Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Steganografie v tištěných dokumentech

Plzeň 2019 David Friesecký

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 27. června 2019

David Friesecký

Abstract

The aim of this bachelor thesis is to create a program that allows to hide the message from the user to the text of the PDF document so that it can be read after printing and taking a photo of the coded text in the document. The program must detect and decode the coded message from the photo submitted to it.

Key words: steganography, print, PDF

Abstrakt

V této bakalářské práci je cílem vytvořit program, který umožní ukrýt zprávu od uživatele do textu v PDF dokumentu tak, aby byla čitelná i po vytisknutí a pořízení fotografie kódovaného textu v dokumentu. Program zakódovanou zprávu musí detekovat a dekódovat z fotografie, která mu je předložena.

Klíčová slova: steganografie, tisk, PDF

Obsah

1 Úvod

Steganografie je metoda, umožňující ukrýt zprávu do libolného přenosového média (např. text, video, audio, obrázky, aj.). V dnešní době se steganografie, díky modernizaci výpočetní techniky, výrazně rozrostla za účelem ochrany autorských práv [3, 4]. Dalším důvodem využívání je, že se dá použít k zabezpečení citlivých informací na místech, kde je z nějakého důvodu omezeno či zakázáno využití kryptografie [5, 14].

Výsledkem práce bude software, který bude pracovat s textovými dokumenty ve formátu PDF. Do předloženého dokumentu s textem zakóduje vybranou zprávu formou kódovací abecedy (tj. znaky převede např. do binární podoby). V opačném případě bude program zakódovanou zprávu detekovat a dekódovat z obrázku vytištěného dokumentu. Je tedy velmi důležité, aby zpráva zůstala čitelná po vytisknutí dokumentu, jeho vyfocení a následnému předání aplikaci pro dekódování. Tento software je určen jako pomůcka např. sjednatelům smluv tak, že do dokumentů s citlivými informacemi, které jsou součástí smlouvy, nechá zákodovat tajnou zprávu. Tu pak lze využít jako důkaz porušení smluvních podmínek o udržení tajemství.

V teoretické části je popsána stegnografie se stručným úvodem do historie, rozbor formátu PDF a jeho vlastností, možné knihovny, text v digitální podobě, možné metody a postupy kódování a dekódování, v poslední řadě jsou popsány bezpečnostní kódy.

V praktické části je popsáno, jaké metody kódování a dekódování byly pro steganografii v programu použity spolu s bezpečnostními kódy. Část je věnována experimentu, zaměřenému na přesnost metod dekódování fotek se šumem a vizuální dojem kódovaného textu na lidské subjekty.

2 Steganografie

Název „steganografie“ vychází z řeckého slova stegos „skrytý“ a grafia „psát“ neboli „skryté psaní“. Je vědní disciplínou spadající do vědního oboru kryptografie a zároveň je jejím přechůdcem [12]. Zabývá se utajením existence probíhající komunikace prostřednictvím libovolného přenosového média.

2.1 Historie

Jedním z prvních, nám známých důkazů použití steganografie je z bitvy u Salamíny, která se odehrála kolem roku 440 př. n. l. Před bitvou, která byla započata Peršany proti Řekům, vyhnanec Řek Damaratus ukryl zprávu na hliněných destičkách zalitých voskem. Řekové díky němu bitvu vyhráli [12, 16]. Dalšími byli Římané, kteří používali speciální inkousty na bázi mléka či ovocné šťavy. Na papíře byly neviditelné, ale po zahřátí zhnědly [16]. V 16. století Ital Giovanni Porta vytvořil inkoust, resp. roztok z kamence a skalice. Zpráva se jím napsala na vejce, to se pak uvařilo na tvrdo a po oloupání byla zpráva na bílku [17]. V naší době, kdy už máme možnost využívat digitální technologie, získáváme úplně jiný rozměr ukryvání zpráv.

2.2 Využití

Lze ji použít k ochraně citlivých informací (např. ochrana před průmyslovou špionáží) a speciálně na místech, kde se nedá volně použít kryptografie [5, 14]. Další základní oblastí je komerční použití pro ochranu autorských práv [3, 4]. Do stejného vědního oboru, které se stará o ukrytí informace, patří pojmy vodotisk (watermark) a otisk (fingerprint) [3], které nepřímo souvisí se steganografií. Rozdíl mezi nimi je takový, že jeden vodoznak je použit na všechny objekty, oproti otisku, kdežto otisk je jejinečný pro každý objekt zvlášť. Jednotlivá odvětví ukrývání informace lze zhlédnout níže (viz obr. 2.1). Více o vodotisku viz kap. 2.4.

2.3 Steganografie a kryptografie

Zatímco steganografie se zaobírá utajením existence přenášené zprávy, tak kryptografie se zabývá šifrováním zprávy, tj. aby zpráva byla čitelná pouze za určitých podmínek, tj. se znalostí šifrovací klíče a šifrovacího algoritmu.

Samotné použití steganografie přináší bezpečností rizika. Zpráva je sice do určité míry chráněna proti získání přenášených dat, ale už není chráněna proti samotnému přečtení zprávy. Za účelem zvýšení bezpečnosti je vhodné steganografii s kryptografií vzájemně kombinovat. Výsledkem je, že jsou data utajená, ale navíc i nesrozumitelná bez znalosti šifrovacího klíče.

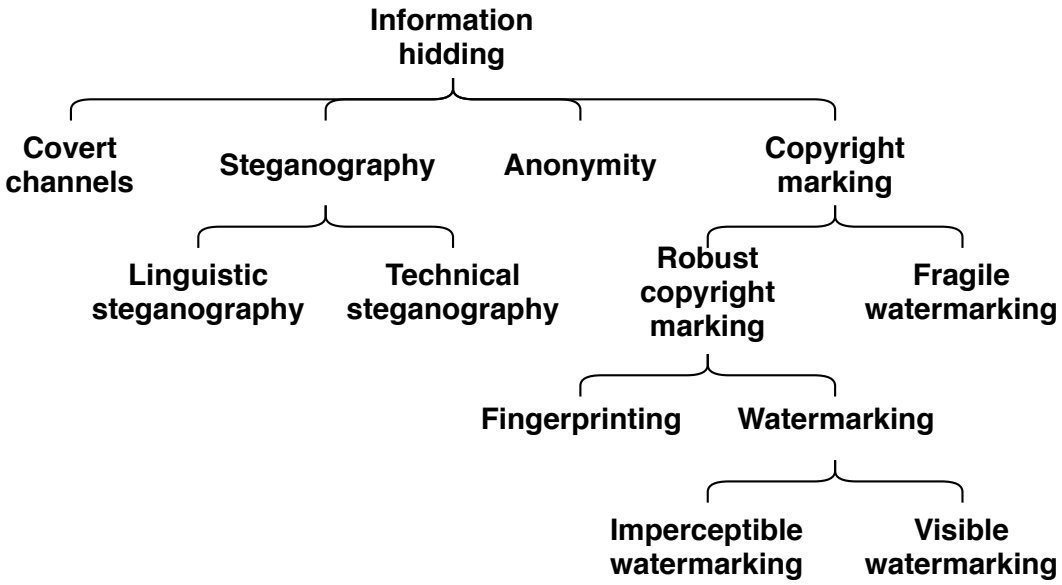
2.4 Vodotisk (Watermark)

Rozdíl mezi vodotiskem a steganografií je ten, že u vodotisku se klade důraz na jeho neodstranitelnost z přenosového média. Oproti tomu je u steganografie cílem přenášená data či informaci plně ukrýt.

Vodotisk se dělí na tři typy (viditelný, neviditelný robustní a neviditelný křehký). Viditelný vodoznak je používán u přenosových médií (např. obrázků), kde chceme, aby bylo vidět, kdo je zdrojem média. Jelikož je tento typ vodoznaku vlastnostní média, tak ho nelze pokládat za steganografii.

Neviditelný robustní vodoznak je určen k odhalení zneužití média a stejně jako u viditelného by měl být schopen odolat vyjmutí nebo případným změnám v médiu, tj. po jeho vyjmutí je nenapravitelně pozměněno i přenosové médium.

Neviditelný křehký vodoznak se může vyskytovat např. na fotografiích pořízených digitálním fotoaparátem a slouží k určení pravosti pořízené fotografie. Používá se např. v publicistice, jako důkaz, že fotografie nebyla pozměněna [10].



Obrázek 2.1: Vizualizace možností ukrytí informace [3]

3 Formát PDF

Zkratka PDF, vycházející z anglického názvu *Portable Document Format*, tj. přenosný formát dokumentu. Slouží k přenášení dat, resp. dokumentů mezi různými platformami, nebo jako nástroj pro tvorbu prezentací. Vychází z PostScriptu a je nezávislý na softwaru, hardwaru i operačním systému. Zajišťuje, aby se obsah souboru typu PDF zobrazil na všech zařízeních beze změny. Podporuje široké množství vkládaných grafických obsahů. Mezi základní patří text a obrázky, ale také i interaktivní formuláře, videa, zvuk a 3D grafika. Roku 2008 byl formát PDF zaveden jako norma ISO 32000-1.

Soubor formátu PDF lze vytvářet a upravovat v komerčních nebo volně dostupných softwarech určených přímo pro práci s PDF, jako jsou např. Adobe Acrobat, ApowerPDF, PDF-XChange Editor nebo webový nástroj Sedja. Dalším způsobem vytvoření PDF je exportování z dokumentů jiného formátu. Často jimi bývají textové dokumenty, jako např. Word nebo LaTeX.

Informace k této kapitole byly čerpány z celkem pěti pramenů [1, 2, 6, 13, 15].

3.1 Historie

Formát PDF byl vytvořen v roce 1993 firmou Adobe a veřejně přístupný je až po zavedení jeho normy ISO 32000-1. S vydáním verze PDF 2.0 firma Adobe zpřístupnila prohlížecí software PDF dokumentů. Díky této distribuci patří dnes mezi nejpoužívanější dokumentové formáty.

3.2 Vlastnosti PDF souborů

Struktura souboru PDF je pevně definována a je tak nezávislá na použitém softwaru, hardwaru i operačním systému. Struktura se do jisté míry podobá PostScriptu, ale nelze použít PostScriptový interpret pro prohlížení PDF souborů.

Do PDF dokumentu lze vkládat kombinace textu, obrázků a grafiky. Mimo těchto pár typických lze vkládat i zvuk, video a hypertextové odkazy.

3.2.1 PDF a PostScript

Jak už bylo zmíněno, tak PDF z PostScriptu vychází, a proto také používá stejné techniky reprezentace dokumentu. Jednotlivé vizuální objekty (písmena, geometrické tvary, obrázky) jsou vykreslovány na definované pozice, přičemž každý objekt je nezávislý na ostatních.

Hlavní rozdíl oproti PostScriptu je, že PDF není programovacím jazykem. Proto neobsahuje žádné procedury ani proměnné a jsou místo toho nahrazeny *operátory*. Ty jsou vykonávány v prohlížečích PDF souborů. Díky tomuto oddělení je urychleno jejich zobrazování a lze snáze vyhledávat text.

3.2.2 Fonty

U mnoha textových dokumentů nám při přenosu na jiné zařízení vzniká problém dostupnosti použitého fontu na obou zařízeních. Pokud se na cílovém zařízení požadovaný font nevyskytuje, tak je většinou nahrazen nějakým příbuzným. To může mít za následek negativní změny v zobrazení. PDF tento problém řeší zavedením *font descriptoru* pro každý použitý font. Obsahuje jméno fontu, velikost a geometrii každého písmene. Ve výsledku, pokud není font v cílovém zařízení, tak se pomocí *font descriptoru* nadefinuje nový font a ten je následně použit.

3.2.3 Přístup k vizuálním objektům

Pro rychlejší vyhledávání stránek (přímým přístupem) je v PDF využívána tabulka křížových odkazů *cross-reference table* (viz kap. 3.4). Ta je umístěna na konci souboru a obsahuje přímé odkazy na umístění všech objektů použitých v PDF souboru.

