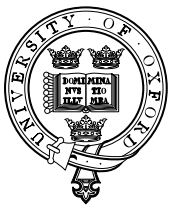
Information Hiding

(complete)

Andrew D. Ker

10 Lectures, Hilary Term 2016



Department of Computer Science, Oxford University

Contents

Chapter 0

Introduction

0.1 Preliminary Material

Advanced Security is a part C fourth year option for undergraduates in Computer Science or Mathematics & Computer Science, and a Schedule C option for the taught Masters in Computer Science. **Information Hiding** forms the second half of the Advanced Security course. There are approximately 10 lectures (three per week in weeks 1–3, one in week 4), two classes (weeks 3 & 5) and one practical (sessions in weeks 3–5).

Prerequisites

Basic discrete probability, including conditional probability and random variables. General background knowledge relating to computer security is assumed.

A small amount of basic linear algebra is used in chapter 3.

The practical work is in C; it does not use advanced techniques and all image handling is done using a standard library, so a basic understanding of the syntax should be sufficient. Memory management is minimal.

Familiarity with digital image formats would be an advantage, but they are covered (briefly) in the lecture notes.

Syllabus

Basic steganography definitions. Examples of hiding in digital images, both spatial and transform domain. Simple detection, advantages and disadvantages of various embedding operations. Hamming codes and wet paper codes, and their applications to hiding. Detection of hidden data via feature vectors, countermeasures. Theory of hidden information: the square root law in the case of i.i.d. discrete sources. Steganography and steganalysis with multiple sources.

Synopsis of Lectures

Steganography. Secret communication, the Prisoners' Problem and the Warden; types of steganography; aims of a cover-modification stegosystem; raw and JPEG digital images; examples of steganography in digital images: spatial-domain LSB replacement, LSB matching, and F5; variations.

Steganalysis. Aims of steganalysis, receiver operating characteristic; examples of structural attacks and payload size estimators on LSB replacement; attacks based on machine learning, example using simplified SPAM features; the wider setting.

Countermeasures. Improving embedding efficiency; syndrome codes, with example based on Hamming codes, theoretical bounds; solving the non-shared selection channel using syndrome codes, applications; optimal embedding.

Theory. Probabilistic models of covers and embedding; using Kullback-Leibler divergence to bound detector performance; the data processing theorem; the square root law for IID covers, and (without proof) extensions.

Reading Material

The main course text is

- "Steganography in Digital Media: Principles, Algorithms and Applications" by Jessica Fridrich (Fridrich, 2010).

A comprehensive presentation with a focus similar to ours: digital media covers, practical methods for cover modification, and detection. The theory section has been superceded by recent research, and the steganalysis section is also somewhat outdated now, but still a good introduction. Contains a lot more material than we can cover in this course. (£50)

Other recommended books are

- "Advanced Statistical Steganalysis" by Rainer Böhme (Böhme, 2010).

An adapted version of his thesis, the specific techniques are now out of date but the chapter on the principles of steganography and steganalysis is a very clear exposition. Does not cover the material from chapters 3-4 of these notes. It is, however, quite expensive. (£80)

- "Machine Learning in Image Steganalysis" by Hans Georg Schaathun (Schaathun, 2012).

Focuses on the steganalysis part, so its relevance is really only to chapter 2 of these notes, but a readable and comprehensive survey of the state-of-art as of four years ago. Unfortunately, seriously superceded by recent research. (£65)

I cannot recommend any other textbooks. There are quite a few published, but most of them are really poor, either covering ancient techniques now known to be insecure or material which is not really steganography at all. Instead, there will be a bibliography at the end of each chapter with references to relevant literature. Where possible, links to freely downloadable copies of the papers have been included.

A note on the literature: this field has matured enormously in the last ten years or so. Early literature may seem naive and use poor notation. It is easy to see this with hindsight! Recent papers will give a better view of the standards for contemporary research.

Course Website

The course page is at `http://www.cs.ox.ac.uk/teaching/courses/advsec/`. At the appropriate time the class exercises, practical manual, lecture slides and lecture notes, will all appear on the "course materials" page.

It will also contain important corrections. When you find errors in the course materials, please send corrections to `adk@cs.ox.ac.uk`.

0.2 Notation

Vectors will be written as boldface lowercase, $\boldsymbol{x}$, indexed as $\boldsymbol{x}[i]$. Indices start at zero or one, depending on which is most convenient. Matrices will be written as boldface uppercase, $\boldsymbol{M}$, indexed as $\boldsymbol{M}[i,j]$.

Some standard functions will be used: the logarithm $\log x$ denotes log to base e and $\log_2 x$ to base 2; rounding down to the nearest integer from x ("floor") is written $\lfloor x \rfloor$, and rounding to the nearest integer $[x]$; the sign function

$$\operatorname{sign}(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ -1, & \text{if } x < 0. \end{cases}$$

The cardinality (size) of a set S is written $\# S$. Tuples will be written (x, y).

0.3 Probability Primer

In case the reader is not familiar with probability, we survey the (very elementary) prerequisite knowledge. We assume that there is a set of events, each of which is assigned a probability in the range $[0, 1]$. We write $\mathrm{P}[A]$ for the probability of event A. If A and B are events then the intersection $A \wedge B$ indicates the event of both A and B, the union $A \vee B$ is the event of either A or B (or both), and the **conditional probability** of A given by B is

$$\mathrm{P}[A \mid B] = \frac{\mathrm{P}[A \wedge B]}{\mathrm{P}[B]}$$

(which is only defined for $\mathrm{P}[B] > 0$). The events $A_1, \ldots, A_n$ are called **independent** if $\mathrm{P}[A_1 \wedge \cdots \wedge A_n] = \mathrm{P}[A_1] \cdots \mathrm{P}[A_n]$.

Probabilities can be broken down using the **partition theorem**: if events $(A_1, A_2, \ldots)$ form a partition (precisely one of the events A_i always occurs) then

$$\mathrm{P}[B] = \sum_{A_i} \mathrm{P}[B \mid A_i] \, \mathrm{P}[A_i].$$

A **random variable** is a function from the set of events to a real number; informally, it is a random number. A **discrete** random variable is one which takes only countably many values, for example the integers; all random variables in this course will be discrete. A discrete random variable is completely defined by its **probability mass function**

$$p(x) = \mathrm{P}[X = x], \text{ for all possible values of } x.$$

Random variables will usually be given letters like X and Y, while x and y will be used to represent particular values of the random variables. The **mean** of a random variable is

$$\mathrm{E}[X] = \sum_x x \, \mathrm{P}[X = x],$$

where the sum is over all the possible values x of X, a conditional mean can be defined by

$$\mathrm{E}[X \mid A] = \sum_x x \, \mathrm{P}[X = x \mid A],$$

and the mean of any function of a random variable is conveniently given by

$$\mathrm{E}[f(X)] = \sum_x f(x) \mathrm{P}[X = x].$$

The variance of X is $\mathrm{Var}[X] = \mathrm{E}\Big[\big(X - \mathrm{E}[X]\big)^2\Big] = E\big[X^2\big] - E[X]^2$.

Random variables X, Y are **independent** if $\mathrm{P}[X = x \wedge Y = y] = \mathrm{P}[X = x] \, \mathrm{P}[Y = y]$ for all x and y. This definition can be extended to any set of random variables.

