Univerzita Komenského, Bratislava

Fakulta Matematiky, Fyziky a Informatiky

Ochrana PDF súborov proti kopírovaniu textu

Bakalárska práca

2012 **Igor Liška**

Univerzita Komenského, Bratislava

Fakulta Matematiky, Fyziky a Informatiky

Ochrana PDF súborov proti kopírovaniu textu

Bakalárska práca

Študijný program: Informatika
Študijný odbor: 2508 Informatika
Školiace pracovisko: Katedra Informatiky
Školiteľ: RNDr. Jaroslav Janáček, PhD.

Bratislava, 2012 **Igor Liška**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Igor Liška
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Ochrana PDF súborov proti kopírovaniu textu

Cieľ: Cieľom práce je implementovať systém na tvorbu dokumentov vo formáte PDF (využitím TeX-u), ktoré bude možné prezerať a tlačiť, no z ktorých nebude možná jednoduchá extrakcia textu použitím štandardných nástrojov. Základnou ideou riešenia je použitie náhodnej permutačnej šifry na vytvorenie špecifického kódovania fontov.

Vedúci: RNDr. Jaroslav Janáček, PhD.
Katedra: FMFI.KI - Katedra informatiky

Dátum zadania: 12.10.2011

Dátum schválenia: 12.10.2011 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestné prehlásenie

Čestne prehlasujem, že prácu som vypracoval sám s pomocou uvedenej literatúry.

Bratislava, 22. 05. 2012 Podpis .....

Ďakujem vedúcemu svojej bakalárskej práce, RNDr. Jaroslavovi Janáčkovi, PhD., za zaujímavú tému a cenné rady, ktoré nemalou mierou prispeli k dotiahnutiu tejto práce do cieľa.

Abstrakt

Autor:	Igor Liška
Názov bakalárskej práce:	Ochrana PDF súborov proti kopírovaniu textu
Škola:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta Matematiky, Fyziky a Informatiky
Katedra:	Katedra Informatiky
Vedúci bakalárskej práce:	RNDr. Jaroslav Janáček, PhD.
Bratislava, jún 2012	

Ochrana autorského vlastníctva trápi nespočetné množstvo autorov publikácií, prezentácií a elektronických kníh. Naskytá sa tu priestor pre systém s jednoduchou obsluhou, ktorý by dokázal znemožniť kopírovanie obsahu chránených diel cez schránku operačného systému. Riešenie, ktoré sme navrhli a implementovali v tejto práci sa síce vzťahuje len na PDF dokumenty vytvorené v sadzbovom programe LaTeX, ale ponúka možnosť tvoriť diela s neskopírovateľným obsahom s využitím šifrovania na úrovni fontov. Presnejšie, kopírovanie nie je znemožnené, ale skopírovaný text zo šifrovanej časti dokumentu je len nezmyselný zhluk znakov. Myšlienka riešenia je založená na technológii virtuálnych fontov v TEXu, ktoré umožňujú prekódovanie znakov a na úprave Adobe Type 1 fontov pre korektný vizuálny vzhľad. Finálny produkt tejto práce je funkčný program schopný vyprodukovať šifrovaný PDF dokument.

Kľúčové slová: PDF, TeX, LaTeX, kopírovanie

Abstract

Author:	Igor Liška
Title:	Protection of PDF files against copying of text
University:	Comenius University in Bratislava
Faculty:	Faculty of Mathematics, Physics and Informatics
Department:	Department of Computer Science
Advisor:	RNDr. Jaroslav Janáček, PhD.
Bratislava, jún 2012	

Enormous number of authors is concerned about the issue of protecting copyright of their publications, presentations and e-books. One can see it as an opportunity to create a system with easy usage that would prevent the unauthorized copying of the content via clipboard of the operating system. Although our solution, which we designed and implemented in this work, is restricted only to PDF documents created in typesetting system LaTeX, it offers possibility to produce protected works utilising font encryption. In other words, copying of the text is possible, but the copied text makes no sense at all. The idea behind is based on TEX virtual font technology that enables us to randomly re-encode all characters and modification of Adobe Type 1 fonts for proper visual appearance. Final product of this work is a program that is able to create encrypted PDF files.

Keywords: PDF, TeX, LaTeX, copying

Obsah

Úvod **1**

1 Prehľad **3**

1.1 TeX . . . 3

1.1.1 LaTeX . . . 3

1.2 PostScript, Portable document format (PDF) . . . 4

1.2.1 PostScript . . . 4

1.2.2 Portable Document Format (PDF) . . . 4

1.3 Adobe Type 1 fonty . . . 5

2 Analýza riešenia **9**

2.1 Popis riešenia . . . 9

2.2 Rozdelenie riešenia . . . 11

2.3 Iné riešenia . . . 11

3 TeX balík **13**

3.1 Prostredie . . . 13

3.2 Módy dokumentu . . . 14

3.3 LaTeX NFSS - New Font Selection Scheme . . . 17

3.4 Detekcia zmeny fontu . . . 18

3.4.1 Počítadlo (counter) . . . 20

3.5 Zavedenie šifrovaných fontov . . . 20

4 Program pdfencrypt **23**
4.1 Utility na konverziu formátu 23
4.1.1 Utilita TFtoPL 23
4.1.2 Utility PLtoTF, VPtoVF 24
4.1.3 Knižnica kpathsea 25
4.2 Virtuálne fonty 27
4.3 Štruktúra virtuálneho fontu 29
4.4 Zašifrovanie textu pomocou virtuálneho fontu 34
4.5 Štruktúra Adobe Type1 fontu 35
4.5.1 Prvá ASCII sekcia 38
4.5.2 Binárna sekcia 39
4.6 Modifikácia Adobe Type1 fontu - zamaskovanie textu 41
4.6.1 Kolízie 41
4.7 Program pdfencrypt 42

5 Manuál **45**
5.1 Inštalácia 45
5.2 Konfigurácia 46
5.3 Návod na použitie 46

Záver **47**

Príloha A - Slovník pojmov **49**

Úvod

Cieľom tejto bakalárskej práce je navrhnúť a implementovať systém umožňujúci ochrániť PDF súbory vytvorené pomocou sadzbového programu LaTex pred kopírovaním obsahu. Dôležité je, aby sa pri prehliadaní obsah súboru zobrazoval korektne (v akomkoľvek PDF prehliadači) a tiež sa dal bez problémov vytlačiť. Nesmie byť možné skopírovať obsah vyznačením nejakej časti obsahu a následne ho preniesť cez schránku (clipboard) operačného systému. Samotnému kopírovaniu síce nezabránime, ale prenesený obsah nebude dávať žiadny zmysel. Takúto ochranu dokážeme zabezpečiť pomocou náhodnej permutačnej šifry. V prvom kroku využijeme funkcionalitu virtuálnych fontov v Texu, aby sme vytvorili zašifrovaný obsah. Pri kompilácii do PDF výstupu Tex najčastejšie používa binárne Adobe Type 1 fonty a práve tieto fonty budeme využívať pre naše PDF súbory. Adobe Type 1 font obsahuje pre nás dôležité PostScript inštrukcie pre vykreslenie každého znaku. Ak tieto inštrukcie vhodne poprehadzujeme tak, aby zodpovedali inverznému zobrazeniu k zobrazeniu z prvého kroku, výsledný PDF súbor bude navonok vyzerať presne tak, ako požadujeme, avšak vykreslené znaky budú reálne zodpovedať zašifrovanému textu. Dôraz je kladený aj na jednoduchosť inštalácie, konfigurácie a použitia nami vytvoreného systému.

Okrem hlavného cieľu tejto práce, zvýšenia bezpečnosti PDF dokumentov vytvorených v LaTeXu, máme stanovený aj vedľajší cieľ - spoznať TEX a LaTeX do väčšej miery, ako iba obyčajný uživateľ. Oblasť virtuálnych fontov, spôsob predefinovania makier a interná réžia pri práci s fontami stoja za preskúmanie a môžu ešte viac prehĺbiť autorove vedomosti o tomto sadzbovom

systéme. V neskorších kapitolách sa budeme zaoberať aj samotnou LaTeX NFSS (New Font Selection Scheme), teda schémou na výber fontov, ktorá zdokonalila prácu s fontami v LaTeXu, oproti klasickému TEXu.

V krátkosti si prejdeme aj históriu okolo TEXu a s ním súvisiacich technológií.

Kapitola 1

Prehľad

V tejto kapitole si priblížime niektoré detaily histórie TEXu a jeho nadstavby LaTeXu. Spomenieme aj jazyk PostScript a formát PDF, spolu s potrebami jeho vzniku. Preskúmame tiež oblasť Adobe Type 1 fontov.

1.1 TeX

TeX je jedným z kľúčových prvkov tejto práce, preto v krátkosti objasníme, čo vlastne TeX je a čo všetko jeho pomocou vieme dosiahnúť. Podľa [Wik12], TeX je sadzbový programovací jazyk, umožňujúci detailné formátovanie textu. Popularitu na akademickej pôde a medzi nadšencami matematiky a informatiky si získal aj schopnosťou profesionálne zobraziť komplikované matematické vzorce. Autorom TeXu je nemálo slávny Donald Knuth, ktorý dokončil prvú verziu okolo roku 1985 pre vlastnú potrebu a údajnú nespokojnosť s vtedajšími sadzbovými technoĺógiami. Nás bude zaujímať hlavne nadstavba TeXu pre zjednodušenú prácu s dokumentmi - LaTeX.

1.1.1 LaTeX

Ako uvádza [Unw], LaTeX uľahčuje autorovi sústrediť sa viac na samotnú štruktúru dokumentu a nestrácať čas so s formátovaním. Obsahuje viaceré prednastavené formáty dokumentu ako kniha, článok, alebo list. Obsahuje

tiež rozširujúci balíček BibTeX pre jednoducho vytvoriteľné bibliografické zápisy [Wika]. V tejto práci sa budeme zaoberať práve spôsobom, ako vytvoriť v LaTeXu PDF súbory, ktoré budú chránené proti kopírovaniu. O formáte PDF si povieme v nasledujúcej podkapitole.

1.2 PostScript, Portable document format (PDF)

Medzi jazykom PostScript a formátom PDF existujú isté súvislosti, ktoré si spomenieme v tejto podkapitole. Začneme jazykom PostScript a ukážeme, ako jeho historický vývoj súvisel s vznikom formátu PDF.

1.2.1 PostScript

Ako je uvedené v [Sys12a], PostScript je programovací jazyk pre presné popisovanie vzhľadu dokumentu. Prvotne bol využívaný na tlačiarniach, ktoré mali zabudovaný interpreter tohoto jazyka, tzv. Raster Image Processor (RIP). RIP bola kombinácia softvéru a hardvéru, schopná prečítať a interpretovať program v jazyku PostScript a vyprodukovať grafickú informáciu na výstupné médium (napríklad papier). Ako napredoval vývoj v oblasti informačných technológií, RIP bol vytvorený aj v čisto softvérovej forme a umožňoval zobraziť PostScript programy aj na obrazovke počítača, čo bolo veľkým prínosom v tlačiarenskom priemysle, nakoľko mohli byť chyby v grafike odstránené ešte pred samotným vytlačením na papier. PostScript je dodnes využívaný v niektorých najmä high-endových tlačiarenskych zariadeniach a na jeho princípoch je postavený aj neskôr vyvinutý formát PDF.