3.3 Struktura formátu PDF

3.3.1 Datové objekty

Ve většině případů se shodují s datovými typy mnoha programovacích jazyků. PDF formát jich využívá celkem osm (*boolean*, *number*, *string*, *name*, *array*, *dictionary*, *stream* a *null*). Datové objekty lze pojmenovávat a následně se na ně odkazovat z jiných objektů. Takové objekty se nazývají *indirect object* neboli *nepřímý objekt*. Pojmenování objektů je závislé na velikosti písmen (tj. *case sensitive*).

Boolean (Boolovská hodnota) Může nabývat hodnot *true* a *false*.

Number (Číslo) Dělí se na celá (*integer*) a reálná čísla (*real*). Přesnost a rozsah čísel jsou omezeny hardwarem, na kterém aplikace pro spuštění PDF běží.

String (Řetězec) Skládá se ze série bajtů a lze objekty tohoto typu zapisovat dvěma způsoby, a to jako sekvenci literálů uzavřených v kulatých závorkách ( ), nebo v hexadecimálním tvaru (sekvence hexadecimálních číslic) uzavřených v lomených závorkách < >.

Name (Jméno) Používá se k pojmenování objektů. Začíná vždy lomítkem /, které není součástí jména, ale pouze indikuje začátek sekvence znaků představujících jméno. Mezi lomítkem a jménem se nesmí vyskytovat žádné bílé znaky. Rozdíl oproti objektu *string* je ten, že je omezen použitelnými znaky.

Array (Pole) Jedná se o kolekci objektů, která může obsahovat objekty navzájem různých typů včetně objektu array. Pole je ohraničeno hranatými závorkami [ ].

Dictionary (Slovník) Kolekce podobající se pomyslné tabulce, která obsahuje dva sloupce a je ohraničena zdvojenými lomenými závorkami << >>. V každém řádku tabulky je dvojice objektů klíč – hotnota (key – value). Klíč musí být typu *name* a hodnota může být, kteréhokoliv typu včetně *name*.

Stream (Proud) Podobně jako u objektu typu *string* se skládá ze série bajtů. Mezi sebou se liší metodou přístupu. Zatímco *string* se musí přečíst celý najednou, tak objekt *stream* je čten po částech. Díky tomu je vhodný pro ukládání velkého množství dat.

Null (Prázdný objekt) Prázdný objekt nerovnající se typem ani hodnout žádnému jinému objektu.

3.3.2 Objekty v kolekcích

Objekty použité v objektech*array* nebo *dictionary* mohou být zapsány jako přímé objekty nebo jako nepřímý odkaz (*indirect reference*) ukazující na nepřímý objekt (*indirect object*).

3.4 Struktura PDF souboru

Header (Hlavička) Obsahuje verzi formátu PDF, která byla v souboru použita.

Body (Tělo) Skládá se z nepřímých objektů, které udávají vzhled dokumentu. Patří sem např. použité textové fonty, obrázky, ale i samotné stránky atd.

Cross-reference table (Tabulka křížových odkazů) Obsahuje „odkazy“ na nepřímé objekty z těla dokumentu. Ty umožňují snadnější přístup k jednotlivým z nich. Každý nepřímý objekt je v tabulce zapsán na jednom řádku a ta obsahuje informaci, kde je objekt umístěn. Každý PDF dokument obsahuje jednu takovouto tabulku, která se může skladá z jedné nebo více sekcí podle počtu editací dokumentu, tj. podle počtu znovuuložení po nějaké změně (editaci) v dokumentu.

Trailer (Patička) Nalezneme v ní odkaz na tabulku křížových odkazů. Dále obsahuje tzv. *trailer dictionary*, ve kterém jsou umístěny odkazy na důležité objekty a informace v dokumentu.

3.5 Struktura PDF dokumentu (vzhled)

Jsou zde popsány objekty z části dokumentu *body* (viz kap. 3.4) definující vzhled dokumentu. Základní položka, která je určena k definici vzhledu, je objekt s názvem*katalag* (*catalog*). Odkaz na něj nalezneme v části dokumentu *trailer* (viz kap. 3.4).

Objekt *catalog* je typu *dictionary* a lze v něm nalézt několik důležitých údajů spolu s odkazy na kořenové uzly *stromových struktur stránek* (*pages tree*) (viz kap. 3.5.1) a osnovy dokumentu.

3.5.1 Pages tree

Pomocí této struktury se lze dostat k jednotlivým stránkám dokumentu. Každý uzel stromu je typu *dictionary* a může vlastnit potomky jakožto objekty typu *page*. Každý takový objekt definuje vlastnostni jedné stránky dokumentu a obsahuje odkazy na text, obrázky a grafiku stránky.

3.5.2 Objekty v kolekcích

Objekty použité v objektech *array* nebo *dictionary* mohou být zapsány jako přímé objekty nebo jako nepřímý odkaz (*indirect reference*) ukazující na nepřímý objekt (*indirect object*).

4 Knihovny pro práci s PDF

Programátorských knihoven pro práci s PDF dokumenty je spousta, ale mnoho z těch komplexnějších je určeno pouze pro komerční použití a právě z tohoto důvodu nemohly být použity pro vývoj aplikace. Mezi ně patří např.*Foxit PDF SDK*, *Quick PDF Library* atd. Mezi volně dostupné knihovny patří např. *libHaru*, *Apache PDFBox* nebo také *Quick PDF Library Lite*. Dalším kritériem pro výběr knihovny byl programovací jazyk, konkrétně *Java*, kterou všechny výše zmíněné knihovny podporují. Na základě obsáhlé dokumentace, komplexnosti i doporučení byla ve finále pro vývoj použita knihovna *Apache PDFBox*.

4.1 Knihovna Apache PDFBox

Mezi základní funkcionality této knihovny patří vkládání a extrahování textu, obrázků a jiných objektů, spojování nebo rozdělování PDF dokumentů. Umožňuje PDF uložit jako obrázek, případně ho vytisknout pomocí standardní *Java API*. Dále umožňuje vyplňovat nebo extrahovat data z PDF formulářů, vytvářet digitální podpisy atd.

Poskytuje širokou škálu možností práce s textem od extrahování po vložení s libovolnými parametry. Např. umístění na specifickou pozici, nastavení barvy, fontu, řezu písma apod. Při načtení textu z dokumentu lze také získat tzv. vložené fonty neboli *embendded fonts*. Jedná se o fonty, kterými je text v dokumentu reprezentován a lze je následně použít pro zpětné vložení textu do PDF ve stejné formátu, aniž bychom měli dané fonty v počítači či jiném zařízení.

Knihovna podporuje celkem 14 základní fontů, kterými jsou *Times Roman*, *Helvetica* a *Courier*, kde každý z nich je rozdělen na *stadardní*, *tučný*, *s kurzívou* a *tučný s kurzívou*. Zbylými dvěma fonty jsou *Symbol Set* a *Dingbat Typeface*.

5 Stránka a sazba textu

5.1 Popis stránky

Obdélníková plocha za textem, kde okraje obdélníku jsou nejzazší okraje textu, se nazývá *zrcadlo sazby* [8]. *Zrcadlo* je umístěno na stránku s větším spodním okrajem a menším horním okrajem. Je to dáno *optickým středem* stránky, který je výše než *střed geometrický* (protnutí úhlopříček stránky). Ná stránce nesmí zůstávat tzv. *sirotek* a *vdova*, což jsou osamocené řádky [11]. *Sirotek* je poslední řádek odstavce, je umístěn jako počáteční řádek na nové stránce. Oproti tomu *vdova* je první řádek nového odstavce, umístěn jako poslední řádek na stránce.

5.2 Sazba textu

Text máme na stránce umístěn v sazebním zrcadle, který tvoří pomyslný obdelník (box). Stejně můžeme ohraničit jednotlivé řádky textu či samotné znaky, které jsou elementárními boxy. Šířka boxu každého řádku se liší podle použitého typu sázení textu.

Sazba do bloku Řádky jsou stejně dlouhé, roztažené na celou šířku sazebního zrcadla a slova jsou oddělena pružnou mezerou (viz kap. 5.2.1).

Sazba na prapor Řádky jsou zarovnány k jednomu z okrajů sazebního zrcadla a mezera mezi slovy je stejně dlouhá (konstatní).

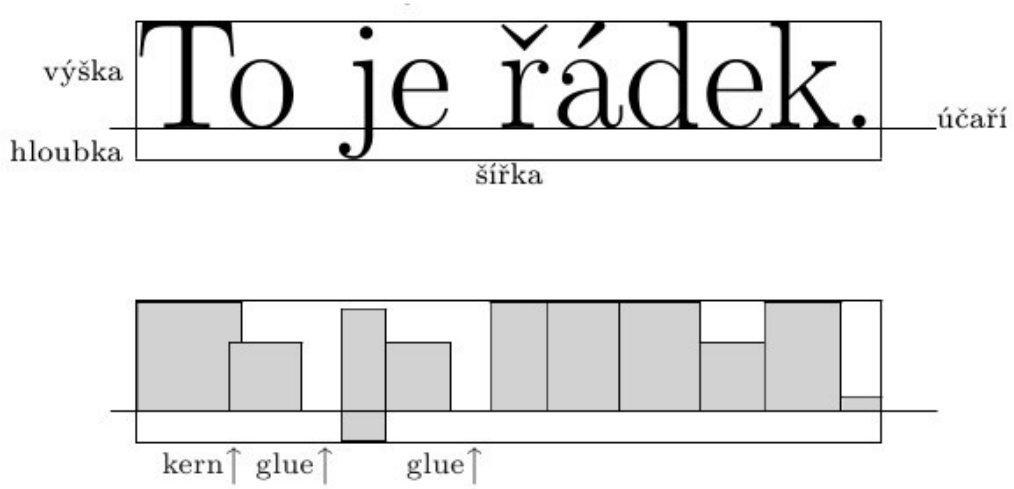
Sazba na střed Podobně jako u sazby na prapor, je mezera mezi slovy stejná, ale řádky jsou umístěny tak, aby vzdálenost od obou okrajů byla stejná.

5.2.1 Pružná mezera

Při *sazbě do bloku* se využívají *pružné mezery* mezi slovy [9]. Mají za následek roztažení řádku na celou šířku *sazebního zrcadla*. Mezery se potom mezi jednotlivými řádky liší.

Dalším obdobným případem pružné mezery je mezera mezi jednotlivými

písmeny ve slovech, kde je cílem, aby se každá mezera jevila opticky stejně. Takový proces se nazývá *kerning* a je dobře patrný např. u písmen „A“ a „V“. Jde o částečné překrytí elementárních boxů písmen, resp. jejich přiblížení až za jejich vzájemnou typografickou hranici. Názornou ukázku pružné mezery a procesu *kerning* si lze prohlédnou níže (viz obr. 5.1).



Obrázek 5.1: Pružná mezera a proces *kerning* [9]

5.3 Použití ve steganografii

Pro zakódování tajné zprávy do textu, je potřeba upravovat sazební nebo grafické parametry textu, případně kombinaci obou. Při zvolení si nějakého parametru je potřeba určit hodnoty, které jsou určeny pro přenos tajné informace a které nesou informaci původního textu. Mezi sazební parametry, které se starají o polohu na stránce, patří např. pozice, rotace, velikost znaků atd. Grafické parametry jsou určeny spíše pro vizuální dojem textu. Mezi ně patří barva, font a řez znaků.

Je nutné, aby hodnoty parametrů pro zakódování zprávy byly nastaveny co nejblíže situaci, kdy lidské oko nepozná rozdíl mezi normální textem a textem s tajnou informací. Také je potřeba brát v úvahu schopnosti dekódovacích mechanismů, prostředků a množství informace, aby se tajná zpráva z textu dala získat.

Jak bylo zmíněno, tak pro zakódování informace můžeme sazební i grafické

parametry kombinovat nebo také lze použít jeden parametr, ale zvolit více hodnot pro přenos tajné informace. Je s tím úzce spjatý pojem *úroveň kódování* (viz kap. 5.3.1). Docílí se tím uložení většíno množství tajných dat na menším prostoru, ale také se tím zvyšuje nápadnost, že nějaká tajná komunikace probíhá. U rozsáhlejších kombinování či použití více hodnot je nutné si vést kódovací tabulku (viz kap. 6).

