Steganografie a možnosti jejího využití

Steganography and possibilities of its use

Bc. Jan Bartl

Diplomová práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

*** nascannované zadání str. 1 ***

*** nascannované zadání str. 2 ***

ABSTRAKT

Tato diplomová práce přináší obecné informace o steganografii, umění ukrývání dat. Steganografie je používána již 2500 let, našla široké uplatnění ve výzvědných službách, armádě, diplomacii, v průmyslové a obchodní sféře a rovněž v osobním použití. Práce poskytuje přehled o dříve používaných technikách ukrývání, o současných moderních postupech i o trendu možného vývoje do budoucna. Informace uvedené v této práci jsou předvedeny na praktické aplikaci, která demonstruje možnost ukrytí zprávy do bitmap a audio souborů. Ukrytí zprávy je založené na technologii LSB (využití nejméně významných bitů informace) a na konkrétní bitmapě je předvedeno, jak ukrytí zprávy bitmapu pozmění.

Klíčová slova: steganografie, problém vězňů , LSB, BMP, PNG, WAV, MP3, lingvistika, kryptografie, CRC, GZIP, bitové operace, Java, SWT

ABSTRACT

This paper presents general information about steganography, the art of data hiding. Steganography has been used in various forms for 2500 years. It has found use in variously in intelligence service, military, diplomatic, industrial and business sphere and personal and intellectual property applications. The paper provides an overview of steganography, general forms of steganography, specific steganographic methods, and recent developments in the field. The information presented in this paper is also applied to a program developed by the author, and some sample runs of the program are presented. The program focuses on the Least Significant Bit (LSB) technique in hiding messages in an image and audio files.

Keywords: steganography, prisoners dilemma, LSB, BMP, PNG, WAV, MP3, linguistic, cryptography, CRC, GZIP, bit operations, Java, SWT

Děkuji vedoucímu mé diplomové práce Ing. Pavlovi Navrátilovi, Ph. D. za pomoc při tvorbě programu, který je součástí této práce, za cenné nápady, rady a kvalitní testování, které odhalily nejednu chybu.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

…….
podpis
diplomanta

OBSAH

ÚVOD

Králové, královny a generálové po tisíce let spoléhali na účinné komunikační systémy, jež jim umožňovaly vládnout jejich zemím a velet armádám. Zároveň si vždy byli vědomi, jaké následky by mělo, kdyby jejich zprávy padly do nepovolaných rukou: vyzrazení cenných tajemství cizincům, odhalení klíčových informací nepříteli. Bylo to právě riziko vyzrazení, co vedlo k rozvoji kódů, šifer a dalších technik určených k ukrytí smyslu zprávy před všemi kromě zamýšleného příjemce. [30]

V dnešní době moderní digitální komunikace, kdy internetem proudí neuvěřitelné množství dat, je požadavek stejný jako v minulosti. Mnohá data putují po internetu bez zabezpečení a pro technicky zdatného hackera není nic jednoduššího, než komunikaci nenápadně odposlouchávat. Uživatel má možnost data chránit kryptovacími technikami, které sice komunikaci zabezpečí, ale na druhou stranu zvýší podezření, že jde o citlivá data, ke kterým se bude chtít hacker za každou cenu dostat. Existuje řada technik, jak kryptovaná data prolomit, i když při dostatečně dlouhém klíči a použití kvalitní kryptovací technologie je pravděpodobnost prolomení malá.

Další možnost zabezpečení je ukrytí zprávy do zcela nenápadně vypadajícího textu nebo multimediálních dat. Není nic jednoduššího, než poslat druhé straně fotografii svého psa, a přitom fotografie může obsahovat zprávu, kterou chci před hackerem skrýt. Útočník při sledování komunikace uvidí „jen“ fotografii.

Techniky skrývání dat používá armáda, policie, ale na druhou stranu je velice pravděpodobné, že podobných technik využívají i teroristé. V únoru 2001, USA Today [19] zveřejnila, že teroristé využívali steganografii založenou na ukrývání zpráv uložených na internetu. Byly vytvořeny techniky, které se specializují přímo na odhalování zpráv v obrázcích uložených v různých veřejných galeriích na internetu, lze tak např. průběžně prohledávat Flickr [3] nebo Facebook [2] a odhalovat skryté zprávy online. V roce 2002 byla pány Niels Provos a Peter Honeyman provedena simulace [1] odhalení ukrytých zpráv v obrázcích a bylo analyzováno přibližně milión obrázků. S tehdejší technikou dokázali denně zpracovat cca 370.000 obrázků, dnes již máme mnohem výkonnější počítače, takže prohledat miliony obrázků denně nebude žádný problém.

Jak tedy nejlépe před zvědavci zprávu ukrýt tak, aby byla co nejméně nápadná?

I. TEORETICKÁ ČÁST

1 POČÁTKY UTAJENÉ KOMUNIKACE

Některé z nejstarších zmínek o utajení komunikace pocházejí od Herodota - „otce historie", jak ho nazval římský filozof a politik Cicero. Ve svých *Dějinách* shrnuje Herodotos konflikty mezi Řeky a Peršany v 5. století př. n. 1. Chápal je jako konfrontaci svobody a otroctví, jako boj mezi nezávislými řeckými státy a perskými utlačovateli. Podle Herodota to bylo právě umění tajných zpráv, co zachránilo Řecko před dobytím Xerxem - Králem králů, který byl despotickým vůdcem Peršanů. [30]

1.1 Voskové psací destičky

Dlouhodobé nepřátelství mezi Řeky a Peršany dosáhlo kritického bodu krátce poté, co Xerxes začal stavět Persepolis, nové hlavní město svého království. Z celé říše a sousedních států sem proudily poplatky a dary. Významnou výjimkou byly Athény a Sparta. Xerxes chtěl takovou opovážlivost ztrestat a začal shromažďovat vojsko. Prohlásil, že „rozšíříme perskou říši tak, že její jedinou hranicí bude nebe a slunce nedohlédne země, jež by nepatřila nám". Po pět let sbíral největší vojenskou sílu v dosavadní historii. V roce 480 př. n. 1. byl připraven na překvapivý úder.

Přípravy perské armády však pozoroval Řek Demaratus, který byl ze své vlasti poslán do vyhnanství a žil v perském městě Susy. Přestože byl vyhnanec, cítil nadále loajalitu k Řecku, a tak se rozhodl poslat do Sparty varování před Xerxovými útočnými plány. Problém však byl, jak zprávu dopravit, aby ji nezachytily perské hlídky. Herodotos píše: „Nebezpečí prozrazení bylo velké a Demaratus přišel jen na jeden způsob, jak zprávu zaslat. Seškrábal vosk ze dvou voskových psacích destiček, sepsal Xerxovy záměry přímo na jejich dřevo a pak zprávu znovu zakryl voskem. Tabulky byly na první pohled prázdné a nevzbudily zájem stráží. Když dorazily do cíle, nikdo nedokázal rozluštit jejich tajemství, až Kleomenova dcera Gorgo uhodla, oč jde, a řekla ostatním, že je třeba seškrabat vosk. Když tak učinili, našli zprávu, přečetli ji a sdělili ostatním Řekům." Kvůli varování se do té doby bezbranní Řekové *začali* ozbrojovat. Zisky státních stříbrných dolů, dosud rozdělované mezi občany, byly použity ke stavbě dvou set válečných lodí.

Xerxes ztratil moment překvapení. Když jeho loďstvo vplulo do zálivu u Salaminy nedaleko Athén, byli Řekové připraveni. Xerxes se domníval, že chytil řecké loďstvo do pasti, avšak byli to naopak Řekové, kteří vlákali nepřítele do úzkého zálivu. Věděli, že jejich malé a méně početné lodě by na otevřeném moři proti perské flotile neobstály, ale v zálivu se uplatnila jejich větší manévrovací schopnost. Když se otočil vítr, zůstali

Peršané uzavřeni v zálivu. Perská princezna Artemisia byla se svou lodí obklíčena ze tří stran, přesto se pokusila uniknout na volné moře, namísto toho však narazila do jedné z vlastních lodí. Vznikla panika, při které došlo k dalším srážkám, a Řekové rozpoutali krvavou řež. Během jediného dne tak byla pokořena ohromná perská vojenská síla. [30]

1.2 Ukrytí zprávy dle Histiaea

Demaratova strategie tajné komunikace spočívala v prostém ukrytí zprávy. Herodotos popisuje i jinou událost, kdy ukrytí textu postačilo k bezpečnému zaslání zprávy. Vypráví příběh, v němž vystupuje Histiaios, který chtěl povzbudit Aristagora Milétského ke vzpouře proti perskému králi. Aby zaslal své poselství bezpečně, oholil Histiaios hlavu svého posla, napsal zprávu na kůži lebky a počkal, až poslovi znovu narostou vlasy. Jak je vidět, v tomto historickém období se menší zpoždění dalo tolerovat. Posel pak mohl cestovat bez potíží, nenesl přece nic závadného. V cíli své cesty si znovu oholil hlavu a ukázal ji příjemci zprávy. [30]

1.3 Steganografie a kryptografie

Komunikace utajená pomocí ukrytí zprávy se nazývá *steganografie,* podle řeckých slov *steganos* (schovaný) a *graphein* (psát). Během dvou tisíc let, jež nás dělí od Herodotových časů, se v různých částech světa rozvinuly různé formy steganografie. Staří Číňané například psali *zprávy* na jemné hedvábí, které pak zmačkali do malé kuličky a zalili voskem. Posel pak voskovou kuličku polkl. Italský vědec Giovanni Porta v 16. století popsal, jak ukrýt zprávu ve vejci vařeném natvrdo pomocí inkoustu vyrobeného z jedné unce kamence a pinty octa. Tím se pak napíše zpráva na skořápku. Roztok pronikne jejími póry a zanechá zprávu na vařeném bílku. Přečíst ji lze, až když vajíčko oloupeme. Do oblasti steganografie patří rovněž neviditelné inkousty. Již z 1. století našeho letopočtu pochází návod Plinia Staršího, jak použít mléko pryšce *(Titbymalus sp.* z čeledi *Euphorbiaceae)* jako neviditelný inkoust. Po zaschnutí je mléko zcela průhledné, když se však lehce zahřeje, zhnědne. I moderní špioni občas improvizovali s použitím vlastní moči, když jim došla zásoba tajného inkoustu.

Dlouhá tradice steganografie jasně ukazuje, že jde o techniku, jež sice poskytuje určitý stupeň utajení, má však zásadní vadu. Když už se zprávu jednou podaří objevit, je prozrazena naráz. Pouhé její zachycení znamená ztrátu veškerého utajení. Důkladná stráž může prohledávat všechny osoby cestující přes hranice, oškrabávat voskové tabulky,

nahřívat čisté listy papíru, loupat vařená vejce, holit lidem hlavy a tak dále. Určité množství zpráv se tak vždy podaří zachytit.

Souběžně se steganografií se proto začala rozvíjet i *kryptografie, jejíž název* pochází z řeckého slova *kryptos* (skrytý). Cílem kryptografie není utajit samu existenci zprávy, ale její význam, a to pomocí šifrování. Aby nešlo zprávu přečíst, pozmění se podle pravidel předem dohodnutých mezi odesilatelem a příjemcem. Pokud taková zpráva padne do rukou nepříteli, je nečitelná. Nezná-li nepřítel použitá šifrovací pravidla, pak se mu podaří zjistit obsah zprávy jen s velkým úsilím, anebo vůbec ne. [30]

1.4 Mikrotečky

Přestože jsou kryptografie a steganografie nezávislé techniky, je možné je, pro větší bezpečnost zprávy, kombinovat. Příkladem takové techniky jsou mikrotečky, používané především během druhé světové války. Němečtí agenti v Latinské Americe dovedli fotografickou cestou zmenšit celou stránku textu do tečky o průměru menším než milimetr a tu pak umístit jako normální tečku za větou do nevinného dopisu. FBI poprvé zachytila mikrotečku roku 1941, když dostala tip, ať hledá na papíře jemný odlesk, způsobený použitým filmovým materiálem. Američané od té doby mohli číst obsah zachycených mikroteček, ovšem s výjimkou případů, kdy němečtí agenti zprávu před zmenšením ještě zašifrovali. V případech, kdy Němci takto kombinovali kryptografii se steganografií, mohli Američané jejich komunikaci monitorovat a občas přerušovat, nezískali však žádné informace o německých špionážních aktivitách. Kryptografie je účinnější než steganografie, protože pomocí ní lze zabránit tomu, aby informace padla do rukou nepřítele. Nicméně některé moderní techniky steganografie jsou natolik účinné, že ukrytou zprávu může mít útočník přímo před očima aniž o ní ví. [30]

1.5 Lingvistická steganografie

Další používanou steganografickou metodou byly tzv. nulové šifry (nezašifrované zprávy). Skutečná zpráva je obsažena v jiném neškodně vypadajícím textu. Mohlo jít o písmena nebo celá slova na určité pozici v textu. Takové nevinně působící zprávy pak nevzbuzovaly pozornost cenzorů. [28]

Na úvod dva starší příklady. Giovanni Boccaccio ukryl ve svých sonetech Amorosa visione další tři sonety - tvoří je začátky tercetu každého sonetu. Dalším příkladem je

anonymní dílo z konce 15. století pojednávající o lásce mnicha k ženě - první písmena všech kapitol tvoří větu *"Bratr Francesco Colonna vášnivě miluje Polii"*.

Zde už je konkrétní příklad zprávy obsahující nulovou šifru:

Fishing freshwater bends and saltwater coasts rewards anyone feeling stressed. Resourceful anglers usually find masterful leapers fun and admit swordfish rank overwhelming any day.

Tato věta vypadá na první pohled poněkud nejasně. Třetí písmena všech slov však dají dohromady jiný text:

Send Lawyers, Guns, and Money.

Podobný příklad, skutečně použitý německým špiónem za druhé světové války:

Apparently neutral's protest is thoroughly discounted and ignored. Isman hard hit. Blockade issue affects pretext for embargo on by products, ejecting suets and vegetable oils.

Zde se tajná zpráva skrývá pod druhými písmeny slov:

Pershing sails from NY June 1.

1.5.1 Čínská mřížka

Písmena nemusí být vždy na stejné pozici v každém slově - pomocí tzv. čínské mřížky lze zprávu rozmístit do krycího textu v podstatě náhodně. Zpráva je vepsána do otvorů v čtvercové mřížce nebo napsána na čtverečkovaný papír na předem domluvená místa a poté je doplněna nezávadným textem. Pro doručení je vhodné ji přepsat jako souvislý dopis. Příjemce vyplní text do čtvercové sítě domluvených rozměrů, přiloží mřížku a v jejích otvorech si přečte předávaný text. [28], [30]

U lingvistické steganografie může být problém vytvořit vždy nový krycí text, který má svůj dostatečně přesvědčivý smysl, aby nevzbuzoval pozornost potenciálního narušitele. Lze ale zkombinovat více různých metod, např. označit významná písmena v novinách neviditelným inkoustem (lingvistická a technická steganografie).

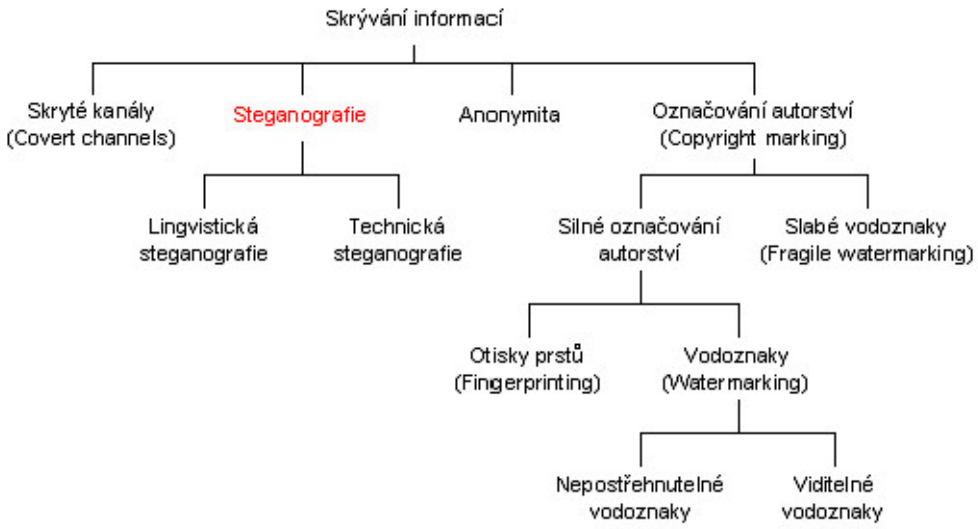
1.6 Další možnosti ukrytí textu

Podle [28] můžeme mezi historicky používané metody skrývání informací zařadit tyto další postupy:

- některá písmena v nezávadném textu byla propíchána špendlíkem, tato písmena tvořila předávaný utajený text
- podobně jsou některá písmena v jinak nezávadném textu psána např. tučněji (nebo jiným sklonem, jsou menší apod.)
- využití formátování textu, skryté znaky a další …
- využívající prázdných míst v textu dokumentu (open space methods) – manipulace s „bílými“ místy v textu (mezery mezi znaky, slovy). Pro představu 1 mezera značí „1“, dvě mezery značí „0“ a kombinací těchto nul a jedniček můžeme zakódovat textovou, ale i binární zprávu. [34]
- syntaktické metody (syntactic methods) využívají metody interpunkci pro kódování zprávy [34]
- sémantické metody (semantic methods) využívají pro vloženou zprávu vzájemnou manipulaci jednotlivých slov, tj. změnu textu bez změny významu nebo používání definovaných synonym v textu atp. [34]
- text lze utajit (ovšem zpravidla již komplikovaněji např. ve spojení s kódováním) i v zápisu šachové partie, návodu na vaření, katalogu objednávaného zboží, háčkování, apod. - Charles Dickens ve své knize A Tale of Two Cities popisuje, jak v době francouzská revoluce ženy s jehlicemi a přízí v rukou do své práce zaplétaly jména a popisy podezřelých osob.

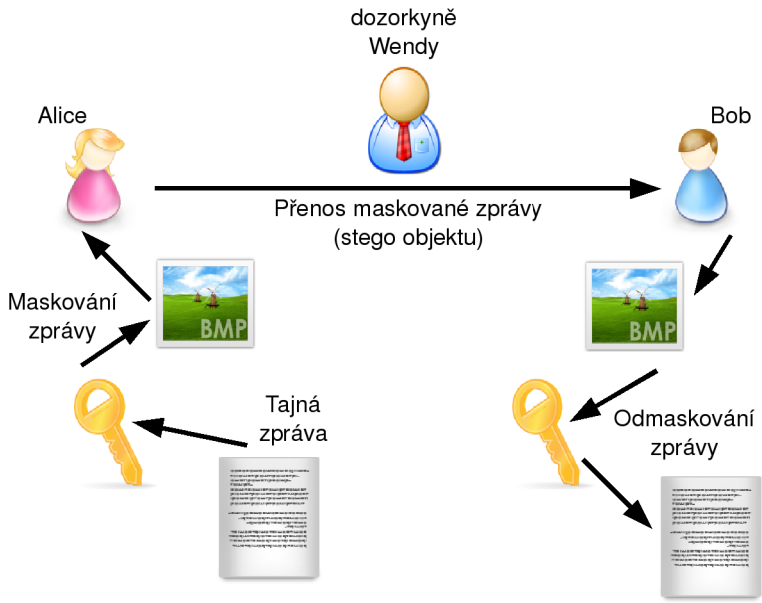
2 MODERNÍ DIGITÁLNÍ STEGANOGRAFICKÉ TECHNIKY

Podle článku Information Hiding - A Survey [25] je steganografie částí větší oblasti, a to skrývání informací:



Obrázek 1: Způsoby ukrývání informací

Princip steganografie si můžeme popsat na následujícím obrázku na příkladu vězeňské komunikace: Alice a Bob jsou vězni, každý sedí v jiné cele. Chtějí spolu komunikovat, což je sice možné, protože dozorkyně Wendy je ochotna jim nosit zprávy, ale současně Wendy zprávy kontroluje, jestli se vězni na něčem nedomlouvají. Alice s



Obrázek 2: Princip steganografie

Bobem se samozřejmě chtějí domlouvat, musí tedy své vzkazy zamaskovat do zpráv, které budou vypadat neškodně. Na obrázku je případ, kdy Alice vybere krycí objekt (text,

obrázek nebo cokoliv jiného), do kterého podle určitého klíče ukryje tajnou zprávu. Tím vznikne tzv. stego-objekt (v literatuře se často nazývá stegogram nebo cover), tedy zpráva, která vypadá nevinně, ale skrývá utajenou informaci. Wendy si stego-objekt prohlédne, ten nevzbudí její podezření a ona ho s klidným srdcem předá Bobovi. Bob zná klíč, na kterém se s Alicí domluvili, a tak ví, jak má ze stego-objektu vypreparovat utajenou zprávu. [28]

Stegoanalýza

Jestliže dokážeme pomocí nějakých způsobů ukrýt zprávu do coveru (bitmapy, audio souboru apod.), musí existovat i zpětný postup a tedy odhalení, že stego-objekt obsahuje ukrytou zprávu. Skutečnost, že stegoanalýza je často založena spíše na hádání než obecně platném matematickém prolomení vlastností stegosystému naznačuje i členění analytických postupů zavedené Peterem Waynerem [33], [35]:

Visual or aural attack

reprezentuje úlohu lidských smyslů v odhalování stegogramu. Některé steganografické algoritmy se spoléhají na to, že jím produkované stegogramy nebudou analyzovány lidským okem či sluchem. Za tento risk si kupují jednodušší algoritmus embeddingu či vyšší steganografickou kapacitu coverů.

Structural attack

je naopak ryze strojovým postupem, který ve vnitřně strukturovaných coverech detekuje takové nestandardní použití této struktury, které vede k zavlečení redundantních dat (tj. tajné zprávy), která by standardně nebyla interpretována, či nebyla vůbec použitelná standardní aplikací. Zde připadají v úvahu typicky ty strukturované covery, které obsahují nějaké volitelné segmenty (které budou standardní aplikace ignorovat) či segmenty pro volné použití, bez jasně udané sémantiky. Příkladem nechť stojí síťové datagramy – viz kap. 2.2 nebo MP3 soubory – viz kap. 2.6. Dodatečné segmenty lze snadno bez narušení původní funkce vložit také např. do spustitelného binárního souboru – viz kap. 2.5.

