha pro výstupní soubor. Aplikace je určena pro přenášení krátkých textových zpráv. Tomu je tedy následně uzpůsobena i struktura souboru na obraz s indexovanou paletou barev. Pro aplikaci vzniká omezení v počtu 256 rozdílných barev pro celý obraz, ale výsledný obraz se většinou zmenší a je proto vhodnější pro přenos. Při takové manipulaci může nastat pozorovatelné zkreslení. Toto zkreslení může být zřetelné, ale nijak nepoukazuje na ukrytá data.

Současně aplikace poskytuje nadstandardní množství podporovaných formátů. Aplikace dokáže načíst 13 různých grafických souborových formátů a zvolit z 5 formátů s indexovanými barvami pro výstupní soubor. To umožňuje zpracovat téměř libovolný obrazový formát, který je na internetu běžně k dispozici. Není tak potřeba využívat aplikace pro převod formátů, pokud uživatel potřebuje výstupní obraz určitého formátu. Šajtar se ve své práci zabýval tvorbou aplikace pro ukrývání utajených zpráv. Tato aplikace dokázala ukrýt textové zprávy do formátu PNG. Autor v rámci diskuse poukazuje na užitečnost rozšíření aplikace o více podporovaných formátů pro ukrytí zprávy a zabezpečení šifrovacím algoritmem (Šajtar, 2010).

V průběhu analýzy bylo zjištěno, že dnes již téměř všechny aplikace podobného typu podporují možnosti šifrování, a proto byla tato funkcionalita přidána, přestože není v zadání přímo specifikována. Proto tato aplikace obsahuje volitelný šifrovací algoritmus, který za pomoci hesla zásadně snižuje riziko získání zprávy osobou, které nebyla určena.

Morkus ve své práci tvrdí, že počet bitových úrovní, které mohou být modifikovány, nelze doporučit pro všechny obrazy jednotně, ale doporučuje využít maximálně 5 bitů z bajtu (Morkus, 2011). V rámci této práce autor umožnil upravit všech 8 bitových úrovní. To sice zcela nahradí původní barvy obrazu, ale vzhledem k využití indexované palety barev se nově vzniklé barvy co nejvěrněji znovu přiřadí podle zdrojového obrazu.

Výsledkem je tedy aplikace, která na rozdíl od běžných šifrovaných komunikačních programů nevyvolává nežádoucí pozornost a současně poskytuje také možnost komunikaci zašifrovat. Vytvořené obrazy mají minimální velikost z důvodu použití palety indexovaných barev a při použití vhodných obrazů nevzniká pozorovatelná změna. Pokud jsou použity fotografie s velkými plynulými přechody barev, může redukce barev a následná úprava vyšších bitů barevných složek způsobit změny, které již budou příliš nápadné, a pro tyto typy obrazů by bylo tedy vhodnější využít alternativní řešení.

9 Závěr

V průběhu zpracování této aplikace se povedlo splnit všechny požadavky specifikované v zadání. Výstupem práce je tedy aplikace pro ukrývání zpráv, která dokáže převést velké množství formátů. Využitý algoritmus se snaží ukrýt maximální množství dat na minimální prostor.

Aplikace podporuje šifrování, které bylo přidáno, přestože není v zadání přímo vyžadováno. Vzhledem k použitým šifrovacím algoritmům je tak aplikace mnohem bezpečnější.

Na základě analýzy lze určit, že aplikace poskytuje na trhu ojedinělou funkčnost a přistupuje ke zpracování ukrytí odlišným způsobem. Protože je aplikace určená pro ukrývání krátkých textových zpráv, využívá obrazy s indexovanými barvami, které mají minimální velikost. Pokud nastane při ukrývání přepsání celé palety barev, aplikace přiřadí nově vzniklé barvy podle původního obrazu. Soubory, které nemají paletu barev, aplikace převede.

Vzhledem ke zpracování v rámci univerzitní práce obsahuje aplikace i log, ve kterém jsou popsány činnosti prováděné aplikací. To umožňuje využít aplikaci pro demonstraci základních principů steganografie. Obsažena je i redukce množství barev v obraze za pomoci obrazově závislých i obrazově nezávislých metod. Proto lze na aplikaci demonstrovat i rozdíl mezi těmito metodami a kvalitou jejich výstupů.