1.2.2 Portable Document Format (PDF)

Portable Document Format (PDF) rozširuje možnosti jazyka PostScript, na ktorom je založený [Sys12a]. Nielenže dokáže presne popísať vzhľad doku-

mentu, ale umožňuje uchovať aj ďalšie informácie popisujúce charakter dokumentu (kľúčové slová), alebo zdroje súvisiace s týmto dokumentom, ako napríklad obrázky, videá, funkčné hyperlinky, fonty atď. PDF formát je v skutočnosti PostScript program, ktorý už je interpretovaný a preložený pomocou RIP. Tvoria ho teda jasne definované objekty, ktoré sú zrozumiteľné pri bežnom pohľade.

PDF formát je v súčasnosti jeden z najpoužívanejších formátov na prenos dokumentov, pretože nám zaručuje, že sa na rôznych platformách bude zobrazovať vždy rovnako a je schopný uchovať v sebe všetky informácie potrebné pre korektné zobrazenie dokumentu, čím je možné zabezpečiť jeho úplnú nezávislosť na externých zdrojoch. Existujú prípady, kedy samotný dokument neobsahuje v sebe všetky potrebné zdroje (napríklad fonty), aby sa ušetrila výsledná veľkosť dokumentu. V takom prípade je nutné, aby tieto zdroje boli dostupné na cieľovej platforme pre korektné zobrazenie dokumentu.

Spomenuté výhody so sebou prinášajú aj niektoré menej pozitívne aspekty, ako napríklad jednoduché kopírovanie informácií z cudzích prác a diel. Práve neoprávneným kopírovaním sa budeme zaoberať v tejto práci a pokúsime sa ho do istej miery znáročniť, aj keď nie úplne znemožniť. Do tejto problematiky sa ponoríme v ďalších kapitolách.

1.3 Adobe Type 1 fonty

Adobe Type 1 Font Format (ďalej už len type 1 formát) je celosvetovo používaný formát pre digitálne znakové sady písma (ďalej už len fonty) navrhnutý firmou Adobe Systems Incorporated [Sys12a]. V roku 1991 sa stal ISO štandardom (ISO 9541). Bol vymyslený v 80-tych rokoch s primárnym využitím na tlačiarniach, ktoré využívali jazyk PostScript na interpretáciu dát určených na tlač.

Neskôr vznikol formát TrueType, vytvorený firmou Apple Computer a

neskôr licencovaný firmou Microsoft. Tento fontový formát mal údajne oslobodiť firmy Apple a Microsoft od závislosti na fontých súboroch firmy Adobe. V dnešnej dobe existuje modernejší fontový formát, OpenType, vyvíjaný spoločným usilím firiem Microsoft a Adobe. Spája viaceré pozitívne vlastnosti starších formátov type 1 a TrueType, ale netrpí niektorými ich limitáciami (type 1 mohol obsahovať najviac 256 glyfov, zatiaľ čo OpenType formát zvláda mapovanie až do 65536 glyfov, čo v dnešnej dobe pokrýva potreby asi všetkých jazykov).

Špecifikácia formátu type 1 nebola vždy voľne dostupná. Spočiatku mali výrobcovia fontov k dispozícii len type 3 formát, ktorý bol schopný využiť všetky možnosti jazyka PostScript, avšak trpel nedokonalým zobrazovaním pri nízkych rozlíšeniach a zaberal viac pamäte oproti formátu type 1. Po nátlaku zo strany konkurencie (TrueType formát) firma Adobe zverejnila špecifikáciu formátu type 1 a vydala tiež Adobe Type Manager (ATM), ktorého úlohou bolo škálovanie type 1 fontov pre zobrazovanie na obrazovkách alebo tlač na tlačiarniach, ktoré nevyužívali jazyk PostScript.

Ako je uvedené v [Wikb], Type 1 formát je v skutočnosti upravená, kompaktnejšia verzia PostScript jazyka s niekoľkými výhodami oproti Type 3 formátu. PostScript interpreter využíva špeciálny rasterizačný algoritmus, ktorý dosahuje krajší vzhľad výstupu pri malých veľkostiach písma. Type 1 font umožnuje vytvárať aj takzvané nápovedy (hints), ktoré pomáhajú lepšie popisovať špeciálne tvary znakov a napovedajú interpreteru, ako takéto glyfy vykresliť a dosiahnuť čo najlepšiu čitateľnosť textu.

V tejto práci sa budeme zaoberať výhradne type 1 fontovým formátom, nakoľko je to natívne používaný formát v sadzbovom programe LaTeX (TEX, aj LaTeX zvládajú pracovať aj s PK (bitmapovými) fontami, ktoré sú takisto natívne používaným formátom, avšak pre ciele tejto práce nie sú podstatné a preto sa nimi nebudeme bližšie zaoberať). V LaTeXu je možné využiť aj iné formáty fontov, ale v zásade je nutné najprv vykonať konverziu do type

1 formátu.

Type 1 formát sa aj v dnešnej dobe teší nemalej popularite medzi používanými fontovými formátmi a je možné ho použiť takmer na všetkých platformách, alebo vytlačiť na každej tlačiarni. Umožňuje to buď interpreter jazyka PostScript, alebo neskôr vytvorený Adobe Type Manager.

Kapitola 2

Analýza riešenia

Predtým, ako sa pustíme do skúmania akéhokoľvek riešenia, jasne si stanovme cieľ tejto práce. Chceme vytvoriť program, ktorý na vstupe dostane LaTeX dokument a jeho úlohou je vytvoriť výstupný PDF dokument, ktorého textový obsah nie je možné skopírovať štandardným vyznačením textu a jeho prenesením cez schránku operačneho systému do iného programu, alebo súboru v zrozumiteľnej podobe. Spôsob, akým vieme takýto PDF dokument vytvoriť, preskúmame v nasledujúcej podkapitole. Nebudeme sa podrobne zaoberať každým detailom, skôr sa na naše riešenie pozrieme „zvrchu" a konkrétne detaily prenecháme ďalším kapitolám.

2.1 Popis riešenia

Naše riešenie musí splniť jednu dôležitú podmienku - výsledný PDF dokument nesmie byť navonok rozpoznateľný od ekvivalentného PDF dokumentu, ktorý vznikol z rovnakého vstupného dokumentu, alebo nebol vygenerovaný naším programom, ale štandardným spôsobom.

Základnou myšlienkou našeho riešenia je šifrovanie na úrovni fontov. Hlavnou výhodou tohoto riešenia je možnosť vyhnúť sa zásahom do vstupného dokumentu (v konečnom dôsledku sa nám síce nepodarilo vyhnúť sa zásahom do vstupného dokumentu úplne, ale naše zásahy sú minimálne a

ľahko odstrániteľné). Myšlienka je jednoduchá - stačí prinútiť každý znak v dokumente, aby navonok vyzeral správne, ale v skutočnosti sa správal ako iný znak. Uveďme si konkrétny príklad. Ak donútime znak „A" vo vstupnom dokumente, aby sa vo výstupnom PDF dokumente správal ako znak „d", tak pri kopírovaní textu cez schránku operačného systému z výsledného PDF dokumentu namiesto znaku „A" prekopírujeme znak „d". Už nám len stačí zabezpečiť, aby znak „d" vo výslednom PDF dokumente vyzeral presne ako znak „A" a naše riešenie bude fungovať. Len samotná myšlienka však nestačí. Počas hľadania spôsobu, ktorým by bolo možné túto myšlienku previesť do reality, sme narazili na mnohé problémy a slepé uličky. Nakoniec sa ako naschodnejšia cesta javila kombinácia virtuálnych fontov a Adobe Type 1 fontov.

Virtuálne fonty umožňujú prekódovanie znakov na iné znaky, čím vieme dosiahnuť zmenu správania znakov - vo výslednom dokumente sa budú tváriť ako znaky s rozdielnymi ordinálnymi hodnotami, preto napríklad znak „A" už nebude obsadzovať podľa ASCII štandardu hodnotu 65, ale inú, náhodne zvolenú.

Pod prekódovaním si môžeme predstaviť náhodné premiešanie sady ordinálnych hodnôt znakov - vygenerovanie náhodnej permutácie kódovacieho vektora, naše riešenie teda využíva náhodne zvolenú permutačnú šifru.

Adobe Type 1 fonty obsahujú inštrukcie, pomocou ktorých vie tlačiareň (napríklad *pdflatex*), ako majú jednotlivé znaky vyzerať. Ak tieto inštrukcie poprehadzujeme vhodným spôsobom, vieme prekódovaným znakom priradiť správny vzhľad, aby sa navonok tvárili ako pôvodné vzory pred prekódovaním.

2.2 Rozdelenie riešenia

Produktom tejto práce budú dva samostatné celky, ktoré spolu budú úzko kooperovať a spolu tvoria riešenie, pomocou ktorého dosiahneme cieľ tejto práce.

Prvý z týchto celkov je TEX balík, jeho funkcionalitu si prejdeme v nasledujúcej kapitole. Druhý celok predstavuje program *pdfencrypt*, napísaný v jazyku Python. Ako to celé funguje?

Samotný program by celú úlohu nezvládol, pretože by nedokázal vykonať dve dôležité úlohy bez rozsiahleho zásahu do vstupného dokumentu - zistiť, kde všade v dokumente nastáva zmena fontu a zaviesť nové, šifrované fonty. Presne tieto úlohy zvládajú makrá, ktoré sme definovali v spomenutom TEX balíku. Po zavedení balíku do vstupného dokumentu (pomocou príkazu *usepackage*) stačí nechať vstupný dokument spracovať napríklad *pdflatexu*. Na jeho výstupe nájdeme kompletný zoznam zmien fontov, na základe ktorého vieme vygenerovať programom *pdfencrypt* príslušné šifrované fonty. Potom prichádza opäť na rad náš TEX balík, ktorý vygenerované šifrované fonty zavedie do dokumentu. Detailnejší pohľad na celý postup nájdeme v nasledujúcich kapitolách.

2.3 Iné riešenia

V tejto podkapitole si uvedieme niektoré alternatívne metódy, ktorými je možné zvýšiť bezpečnosť PDF súborov vo všeobecnosti. Pripomíname, že naše riešenie je použiteľné len pre PDF súbory vytvorené sadzbovým programom LaTeX.

Ako prvú si uvedieme možnosť chrániť PDF súbor heslom. Táto možnosť je vhodná napríklad pri elektronických faktúrach, kedy je žiadané, aby si faktúru vedel otvoriť len samotný adresát. Tento spôsob ochrany je efektívny,

ale má iné prípady použitia, ako naše riešenie. V našom prípade chceme byť schopní voľne distribuovať PDF súbory, ale znemožniť jednoduché kopírovanie ich obsahu. Ochrana heslom by pri voľnej distribúcii nebola veľmi efektívna, nakoľko po prezradení hesla je celkom zbytočná.

Ďalší spôsob je opäť použiteľný pre PDF súbory vo všeobecnosti (nielen pre PDF súbory vytvorené v LaTeXu). Pri vytváraní PDF súboru dostáva autor možnosť deaktivovať funkciu „copy & paste", teda kopírovanie cez schránku operačného systému. Tento spôsob má úplne totožné prípady použitia, ako nami vymyslené riešenie, avšak trpí drobnými nedostatkami. To, či bude kopírovanie skutočne znemožnené závisí čisto na prehliadači, v ktorom je PDF súbor prezeraný. Niektoré open-source prehliadače túto možnosť kompletne ignorujú a preto je v zásade neefektívna.