5.3.1 Úroveň kódování

Pod tímto pojmem se ukrývá číslo, které se získá součtem počtů (větších než jedna) všech použitých hodnot sazebních a grafických parametrů, které byly použity pro zakódování informace (viz vzorec 5.1).

$$x = \sum_{m=1}^{M} a_m + \sum_{n=1}^{N} b_n \tag{5.1}$$

a_m – počet použitých hodnot m-tého sazebního parametru
b_n – počet použitých hodnot n-tého grafického parametru
M – celkový počet sazebních parametrů
N – celkový počet grafických parametrů

Pokud toto číslo dosadíme jako exponent čísla dvě (binární soustava), tak získáme počet všech použitelných variací. Jinak řečeno je tímto číslem dáno, kolik lze použít bitů pro zakódování informace v jednom kroku (např. vysázení jednoho písmene s kódovacími parametry). Vzniká nám však současně omezení, týkající se bezpečnosti dekódování. Jelikož se o dekódování stará rozpoznávací algoritmus a do přeneseného textu může vstupovat nějaká forma šumu, tak algoritmus kódová slova může omylem zaměnit a pozměnit tak výslednou zprávu. S tímto souvisí tzv. vzdálenost kódových slov (viz kap. 7.2). Jelikož se pro kódování snažíme sazební i grafické parametry co nejvíce napodobit, tak v kánále se šumem se díky tomu zvyšuje šance na výskyt chyby, tj. změnu nějakého bitu či bitů. Pokud ale zvětšíme vzdálenost mezi kódovýmislovy, tak se tím sníží i šance na chybu. Na druhou stranu se také snižuje množství kódových slov, které lze použít pro kódování.

6 Kódování a dekódování

6.1 Abeceda

Konečná neprázdná množina, jejíž prvky se nazývají písmena a řetězcem resp. slovem označíme libovolnou konečnou posloupnost písmen abecedy.

Tato množina nám definuje, jaké znaky jsou podporovány v informačním zdroji a kanálu.

6.2 Kódovací tabulka

Jedná se o množinu dvojic, které slouží k převodu z abecedy *zdroje informace* na abecedu *přenosového kanálu* (proces kódování), případně z abecedy *přenosového kanálu* na abecedu *příjemce informace* (proces dekódování). Sloupce této tabulky také můžeme pojmenovat jako *klíč* a *hodnota*. U dekódování se *hodnota* stává *klíčem* a *klíč hodnotou*.

6.3 Proces kódování a dekódování

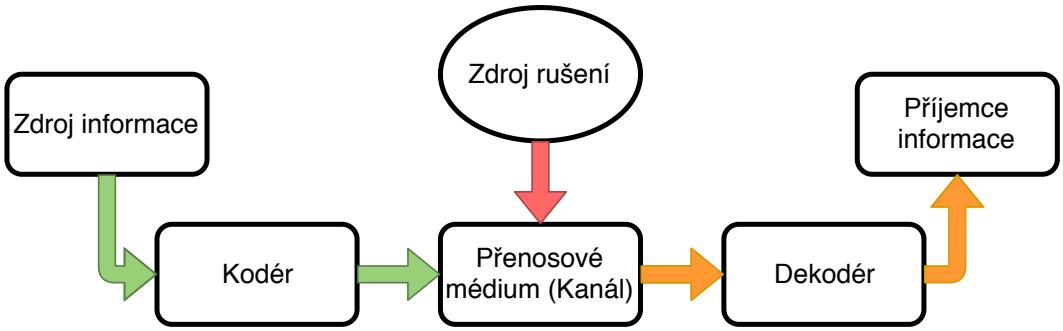
Kódováním je myšleno převedení dat, získaných ze *zdroje informace* s nějakou abecedou (viz kap. 6.1), na data složená z abecedy *přenosového kanálu*. Opačným procesem se nazývá dekódování, kde se data z *přenosového kanálu* převádějí na data určená příjemci (viz obr. 6.1). Výsledkem přenosu zprávy odeslané ze *zdroje informace* skrze procesy kódování a dekódování, je stejná zpráva přijmutá u *příjemce informace*.

Během procesů se používá předem dohodnutá *znaková sada* neboli *kódovací tabulka* (viz kap. 6.2). Speciálním případem kódování je kryptografie, kde obsah zakódované informace je určen pouze příjemcům, kteří vlastní dekódovací klíč a znají kódovací algoritmus.

V případě této práce je *zdrojem informace* uživatelská zpráva a *přenosovým médiem* je text v PDF dokumentu. Dokument je poslán do *přenosového kanálu*, což je jeho vytisknutí a pořízení fotografie kódovaného textu. Poté je zpráva dekódována z fotografie rozpoznávacím algoritmem.

6.4 Přenos dat

Komunikační kanál, kterým je informace přenášena, je v reálném světě často ovlivňován nežádoucími vlivy neboli šumy. Ty jsou reprezentovány *zdrojem rušení* (viz obr. 6.1) a mají za následek nežádoucí změny v datech jdoucích do *dekodéru*. Nechtěným výsledkem jsou nečitelná data u *příjemce informace*. Pro omezení těchto vlivů se často v *kodéru* přidávají redundantní informace za účelem zvýšení odolnosti dat proti těmto vlivům (viz kap. 7).



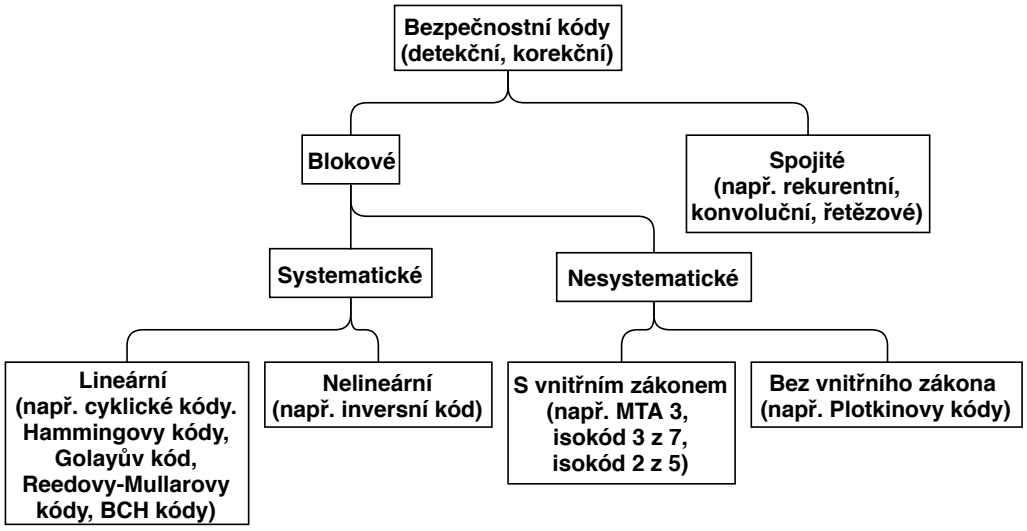
Obrázek 6.1: Diagram procesu kódování a dekódování

7 Bezpečnost kódování

Bezpečnostní mechanismy obecně fungují na bázi přídávání redundantní informace do přenášených dat. Cílem použití bezpečnostních kódů je v lepším případě opravování chyb během přenosu dat nebo alespoň jejich detekování. S tím souvisí *Hammingova vzdálenost* mezi kódovými slovy(viz kap. 7.2), a to tím způsobem, že čím větší je vzdálenost,tím lze detekovat, resp. opravit vícenásobné chyby.

7.1 Dělení bezpečnostních kódů

Bezpečnostní kódy jsou rozděleny do několika skupin (viz obr. 7.1). V dnešní době se hledá ideální kompromis mezi jednoduchostí a stupněm zabezpečení pro detekci, resp. opravu chyb.



Obrázek 7.1: Rozdělení bezpečnostních kódů [7]

7.1.1 Blokové a spojité kódy

U blokových kódů, které se dále dělí na systematické a nesystematické, jsou přenášená binární data rozdělena na bloky, které jsou navzájem nezávislé. Oproti tomu jsou u spojitých kódů datové sekvence bitů posílány na vstup zabezpečovacího kodéru, který na výstupu vrací zabezpečenou sekvenci.

Systematické U této skupiny kódů je rozdělení datových a bezpečnostních prvků v kódové kombinaci dáno podle stanoveného systému, tj. lze přímo určit, jaké prvky v přenášené kódové kombinaci jsou datové a jaké bezpečnostní. Obsahuje-li systematický kód n prvků, tak lze tento počet rozdělit na k informačních a $n - k$ bezpečnostních prvků. Potom lze mluvit o (n, k) kódech. Systematické kódy lze dále dělit na lineární a nelineární. *Lineární kódy* mají tu vlastnost, že výběrem libovolné lineární kombinace kódových prvků, dokážeme získat znovu kódový prvek. Tato vlastnost je dána matematickým základem lineární algebry. *Lineární kódy* se mohou objevovat ve dvou variantách. Prvním typem je posílání informačních a následně zabezpečovacích prvků přenosovým kanálem. Druhou variantou je např. Hammingův kód (viz kap. 7.3), kde jsou informační a bezpečnostní prvky seřazeny specifickým způsobem. *Nelineární kódy* se charakteristikou *lineárních kódů* nevyznačují. Tyto kódy se, jak v definici, tak v jejich vzniku, velmi liší, a proto jsou většinou definovány kódovací tabulkou, obsahující veškeré informační prvky a jejich reprezentující kódové kombinace.

Nesystematické Tato skupina má redundantní informaci rovnoměrně, ale nestejně rozmístěnou ve všech prvcích přenášeného bloku. Dělí se na kódy s/bez vnitřního zákona. *kódů s vnitřním zákonem* se při dekódování ověřuje splnění předem daných zákonů (pravidel), podle kterých se kodové kombinace vytvářejí. Jako příklad lze uvést isokódy k z n, které se vyznačují konstantním počtem nul a jedniček v kódovém slově. **Kódům bez vnitřního zákona** lze přiřadit všechny dvojkové kódy, jejichž Hammigova vzdálenost (viz kap. 7.2) všech kódových kombinací je větší než 1, tj. $d_{min} > 1$. Správnost přijaté informace můžeme ověřit pouze kódovací tabulkou.

7.2 Hammingova vzdálenost

Vzdálenost udávající míru odlišnosti dvou libovolných kódových slov neboli kolik je potřeba změn znaků jednoho slova pro převod na slovo druhé. U binárních kódových slov je tato vzdálenost počet odlišných bitů na stejných pozicích.

$$d_H(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{n} |a_i - b_i| \tag{7.1}$$

a_i – prvky kódového slova $\mathbf{a}$

b_i – prvky kódového slova $\mathbf{b}$

Kód detekuje t-násobné chyby právě tehdy, když je Hammingova vzdále nost kódu $d > t$, a opravuje, když je $d > 2t$.

7.3 Základní bezpečnostní kódy

Paritní kód Za nejzákladnější bezpečnostní kód lze považovat paritní kód nebo také paritní bit, který slouží pouze k detekci chyby. Jedná se o redundatní bit, připojený k přenášenému bloku dat, který je vyplněn 0 nebo 1 podle typu parity. Parita se dělí na lichou a sudou podle počtu jedniček v kódovém slově.

Např. $1101 + P$, kde P značí paritní bit a „1101“ je zpráva pro příjemce. Pokud zpráva bude zabezpečena lichou paritou, tak bude P nahrazeno 0 a kódová kombinace je 11010. V případě sudé parity bude P změněno na 1 a výsledná kódová kombinace bude 11011.

Opakovací kód funguje na bázi n-krát zopakovaného kódového slova. Při jatá data jsou porovnána s kódovými kombinacemi opakovacího kódu tak, že se zjistí Hammingova vzdálenost mezi přijatými daty a kódovými kombinacemi. Pokud vyjde Hammingova vzdálenost rovna nule, tak chyba nenastala nebo nebyla detekována. V opačném případě se hledá nejkratší vzdálenost a kódová kombinace, ke které jsou přijatá data nejblíže, je jejich kódovou kombinací. Pokud známe i spolehlivost přenosového kanálu, tak lze odvodit, že daná kódová kombinace byla vyslána pravděpodobněji než kterákoliv jiná.