Nedostatkem aplikace může být její omezení na textové zprávy. Přestože tyto zprávy mohou obsahovat i libovolné české znaky, mnoho volně dostupných aplikací již tímto omezeno není. Vzhledem k celkovému zaměření aplikace ale není toto omezení považováno za podstatné, protože paleta barev by většinu větších souborů nedokázala efektivně ukrýt.

Při ukrývání zpráv jsou jednotlivé bajty po bitech postupně ukládány na pozice za sebou. Vhodným rozšířením by tedy mohlo být ukrývat bity v nepravidelném vzoru, aby se hůře hledaly jejich správné posloupnosti. V současné verzi aplikace je v případě odhalení komunikace spíše spoléháno na šifrovací algoritmy.

Z testování aplikace je patrný prostor pro budoucí vylepšení v oblasti redukce barev. V rámci této aplikace jsou vybrány do palety vhodné barvy pro sestavení nového obrazu, ale výsledný efekt i tak může u mnoha obrazů působit rušivě. Proto by bylo vhodné implementovat funkce, které by snížili výraznost provedených změn pozorovatelných lidským okem, jako tomu je u ditheringu.

10 Literatura

ADOBE.COM. Adobe Photoshop: File Formats Specification. *Adobe* [online]. Adobe, 2013 [cit. 2016-04-29]. Dostupné z: https://www.adobe.com/devnet-apps/photoshop/fileformatashtml/.

ADOBE.COM. Tiff: Revision 6.0. *Adobe* [online]. 1585 Charleston Road: Adobe, 1992, 1–121 [cit. 2016-02-18]. Dostupné z: http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf.

BÁTORA, Jozef. DeepSound. *JospinSoftware* [online]. Slovakia, 2016 [cit. 2016-03-08]. Dostupné z: http://jpinsoft.net/DeepSound.

CHAUMONT, M., W. PUECH a C. LAHANIER. Securing color information of an image by concealing the color palette. *Journal of Systems and Software* [online]. 2013, **86**(3): 809–825 [cit. 2015-12-11]. DOI: 10.1016/j.jss.2012.11.042. ISSN 01641212. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S016412121200324X.

CHEDDAD, Abbas, Joan CONDELL, Kevin CURRAN a Paul MC KEVITT. Digital image steganography: Survey and analysis of current methods. *Signal Processing* [online]. 2010, **90**(3): 727–752. DOI: 10.1016/j.sigpro.2009.08.010. ISSN 01651684. Dostupné také z: http://linkinghub.elsevier.com/retrieve/pii/S0165168409003648.

CRYPTURE.IT. *Crypture.it* [online]. 2013 [cit. 2016-05-03]. Dostupné z: http://www.crypture.it/Home.html.

ČESKÝ STATISTICKÝ ÚŘAD. Průměrné mzdy - 4. čtvrtletí 2015. In: *Český statistický úřad* [online]. Praha: Český statistický úřad, 2016 [cit. 2016-05-02]. Dostupné z: https://www.czso.cz/csu/czso/cri/prumerne-mzdy-4-ctvrtleti-2015.

EMBEDDEDSW.NET. OpenPuff 4.00: Yet not another steganography SW. *Advanced Embedded Solutions: Security - Software - Hardware - Machinery* [online]. Swtizerland, 2015 [cit. 2016-02-21]. Dostupné z: http://embeddedsw.net/OpenPuff_Steganography_Home.html.

FAROOQ, Umer a M. Faisal ASLAM. Comparative analysis of different AES implementation techniques for efficient resource usage and better performance of an FPGA. *Journal of King Saud University - Computer and Information Sciences* [online]. 2016, , - [cit. 2016-04-11]. DOI: 10.1016/j.jksuci.2016.01.004. ISSN 13191578. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S1319157816300143.

FERNÁNDEZ, Daniel. Java Simplified Encryption. *Jasypt: Java Simplified Encryption* [online]. Spain, 2014 [cit. 2016-03-04]. Dostupné z:

http://www.jasypt.org/.

