OCR of Cryptographic Source Code

Karl Nyberg
Grebyn Corporation
P. O. Box 47
Sterling, VA 20167-0047
karl@nyberg.net
http://karl.nyberg.net
703-406-4161

What I’m Presenting

- Pretty Good Privacy (PGP)
- Optical Character Recognition
- Cost / Benefit of Increased Scanning Resolution
- An Ada95 “application”

What I'm **NOT** Presenting

- New architectural approaches to software development, model-based or otherwise
- Advances in pattern recognition
- Solutions to global warming, world hunger, AIDS, voting in Florida, SARS or the MidEast crises

Obligatory Disclaimer

- Companies mentioned in this presentation are used only for representative purposes and are not meant to imply an endorsement
- On the other hand, I do have financial investments in several of those companies… ☺

Outline

- PGP
- OCR
- The Ada Application
- Results

PGP Background

- What is PGP?
- Why was the source code published?
- What was the result?

What is PGP?

- PGP - Pretty Good Privacy
- A Public Key Encryption program
- Written in 1991 by Phil Zimmerman and released by various means over the years

Why was the source code published? (from the FAQ)

- Make the source code available
- Encourage posts to other platforms
- Remove doubts about the legal status of PGP outside USA / Canada
- Show how stupid the US Export Regulations were

What was the solution?

- An exception in the law allowed for export of printed matter: “A printed book or other printed material setting forth encryption source code is not itself subject to the EAR (see Sec 734.3(b)(2))”
- Lead to the development and publication of “Tools for Publishing Source Code”

More about “Tools…”

- Printed using fixed width OCR-B font
- Special consideration for unprintable characters (spaces, tabs, etc.) and for dealing with line wrapping
- Per-line CRC-16 checksums, with running CRC-32 checksums
- Per-page CRC-32 checksums
- Included training pages

What happened?

- Grand Jury Investigation
- Book reconstruction

Grand Jury Investigation

- Interviewed PKZ, ViaCrypt and Austin Code Works (1993)
- Eventually dropped (January 1996)

Book Reconstruction

- Printed
- Exported
- Scanned
- OCR’d
- Corrected

OCR Background

- What is OCR?
- How does it work?
- How was it applied here?

What is OCR?

- OCR – Optical Character Recognition
- A subfield of Pattern Recognition
- As some have said, “A printer in reverse”
- Takes an image of a page of text and returns the text

How does it work?

- Image acquisition (a scanner)
- Big array of bits (monochrome, grayscale, color)
- “Pre-processing” (deskew, salt / pepper noise removal, text / graphics separation, forms removal, column separation, language identification)
- Component identification
- Component classification
- Output and “post-processing”

Image Acquisition

- Scanning on an HP ScanJet IICX at various resolutions (200, 300, 400 DPI) in monochrome with ADF into TIFF files
- Manual rescanning of skewed images

Some Cost "Parameters"
Time / Space

- Scan times
  - 200 DPI – 19 seconds
  - 300 DPI – 28 seconds
  - 400 DPI – 42 seconds
- Scan sizes
  - 200 DPI – < ½ MB (469294 bytes)
  - 300 DPI – 1 MB (1054047 bytes)
  - 400 DPI – 1.7 MB (1872086 bytes)

Pre-processing

- No text / graphics separation or other pre-processing required
- Skew eliminated by rescan

Component identification

- Estimated “noise” threshhold based upon scanning resolution
- Component identification by connected component analysis
- Components grouped by line segmentation (based upon bounding boxes) and sub-component merge

Sample components

```
                               **           **
                              ******        ****
                             ********      *****
                             *********     ****
                             **** ****    *****
    **********               **** ****    *****
   **************            **** ****   *****
  ***************            *********   ****
  ****************           *********  *****
 *******    ******            *******  *****
 *****       ******            ****    *****
              *****                   *****
            *******                   ****
     **************                  *****
   ****************                 *****
  *****************                 *****
 ******************                *****
*******      ******                ****
******       ******               *****
******      *******              *****
******      *******              *****   ****
******     ********             *****  ********
*******************             *****  ********
 ******************            *****  **********
  ****************            ******  ****  ****
    *******   ****            *****   ****  ****
                             *****    **** *****
                             ****     *********
                             ****      ********
                              **        ******
```