Např. máme množinu kódových slov $X = \{000, 010, 101, 111\}$ a po aplikaci opakovacího kódu s 3násobným opakováním získáme výsledný bezpečnostní kód $Y = \{000000000, 010010010, 101101101, 111111111\}$. V tomto případě lze detekovat 2násobné chyby a opravit 1násobné.

Hammingův kód Jedná se o lineární kód, který dokáže detekovat 2násobné a opravit 1násobné chyby. Pro názornost je použit Hammingův kód $(7, 4)$, který obsahuje celkem 7 bitů, z toho 4 informační a 3 zabezpečovací. Pozice zabezpečovacích bitů jsou dány binární soustavou a řádky, které obsahují jednu 1, tj. mocniny dvou (2^0, 2^1, ..., 2^n). (viz vzorec 7.2).

$$
\begin{aligned}
p_1 &: 1 = 001 \\
p_2 &: 2 = 010 \\
a_1 &: 3 = 011 \\
p_3 &: 4 = 100 \\
a_2 &: 5 = 101 \\
a_3 &: 6 = 110 \\
a_4 &: 7 = 111
\end{aligned}
\tag{7.2}
$$

Prvky p_i značí paritní bity a prvky a_j informační bity. Jednotlivé paritní bity jsou dopočítány podle umístění jedničky jejich pozic v binární soustavě.

První paritní bit p_1 je dopočítán z řádek binární soustavy, které mají 1 na pozici prvního bitu. Rovnice pro výpočet je tedy

$$p_1 + a_1 + a_2 + a_4 = 0 \text{ neboli } p_1 = a_1 + a_2 + a_4.$$

Druhý paritní bit p_2 je vypočítán podobně, ale jsou použity řádky binární soustavy s 1 na pozici druhého bitu. To znamená, že rovnice je

$$p_2 + a_1 + a_3 + a_4 = 0 \text{ neboli } p_1 = a_1 + a_3 + a_4.$$

Nakonec poslední paritní bit p_3 získáme obdobnou rovnicí, ale jsou použity řádky s 1 na místě třetího bitu. Výsledek je rovnice

$$p_3 + a_2 + a_3 + a_4 = 0 \text{ neboli } p_3 = a_2 + a_3 + a_4.$$

Např. pro kódové slovo 1101 (a_1, a_2, a_3, a_4) je výsledek aplikace Hammingova kódu

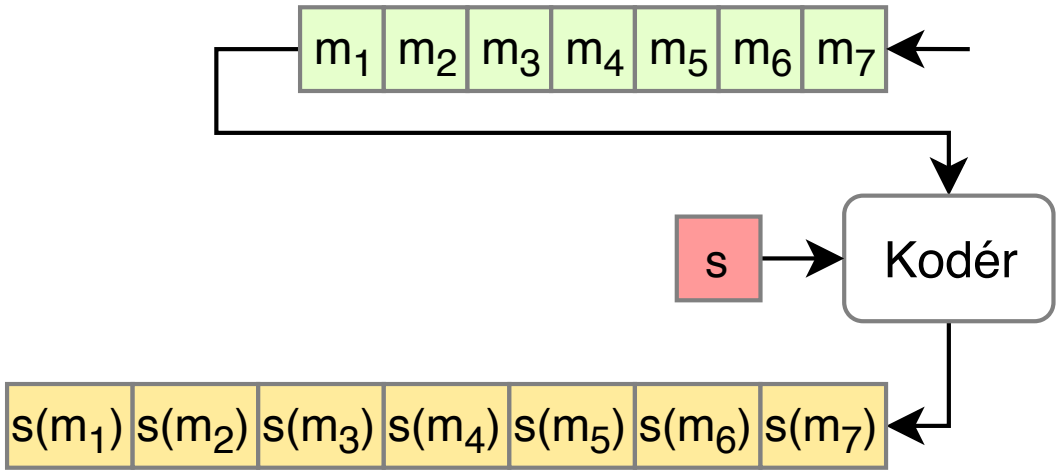
$$
\begin{aligned}
p_1 = 1 \quad &= a_1 + a_2 + a_4 = 1 + 1 + 1 \\
p_2 = 0 \quad &= a_1 + a_3 + a_4 = 1 + 0 + 1 \\
a_1 &= 1 \\
p_3 = 0 \quad &= a_2 + a_3 + a_4 = 1 + 0 + 1 \\
a_2 &= 1 \\
a_3 &= 0 \\
a_4 &= 1
\end{aligned}
$$

8 Popis implementace

Pro testování implementovaných částí aplikace, byly vygenerovány a použity PDF dokumenty, získané exportováním z programu *Microsoft Word 2007*. Dalšími testovacími subjekty byly vygenerované dokumenty pomocí webového nástroje *Google Docs*.

8.1 Popis zpracování zprávy od uživatele

Zpráva od uživatele je při zpracování rozdělována na bloky pevné délky, kterou si uživatel může sám zvolit. Každý blok je poté převeden do binární reprezentace, pokud je to potřeba (viz kap. 8.1.1), a je na něj aplikován zabezpečovací kód, pokud byl nějaký zvolen (viz kap. 8.1.2). Upravené bloky zprávy se poté sekvenčně ukládají do odstavců textu, které si uživatel vybere. Zabezpečená zpráva se do daného odstavce vkládá opakováně, dokud se nevyčerpá místo pro vkládání bitů, přičemž se do odstavce musí vejít alespoň jedenkrát. Níže je uveden popis procesu zabezpečení zprávy (viz obr. 8.1), zelené bloky m_1 až m_7 jsou jednotlivé bloky zprávy, procházející kodérem, kde je na ně aplikována zabezpečovací funkce s.



Obrázek 8.1: Proces zabezpečení zprávy po blocích.

8.1.1 Kódovací abecedy

Jak bylo již zmíněno, tak program vkládá ukrývanou informaci do textu v dokumentech formátu PDF. V důsledku toho jsou v rámci implementace

vloženy podpory dvou kódových abeced. První abecedou je klasická binární reprezentace, kdy uživatel píše binární posloupnost bitů a druhou je abeceda, která se podobá kódování MTA 2 (*Mezinárodní telegrafní abeceda č. 2*). Má pevně definovanou délku kódového slova o šesti bitech pro jeden znak. Jednotlivé kombinace mají mezi sebou navzájem Hammingovu vzdálenost rovnu 2. To znamená, že

$$d(s_i, s_j) = 2; \forall i, j; i \neq j, \text{kde } s_i \text{ a } s_j \text{ jsou libovolná kódová slova.}$$

Ve výsledku má daná abeceda 32 kódových slov a tvoří tak kódovací tabulku.

'a' = 000001	'j' = 010011	's' = 100101
'b' = 000010	'k' = 010101	't' = 100110
'c' = 000100	'l' = 010110	'u' = 101001
'd' = 000111	'm' = 011001	'v' = 101010
'e' = 001000	'n' = 011010	'w' = 101100
'f' = 001011	'o' = 011100	'x' = 101111
'g' = 001101	'p' = 011111	'y' = 110001
'h' = 001110	'q' = 100000	'z' = 110010
'i' = 010000	'r' = 100011	

'0' = 'd' = 000111	'5' = 'k' = 010101
'1' = 'f' = 001011	'6' = 'l' = 010110
'2' = 'g' = 001101	'7' = 'm' = 011001
'3' = 'h' = 001110	'8' = 'n' = 011010
'4' = 'j' = 010011	'9' = 'o' = 011100

'SPACE' = 110100
'LS' = 110111
'FS' = 111000

Abecedu lze rozdělit na tři části. První skupina se skládá z 26 písmen anglické abecedy. Z toho plyne, že české znaky nejsou, pro zakódování do textu, podporovány. Další skupinou jsou číslice, které mají stejná kódová slova jako vybrané znaky z předchozí skupiny. Tím se dostáváme k poslední části, obsahující šest kódových slov, kde tři z nich nejsou využívány. Zbylé tři jsou stejné pro první i druhou skupinu. Jedno kódové slovo reprezentuje mezeru

a zbylé dvě jsou určeny pro změnu mezi písmennou a číslicovou abecedou, tj. LS (písmenná změna, *Letter Shift*) a FS (číslicová změna, *Figure Shift*). Zbylé existující znaky, které nejsou v abecedě uvedeny, a tudíž nejsou ani podporovány programem, jsou nahrazeny mezerami.

Změny LS a FS se do zprávy přidávají během jejího zadávání od uživatele. Sekvenčně se čtou znaky zprávy a mezi každé dva, které jsou navzájem z různých abeced,se vloží LS, případně FS podle druhého znaku. Postupné přidávání těchto speciálních znaků, je z důvodu, aby bylo možné zjistit momentální velikost zprávy v bitech a bylo tak možné určit zda se zpráva vejde do vybraného odstavce.

Např. zpráva od uživatele „Příklad změny 1 abecedy na 2.“ se bude do PDF ukládat jako „Příklad změny 'FS'1 'LS'abecedy na 'FS'2 “, kde jednotlivé změny ('LS' a 'FS') značí jeden speciální znak.

Použití dvou abeced se stejnými kódovými slovy pro písmena i číslice a přepínání mezi nimi, je z důvodu použití optimálního množství bitů na jeden znak. Omezením na pět a méně bitů každého kódového slova, by se nedocílilo zmíněnné Hammingovy vzdálenosti, a v případě zvýšení počtu bitů pro reprezentaci každé abecedy zvlášť by se v některých případech mohla výrazně zvětšit délka zprávy se zabezpečením. Např. zvýšení počtu bitů ze šesti na sedm a aplikování opakovacího kódu s třemi opakováními, je potřeba o tři bity pro zakódování více na jeden znak.

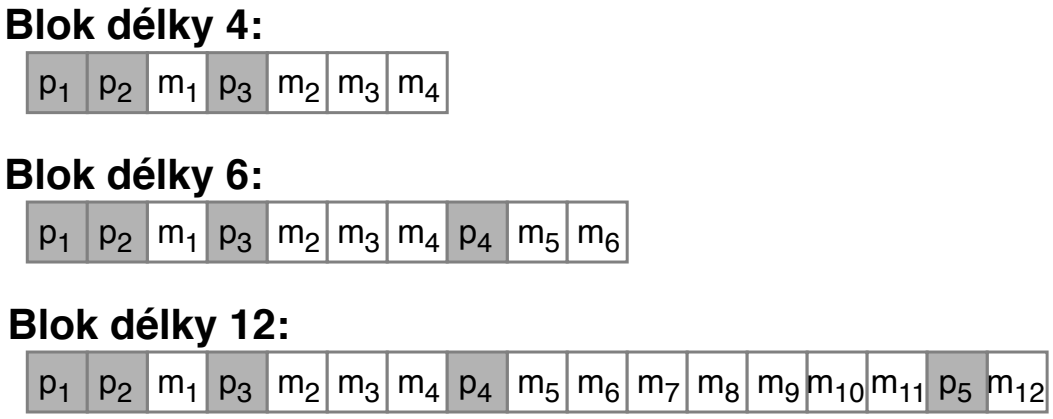
8.1.2 Bezpečnostní kódy

V programu je implementována sada bezpečnostních kódů, skládající se z liché a sudé parity, opakovacího kódu (2×, 3× a 4×) a univerzálního Hammingova kódů, přičemž si uživatel může zvolit mezi těmito zmíněnými nebo nevolit žádný.

Hammingův kód je univerzální ve smyslu práce s blokem různé délky. Obecně se používá Hammingův kód $(7, 4)$, $(15, 11)$ atd., ale lze ho použít na libovolnout délku informačních bitů. Příklad je uveden níže (viz obr. 8.2).