The mean (but not variance!) of a random variable has a **partition theorem**: if events $(A_1, A_2, \ldots)$ form a partition then

$$\mathrm{E}[X] = \sum_{A_i} \mathrm{E}[X \mid A_i] \, \mathrm{P}[A_i].$$

A random variable X is **uniform** over the set $\mathcal{X}$ if

$$\mathrm{P}[X = x] = \frac{1}{\# \mathcal{X}}, \text{ for all } x \in \mathcal{X}.$$

A random variable has the **binomial distribution** with parameters n and p, written $X \sim \mathrm{Bi}(n, p)$, if

$$\mathrm{P}[X = i] = \binom{n}{i} p^i (1 - p)^{n - i}$$

for $0 \leq i \leq n$. This has the physical interpretation of counting the number of n independent events, each of which happens with probability p. In this case the mean of X is np and its variance is $np(1 - p)$. It is useful to note that for any binomial random variable X,

$$\mathrm{Var}[X] \leq \tfrac{n}{4} \tag{0.1}$$

because $p(1 - p) \leq \frac{1}{4}$.

Often we need to prove that a random variable X is unlikely to take a very large, or very small, value. There are many **tail inequalities** for doing this, the simplest of which is **Chebyschev's inequality**:

$$\mathrm{P}\big[\,|X - \mathrm{E}[X]| \geq a\big] \leq \frac{\mathrm{Var}[X]}{a^2} \tag{0.2}$$

for any $a > 0$.

Bibliography

Böhme, R. (2010). *Advanced Statistical Steganalysis*. Springer.

Fridrich, J. (2010). *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge University Press.

Schaathun, H. G. (2012). *Machine Learning in Image Steganalysis*. Wiley.

Chapter 1

Steganography

Reading (course text):	Fridrich, chapters 1, 2, 4, §5.1
Alternatives & further depth:	Böhme, §2.1, 2.3–2.6, 2.7.1, 2.7.2 Schaathun, §2.1–2.2, 8.1

This chapter is about the art of hiding information, what we now call **steganography**[1]. We will briefly survey its history, and then provide a moderately formal definition, focusing on a particular case that is active in the literature and usable in practice: steganography by cover modification in digital media, against a passive Warden.

We will give a couple of simple cases of data hiding in images. They will function as running examples: in chapter 2 we will see how easily they can be detected, and in the following chapter show how they can be improved.

1.1 A Very Brief History of Secret Communication

Humans have needed to communicate securely for thousands of years, and until recently this tended to mean the same thing as communicating in secret. Early ciphers were completely insecure to an opponent who knew the cipher – even if they did not know

[1] στεγανός is Greek for *covered*, and γράφειν means *to write*.

the key, they could soon work it out – and so there are examples of hidden communication from very early history. The 4th century BC writer using the name Aeneas "The Tactician" Tacticus devotes a chapter of his book *How to Survive Under Siege* to secret communication, including:

> A message was once sent in the following manner. A book or some other document, of any size and age, was packed in a bundle or other baggage. In this book the message was written by the process of marking certain letters of the first line, or the second, or the third, with tiny dots, practically invisible to all but the man to whom it was sent: then, when the book reached its destination, the recipient transcribed the dotted letters, and placing together in order those in the first line, and so on with the second line and the rest, was able to read the message.[2]

Similar techniques based on selecting and decoding letters from a text designed to include the hidden message – acrostics, indicator dots, a physical mask, italic letters in early print, encoding of information in musical notation, etc. – were predominant for approximately the next two thousand years. In the 20th century scientific advances allowed invisible inks with specific chemical revealers, microdots, and ways of hiding the message so that it was not physically detectable at all.

We will not say more about the history of the subject here; see (Fridrich, 2010, §1.1) for some more historical examples, and David Kahn's keynote speech at the inaugural Information Hiding Workshop (Kahn, 1996).

1.2 The Prisoners' Problem

One might say that steganography officially entered the realm of computer science in 1983, when a paper by Simmons was published (Simmons, 1983). Simmons introduced the **Prisoners' Problem**, in which two separated prisoners – typically called Alice and Bob – wish to discuss an escape plan through a medium which is monitored by their enemy, the prison **Warden**. Simmons did not mention the word "steganography", but instead described a **subliminal** channel in which the secret information is undetectable amongst some apparently-innocent cover. In his paper the subliminal channel occurs in

[2]Translation taken from `http://www.aeneastacticus.net`, credited to L.W. Hunter and S.A. Handford, 1927.

square roots modulo large composite integers, and the undetectability is based on the difficulty, for the Warden, of determining that multiple possible square roots exist.

The Prisoners' Problem has become the classic analogy for steganography: we suppose that the Warden will take additional security measures (perhaps cutting off Alice's communications) if he detects that secret communication is taking place. We now use the terminology **stego objects** for the communications sent by Alice and **payload** for the hidden information that she wants to communicate. Like Simmons, we assume that Alice and Bob have been able to share a **secret key** before they were imprisoned. And we must adapt **Kerckhoffs' Principle**, which in cryptography states that the enemy should be assumed to know the system, by saying that the Warden is aware of Alice and Bob's communication algorithms[3]. But note that Kerckhoffs' Principle does not necessarily apply to the whole steganographic system. (Does the Warden know that, out of all the prisoners, only Alice might be using steganography? Do they know exactly which messages Alice might be using for payload, and which were simply genuine innocent communications? Do they have a complete and accurate model of her cover source?).

In fact, Simmons' problem is a long way from what we now call steganography. In his original paper, Alice and Bob wanted to authenticate each others' identity, and the aim of the Warden was to *change* the content of their hidden information so as to impersonate one of the parties and infiltrate their plot. In information hiding literature we now make the distinction between

- a **passive** Warden, who can only eavesdrop on Alice and Bob's channel, and whose aim is to detect the presence of hidden communication, and
- an **active** Warden, who can tamper will Alice's stego objects. Their aim is either to disrupt the payload so that Bob cannot recover it, or to impersonate Alice to Bob. The latter is sometimes called a **malicious** Warden.

Steganography research is currently concerned with, almost exclusively, the passive Warden case. That is also the focus of this course. For a little on active wardens, see section 1.6. Instead of mathematical covers such as Simmons' example – and it is difficult to construct a scenario in which Alice and Bob are legitimately exchanging large square roots – we focus on covers which are **digital media**, particularly images. Information hiding in digital media is the most active area of research (hiding in text has

[3]For this reason we do not consider the placement of information in unexpected places, such as file or IP headers, to be true steganography. Unfortunately, there are a number of publications on network packet/timing channel 'steganography' of this sort.

had relatively little success) and image files are the first step (mp3s and video files have too complex a structure to fit within the confines of a lecture course). We shall see that digital media has huge capacity for embedded payload. Furthermore, it is in digital media that illicit use of steganography is known to have occurred.

Finally, a word about the secret payloads. If Alice has any sense, the first thing she will do is losslessly compress, because smaller payloads are always easier to embed and harder to detect. Given that she shares a secret key with Bob, she should also encrypt the compressed payload (just in case the Warden ever does get hold of it). Both operations increase the apparent randomness (entropy) of the messages. It allows us to make the **random payload assumption**: the payload consists of a sequence of bits which are statistically indistinguishable from independent coin flips. We will carry this assumption throughout almost the entire course.

1.3 Types of Steganography

A steganographic communication channel from Alice to Bob begins with a source of covers for Alice. This might be a collection of digital images (or movies, mp3 files, etc.) which she already acquired, a camera with which she can take cover photographs, a live video stream, and so on. The type of cover will be determined by the situation in which Alice and Bob find themselves: it is the type of object that they can legitimately communicate.