Zřejmým slabým místem tohoto typu útoku je moment strojového rozhodování, zda nalezená data v bezportfejních segmentech byla či nebyla injektována stegosystémem. Utajená zpráva se může či nemusí (šifrovaná data) okatě prozradit.

Statistical attack

o zkoumaném objektu shromáždí množinu vytipovaných, kvantitativně vyjádřitelných charakteristik a tato naměřená data porovnává s typickým profilem čistého

objektu, resp. s typickým profilem stegogramu. Výstupem statistického analyzátoru jsou ve vší komplexnosti dvě čísla: pravděpodobnost úspěchu provedení samotného statistického testu a pravděpodobnost správnosti verdiktu (čistý objekt/stegogram). První pravděpodobnost nebude jedničková např. v případě, kdy na zkoumaném objektu nebylo možné určité veličiny naměřit (kupříkladu pokus o naměření počtu výskytů obligátního řetězce „viagra“ v objektu netextového rázu). Druhá pravděpodobnost by teoreticky také neměla být nikdy jedničková, neboť i kdyby zkoumaný objekt vypadal sebepřesvědčivěji jako stegogram, algoritmus musí pamatovat na fakt, že to stále může být jen shoda náhod.

Navzdory tomuto nevyhnutelnému nedeterminismu ve výstupu jsou statistické útoky silným nástrojem. Jejich správná aplikace však může vyžadovat expertní zdatnost v matematice a/nebo v problematice daného coveru. Stejně jako u útoků předchozích dvou typů, ani v této kategorii neexistují pojmenované, natož standardní algoritmy (jako např. DES v kryptografii) odhalující stopy určitého specifického stegosystému. Na znalostech a invenci analytika závisí, jaké statistické ukazatele se rozhodne u objektů pozorovat, jak kvalitně bude schopen data zpracovat, a tím zda bude ve statistickém útoku úspěšný.

Obzvláště vhodná volba statistických ukazatelů je klíčová. U klasického LSB embeddingu například nevyhnutelně selhává idea měření počtu pixelů jednotlivých barev v obrázku a porovnávání tohoto profilu s nějakým standardem pro čisté obrázky a stegogramy. Předně zde není vůbec možné sestavit něco jako referenční barevný profil všech čistých obrázků. Ovšem začne-li si analytik všímat na zkoumaných objektech například počtu párů podobných barev (ať už jsou jakékoliv), může přijít k zajímavým zjištěním.

Stegdetect

Stegdetect [4] je stegoanalytický nástroj určený specificky k odhalování stegogramů vytvořených pomocí programů JSteg s nadstavbou JSteg-Shell, JPHide v. 0.3/v. 0.5 nebo OutGuess v. 0.13b, které akceptují obrázkový cover ve formátu JPEG a tajnou zprávu před embeddingem symetricky šifrují. Algoritmus analýzy je tedy založen na znalosti implementací steganografických algoritmů výše uvedených programů.

Označování autorství

Z obrázku 1 je patrné, že velký význam v technikách skrývání informací mají technologie označování autorství, převážně pak technologie vodoznaku (watermarking) a otisku prstů (fingerprinting). Rozdíl mezi steganografií a vodoznakem spočívá v kladení

důrazu na různé kvalitativní aspekty ukrytí dodatečné informace do coveru: pro steganografii je podstatné, aby přítomnost této informace nebyla nápadná; pro vodoznak, aby dodatečná informace nebyla z coveru odstranitelná (přesněji řečeno, aby její odstranění nutně vedlo v jistém smyslu k destrukci či naprostému znehodnocení coveru) a na její detekovatelnosti buď nezáleží nebo je přímo nezbytná.

Označování autorství tedy klade důraz na robustnost, steganografie na objem (utajených) dat v coveru. I pro samotnou steganografii je jistě vhodné, aby do coveru dané velikosti bylo možné vložit co nejvíce dat tak, že ani poškození stegogramu tajná data nezničí. Existuje však jistá (nejasná) míra bitového objemu, kterou nesmí vkládaný objekt překročit, aby výsledný stegogram stále vypadal dostatečně nenápadně. Tento bitový objem zahrnuje jak samotnou utajovanou zprávu, tak režijní data stegosystému a redundanci, která zajistí robustnost. [35]

Označení autorství lze provést jak technologií vodoznaku tak i pomocí technologie otisku prstů. Obě technologie jsou v dnešní době často používané pro ochranu obrázků, hudby i filmů. Identifikovat použití textu bez schválení autora lze pomocí webových vyhledávačů. Google je natolik silný, že neprohledává již jen webové stránky, ale i obsahy dokumentů, PDF, prezentací a pod. Vyhledat zneužitý obrázek nebo zvukovou nahrávku už tak snadné ale není. Pro tyto účely se hodí právě vodoznak nebo technologie otisku prstů.

Vodoznak (watermarking)

Technologie vodoznaku je založena na přidání značky (loga, názvu) přímo do obrazu (*zvuku, videa*), přičemž značka může být viditelná nebo neviditelná. Technologie je navržena tak, aby v případě modifikace obrazu (změna rozlišení, otočení obrazu, jiná změna histogramu) nebyla značka poškozena. Vodoznak funguje podobně jako metadata, přičemž tato jsou neoddělitelnou součástí obrazu a jsou obtížně odstranitelná.

Nelze dodatečně vodoznakem opatřit již vydaná díla (obrázky, hudba, …) a jeho použití je výrazně dražší, než využití fingerprintu. [12]

Otisky prstů (fingerprinting)

Fingerprinting pracuje na principu, že obrázek (*zvuk, video*) se pomocí hašovací funkce převede na řetězec (otisk). Otisk není umístěn přímo v obrazu, ale uloží se do databáze, která sdružuje jednotlivé autory a jejich autorská díla. Velká výhoda využití

otisku prstů, proti vodoznaku, je v tom, že lze otisk dogenerovat později, tedy i po uvolnění díla na trh (na Web).

Použití otisku je výrazně levnější než použití vodoznaku a existují i verze zdarma (Tineye). Otisk však nemusí být 100% spolehlivý, neboť podobné obrázky (fotografie stejného obsahu) mohou mít přiřazen stejný otisk. [12]

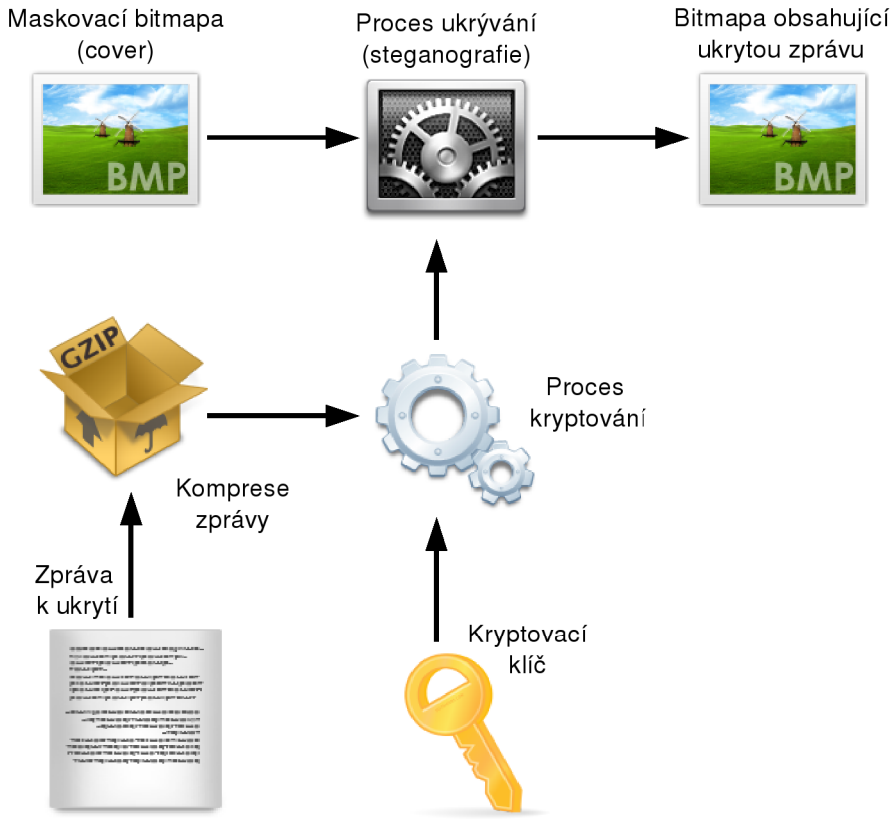
V dalších kapitolách se budeme detailně věnovat konkrétním technikám vytváření stego-objektu. V praktické části si pak ukážeme konkrétní implementace ukrytí zprávy do bitmapy a audio souboru pomocí technologie LSB, tedy nejméně významné informace.

2.1 Využití nejméně významných bitů

Pro skrývání informací v digitálních souborech existují různé metody.

2.1.1 LSB

Technika LSB (Least Significant Bit - nejméně podstatný bit informace) je postavena na nedokonalosti lidských smyslů – sluchu a zraku. Multimediální data lze nepatrně pozměnit a namodelovat do nich tajnou zprávu, ale lidské smysly tuto zprávu nejsou schopny postřehnout. Barva obrazu nebo zvuk se jen nepostřehnutelně (neviditelně,



Obrázek 3: Demonstrace ukrytí zprávy do coveru (např. BMP)

neslyšitelně) změní. Kódování tajných zpráv do posledního bitu každého bajtu audio souborů nebo obrázků sice dává poměrně velký prostor k ukrytí tajných dat, ale je velmi snadné takovou tajnou zprávu najít. Pokud však zpráva před zakódováním do multimediálního obrazu projde ještě kryptováním, podstatně se tak ztíží její odhalení a v případě použití kvalitního kryptovacího algoritmu je pak zpráva i po odhalení těžko rozluštitelná. Ukázka této technologie (LSB) je popsána v praktické části diplomové práce v kapitole 5.1.

2.1.2 LSB Random

Jednoznačně lepší technikou je využívat LSB náhodně vybraných bajtů (nebo pixelů obrázku nebo částí zvukového záznamu). Klíčem k rozluštění skryté zprávy je potom počáteční hodnota (seed) generátoru pseudonáhodných čísel.

Systémy využívající změny LSB mají jednu společnou zranitelnost - použití i velmi jednoduchého digitálního filtrování poruší hodnoty mnoha LSB digitálního objektu a tajná zpráva je nerozluštitelná. To vede k výzkumu dalších možností, jak techniky měnící určité bity posílit proti filtrování.

Řešením je dívat se na filtrovací operace jako na zdroj dodatečného šumu a snažit se co nejlépe využít prostor, který je k dispozici v krycím objektu pro zanesení informace. Nejjednodušším způsobem je opakování - zkrátka každý bit se vloží dostatečně mnohokrát, aby v krycím objektu přežil filtrování. Jiným způsobem může být rozprostření informace do statistiky jasu pixelů - systém může náhodně vybrat několik dvojic pixelů, u nichž zvýší nebo sníží kontrast jasu - průměrný jas obrázku se tím nezmění, ale zvýší se kontrast v rámci množiny vybraných pixelů. Tato metoda údajně při vhodné volbě parametrů vydrží kompresi používanou formátem JPEG.

2.1.3 LSB v MMS zprávách

Dnes již prakticky nelze pořídit mobilní telefon, který by neměl zabudovaný fotoaparát a nepodporoval by technologii MMS *(Multimedia Messaging Service)*. Pomocí MMS zprávy je možné posílat text, obrázky, audio a videoklipy. Přenos zajišťují technologie GPRS , UMTS nebo EDGE.

Pro přenos obrázku se u MMS využívají různé formáty, např. PNG, JPG nebo GIF. V praktické části této práce je implementován postup ukrývání zprávy do grafického formátu PNG. Aplikace, která je součástí praktické části diplomové práce, je napsána v

jazyce Java a lze ji upravit tak, aby fungovala pod Java J2ME (micro edition). Java ME nabízí velice kvalitní API určené pro vývoj aplikací pro malá zařízení, např. mobilní telefony. Dále existuje knihovna Bouncy Castle for J2ME s lepší podporou kryptovacích algoritmů [29]. Bylo by tedy možné, že by odesílatel do MMS zprávy ukryl text a příjemce by si pomocí stejné aplikace z MMS obrázku ukrytý text extrahoval.

2.1.4 Ukrytí zprávy v audio souborech (WAV)

LBE (low bit encoding) je velice podobná technika jako LSB se všemi jejími nepříjemnosti jako je možnost rozeznání ukryté zprávy pouhým uchem v případě, že zpráva moc poškodí původní audio soubor. Pokud využijeme nejméně významný bit stejně jako u bitmap, je riziko rozeznání ukryté zprávy malé, každý člověk má však jinou úroveň vnímání zvuku, takže je to velice individuální. LBE je implementováno v praktické části diplomové práce.

Frekvenční maskování je další metoda ukrytí zprávy uvnitř audio soubor. Tato metoda pracuje na principu přidání náhodného šumu (zprávy) do původního vzorku na podobné frekvenci. Zazní-li totiž současně dva zvuky, které jsou si blízké frekvencí - tišší zvuk (řekněme ukrytá zpráva) zanikne v tom hlasitějším.

Obdobou může být **časové maskování**, které pracuje na principu, že když zazní nízký, tichý zvuk těsně před nebo po silnějším zvuku, nedokáže ho lidský sluch rozeznat.

Další metodou může být využití ozvěny (**echo data hiding**). Pracuje na principu, že lidský sluch nedokáže vnímat krátké ozvěny. Informace se do krycího zvukového signálu vkládá přidáváním krátkých ozvěn s dvěma různými délkami mezery, které znamenají 0 nebo 1.

2.1.5 Struktura multimediálních formátů BMP, PNG a WAV

Abychom mohli využít technologie LSB pro ukrytí zprávy, je potřeba znát strukturu dat, do kterých budeme zprávu ukrývat (maskovat). Popíšeme si proto strukturu grafických formátů BMP a PNG a zvukového formátu WAV. Praktická část diplomové práce demonstruje ukrytí dat právě do těchto typů multimediálních formátů.

BMP

Formát BMP (BitMaP) patří v současnosti mezi nejpoužívanější grafické formáty, což je z technologického pohledu docela paradoxní, protože je poměrně složitý na zpracování a přitom nabízí pouze minimum užitečných vlastností. [31]

Formát BMP je navržen tak, že umožňuje ukládání rastrových dat ve čtyřech formátech:

- 1 bit na pixel – dvoubarevné obrázky (používá se barevná paleta, nemusí se tedy jednat pouze o černobílé grafiky, ale o libovolnou kombinaci dvou barev)
- 4 bity na pixel – 16ti barevné obrázky (taktéž se používá barevná paleta o délce 64 bytů, v minulosti nejpoužívanější typ, zejména na grafických kartách EGA a VGA)
- 8 bitů na pixel – 256ti barevné obrázky (opět se používá barevná paleta, tentokrát o délce 1024 bytů)
- 24 bitů na pixel – TrueColor obrázky (16 milionů barev, barevná paleta se nepoužívá, protože každý pixel je reprezentován přímo svou barvou).

Interní struktura souboru typu BMP

Každý korektní soubor typu BMP obsahuje následující datové struktury:

Tabulka 1: Struktura obrazového formátu BMP

Název struktury	Význam
BITMAPFILEHEADER	hlavička BMP souboru
BITMAPINFOHEADER	informační hlavička o obrázku
RGBQUAD[]	tabulka barev (barvová paleta)
BITS	pole bitů obsahujících vlastní rastrová data (pixely)

- BITMAPFILEHEADER: jedná se o datovou strukturu, která obsahuje základní informace o souboru typu BMP. Velikost této struktury je konstantní a má hodnotu 14 bytů.
- BITMAPINFOHEADER: tato datová struktura obsahuje základní meta informace o uloženém obraze. Velikost této struktury je opět konstantní, zde jde o 40 bytů.
- RGBQUAD[]: pole obsahující barvovou paletu ve formě složek RGB. Typická délka barvové palety, tj. počet barev, je 2, 16 a 256.
- BITS: v této datové struktuře jsou uložena vlastní obrazová data. Konkrétní formát těchto dat závisí na použité komprimační metodě (i na tom, zda je vůbec použita) a

na celkovém počtu barev v obrázku. Pro představu v případě 24 bitového bmp obrázku obsahuje pole bitů jednotlivé rastry ve formátu BGR, přičemž se pole začíná plnit od levého spodního rohu.. V případě 8 bitového bmp obrázku jsou vlastní data jen ukazatel do palety barev. V příloze je ukázka 8 bitového obrázku s rozborem dat.

Tabulka 2: Struktura informační hlavičky obrazového formátu BMP

Název položky	Délka položky	Význam
bfType	2 byty	Identifikátor formátu BMP. Aktuální verze formátu BMP zde obsahuje ASCII kód znaků „BM“, tj. 0×42 a 0×4d hexadecimálně.
bfSize	4 byty	Celková velikost souboru s obrazovými údaji. Některé aplikace tuto položku ignorují a dosazují zde nulu.
bfReserved1	2 byty	Tento údaj je rezervovaný pro pozdější použití. V současné verzi formátu BMP zde musí být uložena nulová hodnota.
bfReserved2	2 byty	I tento údaj je rezervovaný pro pozdější použití. V současné verzi formátu BMP zde musí být uložena nulová hodnota.
bfOffBits	4 byty	Posun struktury *BITMAPFILEHEADER* od začátku vlastních obrazových dat.

Meta informace:

Tabulka 3: Meta informace obrazového formátu BMP

Název položky	Délka položky	Význam
biSize	4 byty	Tato položka specifikuje celkovou velikost datové struktury *BITMAPINFOHEADER*
biWidth	4 byty	Tato položka udává šířku obrazu zadávanou v pixelech
biHeight	4 byty	Tato položka udává výšku obrazu zadávanou taktéž v pixelech
biPlanes	2 byty	V této položce je zadaný počet bitových rovin pro výstupní zařízení. V BMP, jakožto formátu nezávislém na zařízení, je zde vždy hodnota 1. Položka existuje z historických důvodů.
biBitCount	2 byty	V této položce je specifikovaný celkový počet bitů na pixel. Podle počtu barev zde mohou být hodnoty 1, 4, 8 nebo 24 (to odpovídá postupně 2, 16, 256ti barvám popř. plnobarevnému režimu).
biCompression	4 byty	Udává typ komprimační metody obrazových dat. Musí být nastavené na jednu z hodnot: 0 (BI_RGB), 1 (BI_RLE8) nebo 2 (BI_RLE4).
biSizeImage	4 byty	Tato položka udává velikost obrazu v bytech. Pokud je bitmapa nekomprimovaná, může zde být nulová hodnota, protože ji je možno vypočítat z rozměrů obrázků a počtu bitů na pixel.
biXPelsPerMeter	4 byty	Udává horizontální rozlišení výstupního zařízení v pixelech na metr. Tato hodnota může být použita například pro výběr obrazu ze skupiny obrazů, který nejlépe odpovídá rozlišení daného výstupního zařízení. Většina aplikací však nemá potřebné informace o výstupním zařízení, a proto do této položky vkládá hodnotu 0.
biYPelsPerMeter	4 byty	Udává vertikální rozlišení výstupního zařízení v pixelech na metr. Opět, jako u předchozí položky, zde většina programů zapisuje hodnotu 0.
biClrUsed	4 byty	Udává celkový počet barev, které jsou použité v dané bitmapě. Jestliže je tato hodnota nastavena na nulu (což provádí většina aplikací), znamená to, že bitmapa používá maximální počet barev. Tento počet lze jednoduše zjistit z položky biBitCount. Nenulová hodnota může být použita například při optimalizacích zobrazování.
biClrImportant	4 byty	Udává počet barev, které jsou důležité pro vykreslení bitmapy. Pokud je tato hodnota nulová (téměř vždy), jsou všechny barvy důležité. Tento údaj je používán při zobrazování na zařízeních, které mají omezený počet současně zobrazitelných barev (například starší grafické karty se 16ti resp. 256ti barevnými režimy). Ovladač displeje může upravit systémovou paletu tak, aby zobrazil daný obrázek co nejvěrněji. Také je vhodné upravit paletu metodou seřazení jednotlivých barev podle důležitosti.

Není v rozsahu této práce dále popisovat paletu nebo kompresi BMP, detailní informace lze nalézt např. v [31]. V praktické části si provedeme detailní analýzu konkrétního obrázku i s dopadem ukrytí zprávy.

PNG

Grafický formát PNG (*Portable Network Graphics)* je souborový formát určený pro ukládání, přenos i zobrazování rastrových obrázků, tj. obrázků, ve kterých je grafická informace uložena v pravidelné mřížce složené z jednotlivých elementárních grafických plošek nazývaných pixely. Pixely nesou informace o své barvě, popř. i průhlednosti a u PNG mezi nimi není stanoven žádný implicitní vztah. Obrázky jsou při použití PNG ukládány ve zkomprimované podobě, přičemž použitý komprimační algoritmus je bezeztrátový, tj. ani při ukládání ani při načítání či přenosu obrázku nedochází ke ztrátám obrazové informace.

Celý binární soubor s uloženým obrázkem se skládá z hlavičky, která je neměnná, tj. neobsahuje ani číslo verze. Za hlavičkou se nachází libovolné množství takzvaných chunků, což jsou pojmenované bloky, z nichž každý je opatřen svou délkou, typem a kontrolním součtem CRC (Cyclic Redundancy Check/Cyclic Redundancy Code). [32]

Hlavička obrázku

Hlavička souboru PNG má délku 8 bytů a je v každém souboru stejná. Hexadecimální vyjádření: 89 50 4E 47 0D 0A 1A 0A

Tabulka 4: Struktura hlavičky PNG obrázku

Byte	Význam bytu
89	jedná se o byte s nastaveným nejvyšším bitem (detekce sedmibitového přenosu)
50 4e 47	řetězec ‚PNG', spolu s prvním bytem jednoznačně detekuje typ souboru na platformách, které typ rozpoznávají z obsahu a ne z koncovky
0d 0a	znaky **CR+LF**, detekce náhrady za jinou sekvenci: **CR**, **LF** či **LF+CR**
1a	znak **Ctrl+Z**, pro příkaz *type* MS-DOSu
0a	znak **LF**, detekce náhrady za **CR+LF** či **LF+CR**

Po hlavičce přichází série chunků, z nichž každý zprostředkovává jistou informaci o obrazu. Chunky deklarují sebe jako „rozhodující" nebo „pomocné". Pokud se program setká s pomocným chunkem, kterému nerozumí, může ho bezpečně ignorovat. Struktura tohoto chunku je založena tak, aby dovolovala slučitelnost PNG formátů se staršími verzemi.