8.2 Algoritmus kódování

Pro zakódování zprávy do textu je použita modifikace mezery mezi slovy, kde bitová 0 je reprezentovaná standardní mezerou fontu *Helvetica* a pro bi-



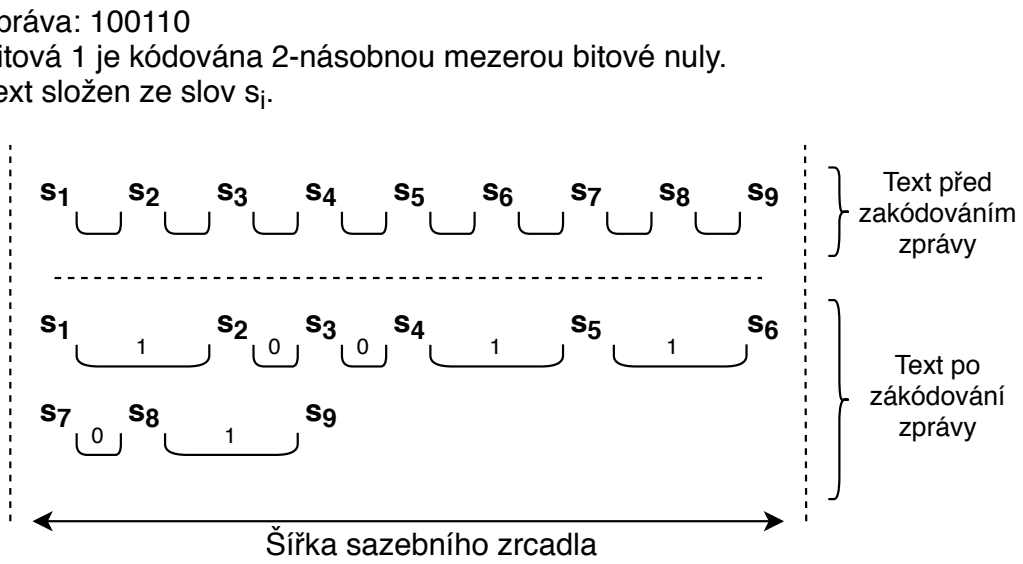
Obrázek 8.2: Příklad univerzálnosti Hammingova kódu.

tovou 1 je použita stejná mezera, která je procentuálně rozšiřována. Po zpracování načteného PDF dokumentu, tj. extrahování textu (viz kap. 8.4.1) a obrázků (viz kap. 8.4.2), je načtený text rozdělen na jednotlivá slova a každé mezeře mezi slovy je přiřazena bitová hodnota výsledné zprávy s nebo bez zabezpečení. Ve výsledku je kódovaná zpráva ukládána do jednotlivých odstavců nezávisle (viz kap. 8.4.1) a v případě většího roztažení některé řádky kvůli většímu počtu mezer, reprezentujících bitovou 1, se slova zalamují na dalším řádku a s tím se také přesouvá i pokračování kódované zprávy. Pokud je zpráva bitově kratší než počet mezer daného odstavce, tak se zpráva opakuje bez ohledu na to, zda se opakovaná zpráva vejde či nikoliv. Příklad procesu zakódování zprávy lze vidět níže (viz obr. 8.3), kde p_i jsou paritní bity a m_j značí informační bity.

8.3 Algoritmus dekódování

Implementovaný dekódovací algoritmus je navržen tak, aby dokázel detekovat a dekódovat tajnou zprávu z obrázku vytištěného dokumentu se šumem. Šum může nastat během tisku daného dokumentu, případně při jeho vyfocení.

Program v režimu dekódování má tedy na vstupu fotografii, která musí mít pevně definovanou strukturu. To znamená, že fotografie obsahuje odstavec, kde je údajně zpráva zakódována a v ideálním případě je fotografie v dokonalé rovině nebo alespoň tak, aby bylo možné protnout obrázek přímkou mezi řádky bez kolize se znaky.



Obrázek 8.3: Příklad procesu zakódování binární informace „100110“ do textu.

Proces dekódování je postaven na ohraničení jednotlivých písmen, případně celých slov podle kvality fotografie. Nejdříve se všechny pixely v obrázku, nezávisle na jiných, přebarví na bílou resp. černou barvu podle jejich odstínu šedi podle vzorce níže (viz vzorec 8.1).

$$\begin{aligned} x &= \sqrt{(255-r)^2 + (255-g)^2 + (255-b)^2} \\ y &= \frac{x}{\sqrt{3(255^2)}} \end{aligned} \tag{8.1}$$

r – červená složka pixelu

g – zelená složka pixelu

b – modrá složka pixelu

Výsledek rovnice y je procentuální hodnota a z ní je zjišťováno zda bude pixel přebarven na bílou, případně černou barvu. Hranice přebarvení je řízena uživatelem. Pokud je výsledek rovnice pod zadanou hranicí, tak je pixel přebarven na bílou a v opačném případě na černou. Následně se obrázek protká sítí, která rozdělí odstavec na jednotlivé řádky horizontálním protnutím (nad a pod každým řádkem) a na jednotlivé znaky vertikálním protnutím (vlevo a vpravo každého znaku) v každém řádku zvlášť. Jakmile je rozdělení odstavce na písmena dokončeno, tak jsou řádky zpracovávány samostatně. Vždy se najdou dva prázdné (bíle) obdelníky neboli mezery, které mají minimální a maximální obsah v pixelech. Obsahy obdelníků jsou poté přepočítány na procenta a tvoří hranice intervalu 0 až 100%. Z tohoto intervalu jsou

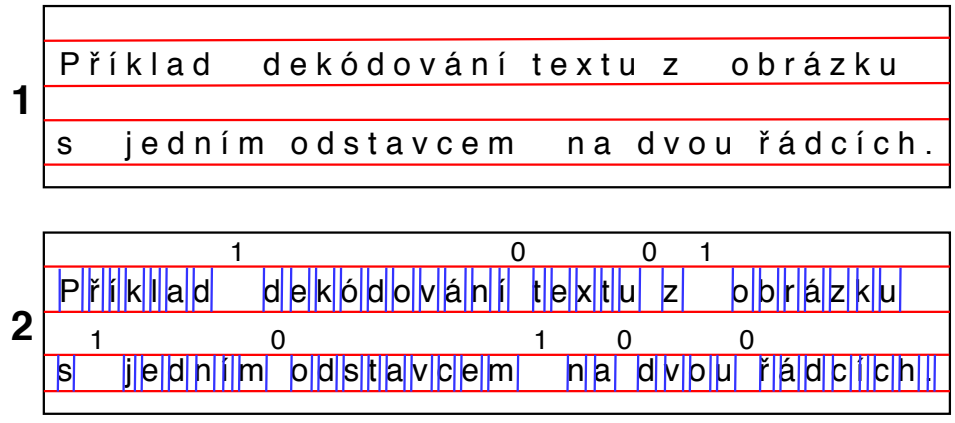
pak všem obdelníkům (stejným způsobem přepočítání) na řádku přiřazeny hodnoty „mezera mezi znaky“, „bitová 0“ nebo „bitová 1“.

V intervalu jsou dva parametry, které jsou v aplikaci uživatelem manipulovatelné a udávají, jak velká část intervalu resp. jak velká mezera je brána jako meziznaková, mezislovní reprezentující 0 nebo mezislovní reprezentující 1. Interval je popsaný níže (viz 8.2).

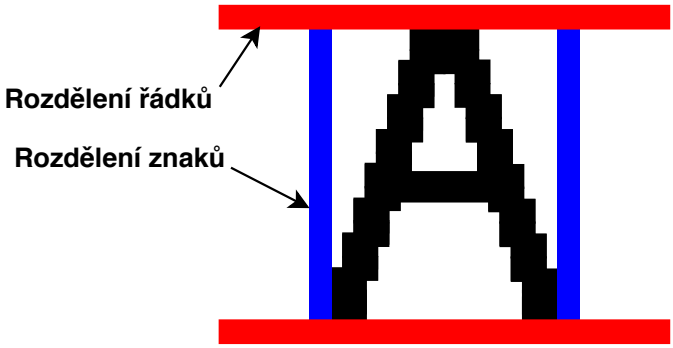
$$[0; a), [a; b], (b; 100] \tag{8.2}$$

a – hranice mezi meziznakovou a mezislovní mezerou (0)
b – hranice mezi mezislovními mezerami (0) a (1)
$[0; a)$ – interval meziznakové mezery
$[a; b]$ – interval mezislovní mezery (0)
$(b; 100]$ – interval mezislovní mezery (1)

V příkladu (viz obr. 8.4), který slouží pouze jako ukázka rozdělování na písmena, je v první části červenými horizontálními čárami ukázáno rozdělení na jednotlivé řádky. Podobně je tomu tak ve druhé části obrázku, kde jsou jednotlivé řádky rozděleny na samostatné znaky vertikálními modrými čárami. V reálném procesu se čáry odsazují o jeden pixel od černého, vždy co nejblíže znaku (viz obr. 8.5). Po dokončení sítě se u každého řádku vypočítávají obsahy mezer a následně na to se extrahuje ukrytá zpráva, metodou která již byla zmíněna.



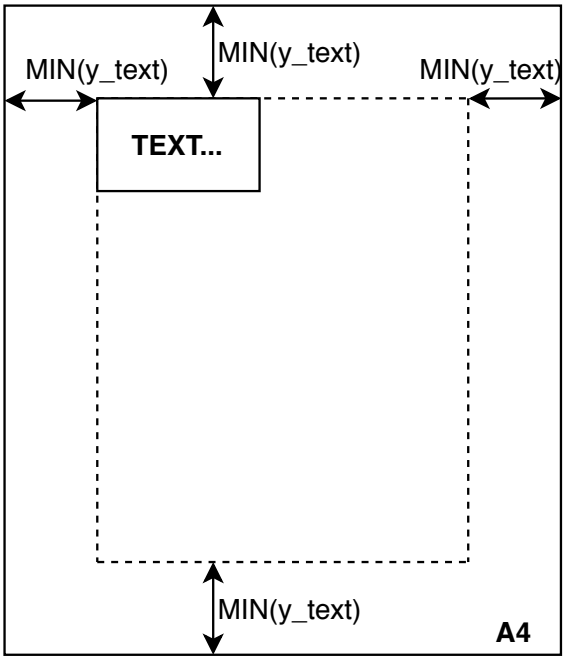
Obrázek 8.4: Příklad procesu dékódování informace „100110“ z obrázku.



Obrázek 8.5: Přiblížený proces dekódování na znaku „A“

8.4 Práce s PDF dokumentem

Pro zakódvání zprávy, je vytvořen nový PDF dokument s formátem stránek A4 a je pro něj zjištěno sazební zrcadlo (viz obr. 8.6), které je odvozeno od nejmenší X-ové souřadnice jako levé odsazení, které je stejné i pro pravou stranu. Stejně to platí i pro horní a dolní odsazení, které je získáno podle nejmenší Y-ové souřadnice.



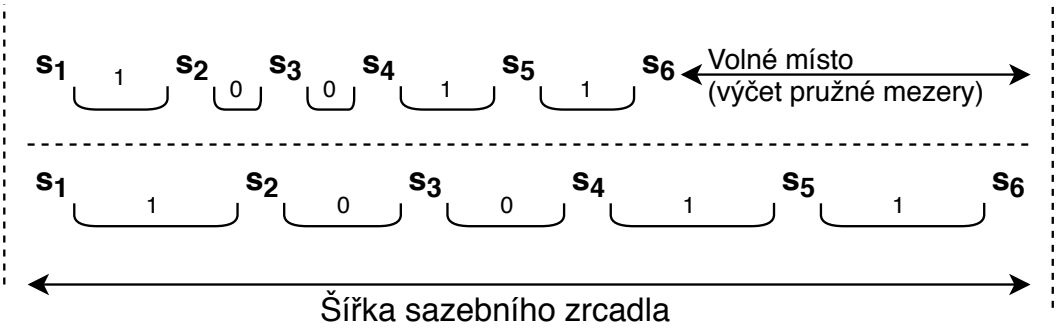
Obrázek 8.6: Sazební zrcadlo pro sazbu textu a obrázků

8.4.1 Extrahování a importování textu

Text je z dokumentu čten po znacích, jejichž data o poloze, fontu a typu znaku jsou uloženy jako instance třídy *TextPosition* z knihovny PDFBox. Nevýhodou této třídy je ta, že neumožňuje zjistit stránku, na které se text nachází. Po získání textu se jednotlivé znaky rozdělují na nadpisy a odstavce podle velikosti písma. Text pro odstavce je klasifikován podle nejmenší nalezené velikosti písma v dokumentu a nadpisy se rozpoznávají podle zbýlých velikost, tj. všech vyšších. U nadpisů si program pamatuje levé a horní odsazení a zbylé parametry jako jsou souřadnice se dopočítávají během sázení (importování) textu. Odstavce si pamatují pouze horní odsazení od předešlého textu a mezery mezi řádky. Text klasifikován jako odstavcový je poté sázen (importován) do celé šířky sazebního zrcadla, přičemž je podporováno pouze psání v jednom sloupci, tj. jedno sazební zrcadlo na stránce.