One type of steganography is **cover selection**, in which Alice selects, or waits for, a cover which already contains the hidden message that she wants to send. This is only going to be reasonable if the hidden message is very short. For example, Alice might send 8 bits of information as the first 8 bits of a keyed hash of a digital image. The key for the hash is shared with Bob. On average it will take Alice 256 images until she finds one with the correct hash, and she sends that one. This is called a **rejection sampler**, an idea introduced by Hopper (Hopper et al., 2002). The advantage of this method is that the object Alice sends is completely natural, unaltered by embedding and therefore innocent by construction (as long as certain technicalities are taken care of).

The disadvantage is that the payload must be extremely small, otherwise Alice will need an infeasibly large library, or will have to wait a vast length of time, to find a suitable object. There is an entire branch of the steganography literature devoted to a complexity-theoretic analysis of techniques following Hopper. We will not take this approach because we consider it unrealistic in practice. As far as we know, nobody has publicly constructed such an embedding method for nontrivial payloads.

A variation on this idea is **cover synthesis**, in which Alice is able to construct a cover which contains hidden data of her choice. Although a few such schemes have been published[4] only a handful apply to digital media, and they are subject to weaknesses because it is very difficult to generate realistic cover examples. It is certainly possible to use a combination of rejection sampling and cover synthesis to send moderate payloads in, for example, mosaic images (each tile conveys a few bits) but there is no version for natural photographs.

The third type of steganography, which is practical for digital media and carries significant payloads, is **cover modification**. Here Alice takes a cover and makes subtle changes in order to embed the hidden payload. This technique is ideal for digital media, which consists of thousands or millions of data points and where small modifications can easily pass inspection. It is by far the most active type of steganography in published research, and there is good theoretical and practical work on both hiding and detection.

Let us formalise the cover modification paradigm as follows. Let $\mathcal{C}$ be the set of all objects of the same type as covers (e.g. digital images of a certain type and size), $\mathcal{K}$ the set of possible secret keys, and $\mathcal{M}$ the set of payloads that Alice might want to transmit. For now, the representation of these objects can be flexible: raw bitmap images might be represented as a vector of pixel intensities, JPEG images as a header concatenated with a vector of quantized DCT coefficients (see subsection 1.5.1), and keys and payloads can simply be sequences of bits.

Alice uses an **embedding function** to modify a cover into a stego object, given the payload and secret embedding key, and Bob uses an **extraction function** to recover the payload given the key. Note that Bob does *not* have access to Alice's original cover for his extraction (it would be make detection too easy if Alice's original cover could ever be seen by the Warden). The embedding and extraction functions have the type

$$\begin{aligned} \text{Emb} &: \mathcal{C} \times \mathcal{K} \times \mathcal{M} \to \mathcal{C}, \\ \text{Ext} &: \mathcal{C} \times \mathcal{K} \to \mathcal{M}. \end{aligned}$$

In fact, the messages which can be hidden in cover $\boldsymbol{c}$ may depend on $\boldsymbol{c}$ itself: even, for example, images with the same size might have different **capacity** (maximum possible payload lengths). Similarly it is conceivable that the set of possible keys depends on the cover as well. We will ignore these technicalities in our formal presentation, but must take care of them in implementations.

[4] Try `http://www.spammimic.com/` for an entertaining example.

A **stegosystem** consists of Alice's cover source, the embedding and extraction functions, the channel by which Alice sends stego objects to Bob, and (usually implicitly) the source of secret keys and payloads.

1.3.1 Aims of a Stegosystem

A cover modification stegosystem has a number of aims. The first is simply **correctness** of the secret communication channel: for all $\boldsymbol{c} \in \mathcal{C}$, $\boldsymbol{k} \in \mathcal{K}$, and $\boldsymbol{m} \in \mathcal{M}$,

$$\mathrm{Ext}(\mathrm{Emb}(\boldsymbol{c}, \boldsymbol{k}, \boldsymbol{m}), \boldsymbol{k}) = \boldsymbol{m}.$$

This is usually simple to verify. Some authors also talk about **robustness**, which is the ability of the payload to survive changes made to the stego object. This is relevant when there is an active Warden, but even with a passive Warden the payload has to be received correctly at the end of the channel through which that Alice sends stego objects. So in the case of steganography in lossy compressed images, some might say that the embedding has to be robust to the effects of lossy compression. In our formulation we do not consider this to be robustness, simply correctness bearing in mind the type of the stego object.

Another aim is **capacity**: $\#\mathcal{M}$ should be large. That is, we transmit as much secret information ($\log_2 \#\mathcal{M}$ bits) as possible, or as we need, in each cover. Capacity is usually measured relative to the size of the cover, a so-called **embedding rate**. We might talk of p secret bits per cover pixel, or p secret bits per nonzero DCT coefficient in the cover.

Capacity is in competition with the unique aim of steganography: informally, the embedding should be undetectable by a Warden. This is difficult to formalize, and we will not come to a mathematical formulation until chapter 4, but we can start by expressing the Warden's task in the language of statistics as a **hypothesis test**. The Warden is given an object $\boldsymbol{x}$ and knows (we assume, following Kerckhoffs' Principle) that one of two states holds:

(the **null hypothesis**)	H_0 :	$\boldsymbol{x}$ is a genuine cover
(the **alternative hypothesis**)	H_1 :	$\boldsymbol{x} = \mathrm{Emb}(\boldsymbol{c}, \boldsymbol{k}, \boldsymbol{m})$ for some unknown cover $\boldsymbol{c}$, key $\boldsymbol{k}$ and payload $\boldsymbol{m}$

The Warden has some **decision function** $D(\boldsymbol{x})$ which determines whether they give a positive detection of covert payload. The statistical terminology is to say that the Warden either accepts the null hypothesis, or rejects it in favour of the alternative. There are two mistakes he might make:

- Rejecting H_0 when H_0 was true. Statisticians call this a **type I error**, and steganographers call it a **false positive** or **false alarm** because the Warden detected something that was not happening. It is typical to write the probability of a false positive as α.
- Accepting H_0 when H_1 was true. Statisticians call this a **type II error**, and steganographers call it a **false negative** or **missed detection** because the Warden failed to detect steganography. It is typical to write the probability of a false negative as β.

The Warden will balance the sensitivity of their decision $D(\boldsymbol{x})$ according to whether they are more worried about false positive or false negative results, but either way it is their aim to keep α and β low. Therefore one way to formalise the **security** of a stegosystem is to say:

For all decision functions $D : \mathcal{C} \to \{\text{Positive}, \text{Negative}\}$, α and β are reasonably large.

Note that we are not truly saying that use of the stegosystem is completely undetectable, rather that it is not *reliably* detectable. The reason that the aims of capacity and undetectability are in competition is that, given constant embedding rates, the Warden can find a decision function such that α and β both tend to zero exponentially fast in the length of the hidden payload! There will be more on this in later chapters.

One way to avoid detection is to make few changes to the cover. So a measure of interest is the **embedding efficiency**, the number of payload bits hidden for each cover element changed. To be more precise, it is the number of bits hidden divided by the *average* number of changes needed. This is only a crude measure of security, because it does not take the magnitude of changes into account, nor their individual detectability, but it is still something to consider.