More Cost Parameters Image Analysis

- Time to perform connected component analysis and line segmentation:
  - 200 DPI – 6 seconds
  - 300 DPI – 10 seconds
  - 400 DPI – 17 seconds

Ada / Design Issues

- TIFF Parsing
- Data Representation and Storage

Component Classification

- Classification based upon feature extraction (height, width, various moments, position relative to baseline, number of “bits”, etc.)
- Limited field “validation” (CRC-16 line checksums, page headers for example)
- Simple ASCII output of “best” candidate

Design Issue
Classification Approach

- Various options considered
  - Template (overlay and compare) Matching
  - Neural networks
  - Feature vectors
    - Exemplar (best match) selection
    - Average values
    - Classification trees (for performance)

More Cost Parameters Component Classification

- Time to perform classification (average)
  - 200 DPI – 10 seconds
  - 300 DPI – 12 seconds
  - 400 DPI – 14 seconds

Training The System

- Available training data included
- Automatically trained

Design Issue
Training Style

- Automatic v. Manual
  - Required pin-for-pin accuracy with character segmentation
  - Doesn’t address component glyphs
- Compiled v. Flat File
  - Extra step in “production” process – could be hidden from the end user
  - Performance improvement, approximately n % (classic space-time tradeoff – increased executable size by y %)

Meta-application

- Image data
- Accuracy measurement
- Line reconstruction
- Performance
- Sizing

Image Data

- Table of Pages

Volume	Training	Test
Tools for Publishing Source Code via OCR	10	85
Pretty Good Privacy 5.5 Platform-Independent Source Code Volume 1	6	446

Design Issue
Where do you keep all this data?

- Page structures
- Components and bounding boxes
- Line structures
- Feature data
- Interrelationships among the above
- Purpose of the data

Accuracy Measurement

- Character accuracy
  - Feedback on ground truth (training data)
  - By count required to match CRC
  - Levenshtein metric (edit distance)
- Line accuracy – CRC16 checksum
- Page accuracy – CRC32 checksum

Ground Truth

- By resolution (training data)
  - 200 dpi
    - Tools - 99.918% (missed 43 out of 52941)
    - Volume 1 - 99.914% (missed 29 out of 33743)
  - 300 dpi
    - Tools - 99.989% (missed 7 out of 52941)
    - Volume 1 - 99.985% (missed 5 out of 33738)
  - 400 dpi
    - Tools - 99.989% (missed 7 out of 52941)
    - Volume 1 - 99.985% (missed 4 out of 33743)

Line Reconstruction

- Consider secondary and tertiary, etc. candidates for reconstruction with CRC-16 checksum
- Running CRC-32 on input stream for additional reconstruction confirmation and page checking
- CRC-16 checked by increasing the number of candidates as a function of the relative scores and deviations of the candidates
- Terminate CRC-16 when CRC-32 fails

Example Candidates

```
cd2a1e sub Fatsl¤      { Qcleanup(); ·print STDERR @_; ·exit(1); }
 b e a xwh EsIeL:      ( &CLasoaD)!_ `Pc1oI s?ED#N G-_ `sc1I)i(_ )
 6   c eak PuZa(_      I 8[(seun#l(J .#F!xZ 2YO[NE Q!J .oz)Zlj!J I
       zU6 [ezo!"      t @I!ouxwR!1| -R!(uz GfCCGP #=1 _a=(Y!)}! {
       cnd hofx||      ) Rt|wweoP{}1 _9¤)sc %toeQ# R:! -wsjT{!1| f
       aoR rxcz).      } $DcxossEIi! ~h|je? EISkSD B|| ~xe{f|]i1 ]
       oeG 1w?wc!      f 0o)nmaUS|/) 'Dn_af a!XKES S') 'no<{I?ji >
```

Example
Character Substitution

```
 cd2a1e sub Fatsl¤      { Qcleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { Qcleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { QCleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { Qcleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { QCleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { Q[leanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { &cleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { &Cleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { &[leanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { Qcleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { QCleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { Q[leanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { &cleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { &Cleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatsl¤      { &[leanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatel¤      { Qcleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatel¤      { QCleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatel¤      { Q[leanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatel¤      { &cleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatel¤      { &Cleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatel¤      { &[leanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatal¤      { Qcleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatal¤      { QCleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatal¤      { Q[leanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatal¤      { &cleanup(); ·print STDERR @_; ·exit(1); }
cd2a1e sub Fatal¤      { &Cleanup(); ·print STDERR @_; ·exit(1); }
```