Při sazbě textu odstavců si uživatel může volit mezi *sazbou na prapor* nebo *sazbou do bloku*. U *sazby do bloku* jsou šířky mezer vypočítávany na každém řádku zvlášť. Na ukázce níže lze vidět příklad dopočítání průžné mezery jednoho řádku (viz obr. 8.7).

Výpočet začíná sečtením šířek slov a všech nulových i jedničkových mezer dohromady. Součet se pak odečte od sazebního zrcadla a zbyde místo pro rozšíření pružných mezer, tj. mezery mezi slovy. Volné místo se vydělí počtem všech mezer, které se poté přičte ke každé z nich.



Obrázek 8.7: Příklad roztažení pružných mezer v řádku do bloku

8.4.2 Extrahování obrázků

Obrázky se získávají nezávisle na extrahování textu, tj. v jiný časový okamžik. Obrázky jsou uloženy jako instance třídy *PDImageXObject* z knihovny

PDFBox. Tato třída obsahuje pouze informace o obrázku jako samotstatném objektu, ale už neobsahuje číslo stránky, na které byl umístěn, a stejně tak ani souřadnice, z kterých bych extrahován. Tyto informace jsou získávány dodatečně pomocí oddděné metody *processOperator* třídy *PDFStreamEngine*, také z knihovny PDFBox.

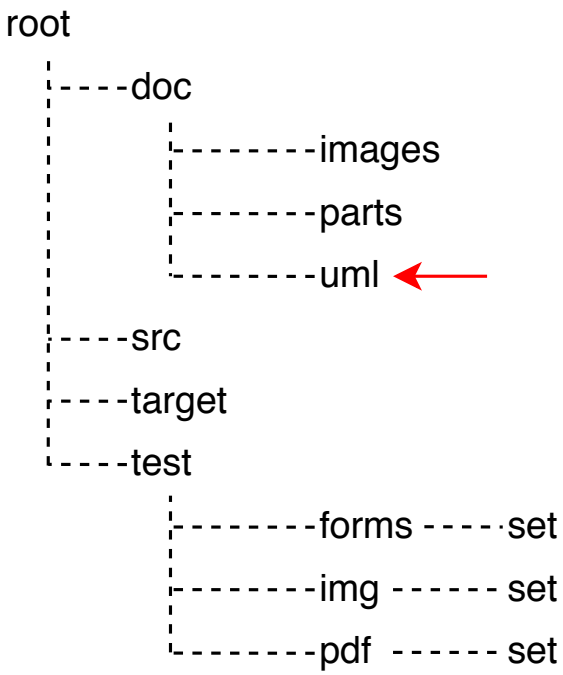
Zpětné vkládání obrázků je kvůli nezávislému extrahování nepřesné, jelikož nelze sjednotit souřadnice a stránky, na kterých se s textem nacházejí.

9 Uživatelská příručka

9.1 Kód programu

Jednotlivé návaznosti tříd jsou popsány UML diagramy, které jsou umístěny v adresáři `/doc/uml` na CD (viz obr. 9.1). Adresář obsahuje UML diagramy ve stejnojmenných souborech ve dvou různých formátech *.uxf* a *.pdf*.

Popis jednotlivých tříd je umístěn v *JavaDoc* dokumentacích ve zdrojových souborech.



Obrázek 9.1: Adresářová struktura

9.2 Překlad a spustění aplikace

Pro překlad aplikace je potřeba mít nainstalovaný software *Java SE Development Kit 8* a nástroj pro správu aplikací *Apache Maven*. Aplikace je poté přeložitelná z příkazového řádku, otevřeného v kořenovém adresáři aplikace, příkazem `mvn clean install`. V adresáři *target* se po přeložení nacházejí

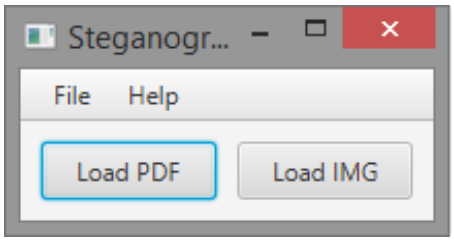
dva spustitelné soubory s příponou *.jar*. První se nazývá *text_steganography-1.0.0-SNAPSHOT.jar*, který je generován automaticky nástrojem *Maven*, ale neobsahuje závislosti na externí knihovny a je tudíž nefunkční. Druhý se jmenuje *text_steganography-1.0.0-SNAPSHOT-jar-with-dependencies.jar*, který už závislosti obsahuje.

Spuštění aplikace se docílí tak, že je druhý JAR soubor spuštěn dvojklikem. Pokud to operační systém nebo systemové nastavení neumožňuje, tak ho lze spustit z příkazového řádku příkazem:

```
java -jar text_steganography-1.0.0-SNAPSHOT-jar-with-dependencies.jar
```

Vstupní bodem aplikace je okno (viz obr. 9.2), které slouží jako počáteční rozcestník mezi režimy. Obsahuje dvě tlačítka, kde první *Load PDF* slouží pro načtení PDF dokumentu a přechod do režimu kódování. Druhé tlačítko *Load IMG* slouží pro načtení obrázku a přechod do režimu dekódování.

Všechna okna ve spuštěné aplikaci sdílejí stejnou hlavičku, která obsahuje dvě menu. První je menu *File*, které obsahuje celkem tři položky (tlačítka) *Load PDF* a *Load IMG*, které jsou totožné s předchozími zmínenými a *Exit* pro ukončení aplikace. Druhé menu se nazývá *Help*, které obsahuje jednu položku *About*, pomocí níž lze získat informace o programu.



Obrázek 9.2: Úvodní okno aplikace

9.3 Režim zakódování zprávy

Po načtení PDF dokumentu tlačítkem *Load PDF* se objeví okno (viz obr. 9.3), ve kterém se nastavují kódovací parametry včetně vyběru zprávy a odstavců pro zákódování. Mezi volitelné parametry patří:

Type of alphabet slouží k výběru mezi kódovacími abecedami (binární nebo znaková).

Size of data block (integer) je určen pro velikost datového bloku, tj. rozdělení zprávy na bloky o n bitech. Zadává se v přirozených číslech.

Type of security je výčet zabezpečovacích kódů, mezi kterými lze volit. Obsahuje sudou a lichou paritu, opakovací kód s 2, 3 a 4 opakováními a univerzální Hammingův kód.

Dispersion space (real) [%] udává rozptyl bitových mezer. Zadává se v procentech jako kladné reálné číslo.

Justify text je zaškrtávací tlačítko, sloužící pro zarovnání textu. Pokud je zaškrtnuto, tak je text zarovnán do bloku, jinak na prapor.

Variable space over objects slouží pro volbu pevné nebo pružné mezery mezi textem a obrázkem, resp. mezi objekty obecně. Pokud je zaškrtnuto, tak to značí pevnou mezeru.

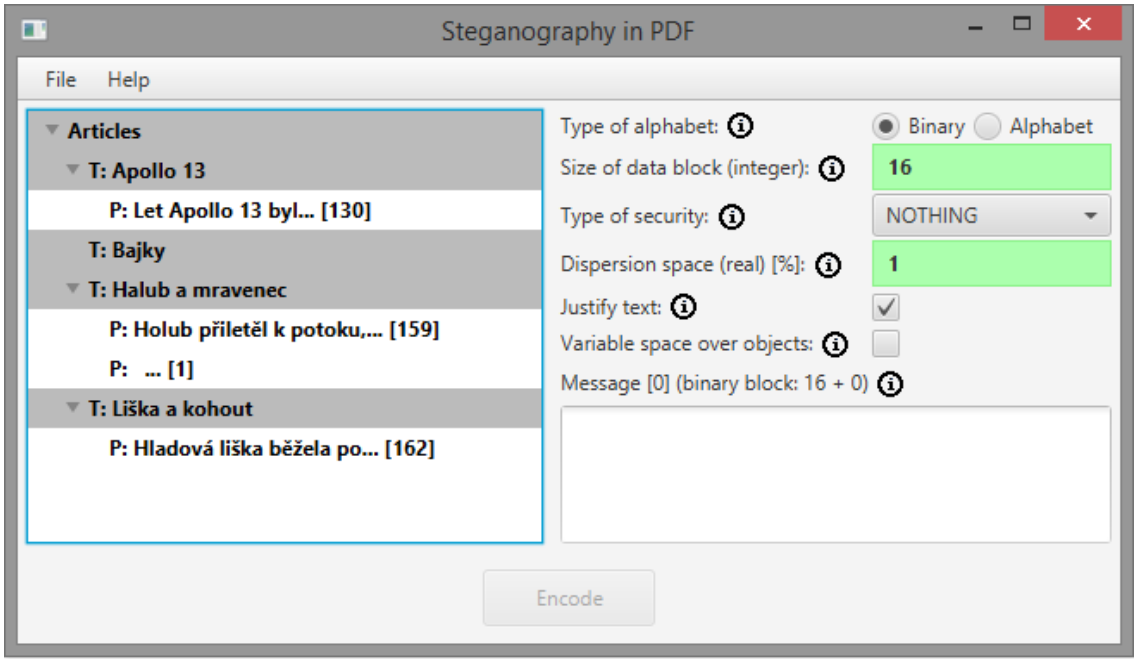
Korektnost parametrů je zabezpečena zablokováním tlačítka *Encode*, který slouží pro zákódování zprávy. Pokud jsou parametry vyplněny špatně, tak jsou podbarveny červeně, v opačném případě zeleně. Pro odblokování tlačítka musí být vybrán alespoň jeden odstavec, pole pro zprávu nesmí být prázdné a všechny parametry musí být zeleně podbarveny. Jednotlivé parametry mají vedle sebe obrázek „`i` v kroužku“. Po najetí myši na takový obrázek, se objeví dodatečné informace daného parametru, případně i vysvětlení čísel jako jsou u popisku nad polem pro zprávu.

9.4 Režim dekódování zprávy

Po načtení obrázku tlačítkem *Load IMG* se objeví okno (viz obr. 9.4). Podobně jako u kódování má i toto okno sadu volitelných parmetrů, kde první tři jsou totožné:

Type of alphabet slouží k výběru mezi kódovacími abecedami (binární nebo znaková).

Size of data block (integer) je určen pro velikost datového bloku, tj. rozdělení zprávy na bloky o n bitech. Zadává se v přirozených číslech.



Obrázek 9.3: Okno v režimu kódování

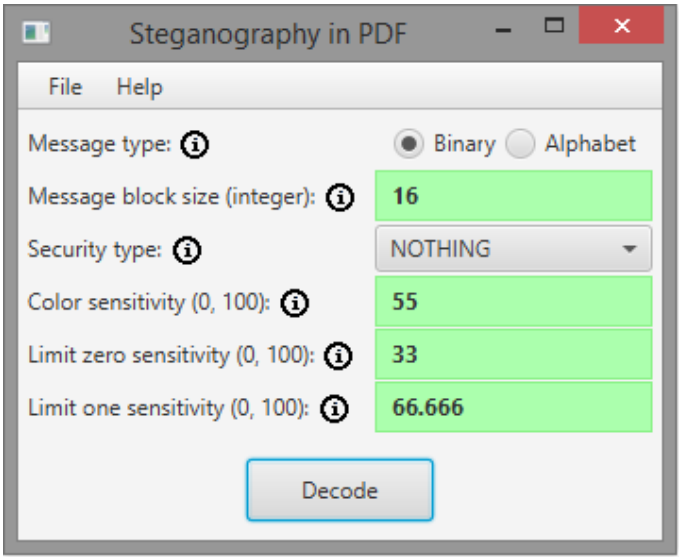
Type of security je výčet zabezpečovacích kódů, mezi kterými lze volit. Obsahuje sudou a lichou paritu, opakovací kód s 2, 3 a 4 opakováními a univerzální Hammingův kód.