Poslednou možnosťou je uložiť jednotlivé strany PDF súboru ako rastrové obrázky a vyrobiť z nich nový PDF súbor. Tento spôsob je rovnako efektívny, ako naše riešenie, čo sa týka bezpečnosti, ale prináša so sebou niekoľko zásadných nevýhod. Veľkosť výsledného PDF súboru enormne vzrastie oproti pôvodnému súboru a pri funkcii „zoom" veľmi rýchlo narazíme na klesajúcu vizuálnu kvalitu dokumentu.

Aby sme to zhrnuli, najvyššiu mieru bezpečnosti poskytuje ochrana heslom. Tento spôsob má však iné prípady využitia, ako naše riešenie a celá bezpečnosť sa vytráca po prezradení hesla. Takáto vlastnosť je pre distribúciu PDF súboru nepraktická. Zvyšné dve riešenia sú síce použiteľné pre PDF súbory vo všeobecnosti, avšak trpia viac či menej závažnými nedostatkami, ktoré eliminujú ich využitie v praxi. Naše riešenie je obmedzené len na PDF súbory vytvorené v LaTeXu, ale poskytuje rozumnú mieru bezpečnosti a netrpí stratou kvality, alebo prehnanou veľkosťou výsledného PDF súboru.

Kapitola 3

TeX balík

Súčasťou spôsobu, akým budeme generovať výsledný šifrovaný PDF súbor, je automatické generovanie šifrovaných fontov, o ktorom si detailne povieme v ďalšej kapitole. Pre naše riešenie však nechceme vždy generovať šifrované fonty pre každý jeden font, ktorý má TEX nainštalovaný. Chceme vybrať len tie fonty, ktoré sú použité v našom dokumente, zahŕňajúc nami explicitne zvolené fonty a tiež TEXom automaticky zavedené fonty. Elegantným riešením je nechať samotný TEX, nech nám presne povie, ktoré fonty potrebuje. Dostali sme sa o krok bližšie k našemu cieľu. TEX nám prezradí zoznam použitých fontov a pomocou našeho programu si dokážeme vygenerovať šifrované verzie použitých fontov. Tu sa dostávame k ďalšej prekážke - ako teraz donútime náš dokument, aby použil šifrované fonty bez toho, aby sme ho museli upraviť? Spomenuté úlohy vyriešime pomocou rôznych makier napísaných priamo v TEXu a nakoniec ich spojíme do jedného TEX balíku.

3.1 Prostredie

Hlavnou súčasťou našeho balíku je nové prostredie (environment), nazvané „encrypted". Výhodou prostredí v LaTeXu je možnosť ohraničiť a presne určiť časti dokumentu, na ktorých chceme vykonať nejaké úpravy. Umožňuje nám to poskytnúť autorovi dokumentu absolútnu flexibilitu - môže sa sám

rozhodnúť, či chce šifrovať celý dokument, alebo len vybrané pasáže. Konkrétny príklad môžeme vidieť v ukážke 3.1.

```
\begin{document}
    Text dokumentu, ktorý nebude šifrovaný.

    \begin{encrypted}

        Text dokumentu, ktorý bude vo výslednom PDF
        dokumente šifrovaný.

    \end{encrypted}
\end{document}
```

Obr. 3.1: Spôsob rozčlenenia dokumentu na šifrované a nešifrované časti

Súčasťou našeho prostredia sú aj definície viacerých makier a tiež predefinovanie niektorých TEX príkazov (*endgroup*, *selectfont*), ktoré nám pomôžu pri detekcii zmeny fontu v dokumente. Detailnejší popis zmien v týchto príkazoch sa dozvieme v následujúcich podkapitolách.

3.2 Módy dokumentu

Ešte predtým, ako si povieme viac o makrách, ktoré obsahuje náš balík, objasníme si jednu z kľúčových schopností našeho balíka - prepínanie módu dokumentu. Najprv si však povieme o dôvode, prečo to vôbec chceme robiť. Predstavme si postup, ktorým sa chceme dopracovať k šifrovanému PDF súboru a spätne si ho prejdime:

1. Pri každej zmene fontu v dokumente si do logu poznačíme, že sa zmenil font a informácie o novom fonte

2. Necháme spracovať dokument *pdflatexu*[1], ktorý nám vygeneruje log súbor obsahujúci kompletný zoznam použitých fontov

3. Na základe zoznamu fontov z log súboru vygenerujeme šifrované fonty

4. Necháme spracovať dokument *pdflatexu*, ktorý použije šifrované fonty a vygeneruje výsledný PDF dokument

Problém nastáva pri spracovaní dokumentu *pdflatexom* - v obidvoch prípadoch sa totiž pracuje s tým istým vstupným dokumentom. Ak by sme tomuto dokumentu nevedeli prepnúť mód, v 2. bode by sme nedostali log súbor, ale chybovú hlášku o chýbajúcich fontoch. Inými slovami, len s jedným módom dokumentu by sme sa ocitli v cykle, pretože na získanie zoznamu fontov potrebujeme nechať prejsť dokument pdflatexom, na to ale potrebujeme mať už vygenerované šifrované fonty, čo však nevieme spraviť skôr, ako dostaneme zoznam použitých fontov z log súboru.

Rozlišujeme dva módy dokumentu, normálny a šifrovací mód. Pri spracovaní *pdflatexom* v šifrovacom móde sa štandardné fonty nahradia ich šifrovanými dvojičkami a preto je očakávané, že sú všetky šifrované fonty dostupné. Cieľom je vytvoriť výsledný šifrovaný PDF dokument.

Implicitne je TeX dokument prepnutý do normálneho módu. Úlohou normálneho módu je nesnažiť sa použiť šifrované fonty, ale detekovať zmeny fontov v dokumente, ktoré môžu nastať všetkými dostupnými spôsobmi. O detekovaní zmien fontu si viac povieme v následujúcich podkapitolách. Všetky detekované zmeny sú následne zapísané do log súboru príkazom $\backslash message$ v nami dopredu určenom formáte, ako môžeme vidieť v ukážke 3.2.

[1]V uvedenom postupe je využitie pdflatexu čisto náhodné, je možné použiť aj iné sadzbové programy, napríklad pdfcslatex

```
bcencryption:fch:cmr10:sf: cmr10----bc1
```

Obr. 3.2: Ukážka formátu informácie o zmene fontu v dokumente - časť log súboru

Rozanalyzujme si informáciu o zmene fontu z ukážky 3.2. Môžeme si všimnúť, že táto informácia pozostáva z viacerých informácií, ktoré sú oddelené znakom „:". Prvá časť je samotný identifikátor, reťazec `bcencryption`, podľa ktorého vieme, že sa jedna o nami vypísanú informáciu. Ďalšia časť je reťazec znakov, označujúci zmenu fontu (reťazec `fch` je skratkou zo slov font change - zmena fontu). Treťou časťou celej informácie je identifikátor nového fontu, v našom príklade ide o štandardný font `cmr10` - Computer Modern Romain, veľkosť 10 pt. Nasleduje informácia o spôsobe, akým došlo k zmene fontu, prípustné hodnoty sú `sf`, `bf` a `eg`. Skratka `sf` je od príkazu *selectfont*, čo značí, že zmena fontu nastala pomocou NFSS (LaTeX New Font Selection Scheme), o ktorej si povieme neskôr. Pod skratkou `bf` sa ukrýva príkaz *bcfont*, ktorý simuluje funkcionalitu TEX primitívy *font*. Posledná prípustná hodnota, `eg`, hovorí o zmene fontu v dôsledku ukončenia skupiny, teda pri zavolaní príkazu *endgroup*. Posledná časť informácie o zmene fontu je názov fontu, ktorým bude nájdený font nahradený v šifrovacom móde dokumentu. Štruktúra názvu tohoto šifrovaného fontu je špeciálne zvolená a má svoje opodstatnenie. Reťazec „`----`" oddeľuje meno pôvodného fontu od sufixu, ktorý obsahuje číslo, inkrementované po každej detekcii zmeny fontu. Takáto štruktúra nám zabezpečuje, že z názvu šifrovaného fontu vieme algoritmicky odvodiť meno pôvodného fontu a tiež nám zaručuje, že každý šifrovaný font bude mať unikátne meno. O poslednú spomenutú vlastnosť sa stará práve číselná hodnota, ktorá je vždy po použití inkrementovaná. Vráťme sa k módom dokumentu.

Prepínanie dokumentu do šifrovacieho módu funguje na báze zadefinovania makra *readcontextbc* na začiatku dokumentu. Ak sa toto makro vyhodnotí ako ekvivalentné s makrom *undefined* (nie je zadefinované), dokument

je v normálnom móde, inak je v šifrovacom móde. Ak chce náš program vygenerovať šifrovaný PDF súbor, jednoducho stačí, ak na začiatok dokumentu doplní definíciu spomenutého makra a nechá dokument prejsť *pdflatexom*. Nakoniec len odstráni makro zo začiatku dokumentu. Bez akéhokoľvek zásahu je vstupný dokument v normálnom móde a po prejdení napríklad *pdflatexom* je súčasťou log súboru a aj výstupu *pdflatexu* zoznam detekovaných zmien fontov vo formáte, aký sme si popísali v ukážke 3.2.

3.3 LaTeX NFSS - New Font Selection Scheme

LaTeX vo verzii LaTeX2e priniesol aj nový systém výberu fontov nazvaný NFSS - New Font Selection Scheme. Pri navrhovaní NFSS bol kladený dôraz na výber fontu na základe jeho atribútov a kontextu. Tento prístup sa časom prejavil ako veľmi úspešný a pohodlný. V NFSS rozlišujeme päť základných atribútov:

1. **encoding** (kódovanie) - tento atribút definuje poradie, v akom sa jednotlivé glyfy nachádzajú

2. **family** (rodina fontov) - skupina fontov patrí do rovnakej rodiny, ak majú spoločné vizuálne charakteristiky. Fonty z rovnakej rodiny sa k sebe „hodia" a je odporúčané nemiešať medzi rozličnými rodinami v rámci jedného dokumentu

3. **series** (séria) - jednotlivé fonty sa môžu odlišovať svojou hrúbkou, alebo šírkou. Tieto dve veličiny NFSS spája pod pojmom `series`.

4. **shape** (tvar) - tento parameter spája fonty s podobnými geometrickými charakteristikami, napríklad kurzíva

5. **size** (veľkosť) - relatívna veľkosť fontu v jednotkách `pt`.

Spôsoby, ktorými vieme v NFSS docieliť zmenu fontu, môžeme kategorizovať do troch úrovní: vysokej, strednej a nízkej.

Vysokoúrovňové spôsoby na zmenu fontu zahŕňajú príkazy ako *textit* pre kurzíva, alebo *textbf* pre tučné písmo. Ich úlohou je jednoduché a promptné použitie, bez ďalších nastavení. Automaticky, v závislosti od aktuálneho kontextu, vedia dosiahnuť požadovanú zmenu.

Na strednej úrovni sa stretneme s príkazmi ako *fontencoding* alebo *fontfamily*, ktoré si už síce vyžadujú viac skúseností autora, ale ponúkajú aj väčšiu kontrolu nad zmenou fontu.