HARMAN, Doug. *Digitální fotografie.* V Praze: Slovart, 2012. ISBN 978-80-7391-553-7.

KUHR, Harald. TwelveMonkeys. *GitHub* [online]. 2015 [cit. 2016-03-04]. Dostupné z: https://github.com/haraldk/TwelveMonkeys.

KWAN, Matthew. The Gifshuffle Home Page. *Darkside Technologies* [online]. 2010 [cit. 2016-02-21]. Dostupné z: http://www.darkside.com.au/gifshuffle/.

LIN, Yih-Kai. A data hiding scheme based upon DCT coefficient modification. *Computer Standards & Interfaces* [online]. 2014, **36**(5), 855–862 [cit. 2016-03-08]. DOI: 10.1016/j.csi.2013.12.013. ISSN 09205489. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0920548914000051.

MARTIŠEK, Dalibor. *Matematické principy grafických systémů.* Vyd. 1. Brno: Littera, 2002. ISBN 80-85763-19-2.

MIANO, John. *Compressed image file formats: JPEG, PNG, GIF, XBM, BMP.* Reading, Mass.: Addison Wesley, 1999. ISBN 0201616572.

MORKUS, Filip. *Program pro skrývání dat v obrazových souborech.* Brno, 2011. Diplomová práce. Vysoké učení technické v Brně. Vedoucí práce Doc. Ing. Karel Burda, CSc.

MURRAY, J a William VANRYPER. *Encyklopedie grafických formátů: obsahující specifikace souborových formátů, řadu konverzních programů a screengrabberů, zdrojové obrázky a kódy pro platformy PC, Unix a Mac.* 2. vyd. Praha: Computer Press, 1997. ISBN 80-7226-033-2.

NETBEANS.ORG. NetBeans IDE Features. *NetBeans* [online]. 2016 [cit. 2016-05-12]. Dostupné z: https://netbeans.org/features/index.html.

NIH.GOV. ImageJ: Image Processing and Analysis in Java. *Research Services Branch* [online]. Bethesda, Maryland, U.S., 2016 [cit. 2016-03-04]. Dostupné z: http://imagej.nih.gov/ij.

NIRAIMATHI, P., M.S. SUDHAKAR a K. BHOOPATHY BAGAN. Efficient reordering algorithm for color palette image using adaptive particle swarm technique. *Applied Soft Computing* [online]. 2012, **12**(8): 2199–2207 [cit. 2015-12-05]. DOI: 10.1016/j.asoc.2012.03.023. ISSN 15684946. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S1568494612001160.

PECINOVSKÝ, Rudolf. *Java 7: učebnice objektové architektury pro začátečníky.* Vyd. 1. Praha: Grada, 2012. Knihovna programátora (Grada). ISBN 978-80-247-3665-5.

PÉREZ-DELGADO, M.L. Colour quantization with Ant-tree. *Applied Soft Computing* [online]. 2015, **36**, 656–669 [cit. 2015-12-07]. DOI: 10.1016/j.asoc.2015.07.048. ISSN 15684946. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S1568494615005086.

QUICKCRYPTO. QuickStego. *QuickCrypto* [online]. United Kingdom, 2015 [cit. 2016-02-29]. Dostupné z: http://www.quickcrypto.com/free-steganography-software.html.

ROY, Souvik a P. VENKATESWARAN. A Text based Steganography Technique with Indian Root. *Procedia Technology* [online]. 2013, **10**, 167-171 [cit. 2016-05-01]. DOI: 10.1016/j.protcy.2013.12.349. ISSN 22120173. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S2212017313005033.

SATIR, Esra a Hakan ISIK. A compression-based text steganography method. *Journal of Systems and Software* [online]. 2012, **85**(10): 2385–2394 [cit. 2015-12-07]. DOI: 10.1016/j.jss.2012.05.027. ISSN 01641212. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0164121212001379.

SOFTPEDIA.COM. Portable SteganoG. *Softpedia* [online]. 2013 [cit. 2016-03-20]. Dostupné z: http://www.softpedia.com/get/PORTABLE-SOFTWARE/Security/Encrypting/Windows-Portable-Applications-Portable-SteganoG.shtml.