1.3.2 Secret Keys

The secret key, shared between Alice and Bob but unknown to the Warden, is crucial to the security of a stegosystem. Without it, no system could possibly survive Kerckhoffs' Principle, because the Warden could copy Bob and read the payload straight out. In cover modification steganography the key often has a straightforward interpretation: it determines where in the cover Bob will find the payload.

The following examples, like most steganography which does not use source coding (see chapter 3), will use the key in the same way. The stego object is represented as a list of elements (e.g. pixels) and the payload as a list of symbols (e.g. bits). One payload

symbol is embedded in one stego element, and the key determines the *order* in which the stego elements are modified. So Alice writes her payload symbols into the elements in the order specified by the key, stopping when the payload has been completed. We will see examples of the **embedding operations**, which perform the symbol-by-symbol embedding, in the following section.

In such a system, the key must determine a permutation of the cover indices. We cannot capture an entire permutation into a key, otherwise the key would very likely be longer than the secret payload being communicated, so instead the key functions as a seed to a pseudorandom number generator, and the stream of pseudorandom numbers generates a pseudorandom permutation of the cover indices. The key should be long enough to prevent the Warden from exhausting over all possible keys (see (Pevný & Ker, 2014) for a recent paper on this topic), and the generation of the pseudorandom permutation needs to avoid bias in the output: a simple algorithm is the famous **Knuth shuffle** (Knuth, 1998, §3.4.2, Algorithm P).

Some early steganography literature seemed unaware of the idea of generating pseudorandom permutations, and instead simply picked random cover indices independently for each payload symbol; the problem is that there is a high likelihood that the same location will be used more than once, which damages the correctness of the embedding.

1.4 Example: LSB Embedding in Uncompressed Images

A digital image is displayed on the screen as a rectangular grid of coloured **pixels**. So the simplest way to store a digital image is directly to represent this grid. In a **raster** image format the image pixels are scanned (usually left-to-right and top-to-bottom), and their colours stored. The raw format, native to most displays, is three bytes of information for each pixel: values 0–255 representing the intensity of red, green, and blue (RGB) which make up the colour[5].

[5]This format is *not* native to printing devices, which must deal with subtractive rather than additive colour, and some digital images are stored in CMYK (cyan, magenta, yellow, black) form, with up to 12 bits of precision for each component.

In this course we are more interested in how the image can be manipulated than the exact format it is written on disk, but here is a particularly simple file format called PNM[6]. The file is simply ASCII text

```
P3      # format identifier
400 300 # width height
255     # RGB value maximum
 37  37  36  74  79  71 150 157 136 161 167 147 172 176 156 181 182 163
189 189 173 198 198 183 206 205 190 210 209 195 212 211 198 212 210 198 ...
#(R   G   B) (R   G   B) (R   G   B) bytes in scan order
```

(This is the beginning of the example cover image in Figure 1.1). The ASCII format is not efficient, and a variation stores the bytes of the image scan directly (but retains the ASCII header). There is an analogous format for images which are only grayscale (one byte of brightness per pixel). The PNM format is very portable but more popular and widespread bitmap image formats include TIFF and BMP; they can store a wider range of image type and metadata (and also palette images, see subsection 1.4.2) but the structure is essentially the same.

So let us represent a raw (as opposed to lossy-compressed) RGB cover image as a sequence of bytes $(\boldsymbol{c}[1], \ldots, \boldsymbol{c}[n])$ where each byte represents, in turn, the red, green and blue intensities of the pixels in scan order. For convenience we will omit from our representation the size of the image, which we might say was already part of Alice and Bob's secret key. We use the secret key $\boldsymbol{k}$ to generate a pseudorandom order $\pi_{\boldsymbol{k}}$ in which to visit these bytes, i.e. for each $\boldsymbol{k} \in \mathcal{K}$,

$$(\pi_{\boldsymbol{k}}(1), \ldots, \pi_{\boldsymbol{k}}(n)) \text{ is a permutation of } (1, \ldots, n).$$

Suppose that we have an m-bit payload represented by a sequence of bits $(\boldsymbol{m}[1], \ldots, \boldsymbol{m}[m])$. In perhaps the simplest type of steganography, we visit the cover RGB bytes in the order given by the key, overwriting the least significant bits (LSB) of each byte with the message until the message is finished. We form the stego vector $\boldsymbol{s}$ by

$$\boldsymbol{s}\big[\pi_{\boldsymbol{k}}(j)\big] = 2\Big\lfloor \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] / 2 \Big\rfloor + \boldsymbol{m}[j] \quad \text{for } j = 1, \ldots, m. \tag{1.1}$$

and $\boldsymbol{s}[i] = \boldsymbol{c}[i]$ for $i \notin \{\pi_{\boldsymbol{k}}(j) \,|\, j = 1, \ldots, m\}$. Finally, $\mathrm{Emb}(\boldsymbol{c}, \boldsymbol{k}, \boldsymbol{m}) = \boldsymbol{s}$.

[6] `http://netpbm.sourceforge.net/doc/pnm.html`

This type of embedding is called **LSB Replacement** (LSBR) and it is one of the earliest types of digital image steganography. The first literature and software to mention LSB replacement is lost in the mists of time, but it was well-established by the time of the first Information Hiding Workshop in 1996, and is implemented in some DOS software (predating Windows 95).

The effect of the LSBR embedding operation is to change the colours very slightly. See Figure 1.1 for an example of LSBR embedding in a small raw cover image; even magnified, a change of ± 1 to RGB values is not perceptible. This is true even in smooth areas of an image (such as blue sky).

How does Bob reconstruct the hidden payload? All he needs is to use the secret key to generate the permutation $\pi_{\boldsymbol{k}}$, and then read

$$\mathrm{Ext}(\boldsymbol{s}, \boldsymbol{k}) = \Big(\boldsymbol{s}\big[\pi_{\boldsymbol{k}}(1)\big] \pmod 2, \ldots, \boldsymbol{s}\big[\pi_{\boldsymbol{k}}(m)\big] \pmod 2\Big).$$

There is a defect with the stegosystem as described: unless it was part of the pre-shared secret key, Bob does not know m, and therefore does not know where to stop reading LSBs to extract the message. An implementation could solve this by adding a header to the payload, which includes its length, but Alice should be careful not to make the message recognisable as this might lead to easier key exhaustion attacks by the Warden[7]. An alternative is for Alice to reserve an unambiguous terminator for the end of her payload.

What is the **capacity** of this embedding algorithm? The largest payload uses all the bytes of the image, which is 3 times the number of cover pixels (we conventionally ignore any overhead caused by either payload or cover headers). We say that the maximum capacity is *3 bits per pixel* (3 bpp), or *1 bit per cover byte*. We can also measure the **embedding efficiency**. Assuming a random message (one which is not correlated with the cover) we will only have to change half of the bytes we visit, because half of them carried the correct LSB already. Thus the embedding efficiency is *2 bits per change*.

Note that the LSB replacement operation (1.1) can be equivalently written to emphasise its isolation of the LSBs of each cover byte (the so-called least significant bitplane):

$$\boldsymbol{s}\big[\pi_{\boldsymbol{k}}(j)\big] = \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] \,\&\, 254 + \boldsymbol{m}[j],$$

[7] Predictable headers were used in this detection study `http://www.citi.umich.edu/u/provos/papers/detecting.pdf`, though the techniques used are now very antiquated. Recent work has revived the topic, more generally, of information leakage about the key.