Každý chunk má hlavičku specifické velikosti a typu. Pak ihned následují vlastní data, posléze probíhá kontrolní součet dat.

WAV

WAV (nebo také WAVE) je zkratka pro zvukový formát, který vytvořily firmy IBM a Microsoft pro ukládání zvuku na PC. Je to speciální varianta obecnějšího formátu RIFF (Resource Interchange File Format) navrženého pro výměnu dat mezi programy. [27]

Hlavním rysem těchto formátů je způsob ukládání vlastních dat - ta jsou dělena do na sobě nezávislých bloků (chunks). Každý blok má svůj prefix, který následující informace nějak popisuje. Prefix je 4-znakový řetězec (hodnota "RIFF") následně doplněný 4-bajtovým číslem udávajícím délku zbytku bloku v bajtech.

V našem případě má WAV soubor jeden blok formátu RIFF, který v sobě obsahuje dva důležité bloky: blok popisující vlastní hlavičku (řetězec s hodnotou "WAVE") a blok se zvukovými daty. Může obsahovat i další (komentáře, copyright atd.), ty však nejsou pro práci ze zvukem důležité.

Hlavička WAV souboru

Tabulka 5: Struktura hlavičky zvukového souboru wav

Jméno	Délka v bajtech	Význam
wID	4	ASCII řetězec "WAVE".
fID	4	ASCII řetězec "frm ". Všechny ID bloků by měly obsahovat 4 znaky, proto ta mezera
fLen	4	Pevná hodnota, musí být vždy 16
wFormatTag	2	Tyto dva bajty vždy definují, jakým způsobem jsou vlastní zvuková data uložena. Většinou se setkáme s hodnotou 1, což znamená PCM (Pulse Code Modulation).
nChannels	2	Počet kanálů. Tudíž 1 = mono, 2 = stereo. Je možné mít i více než dva kanály, tyto případy nejsou však moc časté.
nSamplesPerSec	4	Kmitočet, uvádí se v Hz. Typické hodnoty jsou 11025 (telefonní kvalita), 22050 (hi-fi kvalita), 44100 (CD kvalita). Většinou se nesetkáme z hodnotami nižšími než 8000 a vyššími než 48000.
nAvgBytesPerSec	4	Průměrná datový průtok za sekundu. Informace zejména pro přehrávače. U PCM formátu je tato hodnota však zbytečná, neboť si ji můžeme sami vypočítat vynásobením frekvence počtem kanálů a počtem bajtů na vzorek.
nBitsPerSample	2	Počet bitů na jeden vzorek. Toto číslo většinou nabývá hodnoty 8 nebo 16 bitů. Ačkoliv jsou osmibitové zvuky daleko méně kvalitnější, zabírají dvakrát méně místa, než šestnáctibitové.

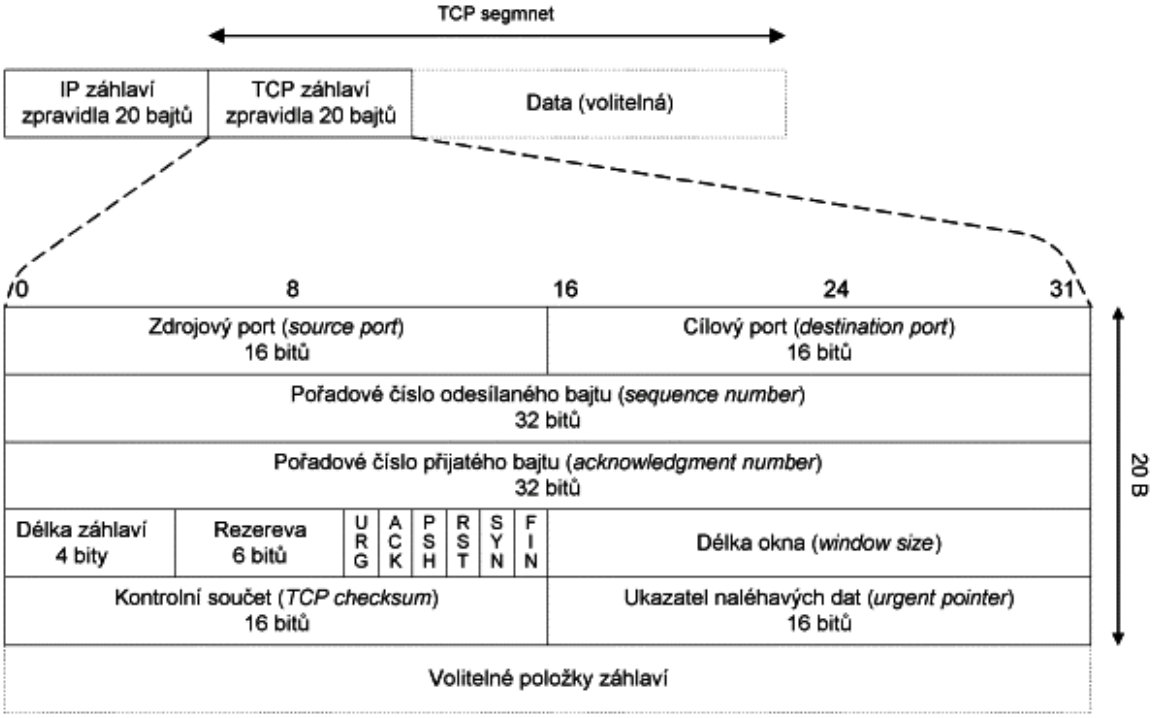
Vlastní data

Za hlavičkou následují vlastní data. Abychom je dokázali přečíst, potřebujeme znát hodnoty nBitsPerSample a nChannels. Pokud je počet bitů na vzorek větší než 0 a menší jak 8, pak každý PCM vzorek bude jednobajtový, a to nabývající hodnoty 0 - 255. Pokud

bude mezi 9ti a 16ti bity, bude nabývat hodnoty od -32768 do +32767. Dále si musíme uvědomit, že pokud bude mít zvuk více jak jeden kanál, jsou data pro jednotlivé kanály každého vzorku uspořádány za sebou.

2.2 Využití volných bitů protokolu TCP

Ukrývání dat do datagramů protokolů TCP nebo IP stojí na okraji pozornosti odborníků. Formát datagramů je definován v obrázku 4. Záleží pouze na dohodě komunikujících stran, která pole datagramů budou tajnou zprávu (či její fragmenty) obsahovat a jak v nich bude zpráva kódovaná. Aby takto generovaný síťový provoz nebudil pozornost, výběr polí je značně omezen a přenosová kapacita tajných bitů malá. Za povšimnutí stojí např. využití rezervních 6 bitů. [14]



Obrázek 4: Struktura hlavičky protokolu TCP

2.3 RSTEG (retransmission steganography)

Polští výzkumníci [17] chtějí v rámci své metody využít toho, že pokud při přenosu paketu v protokolu TCP dojde k chybě, pošle se paket znovu. Odesílatel (ve smyslu zařízení/systému) čeká na to, až příjemce potvrdí, že data byla doručena v pořádku, jinak přenos automaticky zopakuje. Chyba v přenosu není nic zase tak výjimečného, týká se přibližně promile veškerých paketů. Nyní vědci přišli se systémem RSTEG (retransmission steganography). Dejme tomu, že z místa A do B jde e-mail. Systém B pak předstírá, že

přenos selhal a A má poslat informaci znovu. Tentokrát se pošle ale již upravený paket a rozdíl mezi nimi je právě šifrovaným sdělením. Kdyby někdo provoz mezi oběma stranami monitoroval (tedy útok typu man-in-the-middle), neměl by získat žádné podezření – vždyť přece pokud jeden paket dorazil porušený, je logické, že ten samý poslaný příště se od něj bude lišit.

Odhalit by tento systém šel zřejmě až ve chvíli, kdyby představoval statisticky významnou část celého přenosu v rámci protokolu TCP.

2.4 VoIP steganography

Wojciech Mazurczyk a Krzysztof Szczypiorski [24] z Telekomunikačního ústavu ve Varšavě přišli v roce 2009 s myšlenkou, že podobně jako utajení zprávy do obrázku by se dala zpráva utajit i v rámci zvukového záznamu, konkrétně v datech přenášených jako VoIP. Jedna z metod, kterou oba vědci pro přenos zprávy navrhli, spočívá ve využití volných polí v protokolech RTCP a RTP.

"VoIP-steganografii" nemusí využívat pouze telefonující strany. Data může mezi pakety hovoru vkládat i někdo třetí. Podobně jako u steganografie by bez předchozího podezření byl takový postup prakticky neodhalitelný. Ve VoIP se přenáší relativně velké množství dat, takže "střídmá" zpráva by se na kvalitě zvuku neměla nijak projevit.

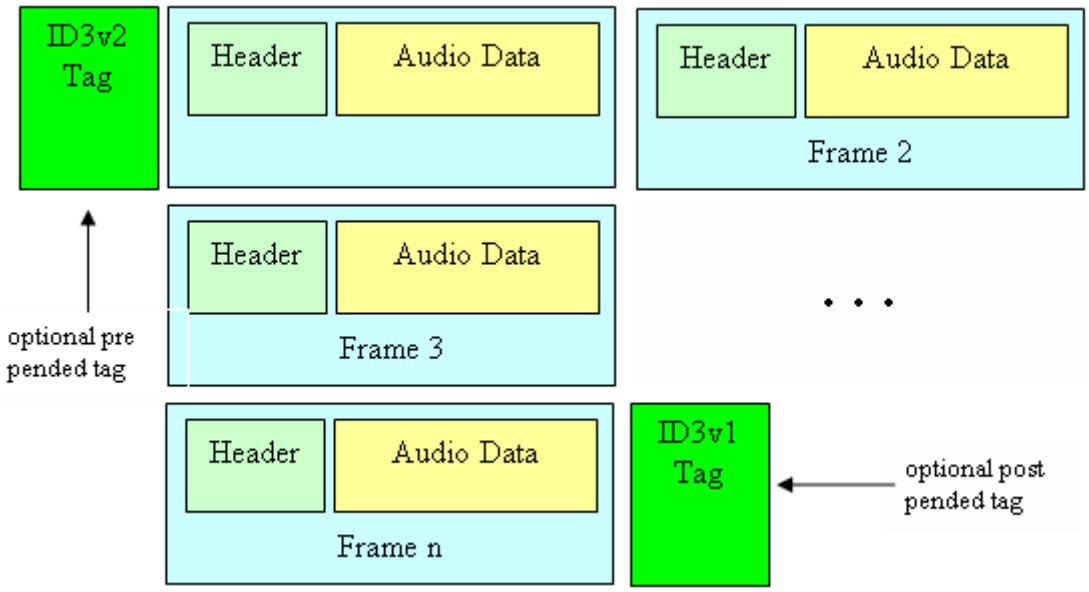
2.5 Ukrytí do spustitelných souborů

Spustitelné binární soubory: např. u Linuxu jsou tyto interně členěny do segmentů, kterých může být libovolný počet a jejichž binární obsah používá buď samotný soubor ke svému běhu (např. segment vlastních instrukcí procesoru, sekce statických dat, …) nebo je používají externí programy pro svoje účely (např. ladicí metadata, ikona). Je snadné novou sekci, jejíž přítomnost nikdo nebude očekávat (ani ji interpretovat), do binárního souboru s programem přidat a použít ji pro tajné uložení, potažmo přenos informace. [35]

2.6 Ukrytí do MP3 audio souborů

Další ze zajímavých technik může být ukrytí zprávy do zvukového formátu MP3. MP3 (MPEG-1 Layer III) je formát ztrátové komprese zvukových souborů, založený na kompresním algoritmu MPEG (Motion Picture Experts Group). Při zachování poměrně vysoké kvality umožňuje zmenšit velikost hudebních souborů v CD kvalitě přibližně na desetinu.

Dle [23] se MPEG audio soubor (.MP3 soubor) skládá z malých částí nazývaných rámce (frames). Každý rámec je nezávislý a má svoji vlastní hlavičku a audio informace. Není zde žádná hlavička celého souboru, díky čemuž lze vyjmout jakoukoliv část souboru a přehrát ji správně.



Obrázek 5: Popis struktury MP3 souboru

Jestliže chceme získat informace o MPEG souboru, obvykle stačí najít první rámec, přečíst jeho hlavičku a předpokládat, že ostatní rámce jsou stejné (to ale nemusí platit vždy).

Hlavička rámce se nachází v prvních čtyřech bytech (tj. 32 bitů) rámce. Prvních jedenáct bitů hlavičky je označeno jako frame sync (synchronizace rámce) a je vždy nastaveno na hodnotu 1. Proto lze najít v souboru první výskyt jedenácti nastavených bitů (tzn. najít první byte s hodnotou 255, který je následován bytem s nastavenými nejdůležitějšími třemi bity) a potom přečíst ze souboru celou hlavičku a zjistit její správnost.

Pokud potřebujeme znát velikost rámce, tak z hlavičky rámce načteme hodnotu datového toku (bitrate), vzorkovací frekvenci (SampleRate) a doplnění (Padding) a použijeme následující vzorec pro určení velikosti rámce:

$$FrameSize = \frac{144 * BitRate}{SampleRate} + Padding \qquad (1)$$

tedy např. pro BitRate = 128000, SampleRate=44100, Padding=0 je velikost rámce 417B.

Tabulka 6: Struktura hlavičky jednoho mp3 rámce

Délka [bit]	Poz. [bit]	Popis informace
11	31-21	Synchronizace rámce (všechny bity nastaveny)
2	20-19	Verze MPEG: 00 - MPEG verze 2.5 01 - rezervováno 10 - MPEG verze 2 11 - MPEG verze 1
2	18-17	Layer: 00 - rezervováno 01 - Layer-3 10 - Layer-2 11 – Layer-1
1	16	0 - chráněno pomocí CRC, 16bitové CRC následuje hlavičku 1 - není chráněno
4	15-12	Bitrate
2	11-10	Vzorkovací frekvence [kHz] bity MPEG1 MPEG2 MPEG2.5 00 44100 22050 11025 01 48000 24000 12000 10 32000 16000 8000 11 rezervováno
1	9	Doplňující bit (padding) 0 - rámec není doplněn 1 - rámec je doplněn jedním bitem navíc
1	8	Tajný bit (private bit)
2	7-6	Mód kanálů 00 - stereo 01 - Joint stereo 10 - dual channel (stereo) 11 - single channel (mono)
1	5	MS stereo (jen pro Joint stereo) 0 - ne 1 – ano
1	4	Intensity stereo (jen pro Joint stereo) 0 - ne 1 – ano
1	3	0 - je chráněno autorským právem 1 - není chráněno autorským právem
1	2	0 - kopie originálního média 1 - originální médium
2	1-0	Zvýraznění: 00 - žádné 01 - 50/15 ms 10 - rezervováno 11 - CCIT J.17

Ze struktury hlavičky mp3 rámce je dle tabulky 6 patrné, že ne všechny informace (bity) bývají použity při zpracování souboru přehrávačem, nebo bývají použity velice zřídka. Dle [23] lze na ukrytí zprávy dobře využít tajný bit, bit autorské ochrany, bit originality a bity zvýraznění. Změna těchto bitů nemá vliv na integritu zvukového rámce, nejhorší co se stane je, že v přehrávači nebude zobrazena informace o autorství nahrávky. Získali jsme tedy 5 bitů z každého rámce mp3 skladby, písnička délky přibližně 4 minuty obsahuje cca 8000 rámců po 5ti bitech, tzn. prostor pro přibližně 5000 znaků.

II. PRAKTICKÁ ČÁST

3 POUŽITÁ TECHNOLOGIE PŘI VÝVOJI APLIKACE

Pro vývoj aplikace Steganography byl zvolen programovací jazyk Java. Nespornou výhodou aplikací vytvořených v Javě je jejich přenositelnost mezi platformami.

3.1 Vývojové prostředí Eclipse

Eclipse je open source vývojové prostředí (IDE) určené pro programování v jazyce Java. Flexibilní návrh této platformy dovoluje rozšířit seznam podporovaných programovacích jazyků za pomoci pluginů, například o C++ nebo PHP. Právě pluginy umožňují toto vývojové prostředí rozšířit například o návrh UML, či zápis HTML nebo XML.

Projekt Eclipse vznikl uvolněním kódu IBM pod EPL licencí. Pro účely tohoto projektu byl vyvinut grafický framework SWT. Výhodou SWT je nativní vzhled aplikací na každé platformě, viz popis dále.

3.2 Grafická knihovna SWT

Java nabízí několik možností na vykreslení grafiky formulářů. První implementací grafiky (Java 1.0) byla knihovna zvaná Abstract Windowing Toolkit (AWT). Knihovna AWT je funkční na všech platformách, které jsou Javou podporovány, ale vzhledově je spíše podprůměrná.

Java 2 přišla s novou GUI knihovnou Java Foundation Classes (JFC), jejíž grafická část se nazývá Swing. Swing má velmi dobrý programovací model. Má všechny komponenty, které by mělo mít každé moderní uživatelské rozhraní. Příklady povedených Swing aplikací lze najít třeba na Sun serveru. Při vývoji Swingu byl ovšem obětován výkon. Všechny komponenty jsou „lehké“ a tak bylo možno dosáhnout toho, že na všech platformách Swing aplikace vypadají stejně, protože každá komponenta je napsána v Javě. AWT oproti tomu využívá nativních služeb daného operačního systému a na každém OS vypadá jinak.

SWT je základní grafická knihovna pro Javu, která nemá žádnou závislost na standardním grafickém API Javy. Jeho výkladní skříň je vývojové prostředí Eclipse a je IBM iniciativou. Je sice napsáno v Javě, ale nesmí používat ochranou známku Java IDE a to proto, že není *Java pure* - obsahuje nativní kód z jednotlivých operačních systémů, na které je portováno. SWT lze s určitou nadsázkou považovat za knihovnu, která leží mezi

AWT a Swingem. Komponenty jsou „vybavenější“ než u AWT, ale nemají takové množství vlastností jako u Swingu.

SWT má podobný přístup jako AWT. Všechny komponenty jsou svázány s nativními službami příslušného operačního systému. Pouze v případě, kdy příslušný operační systém danou službu nepodporuje, SWT ji emuluje. Zřídka užívané vlastnosti jsou vynechané.

Velkou výhodou SWT je, že GUI vypadá na všech platformách jak je jejich uživatel zvyklý a nic ho nepřekvapuje. Když Microsoft uvolnil XP verzi Windows, SWT automaticky převzalo jeho nový vzhled, Swing toho není schopen. V SWT není možné u alokovaných zdrojů jako jsou fonty a barvy se spolehnout na automatickou správu paměti pomocí garbage collector, ale tyto zdroje je nutné uvolňovat ručně. Většinu SWT tříd nelze dědit.

3.3 Bouncy Castle Crypto API

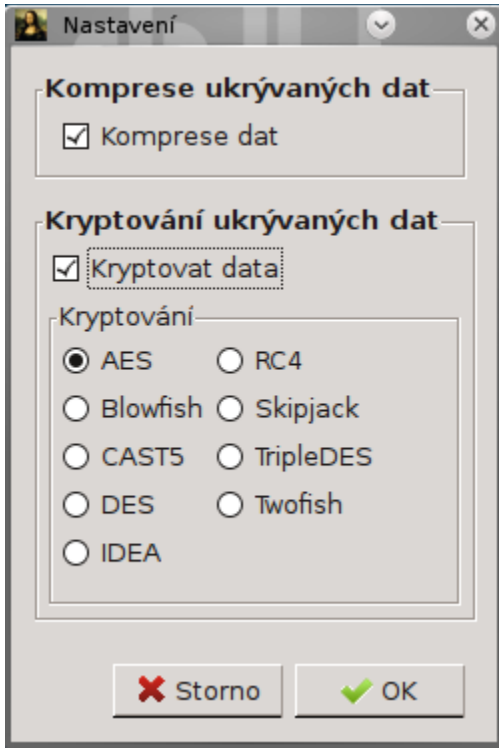
Bouncy Castle je kolekce API pro použití kryptování. Projekt je zaměřen na podporu kryptování v programovacích jazycích Java a C#. Bouncy Castle nabízí implementace bezpečnostních struktur, šifrovacích algoritmů, způsobů zarovnání bloku, hašovacích funkcí, generátorů klíče, digitálního podepisování a mnohé další. Vývoj je řízen australskou skupinou nazývající se The Legion of the Bouncy Castle, a proto se na knihovnu nevztahují americké restrikce na export kryptografického software. Spojením JRUM a BouncyCastle získáme prostředek, jak uskutečnit ověřování klientů a zrcadel na základě sdíleného klíče a symetrické kryptografie.

Jádro Javy obsahuje U.S. restrikce, které zakazují využití silných klíčů v kryptografických algoritmech, pravděpodobně aby nebyly zneužity teroristy. U.S. restrikce lze obejít doinstalováním rozšíření Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 6. Pro správnou funkčnost Bouncy Castle toto budeme potřebovat.

4 IMPLEMENTACE OBECNÝCH FUNKCIONALIT

Aplikace `Steganography`, která je součástí diplomové práce, ukazuje techniky ukrývání informace (zprávy) do bitmapy a do zvukových souborů. Použité techniky jsou popsány níže v samostatných kapitolách. Pro zabezpečení ukrývané informace je možné dále použít techniky kryptování, případně je možné použít kompresi pro získání většího prostoru ukrývaným informacím.

Využití komprese a kryptování je čistě uživatelské rozhodnutí, které lze ovlivnit v nastavení programu, viz následující obrázek.



Obrázek 6: Nastavení programu

4.1 Komprese ukrývané zprávy

Úkolem komprese (též komprimace) dat je zmenšit datový tok nebo zmenšit potřebu zdrojů při ukládání informací. [42] Obecně se jedná o snahu zmenšit velikost datových souborů zvláštními postupy – kódováním, které je dané zvoleným kompresním algoritmem – se ze souboru odstraňují redundantní (nadbytečné) informace, zvyšuje se entropie dat. Komprese dat lze rozdělit do dvou základních kategorií:

Ztrátová komprese – při kompresi jsou některé informace nenávratně ztraceny a nelze je zpět rekonstruovat. Používá se tam, kde je možné ztrátu některých informací tolerovat a kde nevýhoda určitého zkreslení je bohatě vyvážena velmi významným

zmenšením souboru. Používá se pro kompresi zvuku a obrazu (videa), při jejichž vnímání si člověk chybějících údajů nevšimne nebo si je dokáže domyslet (do určité míry). Nehodí se tedy pro naši ukázkovou aplikaci, my potřebujeme bezeztrátovou kompresi.