SOFT32.COM. Hide'N'Send. *Soft32* [online]. 2012 [cit. 2016-03-20]. Dostupné z: http://hide-n-send.soft32.com/.

SOLOMON, Chris a Toby BRECKON. *Fundamentals of digital image processing: a practical approach with examples in Matlab.* 1. vyd. Oxford: Wiley-Blackwell, 2011. ISBN 978-0-470-84473-1.

SUBHEDAR, Mansi S. a Vijay H. MANKAR. Current status and key issues in image steganography: A survey. *Computer Science Review* [online]. 2014, **13-14**, 95-113 [cit. 2016-04-30]. DOI: 10.1016/j.cosrev.2014.09.001. ISSN 15740137. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S1574013714000136.

ŠAJTAR, Lukáš. *Využití obrazových souborových formátů v oblasti steganografie.* Brno, 2010. Bakalářská práce. Mendelova univerzita v Brně. Vedoucí práce Ing. Jan Přichystal, Ph.D.

UMA MAHESWARI, S. a D. JUDE HEMANTH. Frequency domain QR code based image steganography using Fresnelet transform. *AEU - International Journal of Electronics and Communications* [online]. 2015, **69**(2), 539–544 [cit. 2016-04-11]. DOI: 10.1016/j.aeue.2014.11.004. ISSN 14348411. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S1434841114003082.

VAIDYA, Samir. OpenStego: The free steganography solution. *OpenStego* [online]. 2015 [cit. 2016-02-29]. Dostupné z: http://www.openstego.com/.

WANG, Kan, Zhe-Ming LU a Yong-Jian HU. A high capacity lossless data hiding scheme for JPEG images. *Journal of Systems and Software* [online]. 2013, **86**(7): 1965–1975 [cit. 2015-12-11]. DOI: 10.1016/j.jss.2013.03.083. ISSN 01641212. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0164121213000812.

WU, Mei-Yi, Yu-Kun HO a Jia-Hong LEE. An iterative method of palette-based image steganography. *Pattern Recognition Letters* [online]. 2004, **25**(3): 301–309 [cit. 2015-12-05]. DOI: 10.1016/j.patrec.2003.10.013. ISSN 01678655. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0167865503002265.

W3.ORG. Graphics Interchange Format(sm): Version 89a. *W3C* [online]. 1990 [cit. 2016-02-18]. Dostupné z: https://www.w3.org/Graphics/GIF/spec-gif89a.txt.

W3.ORG. JPEG JFIF. *W3C* [online]. 2003a [cit. 2016-04-11]. Dostupné z: https://www.w3.org/Graphics/JPEG/.

W3.ORG. Portable Network Graphics (PNG) Specification (Second Edition). *W3C* [online]. 2003b [cit. 2016-02-17]. Dostupné z: https://www.w3.org/TR/PNG/.

YU, Yuan-Hui, Chin-Chen CHANG a Iuon-Chang LIN. A new steganographic method for color and grayscale image hiding. *Computer Vision and Image Understanding* [online]. 2007, **107**(3): 183–194 [cit. 2015-12-11]. DOI: 10.1016/j.cviu.2006.11.002. ISSN 10773142. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S1077314206001767.

YUE, X.D., D.Q. MIAO, L.B. CAO, Q. WU a Y.F. CHEN. An efficient color quantization based on generic roughness measure. *Pattern Recognition* [online]. 2014, **47**(4), 1777–1789 [cit. 2015-12-05]. DOI: 10.1016/j.patcog.2013.11.017. ISSN 00313203. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0031320313005037.

ŽÁRA, Jiří, Bedřich BENEŠ, Jiří SOCHOR a Petr FELKEL. *Moderní počítačová grafika.* 2., přeprac. a rozš. vyd. Praha: Computer Press, 2004. ISBN 80-251-0454-0.