Nízka úroveň predstavuje rozhranie medzi NFSS a reálnymi fontami, nájdeme tu napríklad príkazy *DeclareFontFamily*, alebo *DeclareFontShape*.

NFSS v sebe ukrýva kľúčovú vlastnosť pre túto prácu - akákoľvek zmena fontu je v konečnom dôsledku zavedená príkazom *selectfont*. Inými slovami, najprv sa zmenia jednotlivé parametre, ktoré sme uviedli na začiatku tejto podkapitoly. Nový font bude aktívny až po zavolaní príkazu *selectfont*, ktorý zmeny parametrov potvrdí. Príkaz *selectfont* nám poskytuje dokonalú príležitosť, ako detekovať zmenu fontu v LaTeXu. K tejto téme sa detailnejšie vyjadríme v nasledujúcej podkapitole. Podrobnejšie informácie k NFSS môžeme nájsť v [Hoe98].

3.4 Detekcia zmeny fontu

V predošlej podkapitole sme si priblížili prácu s fontami vo všeobecnejšom zmysle a teraz si ukážeme, ako je možné niektoré makrá „dodefinovať", aby nám vedeli trošku pomôcť okrem svojej bežnej činnosti. Najprv sa pozrieme na vyššiu úroveň, teda na úroveň LaTeXu. Ak píšeme náš dokument slušne a podľa určitých štandardov, akákoľvek zmena fontu v LaTeXu prebieha najprv zmenou parametrov a ich následovným potvrdením príkazom *selectfont*.

TeX nám poskytuje nástroje, pomocou ktorých je možné dosiahnuť dode-

finovanie ďalšej funkcionality existujúcim príkazom. Najprv starý príkaz premenujeme príkazom *let* a potom zadefinujeme nový príkaz (príkazom *def*) s rovnakým menom a argumentami ako mal pôvodný príkaz a v jeho definícii zavoláme premenovanú verziu a následne môžeme vykonať akúkoľvek operáciu navyše. Presne takýmto spôsobom sme v našom balíku predefinovali makro *selectfont* v našom prostredí „encrypted". Okrem svojej pôvodnej funkcionality teraz navyše dokáže identifikovať a zaznamenať do log súboru nový aktuálne používaný font (kombinácia príkazov *fontname**font* zisťuje názov aktívneho fontu).

Poďme o úroveň nižšie, teda na úroveň TeXu. Tu sa dostávame k najzákladnejšiemu príkazu v súvislosti so zmenou fontu - príkaz *font*. Príkaz *font* je v skutočnosti primitíva (nie makro), čo v praxi znamená, že nie je expandované na ďalšie makrá, ale tvorí rozhranie medzi interpreterom a TEX kódom dokumentu. Všetky ostatné príkazy, ktoré nejakým spôsobom vykonávajú zmenu fontu v konečnom dôsledku používajú príkaz *font*. Nakoľko sa nám aj napriek maximálnemu úsiliu nepodarilo funkčne predefinovať alternatívny príkaz, nazvaný *bcfont*, ktorý má rovnakú funkcionalitu aj syntax ako príkaz *font*, ale zvláda aj naše rozšírené požiadavky rovnakým spôsobom, ako náš príkaz *selectfont*. Takéto riešenie síce so sebou prináša isté nevýhody, no nechceme sa spoliehať na nepredvídateľné následky predefinovania primitívy. Jednou z nevýhod je nutnosť poučiť užívateľa tohoto riešenia o používaní príkazu *bcfont* namiesto príkazu *font*. Ak by užívateľ použil *font* v prostredí „encrypted", táto zmena fontu by nebola zaregistrovaná a vo výslednom PDF dokumente by nebola nahradená svojou šifrovanou dvojičkou, čím by bol zredukovaný počet zmien šifry a v konečnom dôsledku by klesla úroveň bezpečnosti dokumentu.

Trojicu uzatvára príkaz *endgroup*, ktorý funguje na rovnakom princípe, teda zisťuje aktuálne použitý font na konci každej skupiny.

V pôvodnom pláne bolo aj predefinovanie parametra *everypar*, ktorý by

nám zabezpečil zmenu šifry na začiatku každého odstavca. Aj napriek vysokému úsiliu sa nepodarilo tento parameter korektne predefinovať, pretože je využívaný aj samotným LaTeXom, ktorý ho používa v rámci svojej vnútornej réžie a samotná dokumentácia LaTeXu ho neodporúča predefinovávať pre možné nepredvídateľné správanie.

3.4.1 Počítadlo (counter)

Naše prostredie „encrypted" obsahuje aj počítadlo, ktoré nám zaisťuje unikátne mená pre šifrované fonty. Vygenerovaný zoznam použitých fontov totiž väčšinou pri štandardných LaTeX dokumentoch obsahuje veľakrát za sebou rovnaký font. Meno šifrovaného fontu vznikne pridaním sufixu, ktorý obsahuje aktuálny stav počítadla, za pôvodné meno fontu. Následne je stav počítadla inkrementovaný. Týmto spôsobom je zaručené, že pri každej zmene fontu budeme schopní vygenerovať k aktuálnemu fontu jeho šifrovanú dvojičku s unikátnym menom.

3.5 Zavedenie šifrovaných fontov

Poslednou dôležitou úlohou nášho TEX balíku je zavedenie šifrovaných fontov po aktivovaní šifrovacieho módu dokumentu. Mohlo by sa zdať, že jedinou prácou navyše oproti normálnemu módu je po detekcii zmeny fontu ju nielen zapísať, ale ešte aj zavolať primitívu *font* a prostredníctvom nej zaviesť šifrovaný font, ktorý je už v tejto chvíli vygenerovaný a pripravený (generovanie šifrovaných fontov zabezpečuje program *pdfencrypt*, ktorý je produktom tejto práce a je mu venovaná celá ďalšia kapitola). Nastáva tu však menší problém - detekcia fontov v tomto prípade už neprebehne úplne rovnako, pretože sa v dokumente používajú fonty s inými názvami. Jediná vec, ktorá sa nezmení, sú pozície, na ktorých sa zmena detekuje. Riešením tohoto problému sú dve

špeciálne makrá, *fontorderbc* a *fontorderrawbc*. V normálnom móde sú obidve tieto makrá prázdne. Pred spracovaním *pdflatexom* v šifrovacom móde sú obidve tieto makrá dodefinované naším programom *pdfencrypt*, ktorý do nich vloží medzerami oddelené názvy fontov v takom poradí, v akom boli zdetekované v normálnom móde. Makro *fontorderbc* obsahuje názvy šifrovaných fontov a makro *fontorderrawbc* zase názvy pôvodných fontov. Obidve makrá fungujú na princípe fronty, takže z nich vieme vyberať ďalší font v poradí a následne ho odstrániť z fronty, čim pri detekcii zmeny fontu vieme zaviesť šifrovaný font so správnym menom. Po spracovaní dokumentu v šifrovacom móde sú obidve definície z dokumentu opäť odstránené.

V tejto kapitole sme zhrnuli všetky úlohy, ktoré musí náš TEX balík plniť aby sme dosiahli správnu kooperáciu s programom *pdfencrypt*. Práve týmto programom a všetkými jeho súčasťami sa budeme zaoberať v nasledujúcej kapitole.

Kapitola 4

Program pdfencrypt

4.1 Utility na konverziu formátu

Podstatnú úlohu v tejto bakalárskej práci zohrávajú aj rôžne utility na konverziu medzi metrickými fontovými súbormi a ich čitateľnejšou formou a tiež knižnica *kpathsea* na prehľadávanie adresárov. Viac o fontových metrických súboroch si povieme v následujúcich podkapitolách.

Autorom všetkých použitých utilít, okrem knižnice *kpathsea*, je Donald E. Knuth (tvorca TEXu) a všetky spomenuté utility sú voľne dostupné a použiteľné. Prvou utilitou, ktorou sa budeme zaoberať je TFtoPL, ktorej jedinou úlohou je spracovať fontový súbor s metrickými dátami a previesť ho do ľudsky čitateľnej formy, na takzvaný list vlastností (anglicky „property list").

4.1.1 Utilita TFtoPL

Fontové metrické súbory (ďalej TFM súbory) sú pre TeX dôležité, pretože v sebe obsahujú presné rozmery priestoru vyhradeného pre jednotlivé znaky a tiež popisujú, ako sa jednotlivé znaky majú vedľa seba správať pre dosiahnutie dokonalého vzhľadu a čitateľnosti výstupného textu. Aby sme však vedeli s týmito súbormi pracovať a modifikovať ich, musíme si ich najprv previesť

do ľudsky čitateľnej podoby.

Podľa [Knu08], pri navrhovaní štruktúry TFM súborov bol kladený dôraz najmä na ich kompaktnosť, preto sú v nich obsiahnuté dáta uložené v binárnej podobe ako sekvencie bytov. Utilita *tftopl* načíta TFM súbor, ktorý dostala na vstupe a najprv vykoná jeho validáciu. Algoritmus tejto validácie je takmer zhodný s validačným algoritmom samotného TeXu, ako nájdeme v [Knu08]. Pri súboroch, ktoré touto validáciou prejdú bez chýb, si môžeme byť istí, že sú skutočne korektné a aj TeX im bude správne rozumieť. Po validácii nasleduje konverzia do PL formátu, ktorá zahŕňa prepis hlavičkových informácii (rodina fontov, predvolené škálovanie veľkosti, vlastnosti rozmerov, atď.), vytvorenie tabuľky ligatúr a kerningu a nakoniec rozmery (výška a šírka) pre každý znak. O ligatúrach a kerningu si povieme viac v nasledujúcej podkapitole. V poslednom kroku sa takto vytvorené informácie zapíšu do PL (property list) súboru, ktorý môžeme otvoriť a upraviť v štandardnom textovom editore. Formát PL súboru je intuitívny a jednoducho strojovo analyzovateľný. Konkrétnu štruktúru PL súborov si pozrieme v následujúcich podkapitolách a tiež si detailne prejdeme zmeny, ktoré v týchto súboroch vykonáme.

4.1.2 Utility PLtoTF, VPtoVF

Formát PL súborov je vhodný len pre naše úpravy fontových dát, TeX mu nerozumie. Preto potrebujeme nami upravený PL súbor previesť naspäť do binárnej podoby, ktorej TeX už bude rozumieť. Pre tento účel slúži utilita PLtoTF, ktorá dokáže spraviť inverznú konverziu, teda z PL súboru naspäť vytvoriť TFM súbor.

Pre účely tejto práce však obyčajný PL súbor nestačí, preto budeme PL súbory pokladať za VPL súbory, teda virtuálne PL súbory, ktoré nám poskytujú širšiu funkcionalitu. PL súbory sú zároveň aj VPL súbory, ale opačne

to nemusí nutne platiť. VPL súbory nám umožňujú premapovať každý znak na nejaký iný znak bez toho, aby sme museli ručne prerábať jednotlivé metrické údaje. Ako príklad si môžeme uviesť premapovanie znaku "A" na znak "i". Ak by sme nevyužili funkcionalitu premapovania vo VPL súboroch, ale použili by sme len obyčajné PL súbory, museli by sme manuálne prehodiť metrické údaje znaku "i" pod znak "A". Taktiež by sme museli upraviť kerning pre znak "A" (zväčšovanie alebo zmenšovanie medzier medzi znakmi v súvislosti s lepšími vizuálnymi vlastnosťami) a aj všetky asociované ligatúry. V prípade VPL súboru nám stačí znaku "A" oznámiť, že sa má premapovať na znak "i" a už nič iné nemusíme riešiť. Presný popis využitia VPL súborov pre účely tejto práce môžeme nájsť v následujúcich podkapitolách. V tomto okamihu sa dostávame k využitiu utility VPtoVF, ktorej úlohou nie je nič iné, ako vrátiť VPL súbor do podoby, ktorej bude TeX rozumieť (TFM súbor) a zároveň vytvoriť VF (virtual font) súbor.