Bezeztrátová komprese – obvykle není tak účinná jako ztrátová komprese dat. Velkou výhodou je, že komprimovaný soubor lze opačným postupem rekonstruovat do původní podoby. To je nutná podmínka při přenášení počítačových dat, výsledků měření, textu apod., kde by ztráta i jediného znaku mohla znamenat nenávratné poškození souboru.

4.1.1 Gzip komprese

Gzip vznikl jako náhrada za již zastaralý compress, přičemž ve většině případů dosahuje lepších kompresních poměrů a je navíc zbaven všech problémů s patenty. Později byl Gzip zařazen mezi GNU projekty, a tak se stal součástí linuxových distribucí. Jeho autory jsou pánové Jean-loup Gailly a Mark Adler. Je samozřejmě podporován pro všechny platformy.

Gzip je založený na algoritmu DEFLATE, který je kombinací LZ77 a Huffmanova kódování. DEFLATE byl určený k nahrazení LZW a dalších patentem-zatížených algoritmů pro kompresi dat, který měl v té době omezenou použitelnost komprese a dalších populárních archivátorů. [22], [43]

Implementace komprese zprávy

Zpráva k zakódování je uložena v poli bajtů `data`. Výstupem metody `compress` je pole bajtů, které obsahuje zkomprimovanou zprávu, která bude následně ukryta do bitmapy nebo audio souboru.

```
public static byte[] compress (byte[] data) {
     ByteArrayOutputStream buffer = new
                 ByteArrayOutputStream();
     OutputStream deflater = new GZIPOutputStream(buffer);
     deflater.write(data);
     deflater.close();
     return buffer.toByteArray();
}
```

Dekomprese zprávy

Dekomprese zprávy je provedena inverzním postupem.

```
public static byte[] decompress(byte[] data) {
      ByteArrayOutputStream buffer = new ByteArrayOutputStream();
      ByteArrayInputStream in = new ByteArrayInputStream(data);
      InputStream inflater = new GZIPInputStream(in);
      byte[] bbuf = new byte[256];
      while (true) {
            int r = inflater.read(bbuf);
            if (r < 0) {
                  break;
            }
            buffer.write(bbuf, 0, r);
      }
      return buffer.toByteArray();
}
```

4.2 Kryptování (šifrování)

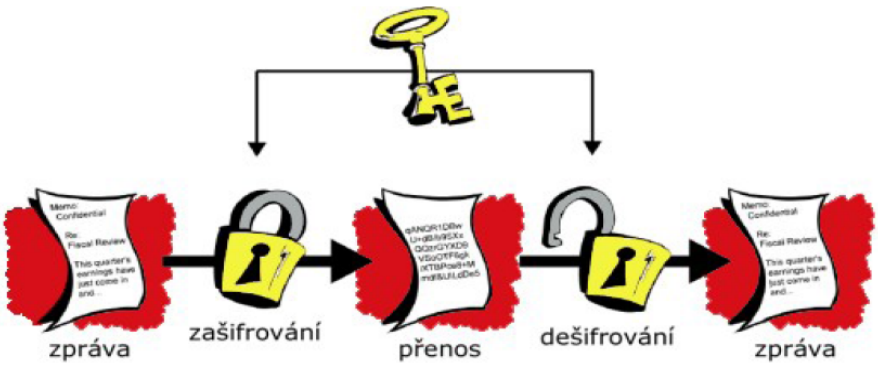
Zpráva těsně před ukrytím do bitmapy nebo audio souboru může projít fází kryptování, které se postará o znesnadnění čitelnosti zprávy v případě, že útočník odhalí použitou technologii ukrývání. Při použití dostatečně kvalitního kryptovacího algoritmu (např. AES) a použití kvalitního klíče dosáhneme prakticky neprolomitelnosti rozluštění zprávy útočníkem.

Pro techniky kryptování byl zvolen provider **Bouncy Castle** a využito **Crypto API** pro Javu (Java Cryptography Extension a Java Cryptography Architecture).

4.2.1 Techniky šifrování

Šifrování z pohledu použití klíčů k zašifrování a dešifrování zprávy můžeme rozdělit na šifrování symetrické, asymetrické a hybridní, které spojuje prvky ze symetrického i asymetrického šifrování. [21]

Symetrické šifrování



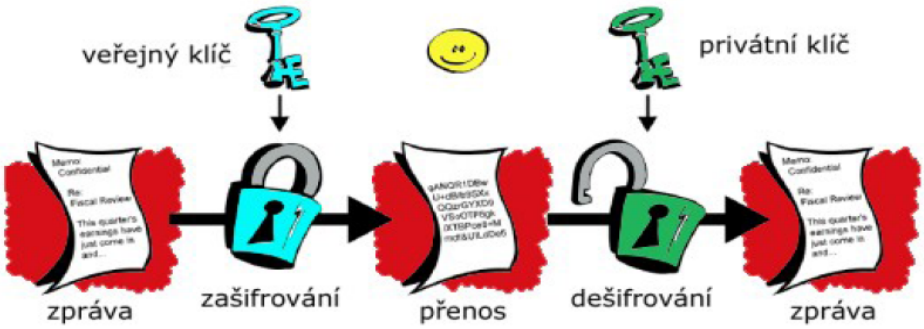
Obrázek 7: Schématické znázornění symetrického šifrování

Pro představu, jak kryptování znepříjemní útočníkovi získat ukrytou zprávu, byly v programu implementovány různé typy symetrických šifer. Symetrické (nebo konvenční) šifrování je metoda, při které je text zašifrován s pomocí jistého klíče a může být obnoven jen se znalostí tohoto klíče, což je zároveň jeho největší slabina.

Symetrické kódy mají jako hlavní výhodu rychlost algoritmu. Na druhou stranu je nutné aby se příjemce i odesílatel dohodli na jednom klíči, který si musí nějakým bezpečným způsobem vyměnit a který budou znát pouze oni dva. Problémem je tedy distribuce klíče - jak dostat klíč k příjemci aniž by se ho chopil někdo nepovolaný? Samotné symetrické šifrování nemůže nikdy problém předání klíče vyřešit.

Kromě problému distribuce klíče má tento typ šifrování další nevýhodu: pokud je účastníků komunikace víc. Pokud chceme mít pro každou dvojici komunikujících stran jiný klíč, potřebujeme pro n účastníků n(n-1)/2 klíčů (jestliže je počet komunikujících malý, příliš to nevadí). Symetrické šifrování je mnohem jednodušší než asymetrické šifrování. Jednak nepotřebuje tak výkonné počítače, jednak je jednodušší jeho princip. Symetrické šifrování vzniklo mnohem dříve.

Asymetrické šifrování



Obrázek 8: Schématické znázornění asymetrického šifrování

Pro úplnost (není implementováno v programu) existuje kromě symetrického šifrování i asymetrické šifrování neboli kryptografie veřejného klíče. Tato metoda byla vyvinutá Whitfieldem Diffiem a Martinem Hellmanem v roce 1975. Každý účastník komunikace má dva klíče. První z nich je veřejný (přístupný všem) a druhý je privátní (soukromý). Cokoli zašifrováno jedním klíčem, lze dešifrovat pouze druhým klíčem a naopak.

Velkou výhodou tohoto přístupu je, že jeden z klíčů můžeme dát k dipozici komukoliv (tedy zveřejnit ho). Kdokoli nám pak chce napsat zprávu, použije k jejímu zašifrování tento veřejný klíč. Ani on sám, ani žádný jiný vlastník našeho veřejného klíče ji nebude schopen dešifrovat. Toho bude schopen pouze držitel druhého páru - privátního klíče, jímž bychom měli být pouze my. Chceme-li poté adresátovi poslat odpověď, nemůžeme ji zašifrovat svým privátním klíčem, neboť by ji byl schopen dešifrovat kdokoli, ale musíme použít veřejný klíč adresáta. Hlavní výhodou asymetrického šifrování je, že soukromé klíče jsou pouze u jejich majitelů a vně se pohybují pouze veřejné klíče. Asymetrické šifrování má jednu velkou nevýhodu - je velmi náročné na matematické operace, tedy i na výkon počítače.

4.2.2 Nastavení providera Bouncy Castle

Třída Bouncy Castle implementuje nejrozšířenější kryptovací algoritmy. Její použití v jazyce Java provedeme následující inicializací:

```
Security.addProvider(new
org.bouncycastle.jce.provider.BouncyCastleProvider());
```

4.2.3 Zakódování zprávy (kryptování) a dekódování zprávy (dekryptování)

Zpráva k zakódování je předávána prostřednictvím pole bajtů. Pole bajtů (zakódovaných vybraným kryptovacím algoritmem) je také návratovou hodnotou metody

`encrypt`. Metoda `init` provede inicializaci vybraného kryptovacího algoritmu. Její popis bude probrán dále v textu.

```
public byte[] encrypt(byte[] data) {
     // Inicializace algoritmu (a klice)
     // viz popis jednotlivých algoritmů
     init ();
     // Nastaveni algoritmu na kódování (encrypt)
     cipher.init(Cipher.ENCRYPT_MODE, key);
     byte[] cipherText = new
                      byte[cipher.getOutputSize(data.length)];
     int ctLength = cipher.update
                     (data, 0, data.length, cipherText, 0);
     ctLength += cipher.doFinal(cipherText, ctLength);
     // cipherText obsahuje zakryptovanou zprávu v poli bajtů
     return cipherText;
}
```

Zakryptovaná zpráva je do metody `decrypr` předávána prostřednictvím pole bajtů `data`. Pole bajtů (dekódovaných správným kryptovacím algoritmem) je také návratovou hodnotou metody `decrypt`. Metoda `init()` provede inicializaci vybraného kryptovacího algoritmu, je pro každý algoritmus specifická a její popis a implementace bude rozebrána dále v textu.

```
public byte[] decrypt(byte[] data) {
     // Délka zprávy
     int ctLength = data.length;
     // Inicializace algoritmu (a klice)
     // viz popis jednotlivých algoritmů
     init ();
     // Nastaveni algoritmu na dekódování (decrypt)
     cipher.init(Cipher.DECRYPT_MODE, key);
     byte[] plainText =
                new byte[cipher.getOutputSize(ctLength)];
     int ptLength = cipher.update
                (data, 0, ctLength, plainText, 0);
     ptLength += cipher.doFinal(plainText, ptLength);
     // plainText obsahuje dekryptovanou zprávu v poli bajtů
     return Arrays.copyOf(plainText, ptLength);
}
```

4.2.4 Implementace symetrických šifer jazykem Java

Do aplikace Steganography byly implementovány následující symetrické šifry:

AES

AES (Advanced Encryption Standard) [38], [39] je schválený standard amerického úřadu pro standardizaci (NIST), který byl udělen symetrické blokové šifře Rijndael. Šifra byla vyvinuta Belgičany Joanem Daemenem a Vincentem Rijmenem a zařazena do veřejné soutěže NIST vyhlášené 2. 1. 1997 o federální šifrovací AES.

Šifra využívá symetrického klíče, tj. stejný klíč je použit pro šifrování i dešifrování. Délka klíče může být **128**, **192** nebo **256 bitů**.

Metoda šifruje data postupně v blocích s pevnou délkou 128 bitů. Šifra se vyznačuje vysokou rychlostí šifrování. V současné době není veřejně znám žádný případ plného prolomení této metody ochrany dat.

AES je nástupcem zastaralého šifrovacího standardu DES, který byl 17.6.1997 prolomen.

Příklad inicializace algoritmu:

```
Byte[] key = new SecretKeySpec(keyBytes, "AES");
Cipher cipher = Cipher.getInstance("AES/ECB/PKCS7Padding", "BC");
```

Blowfish

Šifra Blowfish je navržena B. Schneierem a poprvé byla zveřejněna roku 1994. Jedná se o symetrickou blokovou šifru s velikostí bloku 64 bitů a délkou klíče **8** až **448 bitů** (tj. 56B) v krocích po 8 bitech. Sám autor říká: "Již od začátku jsem zamýšlel, že Blowfish bude zcela volná – nepatentovaná, nelicencovaná a bez copyrightu – alternativa k DESu."

Přestože byla tato šifra poskytnuta veřejnosti a měla k ní tedy přístup celá řada cryptoanalytiků, doposud není veřejně znám případ prolomení této šifry v plném tvaru (full-round). [13], [39]

Příklad inicializace algoritmu:

```
key = new SecretKeySpec(keyBytes, "Blowfish");
cipher = Cipher.getInstance("Blowfish");
```

CAST5 (CAST 128)

Algoritmus šifry byl navržen v roce 1996 dvojicí inženýrů C. Adamsem a S. Tavaresem, je postavena na architektuře jiných šifer CAST rodiny. CAST5 podporuje 12 nebo 16 průchodů cyklem (Feistelova síť) s délkou bloku 64 bitů. Délka klíče šifry je v rozsahu **40** až **128 bitů** (v 8 bitových krocích). Plný průchod - 16 cyklů – je použit v případě, když klíč je větší délky než 80 bitů. [39]

Příklad inicializace algoritmu:

```
key = new SecretKeySpec(keyBytes, "CAST5");
cipher = Cipher.getInstance("CAST5", "BC");
```

DES

Algoritmus **D**ata **E**ncryption **S**tandard byl patentován roku 1975 a do roku 1998 byl (sice s výhradami) standardem pro šifrování informací. [5]

DES je založen na algoritmu firmy IBM, který se jmenoval Lucifer. Tento původní algoritmus byl proprietární a používal klíč délky 128 bitů. Při implementaci DESu došlo ke zkrácení klíče na pouhých **56 bitů**. Toto zkrácení bylo vyvoláno bezpečnostním úřadem NSA kvůli obavám, že by algoritmus byl velmi silný a mohl by být zneužit pro nelegální činnost.

V současnosti je tato šifra považována za nespolehlivou, protože používá klíč pouze o délce 64 bitů, z toho 8 je kontrolních a 56 efektivních. Navíc obsahuje algoritmus slabiny, které dále snižují bezpečnost šifry. Díky tomu je možné šifru prolomit útokem hrubou silou za méně než 24 hodin. [36], [39]

Příklad inicializace algoritmu:

```
key = new SecretKeySpec(keyBytes, "DES");
cipher = Cipher.getInstance("DES/ECB/PKCS7Padding", "BC");
```

IDEA

International **D**ata **E**ncryption **A**lgorithm (Mezinárodní algoritmus pro šifrování dat) je symetrická bloková šifra, kterou navrhli Xuejia Lai a James L. Massey ze Švýcarského národního technologického institutu (ETHZ) v Zürichu. Poprvé byla popsána v roce 1991. Tento algoritmus měl nahradit Data Encryption Standard (DES). IDEA je drobným přepracováním dřívější šifry PES (Proposed Encryption Standard), původně se nazývala IPES (Improved PES).

V současné době je IDEA licencována celosvětově MediaCryptem. IDEA byla používána v Pretty Good Privacy (PGP), do kterého byla začleněna poté, co původní šifra, která byla použita ve verzi 1.0, („Bass-O-Matic“) byla shledána nespolehlivou. IDEA je volitelným algoritmem v OpenPGP.

IDEA pracuje po 64 bitových blocích za použití **128 bitového klíče**. Skládá se z řady osmi identických transformací a vstupní transformace (poloviční průchod). Procesy šifrování a dešifrování jsou podobné. IDEA odvozuje velkou část své bezpečnosti ze střídání operací z různých grup – modulární sčítání a násobení a bitové nonekvivalence (XOR) – které jsou v jistém smyslu algebraicky neslučitelné. [39], [40]

Příklad inicializace algoritmu:

```
key = new SecretKeySpec(keyBytes, "Idea");
cipher = Cipher.getInstance("Idea");
```

RC4

Šifra RC4 (známá také pod názvem ARC4) je široce používaná proudová šifra a je implementována v celé řadě používaných protokolů, jako např. Secure Sockets Layer „SSL“ (zabezpečení komunikace po internetu) nebo WEP (zabezpečení bezdrátových sítí). Autorem šifry je Donald Rivest.

Klíč pro RC4 může mít maximálně 256 bytů (**2048 bitů**). Pro funkčnost se uvažuje o klíčích zarovnaných na byty. V praxi se využívá 128bitový klíč na území USA a 40bitový. Vstupem je klíč o volitelné délce. Princip šifry je míchání bytů klíče spojené s permutací klíče. RC4 je velmi zajímavá a neobyčejná a analytickou metodou zatím nenapadnutá šifra. [39]

Příklad inicializace algoritmu:

```
key = new SecretKeySpec(keyBytes, "RC4");
cipher = Cipher.getInstance("RC4", "BC");
```

Skipjack

Šifra byla v původní verzi používána ministerstvem obrany USA, ale v roce 1998 byl šifrovací algoritmus odtajněn veřejnosti. Skipjack je 64bitová symetrická šifra s 80bitovým klíčem. Celkem 64 bitů otevřeného textu projde 32 šifrovacími rundami. [20], [39]

Příklad inicializace algoritmu:

```
key = new SecretKeySpec(keyBytes, "SKIPJACK");
cipher = Cipher.getInstance("SKIPJACK", "BC");
```

TripleDES

Kvůli vyšší bezpečnosti byl přijat standard Triple DES (též TDES, 3DES). Triple DES je bloková šifra založená na šifrování Data Encryption Standard (DES), které aplikuje třikrát a tak zvyšuje její odolnost proti prolomení. Triple DES je oproti novějším algoritmům (AES) pomalejší, a proto se postupně přestává používat. [37], [39]

Příklad inicializace algoritmu:

```
key = new SecretKeySpec(keyBytes, "DESede");
cipher = Cipher.getInstance("DESede", "BC");
```

Twofish

Twofish je symetrická bloková šifra s délkou klíče **128**, **192** nebo **256** bitů, vyvinutá Brucem Schneierem. Jde o nepatentovanou otevřenou šifru pro volné použití. Šifra Twofish byla jedním z pětice finalistů soutěže standardu AES.

Algoritmus šifry pro klíč o velikosti 128 bitů je nepatrně pomalejší než u Rijndael (AES), ale rychlejší v případě klíče o velikosti 256 bitů.

Základními stavebními kameny této šifry jsou Feistelova síť, S-boxy, MDS matice (zajišťují bezpečné transformace klíčů), pseudo-Hardamanovy transformace (jednoduché mixovací operace převádějící dva vstupy na 32-bitový výstup). [39]

Příklad inicializace algoritmu:

```
key = new SecretKeySpec(keyBytes, "Twofish");
cipher = Cipher.getInstance("Twofish", "BC");
```

4.3 CRC kontrolní součet

Cyklický redundantní součet, označovaný také CRC (zkratka anglického Cyclic redundancy check) je speciální hašovací funkce, používaná k detekci chyb během přenosu či ukládání dat. Pro svou jednoduchost a dobré matematické vlastnosti jde o velmi rozšířený způsob realizace kontrolního součtu. Kontrolní součet bývá odesílán či ukládán společně s daty, při jejichž přenosu nebo uchovávání by mohlo dojít k chybě. Po převzetí dat je znovu nezávisle spočítán. Pokud je nezávisle spočítaný kontrolní součet odlišný od přeneseného nebo uloženého, je zřejmé, že při přenosu nebo uchovávání došlo k chybě. Pokud je shodný, tak téměř jistě k žádné chybě nedošlo. [41]

CRC součet aplikujeme v algoritmu na ukrývanou zprávu (tedy ignorujeme kompresi i kryptování). V případě získávání ukryté zprávy se nejdříve zpráva zrekonstruuje, následně se provede případné dekryptování a dekomprese a CRC součet této výsledné zprávy se porovnává se součtem uloženým v hlavičce, viz obrázek 9. CRC součet tedy odhalí, zda-li se získaná (dekódovaná) zpráva shoduje s původní (ukrytou) zprávou.

Implementace 32 bitové verze CRC:

```
public static int getCRC32 (byte[] data) {
     // vytvori novy CRC-calculating object
     final CRC32 crc = new CRC32();
     crc.update(data);
     // vrati hodnotu CRC v int.
     return (int) crc.getValue();
}
```

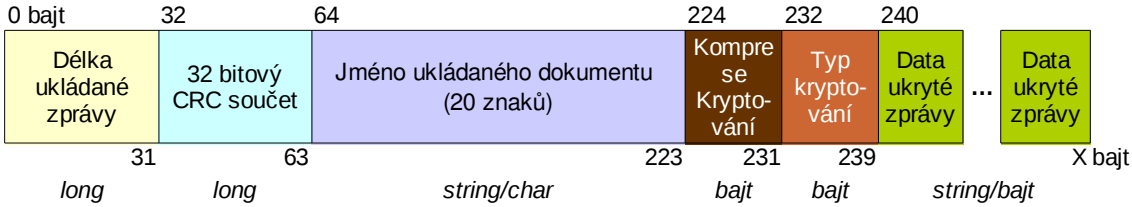
Pro výpočet CRC využívá Java standardní polynom CRC-32-IEEE 802.3:

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1 \quad (2)$$

Popis konkrétního případu výpočtu CRC součtu lze najít např. na webových stránkách Root [16].

4.4 Ukrytí zprávy

Nestačí zprávu do bitmapy nebo audio souboru prostě ukrýt, potřebujeme si kromě zprávy samotné poznačit nejrůznější informace, abychom zprávu mohli později opět z bitmapy (nebo zvukového souboru) lehce získat. Potřebujeme znát délku zprávy, informaci, zda-li je zpráva komprimována či kryptována apod. Z ilustrace níže je zřejmé, že prvních 240 pozic je alokovaných pro „hlavičku“.



Obrázek 9: Struktura hlavičky ukrývané zprávy a samotných ukrývaných dat

4.4.1 Struktura hlavičky ukrývané zprávy

V případě technologie LSB (využití nejméně významného bitu) dokážeme do každého bajtu ukrýt 1 bit. To znamená, že samotná hlavička nám zabere 240 * 1 bajt = 240 bajtů prostoru bitmapy, resp. audio souboru.

Data ukrývané zprávy jsou uložena bezprostředně za hlavičkou, tedy od bajtu 240 dále až po X-tý bajt, který se určí jako 240 (velikost hlavičky) + délka ukládané zprávy.