Character changes required to pass CRC

- Tools
  - 200 DPI - 1510
  - 300 DPI - 275
  - 400 DPI - 126
- Volume 1
  - 200 DPI - 20794
  - 300 DPI - 9941
  - 400 DPI – 711

More Cost Parameters
Time to Incorporate CRC

- Calculate and test for CRC
  - 200 DPI – 13.7 seconds (v. 10)
  - 300 DPI – 13.5 seconds (v. 12)
  - 400 DPI – 14.1 seconds (v. 14)

Levenshtein Metric

- "A measure of the similarity between two strings"
- Based upon the edit distance, or the number of insertions, deletions and substitutions required to change one string into the other
- Sometimes also called the "string-to-string correlation problem"
- Less sensitive to "inserted / deleted" characters than character-by-character comparison

Levenshtein Metric (cont.)

- Example:
  - “cat” -> “bat” has a distance of 1
  - “cant” -> “bat” has a distance of 2
  - “therefore” -> “pinafore” has a distance of 5
  - “xyzzyxy” -> “yzzyxxyx” has a distance of 3

Levenshtein Metric (cont.)

- Tools:
  - 200 DPI - 1186 v 315 (with CRC updates)
  - 300 DPI - 234 v 67 (with CRC updates)
  - 400 DPI - 181 v 73 (with CRC updates)
- Volume 1:
  - 200 DPI - 7906 v 5908 (with CRC updates)
  - 300 DPI - 4230 v 2990 (with CRC updates)
  - 400 DPI - 643 v 313 (with CRC updates)

Line Accuracy

- 200 dpi
  - Tools – 292, with CRC - 2285
  - Volume 1 – 903, with CRC - 9107
- 300 dpi
  - Tools – 4634, with CRC - 5808
  - Volume 1 – 4764, with CRC - 19711
- 400 dpi
  - Tools – 5286, with CRC 5782
  - Volume 1 – 17264, with CRC 27127

Page Accuracy

- 200 dpi
  - Tools - 2%, with CRC - 92%
  - Volume 1 – 1.8%, with CRC 37%
- 300 dpi
  - Tools - 66%, with CRC - 92%
  - Volume 1 - 13%, with CRC - 70%
- 400 dpi
  - Tools - 81%, with CRC - 93%
  - Volume 1 - 53%, with CRC - 91%

Performance

- Average single page recognition time
  - 200 DPI – 19.1 seconds
  - 300 DPI – 23.7 seconds
  - 400 DPI – 30.9 seconds
- Includes image parsing, connected component analysis, component merging, line segmentation, feature extraction, classification and CRC-assisted output

Sizing

- About forty source files, including data analysis tools.
- About eight thousand lines (+34,000 when generated feature tables are compiled in)
- Twenty second compilation on 1.2GHz Pentium 4, Red Hat 8.0 (35 with tables)
- Hours of reading images, calculating features, building feature vector tables, …

Comparison with Export Effort

- Scanning was similar (ADF, rescan)
- OCR was MAC Omnipage
  - Manual Training
  - Omnipage-specific bias to the correction toolset
- Correction - ½ to 4 hours manual effort per 100 pages, 7500 pages, about 150 hours
- Total – two people, roughly 100 hours each

Comparison (cont.)

- Discounting development effort, …
- Approximately 500 pages (v. 7500)
- Manual correction of 12 pages (estimate 200 for all six volumes)
- Would take roughly 4 hours manual effort after scanning

Observations

- Ada – it’s not just for embedded systems ☺
- Benefits of CRC and alternate character considerations combination – great!
- Flat file format painfully slow – consider the implications when going to plain XML

Future Plans

- Infrastructure for pattern recognition work, like building decision trees, neural networks, other pre- and post-processing algorithms
- Other similar documents – DES Cracker
- Other languages, non-CRC documents (e.g., utilizing secondary candidates for spell-checking)