VPtoVF

Utilita VPtoVF je rozšírená verzia utility PLtoTF, ktorá dokáže spracovať mapovacie príkazy vo VPL súboroch a okrem TFM súboru, ktorý je potrebný pre TEX, vytvorí aj VF (virtual font) súbor, potrebný pre *pdflatex*. Vykonáva tiež dôkladnú validáciu VPL súboru, aby výsledný VF súbor bol skutočne korektný [Knu12b].

4.1.3 Knižnica kpathsea

Aby sme pochopili účel knižnice *kpathsea*, rozoberme si najprv adresárovú štruktúru (ďalej už len TDS - anglicky TEX directory structure). Koreňom TDS stromu zvyčajne býva adresár nazvaný `texmf` (v distribúcii TEXlive je nazvaný `texmf-texlive`), tento adresár môže byť umiestnený kdekoľvek v systéme, ale štandardne ho môžeme nájsť napríklad v `/usr/local`, alebo

`/usr/share`. V tomto adresári sa nachádzajú hlavné súčasti celého TEXu, napríklad makrá, fonty a iné pomocné súbory. Nás zaujíma najmä adresár `fonts`, v ktorom sa nachádzajú všetky nainštalované fonty.

V prvých verziách TEXu sa všetky fontové súbory rovnakého typu nachádzali v jednom adresári (napr. všetky TFM súbory sme mohli nájsť pod `texmf/fonts/tfm`). Takáto organizácia súborov bola postupom času neprehľadná a priniesla so sebou viaceré problémy ako napríklad komplikovanú údržbu a aktualizáciu fontov. Spolu s TDS štandardom sa zaviedlia aj štruktúra pre usporiadanie fontov v adresári `fonts`, konkrétna štruktúra vyzerá následovne:

`texmf/fonts/<`*`typ`*`>/<`*`dodávateľ`*`>/<`*`typ písma`*`>/`

`<`*`typ`*`>` môže byť napríklad:

- `afm` - metrické súbory pre Adobe fonty
- `source` - zdrojové súbory pre Metafont
- `tfm` - metrické súbory pre TEX
- `type1` - Adobe Type1 fonty (PFB alebo PFA formát)
- `vf` - virtuálne fonty (VF súbory)

Existujú aj ďalšie typy fontov, ale pre túto prácu nemajú až taký význam, preto ich nebudeme uvádzať. Bližšie informácie možeme nájsť v [Hoe98].

Parameter `<`*`dodávateľ`*`>` odkazuje na tvorcu fontov, typickými príkladmi sú `adobe`, `monospace`, alebo `public`, kde sa nachádzajú voľne redistribuovateľné fonty. Parameter `<`*`typ písma`*`>` je samotný názov písma, napríklad `cm` pre Computer Modern. Ak by sme chceli v TDS nájsť TFM súbor pre štandardne predvolený font v LATEXu - Computer Moder Romain, veľkosť 10pt

(`cmr10.tfm`), podľa uvedenej štruktúry by sme vedeli, že ho máme hľadať v tomto adresári: `texmf/fonts/tfm/public/cm/cmr10.tfm`.

Pre dosiahnutie cieľu tejto práce budeme potrebovať jednoducho vyhľadávať TFM súbory s metrickými údajmi a PFB súbory s grafickými inštrukciami. Jednou možnosťou je napísať si vlastný algoritmus na prehľadávanie adresárov podľa uvedených konvencií. Vstupom takéhoto algoritmu je názov súboru (napr. `cmr10.pfb`) a výstupom by mala byť absolútna cesta k tomuto súboru, prípadne oznámenie, že taký súbor neexistuje. Museli by sme vyriešiť rôzne problémy, ktoré by vznikali niekde medzi starou dokumentáciou a nedodržiavaním konvencií. Keď by sme sa konečne dopracovali k vyladenej funkčnej verzii, podarilo by sa nám znovu vynájsť koleso. Štandardná distribúcia TEXu - TEX live, ktorá bola použitá aj pri vývoji tejto práce, využíva na hľadanie zdrojov knižnicu *kpathsea*, ktorá bola presne pre tento účel vytvorená.

kpsewhich

Súčasťou tejto knižnice je utilita *kpsewhich*, ktorá funguje presne podľa našich potrieb: na vstupe dostane názov hľadaného zdroju a následne postupne prehľadá preddefinované adresáre, začínajúc lokálnym adresárom (`./`) a v prípade, že skončí s chybovým kódom 0, na výstupe dostaneme absolútnu cestu k hľadanému zdroju. V opačnom prípade môžeme podla chybového kódu zistiť, kde nastal problém.

4.2 Virtuálne fonty

V predošlej podkapitole sme sa dozvedeli, aké utility nám uľahčia prácu, ktorá by nám vznikla v prípade, že by sme museli riešiť konverziu medzi formátmi, alebo vyhľadávanie zdrojov sami. V následujúcich podkapitolách sa dozvieme, ako presne vyzerá štruktúra VPL súboru, ktorý nám utilita

tftopl vygenerovala a aké zmeny v obsahu tohoto súboru musíme vykonať, aby sme dosiahli šifrovanie fontu a mohli pomocou *vptovf* vytvoriť virtuálny font. V tejto podkapitole si priblížime koncept virtuálnych fontov a na príklade si ukážeme ich konkrétne využitie v praxi.

Z pohľadu TEXu je virtuálny font úplne obyčajný font, použiteľný v TEX dokumente štandardným spôsobom, ale v skutočnosti je to font poskladaný z jedného alebo viacerých iných fontov (ktoré môžu byť opäť aj virtuálne). Ako je možné tento koncept uplatniť v praxi ilustrujeme následovným príkladom. Pod pojmom font si velmi zjednodušene môžeme predstaviť tabuľku (lepšie povedané pole) indexovanú od 0 po 255 a ku každému indexu je priradený nejaký znak (napríklad znak „A" by mal podľa ASCII štandardu index 65). Vstupný TEX súbor nevie, že v sebe ukrýva písmeno „A", ale len znak s indexom 65. Potom príde napr. *pdflatex* a pozrie sa v PFB súbore na index 65 a zistí, aké grafické inštrukcie má použiť na vykreslenie tohoto znaku do výstupného PDF súboru. Ak by tu nenašiel inštrukcie pre vykreslenie písmena „A", ale napríklad písmena „f", vo výslednom PDF dokumente by všetky výskyty „A" vyzerali ako „f". Pre čísla a znaky malej aj veľkej abecedy sa takmer vždy môžeme spoľahnúť, že príslušné znaky nájdeme pod správnymi indexami, avšak nie je to zaručené. Závisí to len na slušnosti tvorcu konkrétneho fontu. V praxi sa problémy vyskytujú najmä v prípade takzvaných expertných fontov, ktoré obsahujú neštandardné znaky, ako napríklad ligatúry.

Predstavme si následujúcu situáciu: zakúpime si font od nejakej spoločnosti X a dostaneme od nich dva rôzne PFB súbory. Spoločnosť X nepatrí medzi vyznávačov štandardov a preto používajú úplne odlišné kódovanie. V prvom, hlavnom PFB súbore, nájdeme všetky štandardné znaky, ako čísla, abecedu, interpunkčné znamienka a pod, avšak sú indexované od 0 - najprv malá abeceda, potom veľká abeceda, následovne čísla a nakoniec všetky zvyšné znaky. Takto kódovaný font určite nebude mať pod indexom

65 inštrukcie na vykreslenie písmena „A”. V druhom, expertnom fonte, nájdeme všetky špeciálne znaky, ako ligatúry, &, %, $ a pod. Pri použití takéhoto fontu v našom dokumente dochádza hneď k niekoľkým katastrofám. Tou závažnejšou by bol kompletne nezrozumiteľný výstupný PDF dokument, nakoľko by bežné znaky boli vykreslené nesprávnymi inštrukciami. Menej závažným problémom by bolo použitie akéhokoľvek znaku z expertného fontu, pretože by sme tento expertný font museli najprv explicitne zaviesť do dokumentu. Presne v takejto situácii nám prichádzajú na pomoc virtuálne fonty. Z pohľadu autora dokumentu pôjde o obyčajný font, takže nebude musieť nijako meniť svoj vstupný dokument a to ani v prípade, že by chcel použiť ligatúry, alebo niektorý zo špeciálnych znakov expertného fontu. Do virtuálneho fontu dokážeme povyberať a správne poprehadzovať znaky z nášho hlavného a expertného fontu. V prvom rade prehodíme čísla a abecedu tak, aby ich ordinálne hodnoty boli v súlade s ASCII štandardom a potom sa môžeme inšpirovať napríklad fontami Computer Modern, a podľa nich dosadiť zvyšné znaky z expertného fontu, napríklad ligatúra ff by patrila pod index 11 v našom virtuálnom fonte. Takto vytvorený virtuálny font zohráva úlohu medzivrstvy, ktorá umožňuje autorovi dokumentu využívať viacej rôzne kódovaných fontov ako jeden štandardne kódovaný font.

Využitie virtuálnych fontov pre účely tejto práce bude podobné. V našom prípade bude virtuálny font maskovať vždy len jeden klasický font a žiadaná je práve zmena poradia znakov.

4.3 Štruktúra virtuálneho fontu

Z predošlých podkapitol už máme určitú predstavu o tom, ako fungujú virtuálne fonty, a ako TFM súbory prekonvertovať na ľudsky čitateľné VPL súbory (virtual property list), z ktorých spätne vytvoríme TFM súbor a virtuálny font (VF súbor). Poďme sa bližšie pozrieť na štruktúru VPL súborov

a povieme si aj o spôsobe, akým ich budeme v našom programe strojovo analyzovať.

Ako príklad môžeme použiť súbor `cmr10.tfm`, ktorý obsahuje metrické údaje k fontu Computer Modern Romain, veľkosť 10pt. Pomocou utility *tftopl* preložíme súbor `cmr10.tfm` na `cmr10.vpl` a po jeho otvorení na prvých riadkoch uvidíme (ukážka 4.1):

```
(FAMILY CMR)
(FACE O 352)
(CODINGSCHEME TEX TEXT)
(DESIGNSIZE R 10.0)
(COMMENT DESIGNSIZE IS IN POINTS)
(COMMENT OTHER SIZES ARE MULTIPLES OF DESIGNSIZE)
(CHECKSUM O 11374260171)
(FONTDIMEN
   (SLANT R 0.0)
   (SPACE R 0.333334)
   (STRETCH R 0.166667)
   (SHRINK R 0.111112)
   (XHEIGHT R 0.430555)
   (QUAD R 1.000003)
   (EXTRASPACE R 0.111112)
   )
```

Obr. 4.1: Hlavičkové údaje VPL súboru

Nakoľko štruktúra VPL súborov spĺňa charakter dobrého uzátvorkovania jednotlivých vlastností, strojová analýza tohoto formátu je skutočne jednoduchá a úplne postačuje jeden prechod celým súborom. Výsledkom analýzy je zoznam, obsahujúci hierarchicky usporiadané všetky nájdené vlastnosti.