Délka ukládané zprávy

- délka zprávy je uložená v proměnné typu int (4 bajty) což je 4 * 8 bitů = 32 bitů
- v hlavičce je informace uložena na pozici 0 až 31

CRC součet

- 32 bitový kontrolní součet
- v hlavičce je uložen na pozici 32 až 63
- obsahuje kontrolní součet ukrývané zprávy (bez komprese a kryptování)

Jméno ukládaného dokumentu

- Je možné uložit název dokumentu délky až 20 znaků. Delší jména dokumentů jsou automaticky oříznutá (uživatel dostane informativní hlášení)
- v hlavičce je uložen na pozici 64 až 223

- v případě ukrývání prostého typovaného textu je místo jména dokumentu uložen pomocný znak „?“

Komprese a kryptování

- příznak použití komprese a kryptování je realizován následujícími hodnotami:

  0 … komprese NE, kryptování NE **2** … komprese ANO, kryptování NE

  1 … komprese NE, kryptování ANO **3** … komprese ANO, kryptování ANO
- pro uložení příznaku potřebujeme 1 bajt (8 bitů)
- v hlavičce je uložen na pozici 224 až 231

Typ kryptování

- příznak použitého typu kryptování je dán následujícími hodnotami:
  - 1 … AES
  - 2 … Blowfish
  - 3 … CAST
  - 4 … DES
  - 5 … IDEA
  - 6 … RC4
  - 7 … Skipjack
  - 8 … Triple DES
  - 9 … Twofish
  - 0 … není použito kryptování (defaultní hodnota)

4.4.2 Prostor pro ukrytí vlastních dat zprávy

V předchozí kapitole je popsáno, že pomocná hlavička zprávy zabírá 240 bajtů prostoru obrázku, resp. audio souboru. Tzn. že dokážeme ukrýt dokument o následující velikosti:

$$\frac{Velikost\ obrázku\ v\ bajtech - 240\ bajtů\ hlavičky}{8} = Volný\ prostor\ v\ bajtech \qquad (3)$$

5 UKRYTÍ ZPRÁVY DO BITMAPY

V následující kapitole si popíšeme, jak je v programu implementována technologie ukrývání zprávy do bitmapy s praktickou ukázkou zakódování (ukrytí) a dekódování (nalezení) zprávy.

5.1 Implementace technologie LSB

Implementace technologie LSB pro ukrytí dat do bitmapy využívá následujících objektů a operací:

- zprávu reprezentují jednotlivé bajty
- bitové operace: logický AND (&), OR(|)
- třída `BufferedImage` pro úpravy jednotlivých bitů obrázku
- třída `ImageIO` pro ukládání obrázku z paměti na disk a naopak
- třída `Graphics2D` pro práci s grafikou, např. pro získání uživatelského prostoru
- třída `WritableRaster` umožní přístup do bufferu
- třída `DataBufferByte` buffer použitý pro BufferedImage

5.1.1 Práce s bajty

Bajty jsou elementární datové prvky většiny aplikací. Každý bajt je tvořen osmi bity, které mohou nabívat hodnot 0 nebo 1. Číslo vytvořené kombinací nul a jedniček lze jednoduše transformovat z binární soustavy do dekadické a naopak.

Hodnota čísla podle pozice jedničky: `128 64 32 16 8 4 2 1`

Příklady:

- `00000000 = 0`
- `00000010 = 2`
- `00000111 = 7`
- `00001011 = 11`

`Bajt (byte)` lze vytvořit z primitivního datového typu `int` jednoduchým přetypováním:

```
Byte b = (byte)7;
```

Většina tříd v Javě má metody, které vracejí pole dat **`byte[]`**, které lze jednoduše procházet, modifikovat, kopírovat atd.

Příklad převedení String (textu) na pole bajtů:

```
String w = "William";
Byte[] b = w.getBytes();
```

První znak „W“, resp. jeho ascii hodnota 87, je uložen v `b[0]`. Ačkoliv se zdá, že je v poli uloženo číslo (`int`), je to ve skutečnosti bajt složen z 8 bitů, v našem případě `01010111`.

5.1.2 Bitové operace

Technologie LSB je založena na změně hodnoty nejméně významného bitu. Pro tuto činnost využijeme bitové operace.

Logický součin AND

AND (`&`) je binární logická operace jejíž hodnota je pravda, právě když obě vstupní hodnoty jsou pravda (1), v opačném případě nepravda (0). Pokud tedy oba bajty mají na stejné pozici (bitu) hodnotu 1, je ve výsledku na stejné pozici také 1, jinak 0.

Příklad:

```
01010111 = 87
01100101 = 101
01000101 = 69
```

```
Byte b = 87 & 101; //69: 01000101
```

Bitový posun vlevo

Princip bitového posuvu vlevo lze rozpoznat z následujícího příkladu:

```
01010111 = 87
<< 1 // posuv o 1 bit
10101110 = 174
```

Bitový posun vlevo můžeme chápat tak, že na pravou stranu binárního čísla přidáme 0 a bit úplně vlevo odstraníme. Je důležité si uvědomit, že pokud odstraňovaný bit nebyl 1, hodnota bajtu se zdvojnásobí. Pokud je zapotřebí provést bitový posuv o více bitů, operace se v cyklu opakuje. Tzn. pokud pro jakékoliv číslo provedeme bitový posuv 8 krát, dostaneme výslednou hodnotu 0.

Příklad – posuv o 2 bity:

```
01010111 = 87
<< 2
01011100 = 92
```

```
Byte b1 = 87 << 1; //174: 10101110
Byte b2 = 87 << 2; //92: 01011100
```

Bitový posun vpravo

Jedná se o inverzní postup oproti posunu vlevo ve smyslu, že 0 přidáváme vlevo a bit úplně vpravo je odstraněn.

Příklad:

(posun o 1 bit)

```
01010111 = 87
>>> 1
00101011 = 43
```

(posun o 2 bity)

```
01010111 = 87
>>> 2
00010101 = 21
```

5.1.3 Využité třídy

ImageIO

ImageIO je velice užitečná třída na vstupně výstupní operace s obrázky. Nejdůležitější metody využívané v programu jsou metody `read()` a `write()` pro načítání a ukládání obrázku.

Graphics2D

Třída Graphics2D nám zajistí přístup do dat obrázku a to pro čtení i modifikaci, podle toho, jestli potřebujeme ukrývat zprávu do obrázku nebo naopak zprávu hledáme.

WritableRaster

Třída WritableRaster definuje přístup k jednotlivým pixelům obrázku. WritableRaster je podtřída třídy Raster, která řeší konkrétní přístup k bufferu zpracovávaného obrázku.

DataBufferByte

Třída DataBufferByte nám poskytne pole jednotlivých pixelů `byte[]`, ze kterých se skládá zpracovávaný obrázek.

```
private byte[] getByteData(BufferedImage image) {
      WritableRaster raster = image.getRaster();
      DataBufferByte buffer =
                  (DataBufferByte) raster.getDataBuffer();
      return buffer.getData();
}
```

5.1.4 Popis implementovaných metod

Nebudu zde popisovat všechny metody, se kterými program pracuje, ale omezím se na ty, o kterých si myslím, že jsou nejdůležitější.

Uživatelský prostor obrázku

```
private BufferedImage getUserSpace(BufferedImage image){
      BufferedImage img  = new BufferedImage  (
            image.getWidth(),
            image.getHeight(),
            BufferedImage.TYPE_3BYTE_BGR);
      Graphics2D graphics = img.createGraphics();
      graphics.drawRenderedImage(image, null);
      graphics.dispose();
            return img;
}
```

- vytvoří nový objekt (kopii) BufferedImage stejné velikosti a barevné hloubky jako originální obrázek.
- originální obrázek je následně „překreslen“ do kopie
- z důvodu paměťových nároků je následně objekt `graphics` uvolněn
- nyní máme připraven nový BufferedImage, do kterého můžeme začít ukrývat požadované informace

Převod čísla (int) na bajt a zpět

Pro uložení hodnoty `int` potřebujeme 4 bajty. Převod provedeme jednoduše logickým součtem vždy pro příslušný bajt a následným bitovým posunem.

```
private byte[] bitConversion(int i) {
     byte byte3 = (byte)((i & 0xFF000000) >>> 24);
     byte byte2 = (byte)((i & 0x00FF0000) >>> 16);
     byte byte1 = (byte)((i & 0x0000FF00) >>> 8 );
     byte byte0 = (byte)((i & 0x000000FF)       );
     return(new byte[]{byte3,byte2,byte1,byte0});
}
```

Převod z pole bitů zpět na číslo je přesně inverzní, postupně procházíme jednotlivé bajty a skládáme z nich výsledný `int`. Pohledem na následující algoritmus je postup zřejmý:

```
private int getIntFromArray(byte[] b) {
    int i = 0;
    i |= b[0] & 0xFF;
    i <<= 8;
    i |= b[1] & 0xFF;
    i <<= 8;
    i |= b[2] & 0xFF;
    i <<= 8;
    i |= b[3] & 0xFF;
    return i;
}
```

Ukrytí informace do obrázku

Následující algoritmus implementuje techniku ukrytí zprávy `addition` (pole bajtů známe délky) do obrázku `image` reprezentovaného polem jednotlivých pixelů. Zprávu začneme ukládat od místa `offset`. Důvod použití offsetu (posunutí) bude vysvětlen dále v textu.

```
private byte[] encodeText (byte[] image,
                    byte[] addition, int offset) {
     for(int i=0; i<addition.length; ++i) {
          int add = addition[i];
          for(int bit=7; bit>=0; --bit, ++offset) {
             int b = (add >>> bit) & 1;
             image[offset] = (byte)((image[offset] & 0xFE) | b );
          }
     }
}
```

Jednotlivé bity bajtu si můžeme ohodnotit: bit nejvíce vlevo je nejvíce významný bit, naopak bit nejvíce vpravo je nejméně významný. Tato informace nám dává klíč, jak změnit obrázek tak, aby změna byla co nejméně nápadná. Provedeme to tak, že využijeme právě nejméně významného bitu a uložíme místo něj naši ukrývanou zprávu.

`for(int i=0; i<addition.length; ++i)` – projdeme postupně každý bajt zprávy, kterou hodláme do bitmapy ukrýt

`int add = addition[i];` – aktuální bajt přiřadíme do add

`for(int bit=7; bit>=0; --bit, ++offset)` – postupně projde všech 8 bitů bajtu uloženého v add

`int b = (add >>> bit) & 1;` – do `b` je přiřazena hodnota bajtu posunutého doprava a na výsledku je dále proveden logický součin AND 1

Pro lepší představu můžeme algoritmus nasimulovat na následujícím příkladě:

add = 87 = 01010111

První smyčka v cyklu, bit = 7:

```
01010111 = 87
>>> 7
00000000 = 0
```

další, bit = 5:

```
01010111 = 87
>>> 5
00000010 = 2
```

Další smyčka, bit = 6:

```
01010111 = 87
>>> 6
00000001 = 1
```

další, bit = 4:

```
01010111 = 87
>>> 4
00000101 = 5
```

Nyní provedeme logický součin & 1:

bitový posun vpravo o **7 bitů**:

```
00000000 = 0
00000001 = 1
00000000 = 0 = b
```

Další smyčka, bitový posun o **5 bitů**:

```
00000010 = 2
00000001 = 1
00000000 = 0 = b
```

Další smyčka, bitový posun o **6 bitů**:

```
00000001 = 1
00000001 = 1
00000001 = 1 = b
```

Další smyčka, bitový posun o **4 bity**:

```
00000101 = 5
00000001 = 1
00000001 = 1 = b
```

Poznámka k příkladu, `b` je nastaveno na hodnotu 0 nebo 1. Provedením bitové operace **AND 1** nastavíme všechny bity na 0 s výjimkou posledního, který zůstane, jaký byl.

```
image[offset] = (byte)((image[offset] & 0xFE) | b );
```

Tento zápis pracuje na podobném principu. `0xFE` je hexadecimální zápis, který reprezentuje 11111110 v binární soustavě. Dle zdůvodněného výše, tento postup nechá prvních 7 bitů tak jak je a poslední bit nastaví na 0. Následně na tento poslední bit „0“ aplikujeme logický součet OR s hodnotou `b`, která je buď 00000000 nebo 00000001. Tímto způsobem nastavíme poslední bit na hodnotu uloženou v `b`.

Dekódování ukryté informace v obrázku

Dekódování informace z obrázku se provádí inverzním postupem oproti kódování. Základem metody, která vrací hledanou informaci prostřednictvím **`byte[]`** je pole **`imageByte`**, které nese informace o jednotlivých pixelech obrázku. Dále potřebujeme vědět délku hledané informace, tedy **`length`** a posunutí **`offset`**, od jakého místa v poli **`imageByte`** bude informace vyhledávána. Pak se již pouze pole projde a z jednotlivých bitů se složí hledaná informace.

```
private byte[] decodeText(int length, int offset, byte[] imageByte) {
            byte[] result = new byte[length];

            // Prochazi každý byte
            for (int b = 0; b < result.length; ++b) {
                  // postupne prochazi jednotlive bity byte
                  for (int i = 0; i < 8; ++i, ++offset) {
                        result[b] = (byte) ((result[b] << 1) |
                              (imageByte[offset] & 1));
                  }
            }
            return result;
}
```

`for(int b=0; b<result.length; ++b)`

Dopředu jsme si nachystali pole `result` o velikoti hledané informace. Pole se v cyklu bude postupně procházet a plnit.

`for(int i=0; i<8; ++i, ++offset)`

Musíme projít v cyklu a postupně po jednotlivých bitech posbírat celý bajt.

`result[b]=(byte)((result[b] << 1)|(imageByte[offset] & 1));`

Výsledné bajty jsou uloženy v poli `result`. Každý jednotlivý bajt hledané informace (zprávy) je složen z 8 bitů, které jsou jsou uloženy v nejméně významných bitech 8 bajtů obrázku.

Nalezení velikosti ukryté zprávy

Abychom zprávu mohli vyhledat a správně složit, je zapotřebí znát její velikost. Vstupní parametr `imageByte` je pole pixelů, ze kterých se obrázek skládá.

```
private int getHidenMessageSize(byte[] imageByte) {
            int length = 0;
            // V prvnich 32 bajtech je velikost ukryte zpravy
            for (int i = 0; i < 32; ++i) {
                  length = (length << 1) | (imageByte[i] & 1);
            }
            return length;
}
```

```
for(int i=0; i<32; ++i)
```

Jak je popsáno dříve, velikost zprávy je uložena v proměnné typu int, která má velikost 4 bajty, tedy 32 bitů. Jelikož algoritmus počítá s ukrytím 1 bitu do každého jediného bajtu, musíme projít prvních 32 bajtů obrázku.

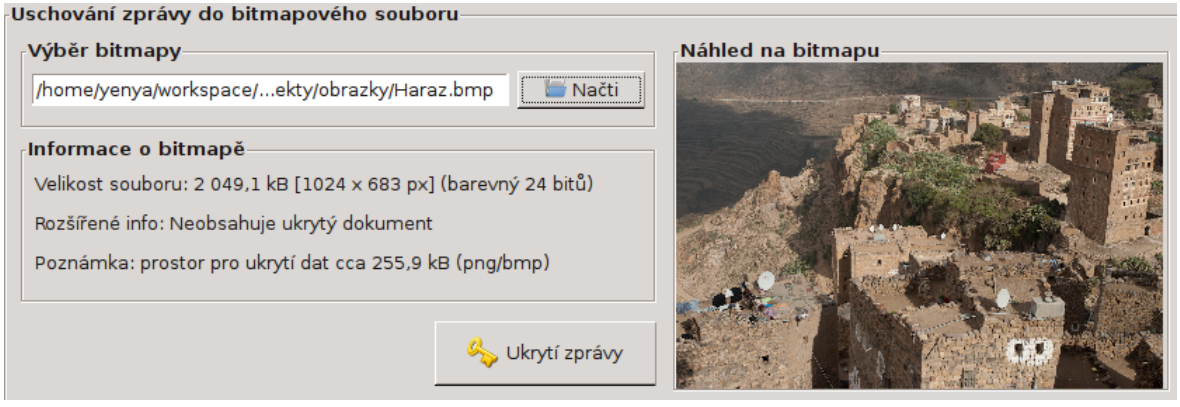
```
length = (length << 1) | (image[i] & 1);
```

V cyklu se postupně provede bitový posun o `1` a následně i bitová operace `OR` s nejméně významným bitem daného bajtu (operace `& 1`) – všechny bity zpracovávaného bajtu se vynulují s výjimkou posledního, který zůstane jak je. Postupně jsou tedy tyto bity „sesbírány“ a vloženy do atributu `length`.

5.2 Ukázka ukrytí zprávy

Popišme si kroky, které musí uživatel provést v případě požadavku ukrytí zprávy do bitmapy:

- výběr bitmapy
- definice ukrývané zprávy
- nastavení komprese a kryptování
- proces ukrytí zprávy



Obrázek 10: Ukázka GUI v případě práce s ukrýváním zprávy do obrázku

5.2.1 Výběr bitmapy

Algoritmus ukrývání dat byl popsán v kapitole 5.1 včetně náhledů na zdrojové kódy, které řeší důležité algoritmy. Víme tedy, že obrázek je uložen v paměti v poli pixelů, které se modifikují. Pro obrázky typu PNG a BMP máme k dispozici plnou podporu, tedy obrázek můžeme načíst, ukrýt do něj zprávu a následně do stejného formátu i uložit.

Můžeme však částečně pracovat i s obrázky typu JPG a GIF. Nicméně algoritmus ukrytí zprávy je převede na PNG/BMP dle výběru uživatele.

Po načtení bitmapy se vykreslí její náhled a zobrazí se praktické informace o bitmapě: rozlišení, velikost, barevná hloubka a další, viz obrázek výše.

Pokud bitmapa již obsahuje ukrytou zprávu, je o tom uživatel informován. Pokud ne, zobrazí se tlačítko pro ukrytí zprávy.

5.2.2 Ukrývaná zpráva

Ukrývaná zpráva může být dvou typů: prostý typovaný text a nebo textový či binární dokument.

Prostý typovaný text

V případě výběru ukrývání prostě typované zprávy je možné tuto zapsat do editačního pole ve spodní části obrazovky. Pro potřeby ukrývání se textová zpráva převede do pole bajtů. V kapitole 5.1.1 jsme si ukázali, jak textovou zprávu snadno do pole bajtů převést.

Textový binární dokument

Kromě typované textové zprávy máme možnost ukrýt i binární dokument, např. Word, Excel, Open Office, PDF apod. Binární soubor je také převeden na bajty, které budou ukládány (kódovány do bitmapy).

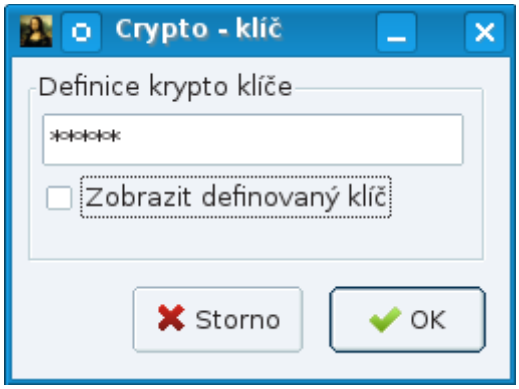
5.2.3 Proces ukrytí zprávy do bitmapy

Před samotným ukrytím zprávy si uživatel nastaví, zda-li má být kryptována a komprimována. Na to slouží globální nastavení aplikace, viz *Obrázek 1: Nastavení komprese a kryptování*.

Vytvoření a ukrytí hlavičky zprávy

- délka zprávy pozice 0 až 31
- crc kontrolní součet 32 až 63
- název ukládaného dokumentu
- příznak použití komprese a kryptování
- příznak typu kryptování

Ukrytí vlastní zprávy



Obrázek 11: Zadání krypto klíče

- Zpráva se ukrývá do bitmapy od pozice 240.
- Provede se komprese a kryptování zprávy v případě, že toto bylo uživatelem nastaveno.
- V případě kryptování zprávy se zobrazí uživateli obrazovka, kde se zadává kryptovací klíč. Z hlediska bezpečnosti si uživatel může deaktivovat průběžné zobrazování typovaní klíče a budou se mu zobrazovat jen hvězdičky. Pokud uživatel zadá heslo, které je v rozporu s vybraným kryptografickým algoritmem, bude o tomto informován a bude mu zobrazena dodatečná informace, jaké jsou podmínky na zadání klíče.
- Pokud ukrývání proběhlo bez komplikací, zobrazí se uživateli informační hlášení, že zpráva je ukryta, v opačném případě se zobrazí chybové hlášení.

5.3 Dekódování ukryté zprávy

Pokud bychom nepotřebovali text z bitmapy znovu dekódovat, obešli bychom se bez hlavičky, ale k čemu by takový program byl …

To, jestli bitmapa obsahuje ukrytou zprávu, zjistíme z hlavičky (struktura hlavičky viz obrázek 9) a to z následujících indícií:

- v hlavičce je zakódovaná délka ukládané zprávy. Pokud délku nelze zpětným algoritmem dekódovat, žádnou zprávu obrázek neobsahuje,
- v hlavičce je zakódovaná informace o případně použité kompresi a kryptování. Tato informace nabývá hodnoty 0 až 3, pokud je uložena jiná informace, žádnou zprávu obrázek neobsahuje,
- pro další kontrolu existence ukryté zprávy můžeme využít i příznak typu kryptování. Tato informace nabývá hodnoty 0 až 9, jiná hodnota znamená, že žádnou zprávu obrázek neobsahuje.

Proč je provedena tak důkladná kontrola existence zprávy v obrázku? Za jisté konstelace, za jisté kombinace barev by mohlo nastat, že algoritmus vrátí smysluplnou

délku zprávy, ale ve skutečnosti bitmapa zprávu obsahovat nemusí. Proto se kontrolují i další příznaky (uložené informace v hlavičce), aby se existence zprávy potvrdila.

Pokud tedy algoritmus rozhodne, že bitmapa zprávu obsahuje, oznámí, zda-li se jedná o ručně typovaný text nebo ukrytý dokument (zobrazí jeho název), zda-li je zpráva komprimována nebo kryptována (včetně způsobu kryptování). Navíc se objeví tlačítko, které zavolá algoritmus na dekódování zprávy.