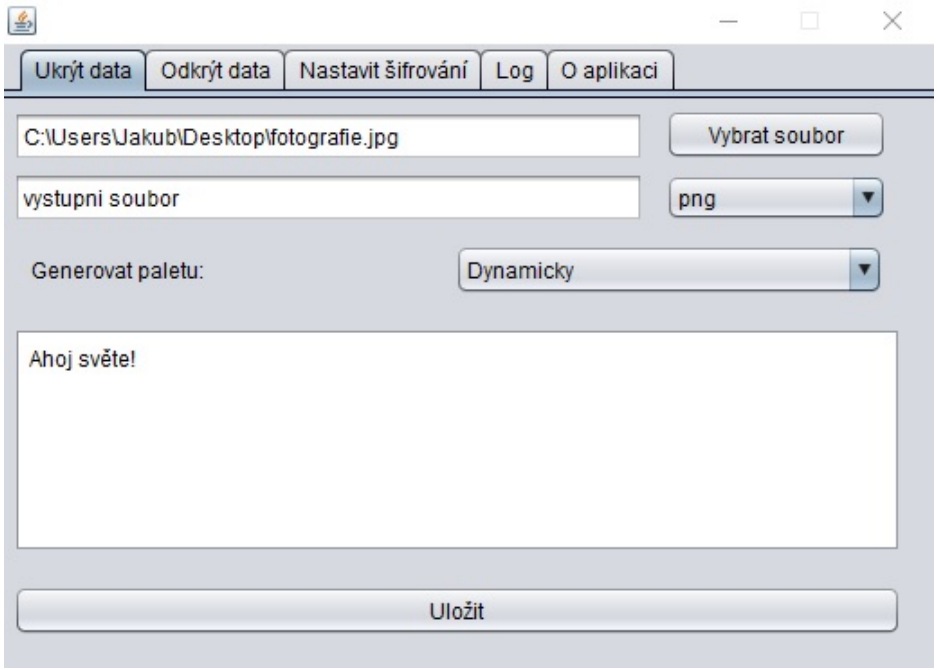
Přílohy

A Datové přílohy

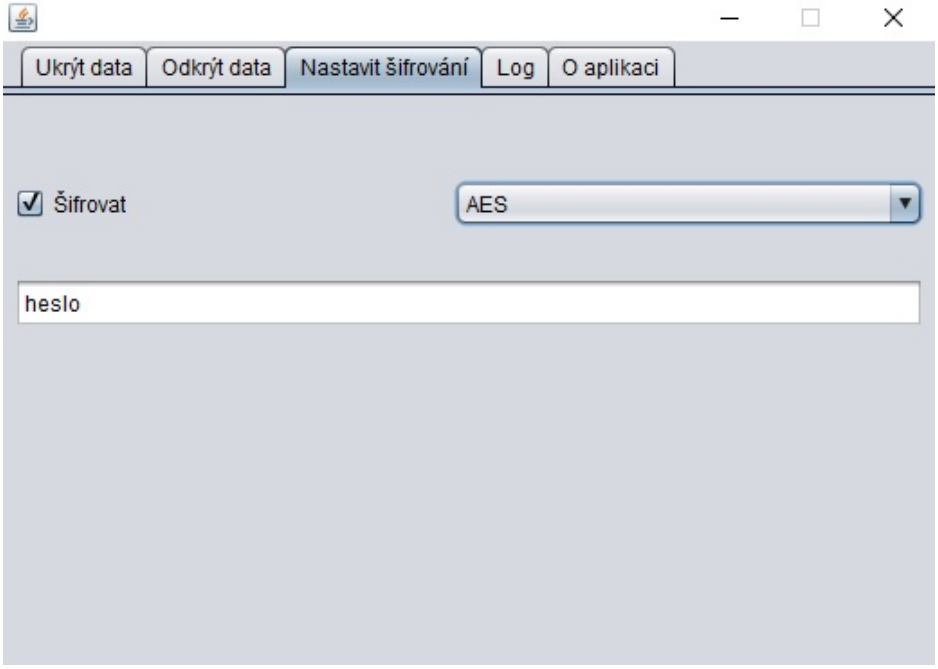
Na přiloženém CD jsou umístěny spustitelný soubor ve formátu JAR a projektová složka se zdrojovými kódy. Pro vytvoření spustitelného souboru lze využít vývojové prostředí NetBeans nebo ve složce se souborem pom.xml použít následující příkaz:

```
mvn clean install
```

