Scott Williams

Computer Science Part II Project Dissertation

STEGANOGRAPHIC FILE SYSTEMS WITHIN VIDEO FILES

Christ's College
University of Cambridge

December 18, 2014

Performa

Name:	Scott Williams
College:	Christ's
Project Title:	Steganographic file systems within video files
Examination:	Part II of the Computer Science Tripos
Year:	2015
Word Count:	12,000
Project Originator:	Scott Williams
Project Supervisor:	Daniel Thomas

Original Aims of the Project

To investigate appropriate steganographic embedding methods for video and to develop a practical steganographic software package to enable the embedding of arbitrary data within video files via a file system interface. Raw AVI video files should be supported and a variety of steganograhpic embedding algorithms should be available. Basic file system commands should work within the presented logical volume and embedding should occur with no perceivable impact on video quality.

Work Completed

A complete software package has been developed enabling the embedding of arbitrary files within many video formats (including MP4 and AVI) via a file system interface. A total of 9 steganographic embedding algorithms are supported, along with encryption and plausible deniability functionality. Basic file system operations work as expected within the mounted volume and the embedding process can operate without any perceivable impact on video quality. Performance of the system is adequate for general use allowing high definition media content to be played directly out of the embedded volume.

Special Difficulties

None.

Declaration of Originality

I, Scott Williams of Christ's College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

I give permission for my dissertation to be made available in the archive area of the Laboratory's website.

Signed:

Date:

Contents

1 Introduction **1**

1.1 Motivation 1

2 Preparation **2**

2.1 Background 2

2.1.1 Steganographic Concepts 2

2.1.2 Steganalysis 4

2.1.3 The AVI file format 4

2.1.4 JPEG compression 6

2.2 Existing tools 6

2.3 Choice of Languages and Tools 6

2.4 Requirements Analysis 6

2.4.1 Core Requirements 6

2.4.2 Possible Extensions 6

3 Implementation **6**

3.1 Introduction 6

3.2 Filesystem 6

3.3 Steganographic Algorithms 6

3.4 Extensions 6

4 Evaluation **6**

4.1 Satisfaction of Requirements 6

4.2 Correctness 6

4.3 Security 6

4.4 Performance 6

5 Conclusions **6**

5.1 Future Project Directions 6

1 ‖ Introduction

Steganography is the art of hiding information in apparently innocuous objects. Whereas cryptography seeks to protect only the content of information, steganography attempts to conceal the fact that the information even exists. This allows steganographic methods to be utilised in countries where encryption is illegal for example, or within the UK where keys for identified encrypted data can be forced to be handed over.

In this project I design and implement a practical steganographic software application - `Stegasis` - which enables users to embed arbitrary files within videos via a file system interface. `Stegasis` can operate with no perceivable impact on video quality and can achieve embedding capacities of upto 200% of the video size. A wide range of video formats are supported[1] along with several steganographic embedding algorithms. Standard encryption algorithms can be used to further protect embedded data and plausible deniability functionality protects users even when the presence of embedded data has been confirmed.

Steganogaphic methods operating on video have had comparatively little attention compared to images and audio. As such, there are few programs currently available which allow data to be steganographically hidden within video. `Stegasis` is the first application to enable the embedding of arbitrary files within videos via a file system interface.

1.1 Motivation

Digital media is ubiquitous on the Internet and high definition video content is now common place on video sharing and social networking websites. Video files of multiple gigabytes in size can reside on users devices without arousing suspicion, providing an ideal hiding place for large collections of sensitive files. Few programs are capitalising on this fact, and those that are, allow the user to embed only a singe chosen file into a specific single video format. As with `TrueCrypt`[2], I believe that a practical system for protecting sensitive files should present the user with a mounted logical volume allowing the use of standard file system operations to create, access and organise embedded data. Furthermore, there exist many commonly used video formats along with many more currently in development. As such, a steganographic program operating on one single video format not only greatly restricts usability, it will require constant development as new video formats inevitably become more popular. Instead, a generic solution applicable to a variety of video formats is preferred.

The many recent global surveillance disclosures show that using certain technologies related to privacy can get you "flagged" by authorities such as the `NSA` - it is no longer the case that simply encrypting data is enough to keep the owner safe.

[1] Including many modern video formats such as `MP4`, `MKV`, `FLV` and `AVI`.

[2] A successful cryptographic program providing on-the-fly encryption and full disc encryption.

2 ‖ Preparation

2.1 Background

In this section steganographic background material, definitions and concepts are introduced. The `AVI` file structure and `JPEG` compression are also discussed. The most important property of any steganographic system is undetectability, that is, it should be impossible to differentiate between ordinary and steganographically modified objects. This requirement is famously formulated within Simmons' prisoners' problem.