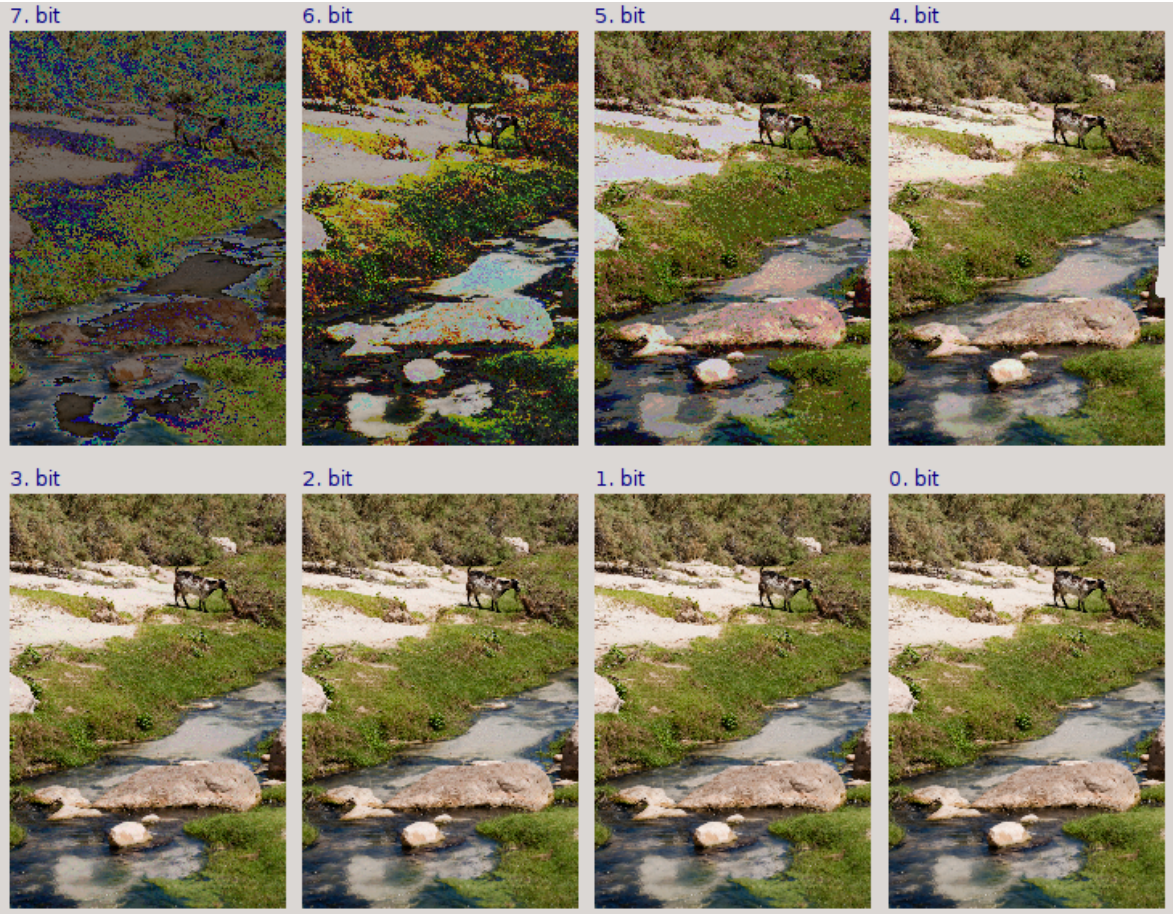
Každá ukrytá zpráva si nese v hlavičce kontrolní součet CRC původní ukrývané zprávy (bez provedené komprese a kryptování). Pokud se tento neshoduje s CRC právě dekódované zprávy, je zpráva poškozena, nebo v případě kryptované zprávy bylo zadáno špatné heslo.

5.4 Simulace ukrytí dat

Abychom si udělali představu o technologii LSB, můžeme si nasimulovat ukrytí zprávy v maximální možné délce a podívat se, jak zpráva poškodí obrázek. Máme k dispozici dva typy simulace: změna jediného bitu v každém bajtu anebo libovolný počet bitů pro ukrytí zprávy v každém bajtu. Změnu lze na obrazovce ihned zobrazit, případně ji uložit do adresáře analýza pro další zkoumání. Simulace neřeší hlavičku zprávy, ta se při simulaci nevytváří.

5.4.1 Využití jediného bitu v bajtu

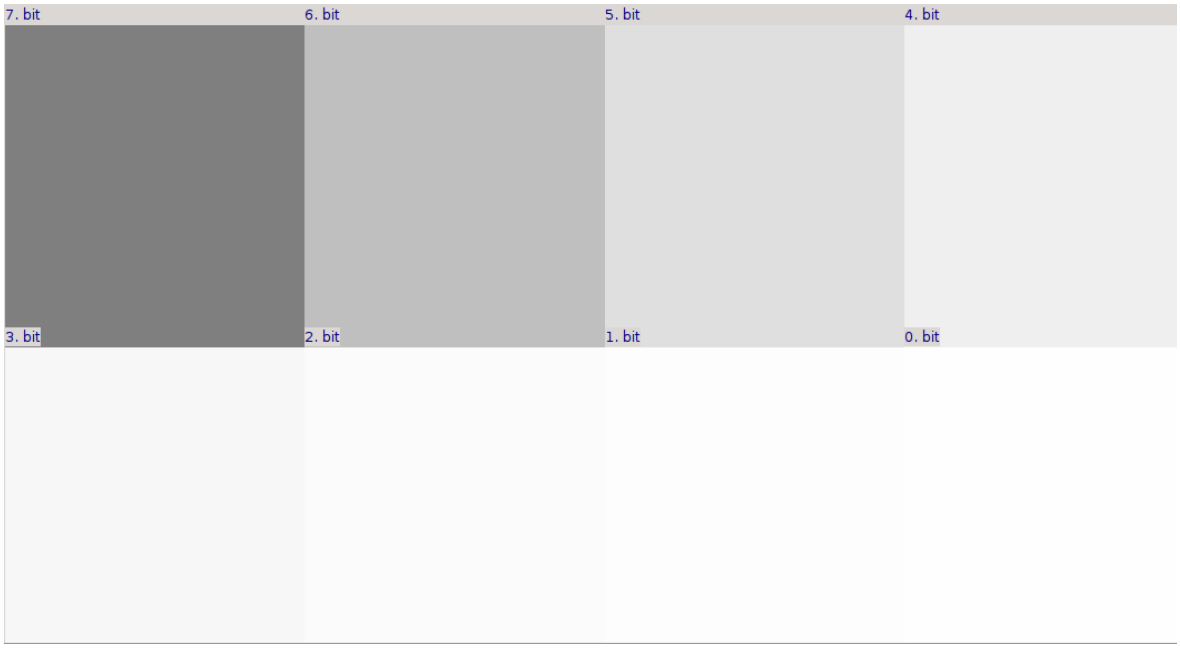
Tato simulace znázorní, jak ukrytá zpráva poškodí obrázek, pokud na zprávu využijeme jediný bit z každého bajtu. Pro názornost zprávu uložíme na všechny pozice (0 až 7) a v náhledu vidíme, co zpráva provedla s kvalitou obrázku.



Obrázek 12: Ukázka simulace ukrytí zprávy do jednoho bitu bajtu

Z ukázky je patrné, že nejvhodnější je ukrýt zprávu do nejméně významného bitu bajtu „0“. Uvědomme si, že 24 bitová bitmapa se skládá z 256 složek červené (R), zelené (G) a modré (B) barvy, jejichž kombinacemi získáme celkem 16 777 216 barev. Drobnou změnu barvy (kterou provedeme právě nastavením bitu „0“) nedokáže lidské oko rozeznat a změna je poznat až v detailním zkoumání bitmapy v hexa náhledu. Pokud však útočník nezná původní strukturu bitmapy a dostane se mu do rukou jen pozměněná bitmapa se zprávou, je vyhledání ukryté zprávy velice náročné. V případě použití komprese a kryptování bude ukrytý text jen nepostřehnutelný náhodný šum v obraze.

Zajímavá ukázka je i u bitmapy, která se skládá z jediné barvy, např. bílé, která je u 24 bitové verze definována kombinací FF FF FF (tedy samé jedničky). Lze pozorovat, že čím vyšší bit z bajtu měníme, tím je výsledná barva tmavší (více šedá):



Obrázek 13: Modifikace 1 bitu u bitmapy bílé barvy

V následující tabulce máme zobrazenu hodnotu původní bílé barvy obrázku a popis, jak změnou bitu postupně přecházíme v šedou barvu:

Tabulka 7: Změna bílé barvy bitmapy po modifikaci bitu

Informace	Zápis barvy	
	Binární	Hexa
Původní barva	11111111	FF
Změna 0. bitu	11111110	FE
Změna 1. bitu	11111101	FD
Změna 2. bitu	11111011	FB
Změna 3. bitu	11110111	F7
Změna 4. bitu	11101111	EF
Změna 5. bitu	11011111	DF
Změna 6. bitu	10111111	BF
Změna 7. bitu	01111111	7F

5.4.2 Využití libovolných bitů v bajtu

Druhá simulace nabízí možnost „naklikat“ si libovolnou kombinaci bitů, které místo dat obrázku ponesou data zprávy. U kvalitního 24 bitového obrázku lze bezpečně ukrýt zprávu do nejnižších třech bitů *(0,1,2)*, aniž by toto lidské oko postřehlo. U 8 bitových obrázků je třeba být opatrnější a využít třeba jen jediný bit.

Simulaci si předvedeme na barevném 24 bitovém obrázku s rozlišením 1000x600 obrazových bodů.

Obrázek 14: Bitmapa (originál) 1000x600 bodů bez ukryté zprávy

Pokud bychom se rozhodli ukrýt zprávu do jediného bitu, máme k dispozici prostor pro celkem 225 000 znaků (1000 * 600 * 3 / 8). Ukážeme si ukrytí do 3 nejnižších bitů:

Obrázek 15: Bitmapa s ukrytou zprávou délky 675 000 znaků (využití 3 bitů)

Obrázky 14 a 15 jsou od sebe pouhým okem těžko rozeznatelné, přesto je v nich již značný rozdíl. Pokud bychom využili další bit, již uvidíme výraznou nepřesnost ve stínech a využití dalšího bitu již vede k velké destrukci obrazu:

Obrázek 16: Bitmapa s ukrytou zprávou délky 1 125 000 znaků (využití 5 bitů)

Čím více bitů původního obrázku využijeme na přenos zprávy, tím je obrázek nekvalitnější, což může být pro útočníka důvod pro jeho analýzu a případné odhalení zprávy. Implementovaný algoritmus LBA popsaný v kapitole 5.1 využívá jen nejméně významného bitu „0“, což nabízí prostor pro ukrytí zprávy velikosti 1/8 kapacity obrázku a přitom vliv na kvalitu obrazu je zanedbatelný.

Je také zajímavé si uvědomit, že obraz se skládá z barev, které si můžeme představit jakousi kombinací „0“ a „1“, viz tabulka 7. Ukrývaná zpráva je také reprezentovaná kombinací „0“ a „1“. Předpokládejme modifikaci nejméně významného bitu (LSB). Pokud tento bit (reprezentující hodnotu barvy, resp. pouze její část) nese hodnotu „0“ a my právě ukládáme část zprávy, jejíž bit je také „0“, nedojde na tomto místě ke změně barvy obrázku, ale přesto je zde uložena jedna z částí zprávy. Stejně tak to platí pro „1“. V ostatních případech, kdy se LSB liší od právě zpracovávaného bitu zprávy, dojde k nahrazení LSB bitem zprávy.

6 UKRYTÍ ZPRÁVY DO AUDIO SOUBORU

V teoretické části diplomové práce jsme se věnovali rovněž možnostem ukrývání zprávy do zvukových formátů. V kapitole 2.4 jsme si ukázali představu polských inženýrů, jak by se dala zpráva ukrýt v rámci VoIP hovoru. V kapitole 2.6 jsme řešili využití rezervních bitů v hlavičkách rámců MP3 souborů. Pro praktickou ukázku ukrytí zprávy do audio souboru jsem si zvolil ukrytí do formátu WAV, popsaného v kapitole 2.1.4 a 2.1.5. Tento formát jsem vybral z toho důvodu, že můžeme použít obdobné technologie jako v případě ukrývání zprávy do bitmapy. Důkladně bude technologie rozebrána v kapitole 6.2.

Zvuk a jeho vlastnosti

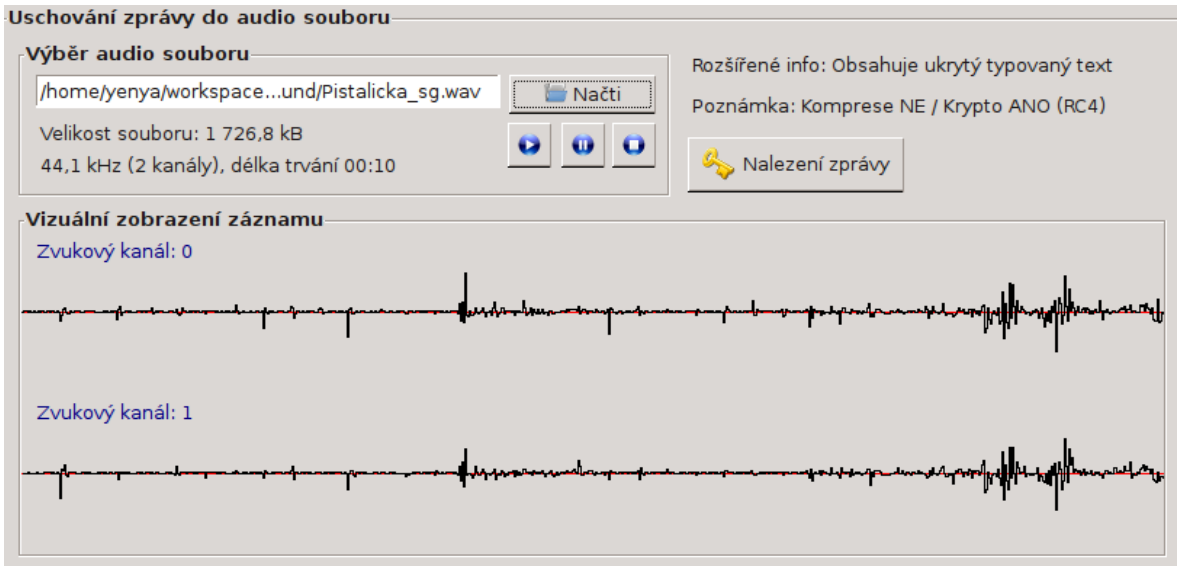
Zvuk je každé podélné mechanické vlnění v prostředí, které je schopno vyvolat sluchový vjem. Lidské ucho je schopno vnímat zvuky o frekvenci 16 Hz až 20 kHz. Nejcitlivější je v oblasti 1000 – 3000 Hz, což je oblast frekvence lidského hlasu. Zvuky pod 16 Hz se nazývá infrazvuk a nad 20 kHz se nazývá ultrazvuk. V závislost na vnímání zvuku lidským uchem, můžeme u jednotlivých zvuků rozlišovat tři základní vlastnosti:

- hlasitost zvuku – velikost amplitud jednotlivých vln, udávána v decibelech (dB, logaritmická stupnice)
- výška zvuku – je dána jeho frekvencí, čím vyšší je frekvence, tím je vyšší výška.
- barva zvuku – tvar vln (sinusoidy, čtvercové, trojúhelníkové, atp.). Zvuky se i při stejné výšce tónu mohou lišit odlišným zabarvením. Barva zvuku je určena počtem vyšších harmonických tónů ve složeném tónu a jejich amplitudami. Sluchem podle barvy zvuku rozeznáváme hudební nástroje a hlasy lidí.

Záznam zvukové stopy se nejčastěji provádí se vzorkovací frekvencí 44100 Hz. Pro popis dat se používá nejčastěji 16bitový integer. Takto zaznamenaný signál může reprodukovat signál, který obsahuje frekvence až do 22050 Hz. [44]

6.1 Grafické rozhraní

Kromě samotného algoritmu ukrývání zprávy do zvukového souboru, byly implementovány základní funkcionality „zvukového editoru“.



Obrázek 17: Uživatelské rozhraní při ukrývání zprávy do audio souboru

Po načtení WAV souboru se uživateli zobrazí ucelené informace o nahrávce (počet zvukových stop, frekvence, délka) a bude znázorněn grafický průběh skladby, a pomocí ovládacích prvků může nahrávku přehrát.

O vykreslení grafického průběhu audio záznamu se stará třída `SingleWaveform`, resp. její metoda `paint`.

```
// nachysta sample na vykresleni
samples = helper.getAudio(channelIndex);
for (; t < samples.length; t += increment) {
     // urceni meritka zobrazeni
     double scaleFactor = helper.getYScaleFactor(height);
     double scaledSample = samples[t] * scaleFactor;
     int y = (int) ((height / 2) - (scaledSample));
     // vykresli aktualni sample
     gc.drawLine(oldX, oldY, xIndex, y);
     xIndex++;
     oldX = xIndex;
     oldY = y;
}
```

V případě, že audio soubor obsahuje ukrytou zprávu, bude o tom informováno v rozšiřujících informacích, jak je patrno z obrázku 17. V poznámce bude zobrazeno, zda-li je ukrytá zpráva komprimována, případně kryptována.

6.2 Implementace technologie LBE

Ukrývání zprávy do zvukového formátu WAV je velice podobné ukrytí zprávy do pixelů bitmapy, viz 2.1.4. Stejně jako u bitmapy, kde se přeskakuje hlavička, která se nemění (velikost bitmapy, rozlišení, počet bitů na pixel a pod.), se i zde přeskočí hlavička WAV a upravují se jen vlastní data (samples) a to technologií nejméně významného bitu. V tabulce 5 je detailně popsána hlavička WAV souboru.

Nemá smysl zde popisovat implementaci ukládání informace do WAV souboru, neboť byla použita obdobná technologie jako u LSB, založena na bitových posuvech a na ukládání zprávy na nejméně významný bit. Zachována zůstala i práce s hlavičkou, viz 4.4.1. Zdrojové kódy jsou součástí DP a jsou dostatečně komentovány.

Pro přístup k datům (konkrétním samplům) se stará třída `RandomAccessFile`. Tato třída umožňuje manipulaci se souborem, která neprobíhá sekvenčně jako v případě použití proudů. Objekt třídy vytvoříme ze jména souboru. Jako druhý argument uvedeme způsob otevření souboru - buď hodnotu 'r' (pokud hledáme ukrytou zprávu a stačí nám přístup pro čtení) nebo hodnotu “rw” pro čtení i zápis (pokud ukrýváme zprávu):

```
wavFile = new RandomAccessFile(fileName, mode);
```

RandomAccesFile tedy dokáže číst/zapisovat do kterékoliv části souboru. Pro nastavení konkrétní pozice čtení/zápisu používáme metodu `seek`, pro přeskočení určitého počtu bajtu využíváme metodu `skipBytes`.

```
public void seek(long pozice)
public int skipBytes(int n)
```

Metoda `getLSB` najde na právě nastavené pozici ve WAV souboru nejméně významný bit. Je tedy zapotřebí dopředu pomocí `seek` nastavit potřebnou pozici.

```
// Vrati nejmene vyznamny bit v samplu
public byte getLSB() throws IOException {
     byte LSByte;
     LSByte = (byte) this.wavFile.read();
     this.seek(this.samplePos);
     return (byte) (LSByte & (byte) 0x01);
}
```

Metoda `setLSB` uloží na právě nastavené pozici ve WAV souboru do nejméně významného bitu „0“ nebo „1“. Zápis do nejméně významného bitu provedeme po mocí logického součtu nebo součinu, podle tohom zda-li ukládáme „1“ nebo „0“.

```
//Nastavi nejmene vyznamny bit na konkretni pozici (konkretni
sample) na true/false (resp. 0/1)
public void setLSB(boolean b) throws IOException {
     byte LSByte;

     // Precte si aktualni hodnotu bajtu
     LSByte = (byte) this.wavFile.read();
     // Zamaskuje ho 0 resp 1
     if (b == true)
          LSByte |= (byte) 0x01;
     else
          LSByte &= (byte) 0xFE;

     // Nastavi novou pozici v samplu a zapise upravenou hodnotu
     this.seek(this.samplePos);
     this.wavFile.write(LSByte);
     this.seek(this.samplePos);
}
```

Při práci s bitmapou jsme pracovali tak, že jsme si do paměti načetli obrazová data (barvy jednotlivých pixelů) a tyto jsme postupně modifikovali a ukrývali do nich zprávu. Na závěr jsme z těchto dat znovu vygenerovali bitmapu a to včetně správné hlavičky.

S WAVem je to složitější, neboť Java nemá žádný konkrétní objekt, který by pracoval přímo se samply, které by se daly modifikovat a na závěr z nich vytvořit znovu zvukový soubor s hlavičkou. Tento nedostatek lze obejít použitím třídy `RandomAccessFile`, která dokáže číst a modifikovat data v libovolné části souboru. Před ukrýváním zprávy do audio souboru si nejdříve tento zkopírujeme do nového souboru a pak postupně modifikujeme jednotlivé zvukové samply (nejméně významné bity) podle toho, jakou zprávu ukrýváme, přičemž hlavička se přeskakuje.

7 DALŠÍ APLIKACE VĚNUJÍCÍ SE UKRÝVÁNÍ ZPRÁV

Před vlastní implementací programu `Steganography` jsem vyzkoušel řadu programů věnujících se obdobné tématice. Vybral jsem si z nich ty lepší vlastnosti a přenesl je do svého programu.

OutGuess 0.2

Je univerzální steganografický nástroj, který umožňuje vkládání skrytých informací do redundantních bitů vstupních dat. Druh vstupních dat není pro OutGuess [18], [11] vůbec důležitý, jelikož program využívá specifické ovladače pro dané grafické formáty, extrahující redundantní bity a následně je po úpravách zapíší zpět. Podporovány jsou formáty JPEG a PNG.

OutGuess u JPEG obrazů odolává statistickým výpočtům založených na frekvenční analýze. Výsledky statistických testů nejsou schopny odhalit přítomnost steganografického obsahu, protože OutGuess před samotným vkládáním dat do obrazu zjišťuje maximální velikost zprávy, která může být skryta a bude stále možné statisticky nezměnit výsledný obraz z pohledu frekvenční analýzy.

OutGuess je dostupný pod BSD licencí, je možné ho používat zdarma a je nadplatformový.

Hide In Picture (HIP)

Hide In Picture [7] je jednoduchý program, který demonstruje techniku LSB. Nepodporuje ukrývání prostého textu, je třeba ukrývat dokument, který se převádí na bajty. Ukrývaný dokument lze podpořit kryptováním Blowfish nebo AES (Rijndael). Grafické prostředí je spíše podprůměrné, nicméně program je funkční.