Color sensitivity (0, 100) je precentuální interval, který slouží jako hranice přebarvování pixelu na bílou nebo černou barvu. Odstín šedi, který je pod danou hranicí se mění na bílou a v opačném případě na černou.

Limit zero sensitivity (0, 100) je hranice, kdy nalezená mezera je označena jako meziznaková, pokud obsah mezery je pod danou hranicí, nebo v opačném případně jako mezislovní mezera reprezentující bitovou 0.

Limit one sensitivity (0, 100) určuje hranici, kdy je nalezená mezera označena jako mezislovní představující bitovou 0 resp. 1. Pokud je obsah mezery pod danou hranicí a zároveň nad hranicí parametru *Limit zero sensitivity (0, 100)*, tak je mezera označena jako 0 a v opačném případě jako 1.

Stejně jako v režimu kódování mají jednotlivé parametry zabezpečení proti nekorektnímu vstupu, tj. zablokování tlačítka *Decode* a stejně tak jsou vedle všech parametrů obrázky s podrobnějším popisem daného parametru.



Obrázek 9.4: Okno v režimu dekódování

9.4.1 Zobrazení dekódovaných dat

Po dokončení procesu dekódování je otevřeno nové okno (viz obr. 9.5) pro znázornění dekódovaných dat v tabulce. Tabulka obsahuje maximalně osm sloupců, které lze rozdělit do tří skupin. Počet sloupců lze řídit malým tlačítkem + v pravém horním rohu tabulky (viz obr. 9.5) (na obrázku v červeném čtverci), kde se sloupce dají nahrát, případně odstranit z tabulky.

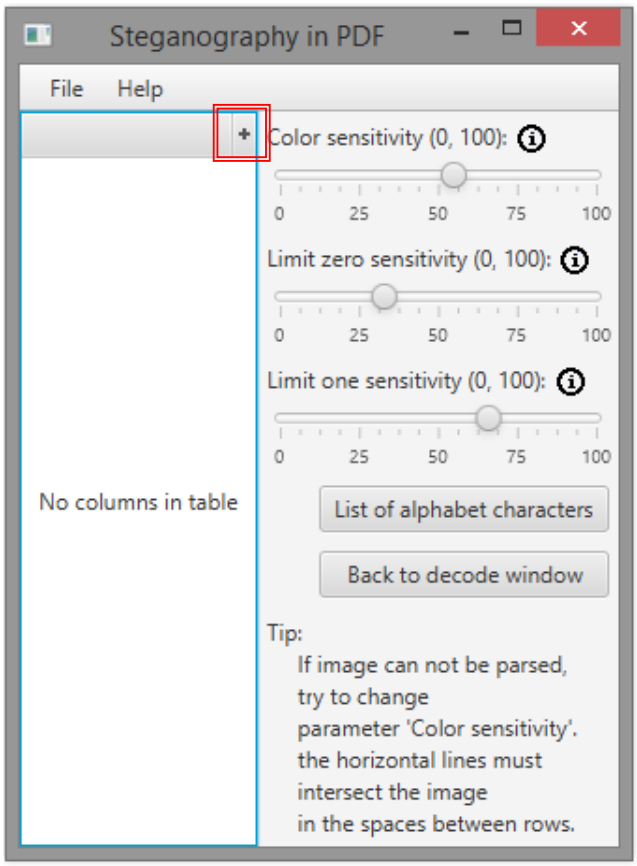
První skupina (viz obr. 9.6) obsahuje sloupec *Parsed rows* s barevně upraveným obrázkem protkaným sítí pro nalezení mezer.

Další skupina (viz obr. 9.7) se zabývá mezerami na jednotlivých řádcích, kde první sloupec *Parsed pixel area [px]* značí kolik pixelů je v nalezených mezerách, druhý *Parsed percentage area [%]* je podobný, ale v procentech vůči ostatním mezerám na řádku. Poslední sloupec *Bits of rows* představuje jaké binární hodnoty byly přiřazeny konkretní mezeře. V této skupině jsou jednotlivé mezery zvýrazněny barvami 2 tmavými a 2 světlými odstíny. Těmito barvami jsou od sebe odděleny jednotlivé bity (v rámci tmavých resp. světlých odstínů) a jednotlivá textová slova (střídání tmavých a světlých odstínů).

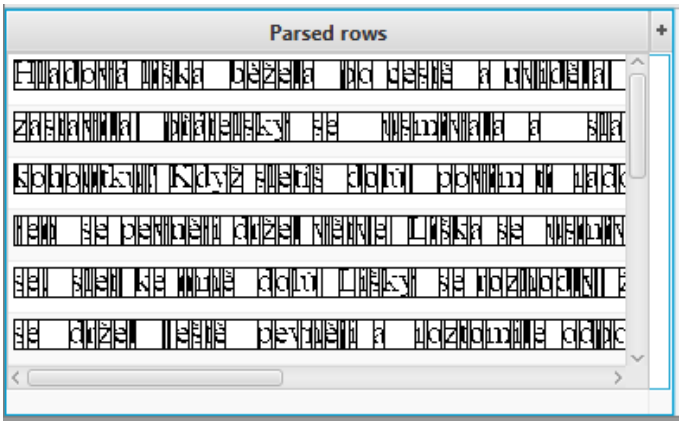
Třetí skupina (viz obr. 9.8) se zabývá načtenými binárními bloky dat, kde první sloupec *Secured original blocks* obsahuje načtené binární hodnoty do jednotlivých bloků a také může obsahovat zabezpečující kód pokud byl v re-

žimu dekódování zvolen. Další sloupec *Secured repaired blocks* obsahuje také bloky se zabezpečením, ale již opravené, pokud to bylo možné. Třetí *Unsecured blocks* a čtvrtý *Character (Alphabet mode)* sloupec na jsou na sebe navázany pokud byla zvolena abeceda znaků. Třetí sloupec obsahuje samostatná získaná data bez zabezpečení a čtvrtý je převádí na znaky. Tato skupina sloupců má specifické podbarvení buněk zelenou, červenou a oranžovou barvou. Zelená značí, že je kódové slovo bez chyby nebo s vícenásobnou chybou. Červená označuje kódová slova s detekovanou chybou a oranžová, že bylo kódové slovo opraveno.

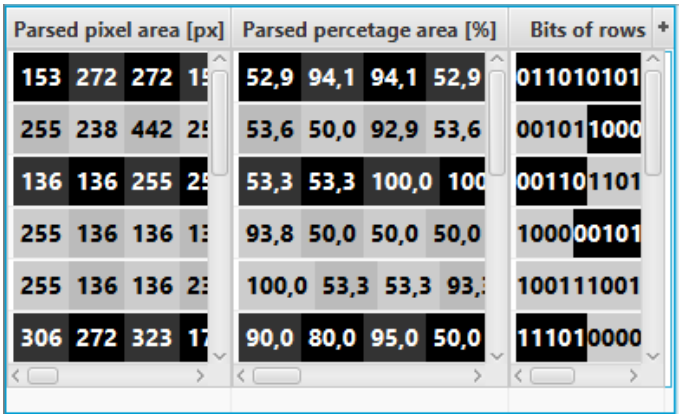
Na pravé straně v okně jsou umístěny posuvníky parametrů, které se shodují s parametry z režimu dekódování a jsou určeny pro manipulaci s dekódovaným obrázkem v reálném čase.



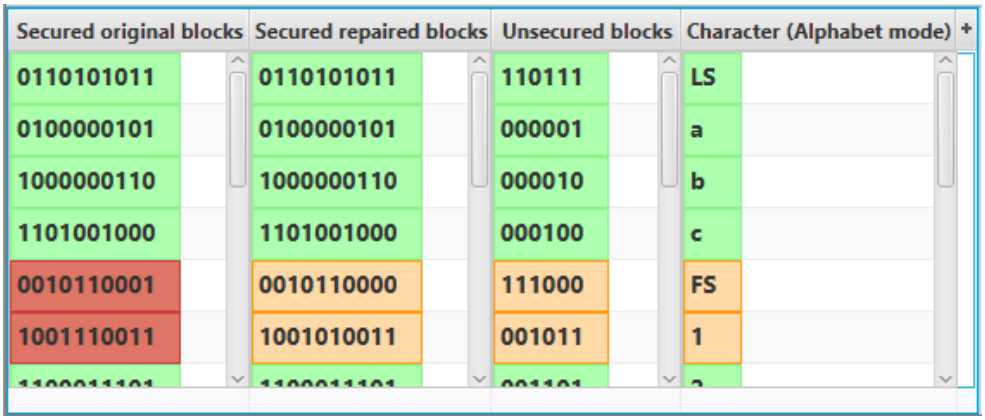
Obrázek 9.5: Okno pro zobrazení dekódovaných dat



Obrázek 9.6: První skupina datových sloupců



Obrázek 9.7: Druhá skupina datových sloupců



Obrázek 9.8: Třetí skupina datových sloupců

10 Statistický průzkum

10.1 Statistika dekódování

Jako subjekty pro experiment sloužilo celkem 50 fotografií, které byly pořízeny v zastíněném světle z vytištěných dokumentů. Do každého dokumentu byla do stejného odstavce zakódována informace s různým rozptylem mezer reprezentujících bity. Jako hodnoty byly použity celá kládná lichá čísla od 1 až po 99. Odstavec, kam byla informace ukryta měl kapacitu 112 bitů a vešlo se do něj 7 celých kódových slov o 16 bitech.

Fotografie poté prošly 5krát procesem dekódování obrázků, kde každé spuštění bylo s jinou hodnotou parametru hranice barev, tj. 55%, 60%, 65% a 70%. Během každého procesu se měnilo intervalové rozmezí mezi slovními mezerami, tj. datové (bitové) mezery. Toto rozmezí bylo pro každý proces stejné a bylo vypočítáno podle vztahu (viz vzorec 10.1).

$$\begin{aligned} x &= \frac{100 - l_0}{2} \\ y &= \frac{99i}{10000} \\ z &= 99 - l_0 + \mid (1 - y)x \mid \end{aligned} \tag{10.1}$$

z – intervalová hranice mezi bit. mezerami 0 a 1

l_0 – intervalová hranice mezi meziznakovou a 0 mezerou

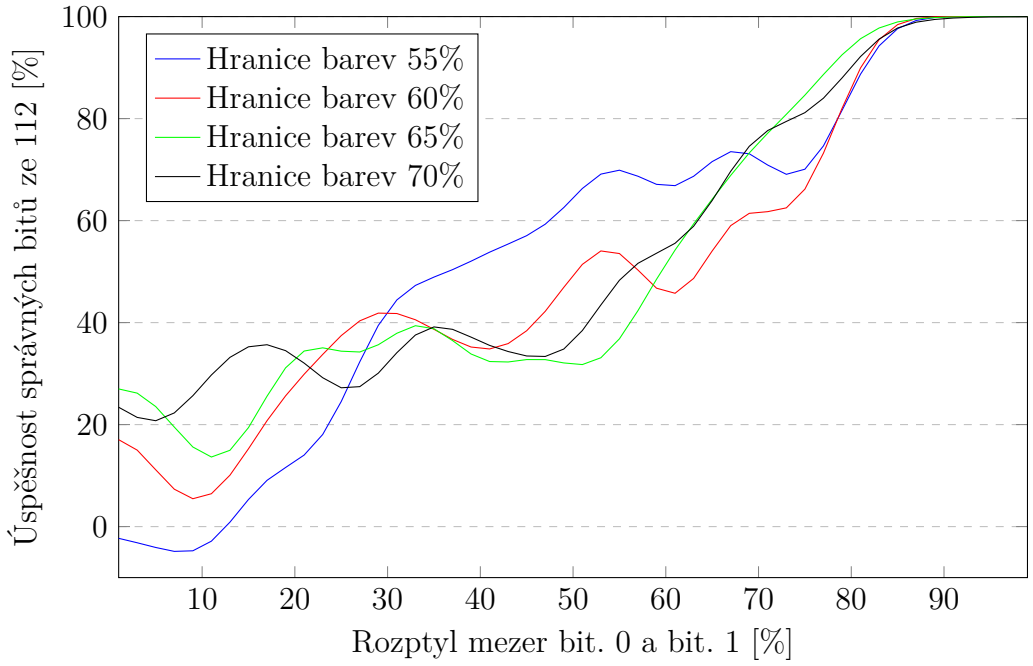
i – rozptyl mezer v zakódovaném dokumentu

Výsledná data z dekódování byla aproximována pro hladší průběh funkcí. Na základě těchto dat byly vytvořeny dva grafy, kde první znázorňuje počet správně dekódovaných bitů (viz graf 10.1) a druhý ukazuje kolik kódových slov bylo správně dekódováno (viz graf 10.2).