Alice and Bob are imprisoned in separate cells and wish to formulate an escape plan. They are allowed to communicate, but all messages must pass through a warden Eve. If Eve suspects the prisoners of secretly communicating their escape plan, the communication channel will be severed and Alice and Bob thrown into solitary confinement. The prisoners attempt to utilise steganograhpy to exchange details of their plan undetected. The steganographic system is considered broken if Eve is able to detect the presence of hidden messages within the prisoners exchanges. It is assumed that Eve has a complete knowledge of the steganographic algorithm being used, with the exception of the stego key, which Alice and Bob have agreed upon beforehand. This is in parallel with Kerckhoff's principle used within cryptography. The warden can be considered one of three categories; *passive*, *active* and *malicious*. A *passive* warden does not modify the exchanged messages in any way, whereas an *active* warden may modify the messages whilst maintaining their original meaning. For example an *active* warden may replace words with synonyms or reorder sentences. If images are being used as a transport medium then an *active* warden may recompress or crop the images. A *malicious* warden attempts to break the steganograhpic system and impersonate the prisoners in an attempt to obtain information.

This project is concerned with only the case of the *passive* warden. As such, any modification of the video files once `stegasis` has embedded data within them, will most likely render the embedded file system corrupt[3].

2.1.1 Steganographic Concepts

A steganographic system consists of a number of individual components:

- A *Cover object* is the original object that the message will be embedded within.
- A *Message* is an arbitrary length sequence of symbols. For this project we consider a message $\mathcal{M} \in \{0,1\}^n$
- A *Stego key* is a secret key used within the embedding process.
- A *Stego object* is the result of embedding a message inside a cover object.

[3] This unfortunately means utilising video sharing websites such as YouTube and Facebook for distribution is not possible due to them performing compression upon video upload.

Definition 2.1. Steganographic System

Let $\mathcal{C}$ be the set of all cover objects. For a given $\boldsymbol{c} \in \mathcal{C}$, let $\mathcal{K}_{\boldsymbol{c}}$ denote the set of all stego keys for $\boldsymbol{c}$, and the set $\mathcal{M}_{\boldsymbol{c}}$ denote all messages that can be communicated in $\boldsymbol{c}$. A steganographic system[4], is then formally defined as a pair of embedding and extracting functions Emb and Ext,

$$Emb : \mathcal{C} \times \mathcal{K} \times \mathcal{M} \to \mathcal{C}$$
$$Ext : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$$

satisfying,

$$\forall\ \boldsymbol{c}, \boldsymbol{k}, \boldsymbol{m}.\ \boldsymbol{c} \in \mathcal{C}\ \wedge\ \boldsymbol{k} \in \mathcal{K}_{\boldsymbol{c}}\ \wedge\ \boldsymbol{m} \in \mathcal{M}_{\boldsymbol{c}} \Rightarrow Ext(Emb(\boldsymbol{c}, \boldsymbol{k}, \boldsymbol{m}), \boldsymbol{k}) = \boldsymbol{m}$$

Definition 2.2. Embedding Capacity

The Embedding Capacity (payload) $\mathcal{P}_{\boldsymbol{c}}$ for a given cover object $\boldsymbol{c} \in \mathcal{C}$ is defined in bits as,

$$\mathcal{P}_{\boldsymbol{c}} = \log_2 |\mathcal{M}(\boldsymbol{c})|$$

The relative embedding capacity $\mathcal{R}_{\boldsymbol{c}}$ for a given cover object $\boldsymbol{c} \in \mathcal{C}$ is defined as,

$$\mathcal{R}_{\boldsymbol{c}} = \frac{\log_2 |\mathcal{M}(\boldsymbol{c})|}{n}$$

where n is the number of elements in $\boldsymbol{c}$.

For example, consider $\mathcal{C}$ to be the set of all 512×512 greyscale images, embedding one bit per pixel gives $\mathcal{M} = \{0,1\}^{512\times512}$ and $\forall\ \mathbf{c} \in \mathcal{C}$. $|\mathcal{M}(\mathbf{c})| = 2^{512\times512}$. The embedding capacity $\forall\ \mathbf{c} \in \mathcal{C}$ is then $512 \times 512 \approx 33$kB as expected. In this case, n is equal to the number of pixels in $\mathbf{c}$ and therefore the relative embedding capacity is equal to 1 bpp (bits per pixel), again as expected.

Using the definitions above, we can define a simple expression for the embedding capacity of a video file.

Definition 2.3. Embedding Capacity for video

With $\mathcal{C}$ as the set of all video files, the embedding capacity $\mathcal{V}_{\boldsymbol{c}}$ for a given video $\boldsymbol{c} \in \mathcal{C}$ can be expressed as,

$$\mathcal{V}_{\boldsymbol{c}} = \sum_{f\ \in\ frames(\boldsymbol{c})} \mathcal{P}_f$$

Note that for certain embedding algorithms, the embedding capacity can depend on both the input data and the cover object[5]. However, in some cases the following expression is also valid,

$$\mathcal{V}_{\mathbf{c}} = |frames(\mathbf{c})| \times \mathcal{P}_{f_0}$$

[4]This is specifically steganography by cover modification.