HIP je přeložen do češtiny a je distribuován pod licenci GNU General Public License.

Camouflage 1.2.1

Aplikace Camouflage [10] pracuje tak, že dokáže zprávu ukrýt k jakémukoliv souboru (obrázek, dokument, exe, ...). Soubor s „přilepenou“ zprávou vypadá a chová se stejně jako „normální“ soubor a může být odeslán příjemci bez případného podezření, že obsahuje ukrytou zprávu.

Provedl jsem test na malé bitmapě 10x10 bodů, která zabírá 374 bajtů. Tajná zpráva „znak A“ se přilepí k obrázku, ale ten následně zabírá už 1230 bajtů. Mnohem efektivnější je kamuflování do souborů, u kterých se nedá spočítat, kolik bajtů ve skutečnosti zabírají. U bitmapy lze velikost snadno zjistit a skrytou zprávu tak odhalit. Dovedu si ale představit využití kamufláže na wordových dokumentech, na PDF dokumentech a jiných podobných mediích …

Pro větší bezpečnost lze ukrytou zprávu podpořit heslem. Jako nevýhodu vidím nutnost instalace programu do počítače, program není samostatně spustitelný a je pevně svázán s kontextovým menu OS Windows. Program není multiplatformní.

SGCrypt 2.0

SGCrypt je velice jednoduchý program, který pomocí LSB techniky dokáže do jakéhokoliv 24-bitového obrázku bmp zakódovat libovolné množství souborů, záleží pouze na velikosti obrázku a volných dat, viz popis LSB v kapitole 2.1.1.

Program je typu freeware a vytvořil ho UFO XTreem development [13].

MP3 Stego

MP3Stego [25] dokáže skrýt informaci do MP3 souboru a to během fáze komprese. Data jsou nejdříve zkomprimována, zakódována a pak skryta do proudu dat MP3 (bit stream). Ačkoliv byl MP3Stego psán jako steganografická aplikace, lze použít i pro watermarking (copyright marking) pro MP3 soubory – slabý, ale přeci jen lepší než současný stav označení autorství v hlavičce „frame“ souboru. Ale i tady existuje způsob, jak informaci poškodit – a to dekompresí mp3 souboru a jeho novou kompresí, tím ukrytá zpráva definitivně zmizí.

Revelation

Revelation [15] je další příklad LSB techniky ukrývání dat do bitmapy typu BMP. Zpráva je zkomprimována a následně kryptována pomocí AES 256 bit. Nelze zvolit vlastní klíč, nelze zvolit typ kryptování, jinak je program funkční.

Další zajímavý software o steganografii

Na stránkách StegoArchive [8] je ucelený seznam programů věnujících se steganografii.

ZÁVĚR

V úvodu diplomové práce jsem představil možnosti ukrytí zprávy tak, aby tato byla bezpečně doručena příjemci, aniž by cestou hrozilo její odtajnění. Dozvěděli jsme se, že vědní disciplína, která se ukrýváním zpráv zabývá, se jmenuje steganografie a na příkladu vězňů Alice a Boba jsme si demonstrovali předávání plánu útěku z vězení, aniž by dozorce Wendy získala jakékoliv podezření. Historicky první použití steganografie se datuje již do 5. století před n. l. a v různých technikách, v práci popsaných, se používá až do dnes.

Hlavní výhoda steganografie je ve schopnosti přenosu ukryté zprávy spolu s nosičem (maskovací objekt nebo-li cover) veřejnými cestami a to naprosto nepozorovaně. Pro ukrytí zprávy existuje v dnešní éře digitální komunikace celá řada vhodných, v internetové komunikaci běžně používaných, maskovacích objektů, jako např. obrázky, zvuk, text, TCP datagram a jiné. Využívá se různých technik práce s covery, každá technika má své výhody a nevýhody a je jen na implementátorovi, jakou technologii se rozhodne využít. Pokud potřebuje skrytě přenést velké množství dat, nabízí se LSB, pokud chce být maximálně nenápadný, využijeme rezervního místa v datagramech TCP nebo prostoru v hlavičce mp3 rámce.

S využitím vhodných technologií jsem vytvořil vlastní aplikaci řešící ukrývání zprávy do obrázku a zvukového souboru. Konkrétně jsem použil algoritmus LSB (využití nejméně důležitého bitu bajtu) a pro ztížení případného odhalení zprávy jsem implementoval kompresi a kryptování zprávy. V příloze jsem detailně rozebral data obrázku a ukázal, na kterých místech je zpráva ukryta. Provedl jsem ověření technologie na celé řadě obrázků, přičemž nejlepších výsledků dává algoritmus u členitých, barevných, nejlépe 24 bitových, obrázků. Zajímavé bylo testování ukrytí zprávy do obrázku vyplněného jedinou barvou, zpráva není pouhým okem patrná ani v případě, že tou barvou je barva bílá. Testování ukrývání zpráv do audio souboru ověřilo, že drobná úprava vzorku opravdu není lidskými smysly postřehnutelná a lze zprávu odhalit pouze detailním zkoumáním vnitřních dat souboru.

Závěrem lze konstatovat, že steganografie je užitečná v případě, kdy potřebujeme nejen zabránit útočníkovi v přečtení obsahu zprávy, ale utajit samu skutečnost, že zprávu odesíláme.

CONCLUSION

The introduction of this paper introduces possibilities of hiding a message so that it can be safely delivered to its recipient without danger of being compromised during delivery. We've learned that message hiding is studied by science called steganography, and demonstrated how imprisoned Alice and Bob can communicate about escape plan without warden Wendy getting suspicious. First uses of steganography is dated to the 5th century BC and various techniques mentioned in the paper are being used.

Main advantage of steganography is transmitting the hidden message along with masking object (called cover) through public channels, without being noticed. Modern era offers us a number of suitable object commonly used in internet communication, such as picture, sound, text or TCP packets. Various techniques of manipulating covers are available, with different advantages and disadvantages. The implementer chooses, which technique to use - LSB for large amount of data, reserved space in TCP datagrams or MP3 frame header for maximum security.

I've used suitable technologies to created my own application for hiding a message in a picture or sound file. Specifically I've used LCB algorithm (using least important bite in a byte), using compression and encryption to make eventual attempts to discover the hidden message more difficult.

An appendix contains detailed analysis of picture data and marked positions the message was hidden in. I've tested the algorithm on number of pictures, getting best results with diverse, colored, 24bit (or more) pictures. Interesting results were gained from experimental hiding of message to picture filled with single color -- the message is not visible to naked eye even in case of white color. Testing on sound files proved that small alternation of sample is not detectable by human senses and message can only be discovered by detailed analysis of file data.

Steganography proved useful in case when we aim to completely hide the fact a message is being sent, rather then merely preventing possible eavesdropper to read its contents.

SEZNAM POUŽITÉ LITERATURY

[1] *CITI - Center for Information Technology Integration* [online]. 4. 1. 2002 [cit. 2010-05-05]. Steganography Press Information. Dostupné z WWW: <http://www.citi.umich.edu/u/provos/stego/faq.html>.

[2] *Facebook* [online]. 2006 [cit. 2010-05-14]. Social Network. Dostupné z WWW: <www.facebook.com>.

[3] *Flickr* [online]. 2004 [cit. 2010-05-14]. Photo Sharing. Dostupné z WWW: <www.flickr.com>.

[4] *Freshmeat* [online]. 0.6. 2004 [cit. 2010-05-23]. Stegdetect. Dostupné z WWW: <http://freshmeat.net/projects/stegdetect/>.

[5] *Pavel Moravčík - VŠB* [online]. 2000 [cit. 2010-05-09]. Šifrovací algoritmus DES. Dostupné z WWW: < http://homel.vsb.cz/~mor196/mal338.pdf >.

[6] *Security portal* [online]. 2.12.2004 [cit. 2010-05-05]. Praktické základy Kryptologie a Steganografie. Dostupné z WWW: < http://www.security-portal.cz/clanky/praktick%C3%A9-z%C3%A1klady-kryptologie-steganografie >.

[7] *Sourceforge* [online]. 2002 [cit. 2010-05-14]. Hide In Picture. Dostupné z WWW: < http://sourceforge.net/projects/hide-in-picture/ >.

[8] *Stego Archiv* [online]. 23. 8. 1998 [cit. 2010-05-06]. Steganography Software. Dostupné z WWW: <http://home.comcast.net/~ebm.md/stego/softwarewindows .html>.

[9] *UFOs workshop* [online]. 2003 [cit. 2010-05-14]. SGCrypt 2 - Downloads: Programy. Dostupné z WWW: <http://ufosworkshop.wz.cz/index.php?page= downloads&kat=1>.

[10] *Unifiction* [online]. 2001 [cit. 2010-05-14]. Camouflage. Dostupné z WWW: < http://camouflage.unfiction.com/ >.

[11] *Universal Steganographic Tool* [online]. 14. 4. 2001, 6. 9. 2004 [cit. 2010-05-05]. OutGuess. Dostupné z WWW: < http://www.outguess.org/ >.

[12] BAILEY, Jonathan. *Plagiarism Today* [online]. 2009 [cit. 2010-05-24]. Image Detection: Watermarking vs. Fingerprinting. Dostupné z WWW: <http://www.plagiarismtoday.com/2009/12/02/image-detection-watermarking-vs-fingerprinting/>.

[13] BARBEC, Tomáš. *Šifra Blowfish* [online]. 2000 [cit. 2010-05-09]. Více o teorii blokových šifer. Dostupné z WWW: < http://moon.felk.cvut.cz/~pjv/Jak/_info/i677/html/BlockCipher.htm >.

[14] DOSTÁLEK, Libor. *Velký průvodce protokoly TCP/IP a systémem DNS* [online]. 2002 [cit. 2010-05-13]. Protokol TCP (Transmision Control Protocol). Dostupné z WWW: < http://www.cpress.cz/knihy/tcp-ip-bezp/CD-0x/9.html >.

[15] HAMLIN, Sean. *Steganography made easy* [online]. 2005 [cit. 2010-05-14]. Revelation. Dostupné z WWW: < http://revelation.atspace.biz/index.html >.

[16] HOLEČEK, Ondřej. *Root* [online]. 2003 [cit. 2010-05-25]. CRC (kontrolní součet). Dostupné z WWW: <http://www.root.cz/clanky/crc-kontrolni-soucet/>.

[17] HOUSER, Pavel. *SecurityWorld* [online]. 2009 [cit. 2010-05-16]. Steganografie v protokolu TCP. Dostupné z WWW: <http://securityworld.cz/securityworld/Steganografie-v-TCP-1695>. [webová stránka]

[18] HOLOŠKA, Jiří. *Odhalování steganografie pomocí neuronových sítí*. Zlín, 2008. 73 s. Diplomová práce. UTB Zlín, Fakulta aplikované informatiky.

[19] KELLEY, Jack. *USA Today* [online]. 2001 [cit. 2010-05-14]. Terror groups hide behind Web encryption. Dostupné z WWW: < http://www.usatoday.com/tech/news/2001-02-05-binladen.htm >.

[20] KLÍMA, Vlastimil. Šifru v pytli neutajíš : Skipjack a KEA. *CHIP*. 1999, 1, s. 46-47.

[21] KOLÁČEK, Michal. *Svět hardware* [online]. 2009 [cit. 2010-05-09]. Šifrování a biometrie pod drobnohledem. Dostupné z WWW: < http://www.svethardware.cz/art_doc-79DA710F91AF8358C125755B002F815D.html >.

[22] KRČMÁŘ, Petr. *Root* [online]. 2003 [cit. 2010-05-15]. Unixová komprese v praxi: Gzip. Dostupné z WWW: <http://www.root.cz/clanky/unixova-komprese-v-praxi-gzip/>.

[23] MACIAK, Lukasz Grzegorz ; PONNIAH, Micheal Alexis; SHARMA, Renu. *Terminally Incoherent* [online]. 2005 [cit. 2010-05-16]. MP3 Steganography. Dostupné z WWW: <http://www.terminally-incoherent.com/stuff/projects/mp3stego/paper.doc>.

[24] MAZURCZYK, Wojciech ; SZCZYPIORSKI, Krzysztof. *SecurityWorld* [online]. 2008 [cit. 2010-05-13]. Moderní steganografie: Zprávu lze ukrýt i do dat VoIP. Dostupné z WWW: < http://securityworld.cz/securityworld/moderni-steganografie-zpravu-lze-ukryt-i-do-dat-voip-550 >.

[25] PETITCOLAS, Fabien. *The information hiding homepage* [online]. 2006 [cit. 2010-05-14]. MP3 Stego. Dostupné z WWW: < http://www.petitcolas.net/fabien/steganography/mp3stego/ >.

[26] PETITCOLAS, Fabien A. P.; ANDERSON, Ross J.; KUHN, Markus G. . Information Hiding - A Survey. *Petitcolas* [online]. 1999, , [cit. 2010-05-13]. Dostupný z WWW: < http://www.cl.cam.ac.uk/~fapp2/publications/ieee99-infohiding.pdf >.

[27] RŮŽIČKA, Rudolf. *Počítače a hudba* [online]. 2003 [cit. 2010-05-16]. Zvukový formát WAV. Dostupné z WWW: <http://www.fi.muni.cz/~qruzicka/Duffek/wav.html>.

[28] RYŠÁNKOVÁ, Alžběta. *Kryptologie - Univerzita Hradec Králové* [online]. 2. 4. 2003 [cit. 2010-05-05]. Steganografie. Dostupné z WWW: < http://kryptologie.uhk.cz/81.htm >.

[29] SESHADRI, Vidhyasri. *Atcomm Enterprises* [online]. 2009 [cit. 2010-05-24]. The J2ME Crypto Recipes. Dostupné z WWW: <http://www.atcomm.com.au/Atcomm-Underground/Java-Mobile-Development/The-J2ME-Crypto-Recipes.html>.

[30] SINGH, Simon. *Kniha kódů a šifer : Tajná komunikace od starého Egypta po kvantovou kryptografii*. Česká Republika : Dokořán, Argo, 2003. 384 s. ISBN 80-86569-18-7.

[31] TIŠNOVSKÝ, Pavel. *Root* [online]. 2006 [cit. 2010-05-16]. Grafický formát BMP - používaný a přitom neoblíbený. Dostupné z WWW: <http://www.root.cz/clanky/graficky-format-bmp-pouzivany-a-pritom-neoblibeny/>.

[32] TIŠNOVSKÝ, Pavel. *Root* [online]. 2006 [cit. 2010-05-16]. Anatomie grafického formátu PNG. Dostupné z WWW: <http://www.root.cz/clanky/anatomie-grafickeho-formatu-png/>.

[33] WAYNER, Peter. *Disappearing cryptography : information hiding : steganography & watermarking*. 2nd ed. Amsterdam : MK/Morgan Kaufmann Publishers, 2002. 413 s. ISBN 1558607692.

[34] ZIGULEC, Štefan. *Steganografická komunikace*. Zlín, 2006. 63 s. Diplomová práce. UTB Zlín, Fakulta aplikované informatiky.

[35] ŽILKA, Roman. *Steganografie a stegoanalýza*. Brno, 2008. 66 s. Diplomová práce. Masarykova univerzita.

[36] Data Encryption Standard In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2007, 2009 [cit. 2010-05-15]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Data_Encryption_Standard>.

[37] TripleDES In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2007, 2009 [cit. 2010-05-15]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/TripleDES>.

[38] AES In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2007, 2010 [cit. 2010-05-15]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/AES>.

[39] *Informační systém MU* [online]. 2009 [cit. 2010-05-15]. Historie Kryptografie. Dostupné z WWW: <http://is.muni.cz/el/1451/jaro2009/cr019/kryptografie.txt>.

[40] International Data Encryption Algorithm In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2006, 2010 [cit. 2010-05-15]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/International_Data_Encryption_Algorithm>.

[41] Cyclic redundancy check In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2004, 2010 [cit. 2010-05-15]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Cyclic_redundancy_check>.

[42] Komprese dat In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2005, 2010 [cit. 2010-05-15]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Komprese_dat>.

[43] Gzip In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2007, 2010 [cit. 2010-05-15]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Gzip>.

[44] Zvuk. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2007, last modified on 2010 [cit. 2010-05-25]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Zvuk>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application programming interface (rozhraní pro programování aplikací)
ASCII	American Standard Code for Information Interchange
AWT	Abstract Window Toolkit
BMP	Microsoft Windows Bitmap
GUI	Graphical user interface
JPEG	Joint Photographic Experts Group
LBE	Low Bit Encoding
LSB	Least Significant Bit - nejméně podstatný bit informace
MMS	Multimedia Messaging Service
MP3	MPEG-1 Audio Layer 3
MPEG	Moving Picture Experts Group
PNG	Portable Network Graphics
SWT	Standard Widget Toolkit
TCP	Transmission Control Protocol
UML	Unified Modeling Language
UTB	Univerzita Tomáše Bati
VoIP	Voice over Internet Protocol
WAV	Waveform Audio File Format

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: POPIS TESTOVACÍCH BITMAP (BMP)

Pro vysvětlení, jakým způsobem a do jakých míst v bitmapě je ukrývána zpráva, si provedeme ukázky ukrytí zprávy do 24 bitové bitmapy šedé barvy.

Tabulka 8: Hexa náhled na 24b bitmapu 10x10 šedé barvy

0x00000000:	42	4d	76	01	00	00	00	00	00	00	36	00	00	00	28	00
0x00000010:	00	00	0a	00	00	00	0a	00	00	00	01	00	18	00	00	00
0x00000020:	00	00	40	01	00	00	13	0b	00	00	13	0b	00	00	00	00
0x00000030:	00	00	00	00	00	00	99	99	99	99	99	99	99	99	99	99
0x00000040:	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
0x00000050:	99	99	99	99	00	00	99	99	99	99	99	99	99	99	99	99
0x00000060:	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
0x00000070:	99	99	99	99	00	00	99	99	99	99	99	99	99	99	99	99
0x00000080:	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
0x00000090:	99	99	99	99	00	00	99	99	99	99	99	99	99	99	99	99
0x000000a0:	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
0x000000b0:	99	99	99	99	00	00	99	99	99	99	99	99	99	99	99	99
0x000000c0:	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
0x000000d0:	99	99	99	99	00	00	99	99	99	99	99	99	99	99	99	99
0x000000e0:	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
0x000000f0:	99	99	99	99	00	00	99	99	99	99	99	99	99	99	99	99
0x00000100:	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
0x00000110:	99	99	99	99	00	00	99	99	99	99	99	99	99	99	99	99
0x00000120:	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
0x00000130:	99	99	99	99	00	00	99	99	99	99	99	99	99	99	99	99
0x00000140:	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
0x00000150:	99	99	99	99	00	00	99	99	99	99	99	99	99	99	99	99
0x00000160:	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
0x00000170:	99	99	99	99	00	00										

Prvních 54 bajtů (šedá políčka) patří hlavičce, zbylá data (bílá políčka) jsou sekvence jednotlivých pixelů, přičemž každý pixel je popsán třemi bajty (ve formátu BGR, nikoliv jak jsme zvyklí RGB). Jeden řádek se skládá z 10 pixelů * 3 bajty (BGR) = 30 bajtů na popis jedno řádku. Každá scanovací řádka je vždy násobkem čtyř bajtů a pokud je potřeba, je počet doplněn nulami. To je vidět i v našem případe, kdy je řádek doplněn 2 bajty „nul“.

Tabulka 9 vysvětluje smysl prvních 54 bajtů. K nejdůležitějším údajům patří:

- začátek vlastních dat (pixelů) obrázku
- velikost hlavičky
- šířka a výška bitmapy
- počet bitů na pixel

Tabulka 9: Struktura hlavičky 24b bitmapy 10x10

Pozice	Bajty				Popis
0	42	4d			Identifikátor formátu BMP. Aktuální verze formátu BMP zde obsahuje ASCII kód znaků „BM“, tj. 0×42 a 0×4d hexadecimálně.
2	76	01	00	00	Velikost souboru 374 bajtů
6	00	00			
8	00	00			
10	36	00	00	00	Hodnota 54 – začátek vlastních dat (pixelů) obrázku
14	28	00	00	00	Hodnota 40 – velikost hlavičky
18	0a	00	00	00	Šířka obrazu 10 pixelů
22	0a	00	00	00	Výška obrazu 10 pixelů
26	01	00			Počet bitových rovin
28	18	00			Počet bitů na pixel, v našem případě 24
30	00	00	00	00	
34	40	01	00	00	Hodnota 320 – velikost obrazu v bajtech. Velikost jednoho řádku je 10 pixelů * 3 barvy = 30 bajtů. Je třeba doplnit nulami tak, aby řádek byl dělitelný bezezbytku hodnotou 4, tedy je třeba doplnit 2 bajty, tj. Řádek se skládá z 32 bajtů. 10 řádků obrázku * 32 bajtů = 320 bajtů.
38	00	00	00	00	
42	00	00	00	00	
46	00	00	00	00	
50	00	00	00	00	

Abychom pochopili princip ukládání jednoho konkrétního bodu, budeme potřebovat bitmapu bílé barvy.

. *Tabulka 10: Hexa náhled na 24b bitmapu 10x10 bílé barvy*

0x00000000:	42	4d	76	01	00	00	00	00	00	00	36	00	00	00	28	00
0x00000010:	00	00	0a	00	00	00	0a	00	00	00	01	00	18	00	00	00
0x00000020:	00	00	40	01	00	00	13	0b	00	00	13	0b	00	00	00	00
0x00000030:	00	00	00	00	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000040:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000050:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000060:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000070:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000080:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000090:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000a0:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000b0:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000c0:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000d0:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000e0:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000f0:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000100:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000110:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000120:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000130:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000140:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000150:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000160:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000170:	ff	ff	ff	ff	00	00										

Všimněme si, že ačkoliv jsou data obrázku bílé bitmapy zcela odlišné od bitmapy šedé, hlavička je shodná.

Práce s 8 bitovými BMP

Hlavní rozdíl práce s 8 bitovými bitmapami (obecně indexovanými, mohou být i 7 bitové, 4 bitové a pod.) je v tom, že jednotlivé body nejsou popisovány konkrétní barvou (RGB), ale odkazem do palety barev, která je vložena bezprostředně za hlavičku.