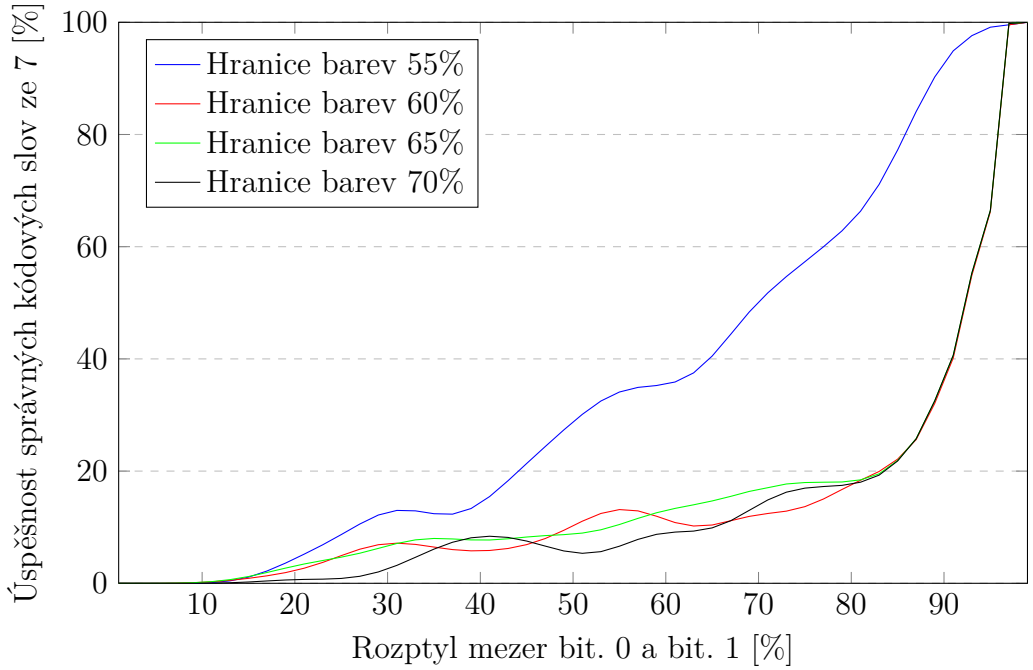
Při rozboru těchto dvou grafů vyplývá, že všechny funkce v hraničních bodech jsou si velmi podobné. Při rozptylu mezer 99% všechny dosáhly maximálního počtu kódových slov, které bylo možné dekódovat. Oproti tomu do rozptylu kolem 20%, byly mezery příliš blízko u sebe a tudíž velkou roli hrálo přebarvování pixelů, jelikož se mohly nesprávné mezery příliš rozšířit nebo naopak zúžit.

Také lze podle modré funkce „Hranice barev 55%“ vidět, že je důležité jaká hranice barev se zvolí. Na začátku kolem 13% rozptylu by v záporných hodnota a to z důvodu, že algoritmus našel méně mezer než bylo jejich maximum. Na druhou stranu tato funkce vyšla z experimentu nejlepé i se středním rozptylem mezer.

Graf 10.1: Úspěšnost správně dekódovaných bitů



Graf 10.2: Úspěšnost správně dekódovaných kódových slov



10.1.1 Závěr statistiky dekódování

Z grafů lze usoudit, že je důležité jaké hodnoty parametry jsou během během procesu zvoleny. Z funkce „Hranice barev 55%“ vyplývá, že je z hlediska ztrátovosti informace bezpečné použít rozptyl od 30 až 40%. Také se lze předpokládat, že při dekódování je dobré použít hranice barev kolem 50% nebo méně, pokud to algoritmus a kvalita obrázku dovolí.

10.2 Uživatelská statistika

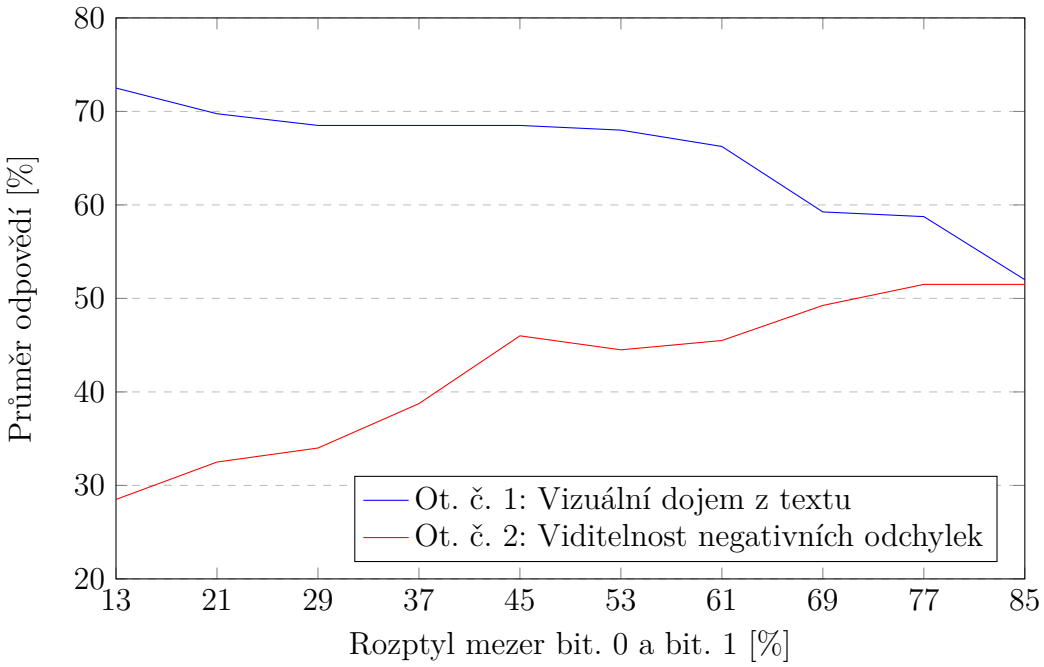
Experimentální data od uživatelů byla získána z předpřipraveného formuláře, ve kterém bylo deset sekcí a v každé z nich vždy dvě stejné otázky. K otázkám byl přiložen obrázek textu s ukrytou informací, přičemž se obrázky v každé sekci lišily rozptylem mezer v rozmezí od 13% do 85%.

Funkce tohoto průzkumu (viz graf 10.3) odpovídají otázkám z formulářů, kde první modrá funkce zkoumá dojem kódovaného textu na uživatele, který ho čte, tj. jakým vizuálním dojmem na něj působí. Druhá červená funkce zkoumá míru viditelnosti grafických odchylek, tj. bitové mezery vůči sobě. Jinak řečeno, jestli uživatel vidí nějaké nesrovnalosti a začíná zjišťovat, že probíhá utajená komunikace. Ve zkratce jsou to dvě stejně položené otázky,

ale jinak formulované, aby se docílilo náhody od uživatel (vidím/nevidím).

Ze samotného grafu nelze určit limitu, kde si je uživatel 100% jistý, že text obsahuje nějakou ukrytou zprávu. Lze ale předpokládat, že zlom nastává mezi 45 až 61%.

Graf 10.3: Dojem stegotextu v obrázku na uživatele



10.2.1 Závěr uživatelské statistiky

Z grafu sice neplyne limita, kdy si je uživatel jistý utajenou komunikací, ale díky předpokladu zlomu 45 až 61%, lze usoudit, že z hlediska nenápadnosti je dobré použít rozptyl mezer zhruba do 50% případně 61% jako maximum.

11 Závěr

Díky nezávislému extrahování obrazků s textem, nelze přímo určit, kde se obrázek má přesně vyskytovat a to má negativní dopad na grafickou podobu kódovaného PDF, které se vytváří od základů. Podobný problém je u zpětného vkládání textu, kdy se nepočítá např. s barvou, tj. text sázen pouze černě. Některé nástroje pro exportování mají zvláštní formát textu, který byl odhalen např. u dokumentů exportovaných z *Google Docs*. Výsledný text byl vysázen s větším fontem než text původní.

Až na tyto nedokonalosti aplikace splňuje zadání uložení zprávy do dokumentu, který je následně vytisknut a vyfotografován. Z fotogragie aplikace dokáže extrahovat ukrytou informaci i s případnými chybami, pokud to zabezpečovací kódy dovolují.

Závěrem statistických experimentů lze předpokládat, že je bezpečné použít rozptyl mezer pro zakódování zprávy od 30 do 50% nebo maximum do 61%. Toto rozmezí je zvoleno jako optimální z hlediska úspěšnosti dekódování a nenápadnosti při čtení.

Literatura

[1] *PDF Reference*. Adobe Systems Incorporated, 6 edition, 11 2006. Version 1.7.

[2] *Document management - Portable document format - Part 1: PDF 1.7*. Adobe Systems Incorporated, 1 edition, 7 2008. Version 1.7.

[3] ŽILKA, R. *Steganografie a stegoanalýza*. PhD thesis, Masarykova univerzita, Fakulta informatiky, 2008.

[4] Isbell, R. Steganography: hidden menace or hidden saviour. *Steganography White Paper*. 2002, 10.

[5] Johnson, N. F. – Jajodia, S. Exploring steganography: Seeing the unseen. *Computer*. 1998, 31, 2, s. 26–34.

[6] Lukan, D. PDF File Format: Basic Structure. Technical report, InfoSec, 5 2018.

[7] Matoušek, R. 3. KÓDY A KÓDOVÁNÍ. *Fakulta strojního inženýrství, Vysoké učení technické v Brně*. 2006.

[8] Nová, I. Základy typografie.

[9] Olšák, P. TeX – 11 (box-penalty-glue a další základy algoritmů sazby). *AbcLinuxu*. 2 2014.

[10] Raja'S, A. – Al Jaber, A. A fragile watermarking algorithm for content authentication. *International journal of computing and information science*. 2004, 2, 1, s. 27–37.

[11] redakce. Základy typografie: Sazba textu a stránka. *Scribus.cz*. 1 2010.

[12] Siper, A. – Farley, R. – Lombardo, C. The rise of steganography. *Proceedings of Student/Faculty Research Day, CSIS, Pace University*. 2005.

[13] Taft, E. et al. The application/pdf Media Type. Technical report, Adobe Systems, 5 2005.

[14] Tarabay, J. Australian Government Passes Contentious Encryption Law. *The New York Times*. 12 2004.

[15] Thomas, K. Portable Document Format: An Introduction for Programmers. *MacTech: The journal of Apple technology*. 1999, 9, 9.

[16] Warkentin, M. – Bekkering, E. – Schmidt, M. B. Steganography: Forensic, security, and legal issues. *Journal of Digital Forensics, Security and Law*. 2008, 3, 2, s. 2.

[17] WATSON, D. L. HIDDEN DATA AND STEGANOGRAPHY. *Handbook of Electronic Security and Digital Forensics*. 2010, s. 427.