[5]Many algorithms operating on `JPEG` images for example will not embedded within zero valued DCT coefficients.

Definition 2.4. Steganographic Capacity

The concept of Steganographic Capacity is loosely defined as the maximum number of bits that can be embedded within a given cover object without introducing statistically detectable artifacts.

For completeness, the least significant bit (LSB) of a given number is defined as follows,

$$\mathrm{LSB}(x) = x \bmod 2$$

2.1.2 Steganalysis

Steganalysis is the study detecting messages embedded using steganographic techniques; this is analogous to cryptanalysis applied to cryptography. A steganalysis attack is considered successful (i.e. the steganography has been broken) if it is possible to correctly distinguish between cover and stego objects with probability better than random guessing. Note that it is not necessary to be able to read the contents of the secret message to break a steganographic system.

A trivial example of steganalysis arises when the steganalyst has access to the original cover object used within the embedding procedure. By computing the difference between the stego and cover objects, the steganalyst can immediately detect the presence of a hidden message. This attack identifies a number of important points to consider when developing a practical steganographic system. Firstly, embedding within popular media content should be discouraged, as the cover object will be likely widely available. Secondly, if a user is embedding within original content, for example a video recorded by them, any copies of the original file should be securely erased after embedding.

Steganalysis methods can be split into two main categories, *Targeted Steganalysis* and *Blind Steganalysis*. *Targeted Steganalysis* occurs when the steganalyst has access to the details of the steganographic algorithm used for embedding. The steganalyst can accordingly target their activity to the specific stegosystem. On the other hand, if the steganalyst has no knowledge of the utilised steganographic algorithm, *Blind Steganalysis* techniques must be applied. In this project, *Targeted Steganalysis* attacks are developed for several of the proposed embedding algorithms.

2.1.3 The AVI file format

As specified within the project proposal, this project only looks at raw uncompressed `AVI` files. Furthermore, only `AVI` version 1.0[6] files are investigated and therefore supported natively[7] by `Stegasis`. Henceforward, references to "`AVI` files" specifically refer to version 1.0, uncompressed.

The `AVI` file format is a Resource Interchange File Format (`RIFF`) file specification developed by Microsoft and originally introduced in November 1992. The data within

[6] Not including the Open-DML extension (version 1.02).

[7] All other video formats (including compressed `AVI`) are supported via the use of `FFmpeg`, as described in section ???

RIFF files is divided into chunks and lists, each of which is identified by a FourCC tag. An AVI file takes the form of a single chunk in a RIFF formatted file, which is then subdivided into two mandatory lists and one optional chunk. The first sub-list is the file header containing metadata about the video (for example framerate, width and height). The second sub-list contains the actual audio/video data and the optional chunk indexes the offsets of the data chunks within the file.

We therefore have an AVI file laid out as follows, see the appendix for a more detailed expanded form.

```
RIFF ('AVI␣'
       LIST ('hdrl' ... )
       LIST ('movi' ... )
       ['idx1' (<AVI Index>)]
      )
```

Listing 2.1: AVI RIFF form

With a RIFF chunk being defined as follows:

```
struct CHUNK {
  char fourCC[4],
  int ckSize,
  char ckData[ckSize] // contains headers or video/audio data
};
```

Listing 2.2: RIFF chunk

And a RIFF list defined as:

```
struct LIST {
  char listCC[4], // Will always be the literal 'LIST'
  int listSize,
  char listType[4],
  char listData[listSize]
};
```

Listing 2.3: RIFF list

An AVI file consists of a number of data streams (usually 2, one for audio and one for video) interleaved within the movi list. Each stream is identified by a FourCC tag consisting of a two-digit stream number followed by a two-character code:

Two-character code	Description
db	Uncompressed video frame
dc	Compressed video frame
pc	Palette change
wb	Audio data

Table 2.1: AVI stream types

Each stream has a corresponding AVI stream header and format chunk within the above mentioned hdrl list. These data structures contain information about the stream including

the codec and compression used (if any). Specifically, the `fccHandler` field contains a `FourCC` tag that identifies a specific data handler. For raw uncompressed video this will equal 'DIB ' (Device Independent Bitmap). Any user provided `AVI` files with a fccHandler not euqal to 'DIB ', that is, the `AVI` contains compressed video, will at this point be rejected and an error message presented to the user.

Definition for movi chunk, top left corner, GBR calculate embedding cap...

2.1.4 JPEG compression
2.2 Existing tools
2.3 Choice of Languages and Tools
2.4 Requirements Analysis
2.4.1 Core Requirements
2.4.2 Possible Extensions
3 || Implementation
3.1 Introduction
3.2 Filesystem
3.3 Steganographic Algorithms
3.4 Extensions
4 || Evaluation
4.1 Satisfaction of Requirements
4.2 Correctness
4.3 Security
4.4 Performance
5 || Conclusions
5.1 Future Project Directions
References

[1] *Steganography in Digital Media*. Jessica Fridrich, 2010.