Tabulka 11: Hexa náhled na 8b bitmapu 16x6 (paleta 17 barev)

0x00000000:	42	4d	7a	01	00	00	00	00	00	00	7a	00	00	00	28	00
0x00000010:	00	00	10	00	00	00	10	00	00	00	01	00	08	00	00	00
0x00000020:	00	00	00	01	00	00	13	0b	00	00	13	0b	00	00	11	00
0x00000030:	00	00	11	00	00	00	00	00	00	00	7f	00	00	00	ff	00
0x00000040:	00	00	00	00	7f	00	7f	00	7f	00	33	33	33	00	00	00
0x00000050:	ff	00	ff	00	ff	00	4c	4c	4c	00	00	7f	00	00	7f	7f
0x00000060:	00	00	00	7f	82	00	99	99	99	00	00	ff	00	00	ff	ff
0x00000070:	00	00	00	ff	ff	00	ff	ff	ff	00	06	10	10	10	10	10
0x00000080:	10	10	10	10	10	10	10	10	10	02	0e	10	10	10	10	10
0x00000090:	10	10	10	10	10	10	10	10	10	0d	0c	10	10	10	10	10
0x000000a0:	10	10	10	10	10	10	10	10	10	03	07	10	10	10	10	10
0x000000b0:	10	10	10	10	10	10	10	10	10	04	01	10	10	10	10	10
0x000000c0:	10	10	10	10	10	10	10	10	10	0a	09	10	10	10	10	10
0x000000d0:	10	10	10	10	10	10	10	10	10	0b	05	10	10	10	10	10
0x000000e0:	10	10	10	10	10	10	10	10	10	08	10	10	10	10	10	10
0x000000f0:	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
0x00000100:	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
0x00000110:	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
0x00000120:	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
0x00000130:	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
0x00000140:	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
0x00000150:	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
0x00000160:	10	10	10	10	10	10	10	10	10	10	0f	10	10	10	10	10
0x00000170:	10	10	10	10	10	10	10	10	10	00						

Šedá políčka (bajt 0 až 53) je hlavička bitmapy, žlutá zvýrazněná část (bajt 54 až 121) je barevná paleta. Vlastní data tedy začínají od pozice 122, jak lze vyčíst i z hlavičky.

Tabulka 12: Struktura hlavičky 8b bitmapy 16x16 (paleta 17 barev)

Pozice	Bajty				Popis
0	42	4d			Identifikátor formátu BMP.
2	7a	01	00	00	Velikost souboru 17a = 378 bajtů
6	00	00			-
8	00	00			-
10	7a	00	00	00	Hodnota 122 – začátek vlastních dat (pixelů) obrázku
14	28	00	00	00	Hodnota 40 – velikost hlavičky
18	10	00	00	00	Šířka obrazu 16 pixelů
22	10	00	00	00	Výška obrazu 16 pixelů
26	01	00			Počet bitových rovin
28	08	00			Počet bitů na pixel, v našem případě 8
30	00	00	00	00	Metoda komprimace=žádná
34	00	01	00	00	Hodnota 256 – velikost obrazu v bajtech. 16 řádků x 16 sloupců = 256
38	13	0b	00	00	horizontální počet pixelů na metr
42	13	0b	00	00	vertikální počet pixelů na metr
46	11	00	00	00	Celkový počet barev 17
50	11	00	00	00	Počet důležitých barev 17

PŘÍLOHA P II: EXPERIMENTY S BITMAPAMI

Vložení testovacích bodů do bílé bitmapy

Abychom pochopili uložení obrazových bodů v bitmapě, provedeme experiment s uložením červeného (BGR: 00 00 FF) a modrého (BGR: FF 00 00) pixelu do bitmapy bílé barvy (abychom pixely snadněji našli).

Tabulka 13: Významový náhled na 24b bitmapu 10x10 s červeným a modrým pixelem

	1	2	3	4	5	6	7	8	9	10
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

Když porovnáme tabulku 13 a 14 zjistíme, že obrazové pixely jsou v bitmapě ukládány od levého spodního rohu (řádek 10, sloupec 1)a postupují po řádcích výše.

Tabulka 14: Hexa náhled na 24b bitmapu 10x10 bílé barvy s uloženým červeným a modrým pixelem

0x00000000:	42	4d	76	01	00	00	00	00	00	00	36	00	00	00	28	00
0x00000010:	00	00	0a	00	00	00	0a	00	00	00	01	00	18	00	00	00
0x00000020:	00	00	40	01	00	00	13	0b	00	00	13	0b	00	00	00	00
0x00000030:	00	00	00	00	00	00	00	00	ff	ff	ff	ff	ff	ff	ff	ff
0x00000040:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000050:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000060:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000070:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000080:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000090:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000a0:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000b0:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000c0:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000d0:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000e0:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x000000f0:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000100:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000110:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000120:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000130:	ff	ff	ff	ff	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000140:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000150:	ff	ff	ff	ff	00	00	ff	00	00	ff	ff	ff	ff	ff	ff	ff
0x00000160:	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0x00000170:	ff	ff	ff	ff	00	00										

Přístup Javy k obrazovým bodům

V předchozím experimentu jsme si ukázali, že obrazové pixely jsou do BMP formátu ukládány od spodního řádku postupně nahoru. Při implementaci technologie LSB načteme obrazové pixely BGR do pole, ale s tím rozdílem, že třída BufferedImage zajistí, že pole začíná levým horním řádkem a postupuje dolů. První obrazový pixel z pohledu Javy je tedy podle tabulky 13 modrý bod.

Vytvoříme si simulaci a pole (BufferedImage) naplníme hodnotami 00 až FE (namísto původních hodnot BGR), zbylé pixely nastavíme hodnotou 00. Takto zmodifikované pole uložíme a podíváme se, kde přesně v bitmapě jsou data uložena.

Tabulka 15: Hexa náhled na uměle generovanou (experimentální) 24b bitmapu 10x10

0x00000000:	42	4d	76	01	00	00	00	00	00	00	36	00	00	00	28	00
0x00000010:	00	00	0a	00	00	00	0a	00	00	00	01	00	18	00	00	00
0x00000020:	00	00	40	01	00	00	00	00	00	00	00	00	00	00	00	00
0x00000030:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x00000040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x00000050:	00	00	00	00	00	00	f0	f1	f2	f3	f4	f5	f6	f7	f8	f9
0x00000060:	fa	fb	fc	fd	fe	00	00	00	00	00	00	00	00	00	00	00
0x00000070:	00	00	00	00	00	00	d2	d3	d4	d5	d6	d7	d8	d9	da	db
0x00000080:	dc	dd	de	df	e0	e1	e2	e3	e4	e5	e6	e7	e8	e9	ea	eb
0x00000090:	ec	ed	ee	ef	00	00	b4	b5	b6	b7	b8	b9	ba	bb	bc	bd
0x000000a0:	be	bf	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	ca	cb	cc	cd
0x000000b0:	ce	cf	d0	d1	00	00	96	97	98	99	9a	9b	9c	9d	9e	9f
0x000000c0:	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	aa	ab	ac	ad	ae	af
0x000000d0:	b0	b1	b2	b3	00	00	78	79	7a	7b	7c	7d	7e	7f	80	81
0x000000e0:	82	83	84	85	86	87	88	89	8a	8b	8c	8d	8e	8f	90	91
0x000000f0:	92	93	94	95	00	00	5a	5b	5c	5d	5e	5f	60	61	62	63
0x00000100:	64	65	66	67	68	69	6a	6b	6c	6d	6e	6f	70	71	72	73
0x00000110:	74	75	76	77	00	00	3c	3d	3e	3f	40	41	42	43	44	45
0x00000120:	46	47	48	49	4a	4b	4c	4d	4e	4f	50	51	52	53	54	55
0x00000130:	56	57	58	59	00	00	1e	1f	20	21	22	23	24	25	26	27
0x00000140:	28	29	2a	2b	2c	2d	2e	2f	30	31	32	33	34	35	36	37
0x00000150:	38	39	3a	3b	00	00	00	01	02	03	04	05	06	07	08	09
0x00000160:	0a	0b	0c	0d	0e	0f	10	11	12	13	14	15	16	17	18	19
0x00000170:	1a	1b	1c	1d	00	00										

Význam zvýrazněných polí je vysvětlen v následující tabulce 16. Všimněme si, že doplněné nuly (zarovnání na 4 bajty) zůstávají (nebyly nijak modifikovány), v poli (BufferedImage) nejsou načteny, nemusíme se o ně vůbec starat.

Tabulka 16: Význam uložených dat ve 24b bitmapě (postupka 00 až FE)

00 01 02	První obrazový bod vlevo nahoře (1. řádek, 1. sloupec)
03 04 05	Druhý obrazový bod vlevo nahoře (1. řádek, 2. sloupec)
1b 1c 1d	Desátý obrazový bod (1. řádek, 10. sloupec)
1e 1f 20	Jedenáctý obrazový bod (2. řádek, 1. sloupec)
00 00 00	Stý (poslední) obrazový bod (10. řádek, 10. sloupec)

Ukrytí zprávy do bitmapy, část I

Provedeme si ukázkové ukrytí slova „UTB“ do testovací 24 bitové bitmapy 10x10 šedé barvy, viz tabulka 8 (BGR 99 99 99). Máme k dispozici celkem 300 bajtů pro uložení nějaké informace (10 řádků x 10 sloupců * 3 BGR). Kromě samotného slova je zapotřebí ukrýt i hlavičku, abychom dokázali ukrytý text znovu nalézt. Dle kapitoly 4.4.1 je zapotřebí 240 bajtů pro uložení hlavičky.

Tzn. že pro ukrytí zprávy máme k dispozici prostor délky (300 – 240) / 8, tj. 7,5 znaku. Volné znaky jsou v následující tabulce 17 reprezentovány šedými „99“. Ukrytá zpráva a pomocná hlavička je v tabulce barevně zvýrazněna a detailně popsána v tabulce 18

Tabulka 17: Hexa náhled na 24b bitmapu šedé barvy se zakódovaným slovem UTB

0x00000000:	42	4d	76	01	00	00	00	00	00	00	36	00	00	00	28	00
0x00000010:	00	00	0a	00	00	00	0a	00	00	00	01	00	18	00	00	00
0x00000020:	00	00	40	01	00	00	00	00	00	00	00	00	00	00	00	00
0x00000030:	00	00	00	00	00	00	99	99	99	99	99	99	99	99	99	99
0x00000040:	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
0x00000050:	99	99	99	99	00	00	98	99	98	99	98	99	98	99	98	99
0x00000060:	98	99	98	99	98	98	98	99	98	98	98	98	99	98	99	99
0x00000070:	99	99	99	99	00	00	99	98	98	98	98	98	98	98	99	98
0x00000080:	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98
0x00000090:	98	98	98	98	00	00	98	98	98	98	98	98	99	98	98	98
0x000000a0:	98	98	98	98	99	98	98	98	98	98	98	98	99	98	98	98
0x000000b0:	98	98	98	98	00	00	98	98	98	98	99	98	98	98	98	98
0x000000c0:	98	98	99	98	98	98	98	98	98	98	99	98	98	98	98	98
0x000000d0:	98	98	99	98	00	00	98	98	99	98	98	98	98	98	98	98
0x000000e0:	99	98	98	98	98	98	98	98	99	98	98	98	98	98	98	98
0x000000f0:	99	98	98	98	00	00	99	98	98	98	98	98	98	98	99	98
0x00000100:	98	98	98	98	98	98	99	98	98	98	98	98	98	98	99	98
0x00000110:	98	98	98	98	00	00	98	99	98	99	98	98	99	99	99	99
0x00000120:	99	99	98	98	99	98	98	98	98	98	98	98	99	98	98	98
0x00000130:	98	98	98	98	00	00	99	99	99	99	98	99	98	98	99	99
0x00000140:	98	98	99	99	98	98	99	99	99	98	98	98	99	99	98	99
0x00000150:	99	98	99	98	00	00	98	98	98	98	98	98	98	98	98	98
0x00000160:	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98	98
0x00000170:	98	98	98	98	00	00										

Díky jinému náhledu Javy na uložená data bitmapy jsme docílili vedlejšího efektu a to zhoršení čitelnosti uložených dat pro případného útočníka. Pokud by útočník chtěl data začít zkoumat, musel by si je přeorganizovat do pohledu „od levého horního pixelu po pravý spodní“ a tato data teprve podrobit steganografické analýze.

Ukrytí zprávy do bitmapy, část II

Tabulka 18: Význam uložených dat ve 24b bitmapě (zakódované slovo UTB)

99	Volné nevyužité místo v bitmapě (bez uložené zprávy). Původní hodnota barvy 99 je nezměněna
xx	Otazník (symbol ručně typované zprávy). Je uložen ve formě ASCII znaku „?“ (dekadicky 67, binárně 00111111) v 8 bajtech. V našem případě následující sekvence: 98 98 99 99 99 99 99 99. Ve zbývajících 152 bajtech, tj. ve 152 posledních bitech dané sekvence bajtů, což představuje 19 bajtů je uložen zbytek znaků doplňujících název souboru, v tomto případě se jedná o znaky mezery, tj. (hodnota jednoho získaného bajtu je dekadicky 32, což představuje binárně hodnotu 00100000).
xx	Hodnota CRC součtu původní zprávy (tedy nekryptované a nekomprimované). Je zakódována ve 32 bajtech (v každém bajtu 1 bit), má tedy velikost 4 bajty
xx	Velikost ukrývané zprávy (binárně zakódovaná 3). Hodnota je zakódována ve 32 bajtech (v každém bajtu 1 bit), Má tedy velikost 4 bajty
xx	Znak U (první znak ukrývané zprávy) – je uložen ve formě ASCII hodnoty znaku U, tedy ASCII hodnoty 85, což je binárně 10101010, v našem případě tedy představuje poslední bity sekvence 98 99 98 99 98 99 98 99
xx	Znak T (druhý znak ukrývané zprávy) – je uložen ve formě ASCII hodnoty znaku T, tedy ASCII hodnoty 84, což je binárně 01010100, v našem případě tedy představuje poslední bity sekvence 98 99 98 99 98 99 98 98
xx	Znak B (třetí znak ukrývané zprávy) – je uložen ve formě ASCII hodnoty znaku B, tedy ASCII hodnoty 66, což je binárně 01000010, v našem případě tedy představuje poslední bity sekvence 98 99 98 98 98 98 99 98
xx	Typ použitého kryptováni. Z důvodu snadné demonstrace kryptování nebylo použito, je zde uložena „0“, tedy sekvence hodnot 98 98 98 98 98 98 98 98
xx	Příznak použití komprese a kryptování. Z důvodu snadné demonstrace nebyla komprese (ani kryptování) použita, je zde uložena „0“, tedy sekvence hodnot 98 98 98 98 98 98 98 98

Rozkódování ukryté zprávy v bitmapě

Tabulku 17 můžeme přeorganizovat do struktury po 8 bajtech a každou „osmici“ převedeme na binární tvar, přičemž k hodnotě „98“ přistupujeme jako k „0“, hodnota „99“ je „1“. Postupně si zrekonstruujeme hlavičku, tedy zjistíme délku ukládané zprávy, její CRC součet, informaci, zda-li se jedná o ručně typovaný text nebo binární dokument, zjistíme použití komprese případně typu kryptování.

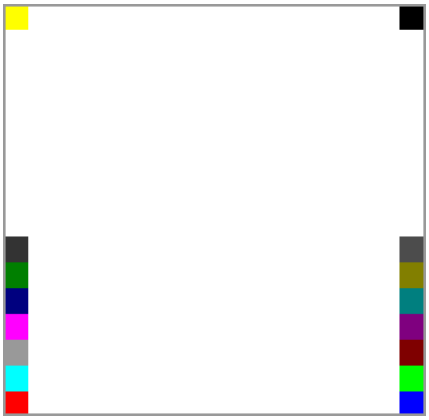
Díky hlavičce se již můžeme dostat jednoduše k ukryté zprávě a odtajnit ji.

Tabulka 19: Rozkódování ukryté zprávy UTB (po jednotlivých bajtech) 24b bitmapa

Poř.	B7	B6	B5	B4	B3	B2	B1	B0	Binární zápis	Význam informace
1	98	98	98	98	98	98	98	98	00000000	3 – délka zprávy
2	98	98	98	98	98	98	98	98	00000000	
3	98	98	98	98	98	98	98	98	00000000	
4	98	98	98	98	98	98	99	99	00000011	
5	99	99	98	99	98	98	99	99	11010011	CRC součet
6	98	98	99	99	98	98	99	99	00110011	
7	99	98	98	98	99	99	98	99	10001101	
8	99	98	99	98	98	99	98	99	10100101	
9	98	98	99	99	99	99	99	99	00111111	? - příznak ručně typovaného textu
10	98	98	99	98	98	98	98	98	00100000	*mezera*
11	98	98	99	98	98	98	98	98	00100000	*mezera*
12	98	98	99	98	98	98	98	98	00100000	*mezera*
13	98	98	99	98	98	98	98	98	00100000	*mezera*
14	98	98	99	98	98	98	98	98	00100000	*mezera*
15	98	98	99	98	98	98	98	98	00100000	*mezera*
16	98	98	99	98	98	98	98	98	00100000	*mezera*
17	98	98	99	98	98	98	98	98	00100000	*mezera*
18	98	98	99	98	98	98	98	98	00100000	*mezera*
19	98	98	99	98	98	98	98	98	00100000	*mezera*
20	98	98	99	98	98	98	98	98	00100000	*mezera*
21	98	98	99	98	98	98	98	98	00100000	*mezera*
22	98	98	99	98	98	98	98	98	00100000	*mezera*
23	98	98	99	98	98	98	98	98	00100000	*mezera*
24	98	98	99	98	98	98	98	98	00100000	*mezera*
25	98	98	99	98	98	98	98	98	00100000	*mezera*
26	98	98	99	98	98	98	98	98	00100000	*mezera*
27	98	98	99	98	98	98	98	98	00100000	*mezera*
28	98	98	99	98	98	98	98	98	00100000	*mezera*
29	98	98	98	98	98	98	98	98	00000000	0 – komprese NE, kryptováni NE
30	98	98	98	98	98	98	98	98	00000000	0 – není definován typ kryptování
31	98	99	98	99	98	99	98	99	01010101	U – dekódovaný znak zprávy (1)
32	98	99	98	99	98	99	98	98	01010100	T – dekódovaný znak zprávy (2)
33	98	99	98	98	98	98	99	98	01000010	B – dekódovaný znak zprávy (3)
34	99	99	99	99	99	99	99	99	11111111	-
35	99	99	99	99	99	99	99	99	11111111	-
36	99	99	99	99	99	99	99	99	11111111	-
37	99	99	99	99	99	99	99	99	11111111	-
38	99	99	99	99					1111	-

Prostor pro ukrytí dat u 8 bitového BMP, část I

V tabulce 11 jsme si představili testovací 8 bitovou bitmapu 16 x 16 bodů. Je to bitmapa bílé barvy, do které bylo z ladících důvodů přidáno 16 barevných teček. Bitmapa vypadá následovně:



Obrázek 18: Náhled na 8b bitmapu 16x16 (paleta 17 barev)

Dle 2.1.5 lze bitmapu se 16 barvami popsat 4 bity. Abychom dostali bitmapu 8 bitovou (viz rozbor hlavičky v tabulce 12, pozice 28), je zapotřebí, aby bitmapa obsahovala barev minimálně 17. Paleta může být pak buď úplná (256 barev) nebo optimalizovaná (jako v našem případě, paleta obsahuje jen použité barvy, tedy 17 barev).

Tabulka 20: Popis palety barev testované 8b bitmapy 16x16

Č.	Binární zápis	Definice barvy BGR			
00	00000000	00	00	00	*00*
01	00000001	7f	00	00	*00*
02	00000010	ff	00	00	*00*
03	00000011	00	00	7f	*00*
04	00000100	7f	00	7f	*00*
05	00000101	33	33	33	*00*
06	00000110	00	00	ff	*00*
07	00000111	ff	00	ff	*00*
08	00001000	4c	4c	4c	*00*
09	00001001	00	7f	00	*00*
0a	00001010	7f	7f	00	*00*
0b	00001011	00	7f	82	*00*
0c	00001100	99	99	99	*00*
0d	00001101	00	ff	00	*00*
0e	00001110	ff	ff	00	*00*
0f	00001111	00	ff	ff	*00*
10	00010000	ff	ff	ff	*00*

Paleta definuje použité barvy v obrázku. Každá barva se skládá ze 4 bajtů: 3 bajty jsou složky barev ve formátu BGR, čtvrtý bajt je nevyužitý – v tomto případě je 00, ale může být i jiný. Java při ukládání palety používá pro čtvrtý bajt FF.

Prostor pro ukrytí dat u 8 bitového BMP, část II

Tabulka 21 je detailní náhled na jednotlivé obrazové body rozdělené do řádků a sloupců. Barva každého bodu je reprezentována odkazem do palety barev, kterou jsme si definovali v tabulce 20.

Tabulka 21: Rozbor dat 8b BMP 16x16

Řádky \ Sloupce	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	06	10	10	10	10	10	10	10	10	10	10	10	10	10	10	02
15	0e	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0d
14	0c	10	10	10	10	10	10	10	10	10	10	10	10	10	10	03
13	07	10	10	10	10	10	10	10	10	10	10	10	10	10	10	04
12	01	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0a
11	09	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0b
10	05	10	10	10	10	10	10	10	10	10	10	10	10	10	10	08
9	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
8	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
7	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
6	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
5	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
4	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
3	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
2	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
1	0f	10	10	10	10	10	10	10	10	10	10	10	10	10	10	00

Ukrytí textu do takovéto bitmapy je, proti 24b bitmapě, trochu složitější. Představme si např. ukrytí informace do bodu [1,1]. Tento bod se odkazuje do palety barev na pozici 0F, tedy binárně 00001111 (žlutá barva). Pokud bychom na toto místo chtěli ukrýt „0“, modifikujeme hodnotu na 00001110, což znamená odkaz na barvu 0E, tedy v našem případě dle palety barev světle modrá. Takovéto změny jsou velice nápadné a mohou dost výrazně změnit vzhled obrázku. V případě úplné palety (neoptimalizované), která je korektně seřazena dle odstínů (sytosti) barev, modifikace ukazatelů do palety nevadí, ale ne každý 8bitový obrázek má takovou paletu.

V aplikaci jsem se rozhodl tento problém vyřešit tak, že do paměti nenačítám a nemodifikuji odkazy do palety barev, ale přímo barvy každého pixelu, následně pro upravené barvy vygeneruji novou paletu.