Prejdime naspäť k štruktúre VPL súboru. Na prvom riadku si môžeme všimnúť atribút `FAMILY`, ktorý pomenuváva rodinu, do ktorej patrí analyzovaný font, v tomto prípade ide o rodinu CMR - Computer Moder Romain. Kurzíva aj tučné fonty môžu mať rovnakú rodinu, ale odlišujú sa následujúcim atribútom - `FACE`. Tento atribút môže nadobúdať hodnoty medzi 0 až 255 v decimálnej sústave. Hodnota `O 352` indikuje kódovanie v osmičkovej sústave a po prevedení do decimálnej sústavy dostaneme hodnotu 234. Atribút `CODINGSCHEME` popisuje aké kódovanie je použité v analyzovanom fonte. Tento atribút je ignorovaný TEXom, ale môžu ho využívať iné programy. `DESIGNSIZE` určuje preddefinovanú veľkosť fontu, keď sa v TEXu explicitne nenastaví škálovanie fontu. Táto veličina je určená v jednotke pt (points). `COMMENT` obsahuje textový reťazec a nemá žiadnu zvláštnu funkciu, je to obyčajný komentár. `CHECKSUM` je využívaný na identifikáciu verzie fontu. Posledný je atribút `FONTDIMEN`, obsahujúci zoznam ďalších atribútov ktoré priamo korešpondujú s TEXovskými *fontdimen* parametrami. Ich význam nie je podstatný pre túto prácu, preto sa nimi nebudeme bližšie zaoberať. Detailnejšie informácie ohľadom uvedených atribútov môžeme nájsť v [Knu12a]. Prejdime k ďalšej dôležitej časti VPL súboru - k tabuľke ligatúr a kerningu.

V typografii pod pojmom ligatúra rozumieme zviazanie dvoch, alebo viacerých znakov do jedného znaku pre dosiahnutie lepšieho vizuálneho efektu. Kerningom nazývame upravovanie (zväčšovanie alebo zmenšovanie) medzier medzi jednotlivými znakmi opäť pre lepšie a prirodzenejsie vizuálne vlastnosti a lepšiu čitateľnosť. Tieto hodnoty nájdeme práve v tabuľke ligatúr (ukážka 4.2).

Začiatok tabuľky označuje atribút `LIGTABLE`, obsahujúci zoznam ďalších atribútov. Atribút `LABEL` nám určuje znak, ktorého vzťahy vzhľadom na okolité znaky ideme popisovať. V tomto, a aj v následujúcich atribútoch si môžeme všimnúť, že hodnota atribútu sa určuje dvomi spôsobmi: môže byť v tvare `C <znak>` alebo `O <číslo>`. Prvý tvar popisuje znak priamo (`C f`

znamená, že sa ideme zaoberať znakom „f"), zatiaľ čo druhý tvar popisuje znak podľa jeho ordinálnej hodnoty v osmičkovej sústave. Po atribúte `LABEL` následujú atribúty `LIG`, alebo `KRN`. `LIG` popisuje, s ktorými znakmi tvorí v našom prípade písmeno „f" ligatúru a na aký znak sa má takéto zoskupenie prepísať. Konkrétne `LIG C i O 14` spôsobí, že ak sa hneď po písmene „f" vyskytne písmeno „i", tak sa táto dvojica prepíše na znak s ordinálnou hodnotou 12 (číslo 14 v osmičkovej sústave). Pri detailnejšom pohľade na štruktúru PFB súborov v závere tejto kapitoly uvidíme, že pod znakom s hodnotou 12 nájdeme inštrukcie na vykreslenie ligatúry fi.

```
(LIGTABLE
   ...
   (LABEL C f)
   (LIG C i O 14)
   (LIG C f O 13)
   (LIG C l O 15)
   (KRN O 47 R 0.077779)
   (KRN O 77 R 0.077779)
   (KRN O 41 R 0.077779)
   (KRN O 51 R 0.077779)
   (KRN O 135 R 0.077779)
   (STOP)
   ...
   )
```

Obr. 4.2: Tabuľka ligatúr vo VPL súbore

Atribút `KRN` popisuje kerning vo vzťahu k inému znaku. Napríklad `KRN O 51 R 0.077779` v tomto prípade vloží 0.077779 jednotiek voľného priestoru za písmeno „f" v prípade, že za ním bude nasledovať znak „)". Veľkosť

medzery môže byť aj negatívna pre dosiahnutie menšej medzery. Po tabuľke ligatúr prichádzajú na rad atribúty popisujúce rozmery samotných znakov - metrické údaje. Pre tento účel slúži atribút `CHARACTER` (ukážka 4.3), ktorého hodnotou je buď priamo popisovaný znak, alebo jeho ordinálna hodnota v osmičkovej sústave, zapísaná v rovnakom tvare, ako v predošlých prípadoch. Obsahuje tiež zoznam ďalších vlastností - výšku a šírku obdĺžnika, ktorý tento znak zaberie vo výslednom dokumente (atribúty `CHARWD` a `CHARHT`). Atribút `COMMENT` obsahuje len prekopírované údaje o kerningu a ligatúrach z tabuľky ligatúr, čisto pre informačné účely. Podrobnejšie informácie k VPL súborom môžeme nájsť priamo v poznámkach Donalda Knutha v [Knu12a], kde sa navyše môžeme dozvedieť aj viaceré detaily ohľadom histórie virtuálnych fontov.

```
(CHARACTER C K
   (CHARWD R 0.777781)
   (CHARHT R 0.683332)
   (COMMENT
      (KRN C O R -0.027779)
      (KRN C C R -0.027779)
      (KRN C G R -0.027779)
      (KRN C Q R -0.027779)
      )
   )
```

Obr. 4.3: Metrické údaje pre písmeno „K” vo VPL súbore

4.4 Zašifrovanie textu pomocou virtuálneho fontu

V predošlých kapitolách sme si vysvetlili koncept virtuálnych fontov a tiež sme si ukázali, ako vyzerá štruktúra VPL súborov, z ktorých budeme virtuálne fonty vytvárať. V tejto kapitole si ukážeme, ako vytvoríme šifrovaný font `cmr10e`, teda šifrovanú verziu fontu `cmr10`. Najprv potrebujeme vytvoriť VPL súbor `cmr10e.vpl` z TFM súboru `cmr10.tfm` pomocou utility *tftopl*. Spomenuli sme, že virtuálny font je vrstva nad iným fontom a preto si hneď na začiatok VPL súboru musíme zadefinovať, ktorý font chceme maskovať. Poďme sa pozrieť na atribút `MAPFONT`.

```
(MAPFONT D 0 (FONTNAME cmr10))
(FAMILY CMR)
(FACE O 352)
(CODINGSCHEME TEX TEXT)
(DESIGNSIZE R 10.0)
...
```

Obr. 4.4: Zmenené hlavičkové údaje VPL súboru, dodefinovaný maskovaný font

Atribút `MAPFONT` slúži na definovanie fontov, z ktorých bude virtuálny font vychádzať. Jeho hodnotou je číselný index, podľa ktorého sa budeme neskôr na tento font odkazovať a okrem toho obsahuje zoznam vlastností, ktorého súčasťou je atribút `FONTNAME`, popisujúci meno maskovaného súboru, v našom prípade má hodnotu `cmr10`. Pridaný `MAPFONT` do hlavičkových údajov VPL súboru môžeme vidieť v ukážke 4.4.

Prichádzame k prvej časti šifrovania fontov - k náhodnému prekódovaniu

znakov na iné znaky. Chceme dosiahnuť aby sa náhodne zmenili ordinálne hodnoty jednotlivých znakov, čím zabezpečíme, že skopírovaný text z nášho dokumentu sa bude javiť ako nezmyselný, pretože znaky sú pri kopírovaní interpretované podľa ich ordinálnych hodnôt. Ak napríklad znaku „A" priradíme hodnotu 66, tak pri kopírovaní textu z nášho dokumentu sa znak „A" bude správať ako znak „B", čo je presne želané správanie. Prekódovanie ordinálnych hodnôt znakov vieme dosiahnuť úplne jednoducho využitím virtuálnych fontov, pretože presne pre tento účel boli vymyslené. Bez nich by sa nejednalo o triviálnu záležitosť, pretože by nestačilo vykonať len jednoduché prekódovanie znakov, ale museli by sme ručne riešiť aj metrické údaje, ligatúry a kerning tak, aby to sedelo s prekódovanými znakmi.

Prejdime k samotnému prekódovaniu znakov a k zmenám, ktoré musíme spraviť v atribúte `CHARACTER`, ktorého fungovanie sme si vysvetlili v predošlej podkapitole.

Ako môžeme vidieť v ukážke 4.5, v atribúte `CHARACTER` nám pribudol atribút `MAP` obsahujúci ďalšie dva atribúty `SELECTFONT` a `SETCHAR`. Pomocou atribútu `SELECTFONT` vyberieme font pomocou indexu, ktorý sme si na zadefinovali na začiatku VPL súboru a potom atribútom `SETCHAR` určíme, na aký znak chceme znak „K" prekódovať (opäť buď priamo, alebo pomocou ordinálnej hodnoty v osmičkovej sústave).

4.5 Štruktúra Adobe Type1 fontu

O Adobe Type 1 fontoch (ďalej už len T1 font) sme si v krátkosti povedali v úvodnej kapitole. T1 font súbory sú dôležité až pri tlačení dokumentu (či už na fyzickej tlačiarni, alebo na virtuálnej - do PDF dokumentu). Ich najpodstatnejšou súčasťou sú PostScript inštrukcie, pomocou ktorých tlačiareň vie, ako vyzerajú jednotlivé znaky a ako ich korektne vytlačiť.

V predošlých podkapitolách sme si vysvetlili, ako pomocou virtuálnych

```
(CHARACTER C K
   (CHARWD R 0.777781)
   (CHARHT R 0.683332)
   (COMMENT
      (KRN C O R -0.027779)
      (KRN C C R -0.027779)
      (KRN C G R -0.027779)
      (KRN C Q R -0.027779)
      )
   (MAP
      (SELECTFONT D 0)
      (SETCHAR C A)
      )
   )
```

Obr. 4.5: Prekódovanie písmena „K” na písmeno „A”

fontov prekódujeme jednotlivé znaky na iné znaky, napríklad znak „A” na znak „f”. Bez akejkoľvek ďalšej práce by sme pomocou takto upravených virtuálnych fontov dostali vo výslednom PDF dokumente znak „f” na všetkých miestach, kde sa vo vstupnom dokumente vyskytoval znak „A”. Ak by sme dokázali „oklamať” našu tlačiareň (v tomto prípade *pdflatex*) a presvedčili by sme ju, že znak „f” v skutočnosti vyzerá presne ako znak „A”, vo výslednom PDF dokumente by sme znak „A” našli korektne vykreslený, ale jeho ordinálna hodnota by bola zhodná so znakom „f”. Tento „trik” dokážeme dosiahnuť vhodnou úpravou príslušného PFB súboru (PFB súbor je binárna forma T1 súboru, existuje aj PFA súbor, ktorý namiesto binárnych dát obsahuje tieto dáta v hexadecimálnej forme).