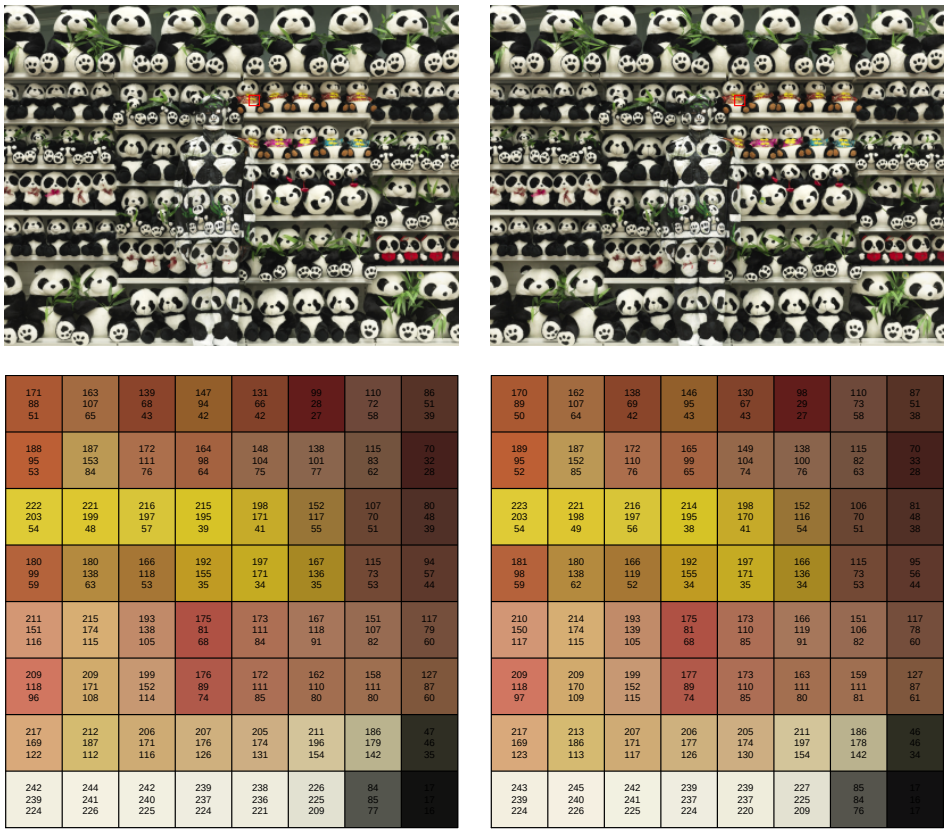


Figure 1.1: Left, example cover image (400×300 pixels, 24-bit RGB) with 8×8 detail below. Right, stego image with 3 bits per pixel embedded by LSBR, with 8×8 detail below. The magnified blocks are annotated with R, G, B values.

Cover image courtesy of Eli Klein Fine Art, © Liu Bolin

where **&** represents bitwise logical 'and'. Or it can be written to emphasise the LSB flipping operation:

$$\boldsymbol{s}\big[\pi_{\boldsymbol{k}}(j)\big] = \begin{cases} \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big], & \text{if } \boldsymbol{m}[j] = \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] \pmod 2 \\ \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] + (-1)^{\boldsymbol{c}[\pi_{\boldsymbol{k}}(j)]}, & \text{otherwise.} \end{cases}$$

The latter exposes a weakness (even-value bytes can only ever be left alone or incremented; odd-value bytes can only be left alone or decremented) which we can exploit in the next chapter. Despite its weaknesses, LSBR has continued to be studied widely, partly due to its extreme ease of implementation: in (Ker, 2004) I gave a two-line Perl script which avoids the need for any special software by typing into a Linux shell

```
perl -n0777e '$_=unpack"b*",$_;split/(\s+)/,<STDIN>,5;
              @_[8]=~s{.}{$&&v254|chop()&v1}ge;print@_'
              <input.pgm >output.pgm secrettextfile
```

This will perform LSB replacement in grayscale PNM files (if the pixel information is in raw byte rather than ASCII encoding). It does not use a pseudorandom permutation to spread the payload through the cover and has no secret key, but a four-line script can achieve this (Ker, 2005).

1.4.1 Other Embedding Operations

We will soon see that LSB replacement in images is surprisingly easy to detect. But there are alternative embedding operations, hardly any more complex, which are significantly more secure.

The second simplest embedding operation, after LSBR, is **LSB Matching** (LSBM), also sometimes called **±1 embedding**. The image is traversed in the same way and the payload bits are still carried in the LSBs of the intensity bytes, so the extraction function is identical. It has the same capacity and embedding efficiency. But the change when a modification is required, i.e. if $\boldsymbol{m}[j] \neq \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] \pmod 2$, is given by

$$\boldsymbol{s}\big[\pi_{\boldsymbol{k}}(j)\big] = \begin{cases} \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] + 1, & \text{if } \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] = 0 \\ \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] - 1, & \text{if } \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] = 255 \\ \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] \pm 1, & \text{otherwise, where } \pm 1 \text{ is picked uniformly at random} \end{cases} \tag{1.2}$$

This avoids the predictable structure of LSBR, and is significantly more difficult to detect. As far as the lecturer knows, the first publication to describe LSB matching (not under that name) is (Sharp, 2001), but it was almost certainly known, and its advantages vaguely understood, well before this.

Once we are prepared to make ± 1 changes to coefficients, why not use the choice to convey more payload, increasing capacity and embedding efficiency? We come to **ternary embedding**, which requires the payload to be formatted as a sequence of base-3 (ternary) symbols $\{0, 1, 2\}$. The embedding operation selects the value of $\boldsymbol{s}\big[\pi_{\boldsymbol{k}}(j)\big]$ nearest to $\boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big]$ such that

$$\boldsymbol{s}\big[\pi_{\boldsymbol{k}}(j)\big] \bmod 3 = \boldsymbol{m}[j].$$

This only requires a change of 0, $+1$, or -1 (apart from the exceptional case when the cover value is 0 or 255 which, we shall see later, steganographers should be avoiding in any case). Ternary embedding has a maximum capacity of one ternary symbol, $\log_2 3$ bits, per cover byte (or $3 \log_2 3$ bits per pixel in RGB colour images). Given random payload, the cover byte will need to be changed with probability $\frac{2}{3}$, meaning that the embedding efficiency is $\frac{3}{2} \log_2 3 \approx 2.377$ bits per change.

Assuming that cover images which have been over- or underexposed are excluded (thus preventing 0s and 255s), ternary embedding is always a superior choice to LSBM, because it requires fewer changes of identical magnitude to convey the same payload. The drawback is that the payload must be converted to ternary symbols, though there is a nice solution to avoid this, which we shall see in chapter 3.

1.4.2 Palette Images

So far we have only dealt with RGB colour images. Obviously the same technique applies directly to brightness values of a grayscale image. But a different type of uncompressed image is the palette image, in which a limited number of colours are available (often 256), with the colours listed in a header and the palette indices stored in an image scan. The rationale is to save disk space and/or bandwidth.

The format GIF is most commonly used, and GIF files were predominant in the early years of the internet, but are now becoming more and more scarce. Hiding in palette images poses additional difficulties, because there is no guarantee that colours with similar indices will look similar, so modifications must be done with the palette in mind. Some palette hiding methods are discussed in (Fridrich, 2010, §5.2) but we will not consider them further in this course.