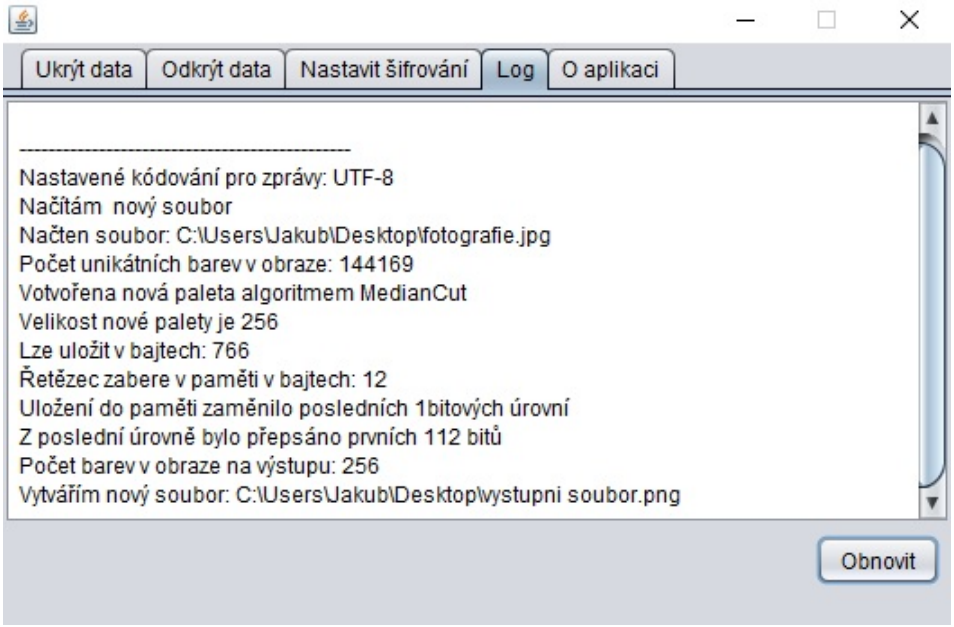
B Ukázka prostředí aplikace



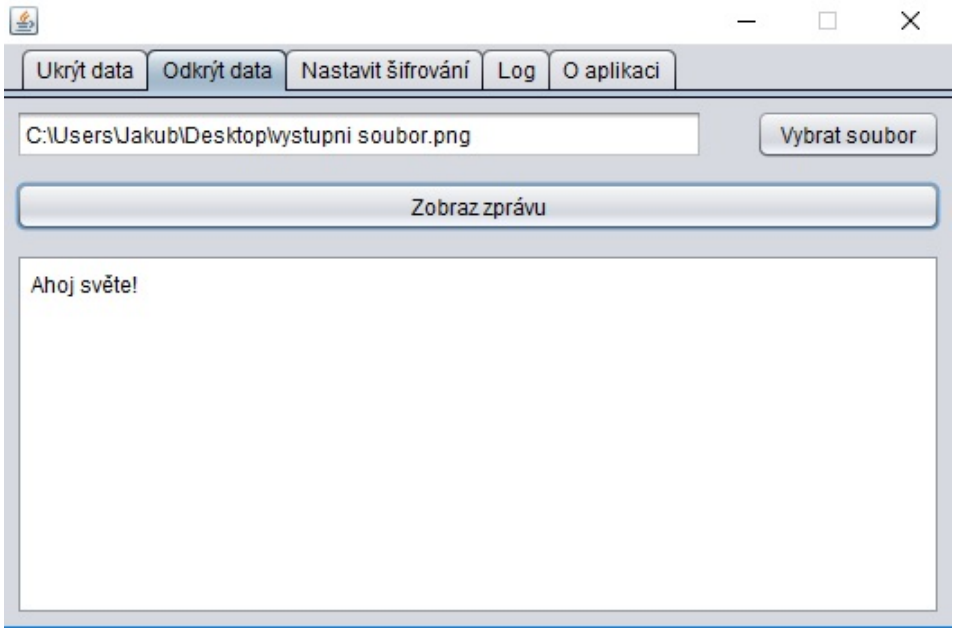
Obrázek 17: Vytvoření nového souboru



Obrázek 18: Zapnutí volitelného šifrování



Obrázek 19: Informace o průběhu tvorby nového souboru



Obrázek 20: Odkrytí dat z nového souboru