Predtým, ako sa pustíme do upravovania PFB súboru, pozrieme sa najprv

zbežne na jeho štruktúru, avšak bližšie sa budeme zaoberať len tými časťami, ktoré sú dôležité pre túto prácu.

T1 font je organizovaný do troch hlavných sekcií:

1. Prvá ASCII sekcia obsahujúca hlavičkové údaje o fonte, ako meno fontu, verzia a kódovací vektor (Encoding array), ktorý asociuje ordinálne hodnoty znakov s názvami glyfov.

2. Šifrovaná sekcia, obsahujúca slovník (CharStrings dictionary), v ktorom môžeme nájsť názvy glyfov a k nim prislúchajúce PostScript inštrukcie pre ich vykreslenie. Táto sekcia je šifrovaná pomocou šifrovacieho algoritmu `eexec`, o ktorom si neskôr povieme viac. Ako sme spomenuli v úvodnej kapitole, T1 font nebol vždy otvorený formát a práve toto šifrovanie znemožňovalo jeho voľnú úpravu.

3. Druhá ASCII sekcia označujúca koniec T1 fontu.

T1 font väčšinou nájdeme len v dvoch formátoch - PFB formáte a PFA formáte. Jediný rozdiel medzi týmito formátmi je spôsob, akým je uložená šifrovaná sekcia T1 fontu. V PFB súbore sú to čisté binárne dáta, zatiaľ čo v PFA súbore je každý bajt uložený v hexadecimálnej podobe, teda reálne sú na jeho uloženie potrebné dva bajty. Výhodou PFB formátu je jeho kompaktnejšia veľkosť, nevýhodou je nemožnosť upraviť hlavičkové údaje v prvej ASCII sekcii v obyčajnom textovom editore. Vzhľadom na miešanie ASCII a binárnych dát v PFB súboroch boli zavedené isté označenia jednotlivých sekcií pre jednoduchšiu analýzu PFB súborov.

Spomenuli sme, že T1 font je zostavený z troch sekcií. V prípade PFB formátu obsahuje každá z týchto sekcií svoju hlavičku pozostávajúcu zo šiestich bajtov. Formát hlavičky je nasledovný:

1. Prvý bajt hlavičky je vždy `0x80`

2. Druhý bajt popisuje typ sekcie, *1* označuje ASCII sekciu a *2* označuje binárnu sekciu

3. Následujúce štyri bajty kódujú veľkosť sekcie vo formáte *little-endian*.

Za druhou ASCII sekciou v PFB súbore môžeme nájsť špeciálnu hlavičku označujúcu koniec PFB súboru, jej veľkosť sú len dva bajty. Prvý bajt je zhodný s prvým bajtom ostatných hlavičiek (*0x80*) a druhý bajt má hodnotu 3, teda označuje koniec súboru. Pomocou týchto hlavičiek je veľmi jednoduché pri analýze PFB súboru správne rozdeliť tento súbor na jednotlivé sekcie a vyhnúť sa tak pomiešaným ASCII a binárnym dátam.

4.5.1 Prvá ASCII sekcia

Prejdime si najprv podstatné časti prvej ASCII sekcie. Táto sekcia je už v ľudsky čitateľnej forme, preto sa rovno môžeme pozrieť, aké údaje v sebe obsahuje. Ako príklad opäť použijeme font `cmr10`.

V jazyku PostScript označuje znak `%` komentár a kazdý PostScript súbor musí začať komentárom `%!`, čím je jasne identifikovateľný. Nakoľko T1 font je tiež PostScript program, aj v tomto prípade musí byť na začiatku prvej ASCII sekcie komentár `%!`. Za ním následuje identifikácia fontu a jeho verzia. Zvyšné hlavičkové údaje nemajú pre túto prácu zvláštny význam, preto ich nebudeme ďalej skúmať. Podrobný popis týchto parametrov môžeme nájsť v samotnom manuáli k Type 1 fontom priamo od Adobe [Inc90]. Zaujímavý pre nás je až kódovací vektor (Encoding array), ktorý priraďuje k ordinálnym hodnotám znakov názvy glyfov. Začína príkazom `/Encoding` a následne pokračuje riadkami v tvare `dup` *`<ordinálna hodnota znaku>`* *`<názov glyfu>`* `put`. V následujúcej podkapitole si povieme o probléme, ktorý tento vektor spôsobuje a aj o zmenách, ktoré musíme v tomto vektore vykonať, aby sme problém odstránili a naše riešenie bolo funkčné.

```
%!PS-AdobeFont-1.0: CMR10 003.002
...
%%EndComments

/FontType 1 def
/FontInfo ...
/Encoding 256 array
dup 0 /Gamma put
...
dup 196 /dieresis put
readonly def
currentdict end
currentfile eexec
```

Obr. 4.6: Hlavičkové údaje T1 fontu

4.5.2 Binárna sekcia

Prejdime teraz k binárnej sekcii T1 fontu. Ako sme už spomenuli, táto sekcia je šifrovaná pomocou `eexec` algoritmu, preto si najprv poďme v krátkosti niečo povedať o tomto algoritme.

eexec šifrovanie

Eexec šifrovanie je obyčajná bloková šifra, ktorá pracuje s blokom o veľkosti jedného bajtu. Módom tejto šifry je CFB (Cipher Feedback), čo v praxi znamená, že na vygenerovanie následujúceho kľúča je použitý predchádzajúci bajt šifrového textu. Eexec šifra je aplikovaná na celú binárnu sekciu, preto každý bajt binárnej sekcie je považovaný za bajt šifrového textu. Presný popis šifrovacieho a dešifrovacieho algoritmu, rovnako ako hodnotu inicializačného

vektora môžeme nájsť v [Inc90], kde je tejto téme venovaná celá kapitola. primárnym cieľom eexec šifrovania bolo dosiahnuť kompaktnú veľkosť výsledného PFB súboru a tiež ochrániť hinty (krátky popis hintov sme uviedli v úvodnej kapitole) pred zbežným prezeraním.

Vráťme sa k štruktúre binárnej sekcie. Po aplikovaní dešifrovacieho eexec algoritmu dostaneme binárnu sekciu T1 fontu v takmer ľudsky čitateľnej podobe. Slovo takmer je použité preto, lebo časti tejto sekcie ostávajú šifrované druhou vrstvou šifrovania, takzvaným `charstring` šifrovaním. Nakoľko naše modifikácie binárnej sekcie nevyžadujú dešifrovanie aj na tejto úrovni, nebudeme sa týmto šifrovaním bližšie zaoberať.

Binárna sekcia obsahuje privátny slovník (Private dictionary), v ktorom nájdeme pre nás zaujímavý `Charstring dictionary`, teda slovník, v ktorom sú ako kľúče použité názvy glyfov a hodnotami sú šifrované reťazce bajtov, obsahujúce inštrukcie na vykreslenie týchto glyfov. Spojením kódovacieho vektora (opísaného v predošlých častiach) a tohoto slovníka dostávame prepojenie medzi ordinálnymi hodnotami znakov a inštrukciami na ich vykreslenie, takže nám všetky časti začínajú do seba zapadať. Pozrime sa na štruktúru `Charstring dictionary` v ukážke 4.7:

```
/CharStrings 132 dict dup begin
...
/C 41 RD <41 bajtov s PS inštrukciami> ND
...
```

Obr. 4.7: Charstring dictionary

Číslo 132 za príkazom `/CharStrings` hovorí, že v tomto slovníku sa nachádza *najviac* 132 glyfov. Štruktúra jednotlivých záznamov je jednoduchá,

najprv je uvedený názov glyfu (napríklad `/C`), potom je číselne vyjadrená dlžka reťazca bajtov, ktorý obsahuje inštrukcie na vykreslenie konkrétneho glyfu. Následuje príkaz `RD` a za ním bajtový reťazec uvedenej dĺžky a záznam je ukončený príkazom `ND`.

Štruktúra PFB súboru je oveľa bohatšia, ale zvyšné nepopísané časti nie sú relevantné pre cieľe tejto práce. Pre prípadný záujem o podrobnejšie detaily k T1 fontom odkazujeme cteného čitateľa priamo na manuál k Adobe Type 1 fontom [Inc90].

4.6 Modifikácia Adobe Type1 fontu - zamaskovanie textu

Dostali sme sa k poslednej dôležitej časti, týkajúcej sa vytvárania špeciálnych šifrovaných fontov - k úprave PFB súborov takým spôsobom, aby sme dosiahli správne vykreslenie znakov vo finálnom PDF súbore a zároveň zmenu ich ordinálnej hodnoty, ktorá je dôležitá pri ich kopírovaní mimo PDF súboru.

V PFB súboroch musíme vykonať dva typy zmien, prvým typom je úprava kódovacieho vektora tak, aby sme odstránili problém, o ktorom si hneď povieme a druhým typom je správne poprehadzovanie vykreslovacích inštrukcií glyfov.

4.6.1 Kolízie

Ako sme si uviedli v predošlej podkapitole, kódovací vektor má na starosti mapovanie medzi ordinálnymi hodnotami znakov a názvami glyfov a nachádza sa v prvej ASCII sekcii PFB súboru. Po podrobnejšom preskúmaní tohoto vektora si môžeme všimnúť, že niektoré ordinálne hodnoty sa zobrazujú na rovnaké názvy glyfov. V prípade fontu `cmr10` si tento jav môžeme všimnúť napríklad pri znakoch s hodnotami 0 a 161, obidva sa zobrazujú na glyf

`/Gamma`. Teraz si predstavme postup prekódovania znakov na úrovni virtuálnych fontov, ilustrovaný v predošlých podkapitolách. Prekódujme znak „a" na znak s hodnotou 0 a prekódujme znak „b" na znak s hodnotou 161. Pri prehadzovaní vykreslovacích inštrukcií v binárnej sekcii PFB súboru musíme zabezpečiť presne opačné zobrazenie, teda pod znak s hodnotou 0 dosadiť inštrukcie na vykreslenie znaku „a" a pod znak s hodnotou 161 dosadiť inštrukcie na vykreslenie znaku „b". Takéto niečo však nie je možné, pretože znak 0 a aj znak 161 sa vykreslujú rovnakými inštrukciami uloženými pod názvom `/Gamma`.

Aby sme sa vyhli takýmto kolíziam, najjednoduchším riešením v tomto konkrétnom prípade je namapovať znaky 0 a 161 na dva rôzne glyfy, napríklad `/Gamma1` a `/Gamma2`. Týmto krokom sme vlastne odstránili glyf `/Gamma` a nahradili sme ho dvomi novými glyfmi `/Gamma1` a `/Gamma2` a preto s každým odstránením kolízie z kódovacieho vektora musíme patrične upraviť aj `Charstring dictionary` v binárnej sekcii PFB súboru.

Po úplnom odstránení kolízií z PFB súboru prichádzame k finálnej fáze, ktorou je správne poprehadzovanie inštrukcií v `Charstring dictionary`. Správnym poprehadzovaním rozumejme inverzné prekódovanie k prekódovaniu na úrovni virtuálnych fontov. Ak sme vo virtuálnom fonte prekódovali znak „S" na znak „k", tak pod ordinálnu hodnotu znaku „k" musíme presunúť inštrukcie na vykreslenie znaku „S".

Binárnu sekciu takto upraveného PFB súboru spätne zašifrujeme eexec algoritmom a dostávame posledný kúsok skladačky.