1.5 Example: Embedding in JPEG Images

Unfortunately, raw bitmap images are transmitted relatively rarely. The reason is simply their size: a 1 megapixel (1 Mpix) colour image takes 3MB of bandwidth and storage, which is not insignificant, even in the present time of cheap storage and fast networks. Almost all images transmitted over the internet, including all social media and in practically every web page, use lossy-compressed images in **JPEG** format. Embedding in JPEG images is conceptually quite similar to embedding in raw images, but there are a few important differences. We need to understand what is actually stored in a JPEG file, before we can modify it to carry a payload.

1.5.1 JPEG Essentials

The JPEG image format, and the lossy codec associated with it, date back to 1992. Based on signal-processing techniques, the aim is to create an approximation of an image, which is perceptually almost identical to the original but can be stored in much less space.

The key to storing information approximately is **quantization**, where a real number x is approximated by $q\lfloor x/q \rceil$ for a fixed **quantization factor** q. Only the integer $\lfloor x/q \rceil$ needs to be stored, from which $q\lfloor x/q \rceil$ can be reconstructed later. Another way of saying this is that quantizing x means storing it "to the nearest q". The smaller the value of q, the more precise the representation of x. With JPEG compression, quantization is chosen so that perceptually important information (low frequency, brightness) is stored with higher resolution and lower quantization factors than the less important information (high frequency, colour).

We will briefly outline the steps involved in JPEG compressing a bitmap image:

(i) The colour space undergoes a linear transformation to separate intensity (**luminance**) from colour (**chrominance**). They are treated separately but similarly. Typically the colour components are reduced in resolution by a factor of two.

(ii) The image is divided into 8×8 pixel blocks (padded as need be). Each block is treated separately.

(iii) Each block undergoes a **discrete cosine transformation** (DCT), which expresses the block as a linear combination of different (2-dimensional) frequencies.

(iv) The frequencies, which are called **coefficients**, are quantized; this step is where information is lost, the others are exactly reversible. The quantization factors are set at time of compression and vary for different frequencies: higher frequencies

get higher quantization factors and many high frequency coefficients will end up quantized to zero.

(v) The quantized coefficients from all blocks are arranged in a particular order, then losslessly compressed. The compressed file, along with headers specifying the size and the quantization factors, is the `.jpg` file stored.

To decompress a `.jpg` image, the steps are reversed: the format is parsed, the lossless compression undone, the coefficients multiplied by the quantization factors, the DCT inverted, colour components recombined and upsampled, and at the final stage the pixel values must be rounded back to integers and clamped into the range 0–255 (which they may have exceeded due to the lossy compression).

The above schema describes the most common use of the JPEG format. Many variations exist in the specification: different colour spaces may be stored, the colour component resolution can be varied, the data can be stored in different orders, additional header information can be included, and so on.

We will give a bit more detail about the JPEG steps. In step (i), the (R, G, B) values for each pixel undergo the linear transformation

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} + \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}.$$

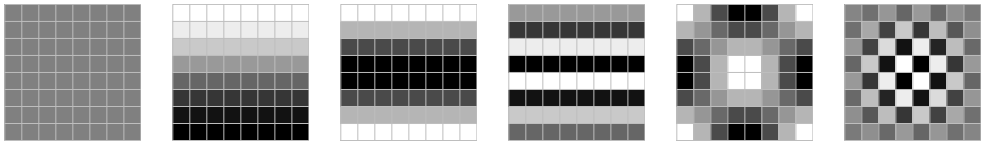
The luminance is Y, which corresponds roughly to the eye's perception of brightness (depending on the display device being used). For images which had no colour in the first place, grayscale levels are interpreted directly as Y. The chrominance coefficients C_b and C_r indicate shift towards blue and red, respectively.

Step (ii) is straightforward. Let us take the luminance channel of an $M \times N$ image and call it $\boldsymbol{Y}[m, n]$ for $m = 0, \ldots, M-1$ and $n = 0, \ldots, N-1$[8]. There are $\lceil M/8 \rceil \cdot \lceil N/8 \rceil$ blocks, let us call them $\boldsymbol{B}^{\boldsymbol{p},\boldsymbol{r}}$ for $0 \le p < \lceil M/8 \rceil$ and $0 \le r < \lceil N/8 \rceil$, defined by

$$\boldsymbol{B}^{\boldsymbol{p},\boldsymbol{r}}[i, j] = \boldsymbol{Y}[8p + i, 8r + j], \quad \text{for } i, j \in \{0, 1, \ldots, 7\}$$

with zero padding for $\boldsymbol{Y}$ if the image dimensions are not multiples of 8.

[8] In this section it is more convenient to base array and matrix indices at zero.



$\boldsymbol{A_{0,0}}$ $\boldsymbol{A_{1,0}}$ $\boldsymbol{A_{2,0}}$ $\boldsymbol{A_{7,0}}$ $\boldsymbol{A_{2,2}}$ $\boldsymbol{A_{7,7}}$

Figure 1.2: Some of the basis blocks for the discrete cosine transform.

The 2-dimensional discrete cosine transform at step (iii) is an analogue of the Fourier transform, but designed for real-valued signals. It amounts to an orthonormal change of basis. Given an 8×8 block $\boldsymbol{B}$ it produces an 8×8 block of real numbers $\boldsymbol{C}$ by

$$\boldsymbol{C}[u,v] = \sum_{i=0}^{7}\sum_{j=0}^{7} \boldsymbol{B}[i,j]\frac{c_u c_v}{8}\cos\left(\frac{\pi}{8}\left(i+\tfrac{1}{2}\right)u\right)\cos\left(\frac{\pi}{8}\left(j+\tfrac{1}{2}\right)v\right)$$

where $c_0 = 1$ and $c_i = \sqrt{2}$ otherwise. It is easier to understand the effect of the DCT by its inverse,

$$\boldsymbol{B} = \sum_{u=0}^{7}\sum_{v=0}^{7} \boldsymbol{C}[u,v]\boldsymbol{A_{u,v}} \tag{1.3}$$

where blocks $\boldsymbol{A_{u,v}}$ make up the **basis**,

$$\boldsymbol{A_{u,v}}[i,j] = \frac{c_u c_v}{8}\cos\left(\frac{\pi}{8}\left(i+\tfrac{1}{2}\right)u\right)\cos\left(\frac{\pi}{8}\left(j+\tfrac{1}{2}\right)v\right);$$

(here (u,v) are called the **DCT modes**).

We can see from (1.3) that the block $\boldsymbol{B}$ has been expressed as a linear combination of the basis blocks, each of which are combinations of gradients in x and y directions, where the linear combination is given by the **coefficients** $\boldsymbol{C}$. Some of the basis blocks are pictured in Figure 1.2.

The rationale for using the DCT is a) to attempt to **decorrelate** the coefficients, and b) to separate low-frequency parts of the block from high-frequency parts.

Step (iv) is quantization. Every block is treated the same, but within each block every coefficient is differently quantized, the higher frequencies receiving higher quantization factors than lower. The overall harshness of the quantization can be different for every

image, but most software uses standard quantization tables generated by a "**quality factor**" between 1 and 100. The matrix of quantization factors for the luminance component, at "quality factor" 50, is

$$\boldsymbol{Q_{50}} = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \tag{1.4}$$

and tables for higher and lower "quality factors" can be determined from this matrix[9].

Step (v) will be irrelevant to steganographers, because it is completely reversible and effectively provides a container for the quantized coefficients. There is a zig-zag ordering, designed to make runs of zeros more likely, in turn making the data more compressible.