4.7 Program pdfencrypt

V poslednej podkapitole prejdeme celý postup, ako sa od vstupného TEX dokumentu pomocou programu *pdfencrypt*, ktorý vznikol ako produkt tejto práce, dopracujeme až k šifrovanému PDF dokumentu. Inštaláciou, konfigu-

ráciou a použitím tohoto programu sa budeme zaoberať v záverečnej kapitole tejto práce.

Jediným vstupom celého algoritmu je samotný TEX dokument. Ako sme spomínali v kapitole o TEX balíku, vstupný dokument má dva módy, v ktorých vie fungovať. Implicitne je aktívny normálny mód, ktorý pri spracovaní vypisuje na výstup všetky zmeny fontov v dokumente. Prvým krokom je získanie tohoto zoznamu zmien fontov, preto najprv necháme spracovať vstupný TEX dokument predkonfigurovanému programu, ako napríklad *pdflatex*, a vykonáme analýzu výstupu tohoto programu. Nakoľko zmeny fontov majú presne definovanú a ľahko vypátrateľnú štruktúru, získanie zoznamu zmien fontov do nejakej dátovej štruktúry je triviálne. Štruktúru výpisu zmien fontov môžeme nájsť v kapitole o TEX balíku.

Následne pomocou knižnice *kpathsea* nájdeme a spracujeme súbor `pdftex.map`, obsahujúci mapovanie medzi TFM a PFB súbormi. Toto mapovanie uchováme v pamäti formou asociatívneho poľa (trieda `Hash` v jazyku Python).

V tomto bode máme pripravené všetko, aby sme mohli začať generovať šifrované fonty. Postupne prechádzame zoznamom nájdených zmien fontov. Zmena fontu je len informácia, aký bol pôvodný font a na aký font sa zmenil. Pre nový font pomocou knižnice *kpathsea* nájdeme príslušné TFM a PFB súbory. Z TFM súboru pomocou utility *tftopl* vygenerujeme VPL súbor a podľa uvedených princípov ho prekódujeme podľa náhodnej permutácie. Využijeme utilitu *vptovf*, ktorá nám vygeneruje dvojicu súborov, TFM súbor a zároveň VF súbor pre prekódovaný font. Následuje dešifrovanie a patričná úprava PFB súboru, teda správne pophrehadzovanie vykreslovacích inštrukcií a spätné zašifrovanie. Tento postup zopakujeme pre všetky detekované zmeny fontov a vygenerujeme tak sériu šifrovaných fontov.

Po príprave všetkých potrebných zdrojov (šifrované fonty) môžeme pristúpiť k prepnutiu módu dokumentu z normálneho do šifrovacieho a opäť nechať

spracovať vstupný dokument programom, ako napríklad *pdflatex*. Ten však v dôsledku šifrovacieho módu bude brať do úvahy šifrované fonty a nahradí nimi tie pôvodné, takže vo výstupnom PDF dokumente budú použité nami vygenerované šifrované fonty.

Ostáva dodržať pravidlá slušnosti a poupratovať po sebe - vymazať všetky vygenerované súbory, okrem výstupného PDF dokumentu, pravdaže. Výstupom kompletného algoritmu je rovnaká sada súborov, ako v prípade štandardného *pdflatexu*, v prípade, že bol použitý na spracovanie dokumentu práve ten (pre slovenské, alebo české dokumenty je možné použiť napríklad *pdfcslatex*, o tejto možnosti sa viac dozvieme v záverečnej kapitole tejto práce).

Kapitola 5

Manuál

Záverečná kapitola bude kratšia, nakoľko jedným z cieľov tejto práce bolo vytvoriť jednoducho použiteľný a ľahko konfigurovateľný nástroj. V tejto kapitole nájdeme všetky potrebné informácie pre správne nainštalovanie, konfiguráciu a použitie programu *pdfencrypt*. Prejdime k inštalácii.

5.1 Inštalácia

Program *pdfencrypt* je naprogramovaný v jazyku Python, verzia 2.7. Preto by sme sa mali najprv uistiť, že sa na našom stroji nachádza Python interpreter pre túto verziu. Náš program využíva voľne dostupné utility *tftopl*, *vptovf* a *kpsewhich*, ktoré je potrebné mať nainštalované pred spustením (tieto utility by mali byť automaticky dostupné so štandardnou distribúciou TEX Live).

Na priloženom CD sa nachádza priečinok `pdfencrypt`, ktorý obsahuje všetky zdrojové kódy k programu *pdfencrypt*. Hlavný skript sa nachádza v súbore `pdfencrypt.py` v tomto priečinku. Celá inštalácia spočíva v skopírovaní tohoto priečinku do preferovaného adresára. Pre jednoduché používanie odporúčame vytvoriť symbolický link s názvom *pdfencrypt* napríklad v `/usr/bin`. Tento link bude odkazovať práve na hlavný skript `pdfencrypt.py`.

5.2 Konfigurácia

Konfigurácia tohoto programu je skutočne jednoduchá, jediný nastaviteľný parameter je program na spracovanie dokumentu, ktorý je implicitne nastavený na *pdflatex*. Túto hodnotu môžeme nastaviť v súbore `config.py` pod kľúčom `RENDERER`. Hodnotou je reťazec obsahujúci príkaz na spracovanie dokumentu. Program *pdfencrypt* bol testovaný najmä s *pdflatexom* a *pdfcslatexom*. Názov dokumentu ja dosadený namiesto reťazca `%s`.

5.3 Návod na použitie

Návod na použitie má dve časti. V prvej časti potrebujeme správne upraviť vstupný dokument - zaviesť náš TEX balík. Ideálne miesto je v preambule dokumentu, teda pred príkazom $\backslash begin\{document\}$. Náš balík zavedieme štandardným LaTeXovým príkazom $\backslash usepackage\{encrypted\}$. Nesmieme zabudnúť, že na to, aby bol LaTeX schopný nájsť náš balík, musí sa nachádzať buď v rovnakom priečinku, ako vstupný dokument, alebo v štandardnom adresári v TDS (TEX Directory Structure). Náš TEX balík sa nachádza na priloženom CD v priečinku `tex_package`, meno súboru je `encrypted.sty`. Tento súbor je potrebné prekopírovať do našeho pracovného adresára, kde sa nachádza aj náš vstupný dokument. Potom už len ostáva v dokumente označiť šifrované časti pomocou prostredia `encrypted`.

Pre vytvorenie šifrovaného PDF dokumentu stačí zavolať príkaz *pdfencrypt [názov_dokumentu.tex]*. Náš program sa postará o zvyšnú prácu.

Existujú dokumenty, ktoré pre vytvorenie správneho výstupného PDF dokumentu potrebujú viac ako jeden prechod *pdflatexom*. Ani takéto dokumenty nespôsobujú žiadny problém, jednoducho stačí spustiť program *pdfencrypt* toľkokrát, koľko je potrebné.

Záver

V závere tejto práce sa poďme pozrieť na dosiahnuté výsledky. Naším hlavným cieľom bolo vytvoriť jednoducho použiteľný nástroj, pomocou ktorého by sme dokázali zamedziť kopírovaniu obsahu PDF dokumentov vytvorených v sadzbovom programe LaTeX. Museli sme sa pri tom držať zásadného kritéria - vizuálny vzhľad šifrovaného dokumentu sa nesmel líšiť od jeho nešifrovanej verzie.

Cieľ tejto práce sa nám podarilo naplniť vytvorením programu *pdfencrypt*, ktorý implementuje myšlienku nášho riešenia. Metóda šifrovania fontov sa formovala postupne, prekonávaním najrozličnejších prekážok. Na konci však priniesla žiadaný výsledok a umožnila dosiahnuť zvýšenú bezpečnosť výstupného PDF dokumentu bez akéhokoľvek vizuálneho zásahu.

V práci spomíname aj alternatívne metódy pre dosiahnutie určitej miery bezpečnosti, ale všetky spomenuté alternatívne metódy sú buď určené na iné prípady použitia, alebo so sebou prinášajú nepraktické nevýhody.

Druhotným cieľom tejto práce bola aj snaha autora získať prehľad o vnútornom fungovaní TEXu a LaTeXu, tento cieľ sa tiež podarilo naplniť. Preskúmali sme rôzne oblasti tohoto sadzbového systému, najdetailnejšie sme prešli cez oblasti virtuálnych fontov a Adobe Type 1 fontov, ktoré s TEXom úzko súvisia. Počas realizácie sme prekonali viaceré výzvy a úskalia, z ktorých sa nám väčšinu aj podarilo zdolať. Jedinou neprekonanou prekážkou zostalo predefinovanie TEX primitívy *font*, ktorá pri predefinovaní spôsobovala problémy, ktoré sa nepodarilo aj napriek maximálnej snahe a niekoľkým prebdeným nociam odstrániť.

Aj napriek tejto prekážke je finálny produkt tejto práce - program *pdfencrypt* a TEX balík *encrypted* funkčný a úspečný celok, pripravený na použitie.

Príloha A - Slovník pojmov

glyf - vizuálny vzhľad znaku alebo písmena
utilita - program s presne určeným účelom, nástroj
TFM súbor - TEX Font Metrics súbor, súbor s metrickými údajmi o fonte
PFB súbor - PostScript Font Binary súbor - binárna verzia Adobe Type 1 fontu
PFA súbor - PostScript Font ASCII súbor - ASCII verzia Adobe Type 1 fontu
PL súbor - Property List súbor - súbor obsahujúci list vlastností, alebo atribútov
VPL súbor - Virtual Property List súbor
VF súbor - Virtual Font súbor

Literatúra

[Hoe98] Alan Hoenig. *TeX Unbound: LaTeX & TeX strategies for fonts, graphics & more*. Oxford University Press, Inc., 1998.

[Inc90] Adobe Systems Incorporated. *Adobe Type 1 Font Format*. Addison-Wesley Publishing Company, Inc., 1990.

[Knu08] D. E. Knuth. The TFtoPL processor. `https://www.tug.org/texlive/devsrc/Master/texmf-dist/doc/generic/knuth/texware/tftopl.pdf`, 2008. [Online; accessed 17-April-2012].

[Knu12a] D. E. Knuth. Virtual Fonts. `http://ftp.cstug.cz/pub/tex/CTAN/info/knuth/virtual-fonts`, 2012. [Online; accessed 21-April-2012].

[Knu12b] D. E. Knuth. VPtoVF. `http://www.tex.ac.uk/ctan/systems/knuth/dist/etc/vptovf.web`, 2012. [Online; accessed 17-April-2012].

[Sys12a] Adobe Systems. Adobe Font Formats. `http://www.adobe.com/type/topics/info9.html`, 2012. [Online; accessed 17-February-2012].

[Sys12b] Adobe Systems. PostScript vs. PDF. `http://www.adobe.com/print/features/psvspdf/index.html`, 2012. [Online; accessed 25-February-2012].

[Unw] Mike Unwalla. Latex: an introduction. http://www.techscribe.co.uk/ta/latex-introduction.pdf.

[Wika] WikiBooks. Latex. http://en.wikibooks.org/wiki/LaTeX.

[Wikb] Wikipedia. Postscript fonts. http://en.wikipedia.org/wiki/PostScript_fonts.

[Wik12] Wikipedia. TeX. http://en.wikipedia.org/wiki/TeX, 2012. [Online; accessed 05-March-2012].