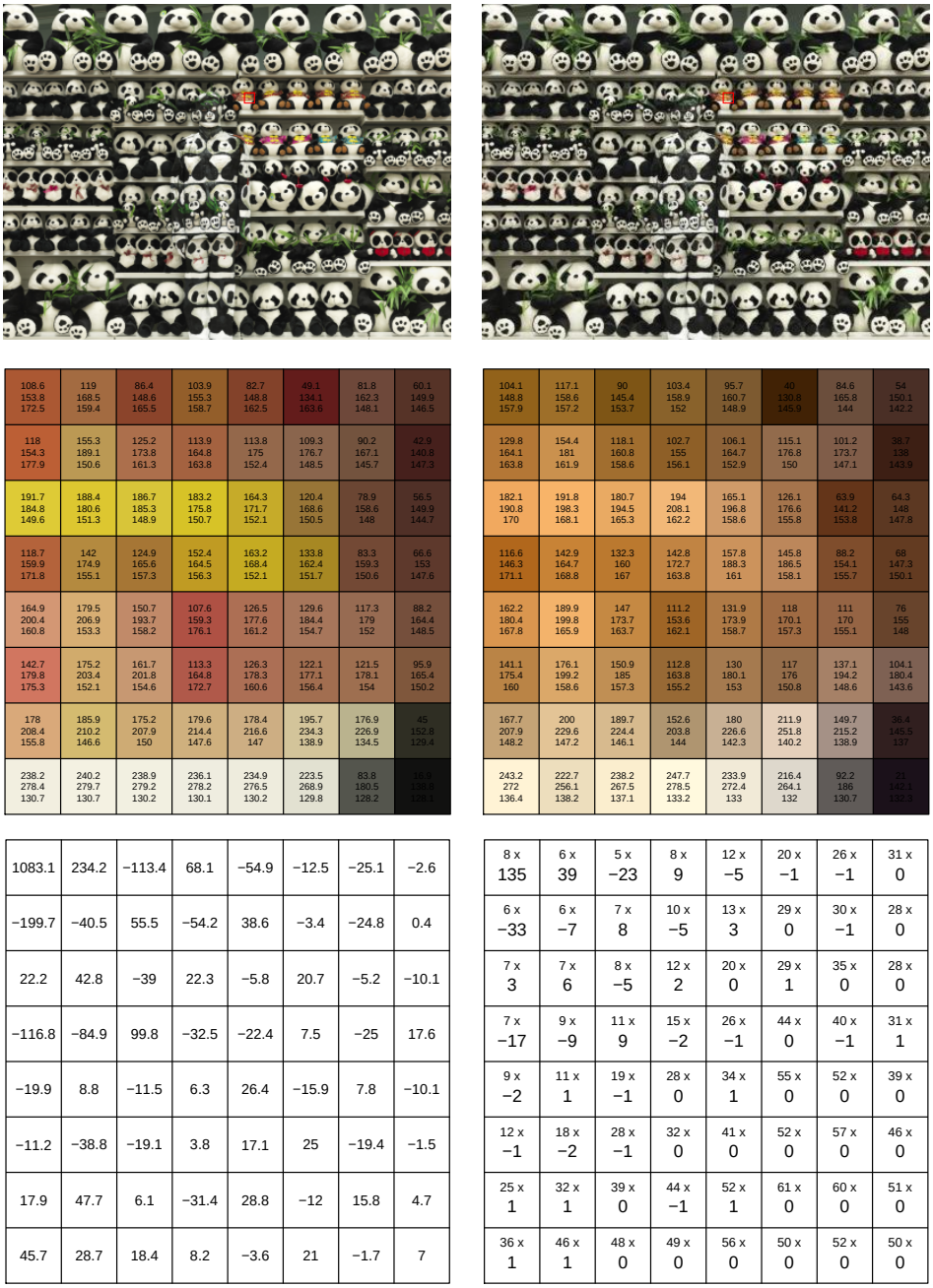
The size of a JPEG file depends on its contents and quality factor. With lower quality factors, more coefficients are quantized to zero and the lossless compression becomes much more efficient, but this happens to a lesser extent in images with lots of texture (which have a larger high-frequency content) than images with large smooth regions. In one of the image sets I use, for testing steganography and steganalysis, there are 1600 colour images sized 2000×1500 (mostly of beaches and flowers). Uncompressed files are 8789KB each, and compressed JPEGs at quality factor 50 average 274KB – about 32 times smaller.

The uncompressed cover image from Figure 1.1 is compared with a JPEG-compressed version in Figure 1.3, including a magnification of, and luminance DCT coefficients for, one 8×8 block.

1.5.2 Embedding Operation

Embedding in JPEGs is called **transform-domain** embedding, because the embedding is applied to a transformation of the pixels (in contrast, embedding in raw images is

[9] We do not need to know the formula in detail, but for "quality factor" $qf > 50$ the quantization matrix is approximately $\boldsymbol{Q_{qf}} = \left[\boldsymbol{Q_{50}} \frac{100-qf}{50}\right]$.



1083.1	234.2	−113.4	68.1	−54.9	−12.5	−25.1	−2.6
−199.7	−40.5	55.5	−54.2	38.6	−3.4	−24.8	0.4
22.2	42.8	−39	22.3	−5.8	20.7	−5.2	−10.1
−116.8	−84.9	99.8	−32.5	−22.4	7.5	−25	17.6
−19.9	8.8	−11.5	6.3	26.4	−15.9	7.8	−10.1
−11.2	−38.8	−19.1	3.8	17.1	25	−19.4	−1.5
17.9	47.7	6.1	−31.4	28.8	−12	15.8	4.7
45.7	28.7	18.4	8.2	−3.6	21	−1.7	7

8 x 135	6 x 39	5 x −23	8 x 9	12 x −5	20 x −1	26 x −1	31 x 0
6 x −33	6 x −7	7 x 8	10 x −5	13 x 3	29 x 0	30 x −1	28 x 0
7 x 3	7 x 6	8 x −5	12 x 2	20 x 0	29 x 1	35 x 0	28 x 0
7 x −17	9 x −9	11 x 9	15 x −2	26 x −1	44 x 0	40 x −1	31 x 1
9 x −2	11 x 1	19 x −1	28 x 0	34 x 1	55 x 0	52 x 0	39 x 0
12 x −1	18 x −2	28 x −1	32 x 0	41 x 0	52 x 0	57 x 0	46 x 0
25 x 1	32 x 1	39 x 0	44 x −1	52 x 1	61 x 0	60 x 0	51 x 0
36 x 1	46 x 1	48 x 0	49 x 0	56 x 0	50 x 0	52 x 0	50 x 0

Figure 1.3: Left, example cover image. Right, the same image JPEG compressed using "quality factor 75" (quantization factors half of those in (1.4)). Corresponding 8×8 blocks are magnified below the image, annotated with their Y, C_b, C_r values (to 1 d.p.): the very rough colour approximation is evident. At the bottom, the DCT coefficients for the luminance component of the same block are displayed; on the left for the original image, and how they are approximated in the JPEG file.

called **spatial-domain** embedding). Once we understand what a JPEG file is, cover modification is not too difficult. Clearly we cannot change LSBs of pixel values in an image and expect these bits to remain unscathed when the file is JPEG compressed. The modifications need to be at the level *after* information is lost, so that the payload is not lost, and that means modifying quantized coefficients. If we perform steps (i)-(iv) of JPEG compression, we find ourselves with 8×8 blocks of integers which are the quantized coefficients:

$$\boldsymbol{C}^{p,r}[u,v], \quad \text{for } u,v \in \{0,1,\ldots,7\},\ 0 \leq p < \lceil M/8 \rceil,\ 0 \leq r < \lceil N/8 \rceil.$$

They are reconstructed exactly by the receiver/decompressor/extractor (re-ordering and lossless compression are completely reversible), so we modify them to encode the payload.

In principle we could use the same embedding operation as for raw images, visiting the quantized (luminance[10]) coefficients in pseudorandom order, and overwriting the least significant bits with the payload. Unfortunately this tends to lead to visible artefacts in the images unless the quantization factors are very small: suppose that we take a JPEG-compressed image and change just one DCT coefficient in one block, say changing $\boldsymbol{C}[u,v]$ by adding 1 to it. Decompression is a linear operation, so the effect on the pixels is to add to the corresponding luminance block $\boldsymbol{B}$ the matrix

$$\boldsymbol{Q}[u,v]\boldsymbol{A}_{\boldsymbol{u},\boldsymbol{v}}.$$

So the effect of the single DCT change is to add an oscillating 8×8 block to the cover. Consider that if u and v are not both small, $\boldsymbol{Q}[u,v]$ might be as high as 25, 50, or even higher (depending on the quality factor). This means that the magnitude of the oscillations might be 25 or more, out of a possible range 0–255: this is going to be particularly visible in parts of an image which did not have much gradient to start with (e.g. sky).

A proper solution to this problem took many years to develop, and we will come to it in chapter 3. But an early attempt, which proved quite effective, was the following rule of thumb: *don't change zero quantized coefficients*. By and large, coefficients which are not quantized to zero have low quantization factors, or occur in blocks which are already very noisy. There is another reason to avoid creating new nonzero coefficients: it will change the size of the `.jpg` file, and that *might* lead to a detectable signature. Another

[10] In the literature, it is almost always the luminance component which is used for embedding, with the chrominance channels left unaltered. Partly this is because changes to the chrominance channels can be quite visible, but use of chrominance channels has not been properly explored.

heuristic, usually followed in JPEG steganography, is to ignore the 'DC' (flat basis block) (0,0) modes altogether, because changes to them are visible as brighter or dimmer 8×8 squares.

So how about visiting all the *nonzero* non-DC coefficients $\boldsymbol{C}^{p,r}[u,v]$, using something like the LSB to convey payload? Now there is the **non-shared selection channel** problem: when reading the stego object, Bob cannot tell the difference between zeros which were not used for embedding and zeros which were created by embedding changes (he should skip the former and read LSBs from the latter). A poor solution was adopted by the program JSteg[11], probably the first software to offer steganography in JPEGs; it avoids both 0 and 1 quantized coefficients, and then uses LSBR on the rest. As we shall see in the next chapter, that breaks a strong symmetry of quantized coefficients (quantized value $-a$ occurs almost equally as often as $+a$, for any a) and is very easily detectable.

An alternative embedding operation which uses LSBs but preserves symmetry is

$$\boldsymbol{s}\big[\pi_{\boldsymbol{k}}(j)\big] = \begin{cases} \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big], & \text{if } \boldsymbol{m}[j] = \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] \pmod 2 \\ \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] - 1, & \text{otherwise, if } \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] > 0 \\ \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] + 1, & \text{otherwise, if } \boldsymbol{c}\big[\pi_{\boldsymbol{k}}(j)\big] < 0 \end{cases} \tag{1.5}$$

It is known as F5, and was described in the publication (Westfeld, 2001). There, the extractor skips *all* zero coefficients. So we need to pair it with a more complicated method for embedding:

(i) Traverse the quantized DCT coefficients in pseudorandom order as determined by the secret key, skipping all zeros.

(ii) Use the F5 embedding operation to embed one bit of payload in each nonzero coefficient.

(iii) But if this creates a new zero it will be skipped by the extractor, so repeat embedding the same bit at the next nonzero coefficient.

Capacity in JPEG images is usually measured as *bits per nonzero DCT coefficient* (abbreviated **bpnc**) and the repeated embedding of the F5 algorithm means that the capacity is usually around 0.7–0.8 bpnc. We will call (1.5) the **F5 embedding operation**, and the algorithm above the **F5 algorithm**. JPEG images have smaller capacity than raw: our cover example contained 120 000 pixels and under LSBR its capacity is 360 000 payload

[11] `http://zooid.org/~paul/crypto/jsteg/`

bits; In the JPEG version there are $\lceil 400/8 \rceil \cdot \lceil 300/8 \rceil \cdot 64 = 121\,600$ DCT coefficients but only 54 629 are nonzero and the published F5 implementation[12] reports an estimated capacity of 36 840 bits.

The effect of the F5 embedding operation is to pull coefficients towards zero; this weakness leads to detectors, but they are nowhere near as sensitive as for JSteg or spatial-domain LSBR embedding. For the F5 algorithm, the embedding efficiency is lower than 2 bits per change and the repeated embedding causes an even greater increase in the number of zero DCT coefficients, an effect known as **shrinkage**, which makes it more detectable. Later versions of the F5 software include a method known as **matrix embedding**, which is a method to increase embedding efficiency at small payloads; we will come to this in chapter 3.

In chapter 3 we will solve the non-shared selection channel problem and be able to use the F5 embedding operation without shrinkage (re-embedding the same bit multiple times). We will also be able to perform **adaptive embedding**, which locates the embedding changes in those parts of an image where, we hope, they are least detectable. Typically this would mean edge areas and noisy textures. Before these problems were properly solved, the F5 algorithm was a *de facto* standard (for at least 2001–2007) and has been widely studied, so it makes for a good example in this course.

1.6 Active Wardens

Active Wardens are not within the scope of this syllabus. In realistic applications of steganography, it is not very likely that a Warden will be active (can one imagine an intelligence agency tampering with digital media transmitted over the internet, trying to disrupt covert communication?). However, social media websites do tend to recompress, and sometimes resize, the images and videos uploaded to them, which is not an active Warden *trying* to damage the payload but nonetheless presents a challenge for the steganographer; as yet, there is relatively little literature on this topic.

Active Wardens are central to the part of information hiding known as **digital watermarking**. The aim of watermarking is to include secret information in the cover which an enemy cannot remove unless they completely ruin the quality of the cover (one can, of course, always destroy hidden information by blanking an image completely). The main

[12] `http://code.google.com/p/f5-steganography/`

application is to identify and protect copyrights of digital media, for example by embedding an individual watermark in each copy of a pre-release video, so that a leaker can be identified. A non-security application, recently proposed by Digimarc Corporation, is to replace barcodes scanned at supermarket checkouts by watermarks spread throughout the packaging: then the item can go through the scanner in any orientation.

The payload will be a lot smaller than in steganography, and must be embedded in a redundant way, often using techniques related to **spread spectrum** communications. Interesting contemporary research includes finding ways to defeat the **collusion attack**, in which a number of differently-watermarked copies of the same cover are merged by colluding pirates.

In order to avoid confusion, we repeat the essential difference between steganography and watermarking: in the former, the enemy tries to detect the hidden payload; in the latter, the enemy tries to disrupt the hidden payload. Fundamentally, the limitations of steganography are about the decision functions of the enemy, which through information theory is susceptible to mathematical analysis. On the other hand, the limitation in digital watermarking is how much the Warden is allowed to tamper with the carrier, which depends on human perception and thus is difficult to capture formally.

Bibliography

Fridrich, J. (2010). *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge University Press.

Hopper, N. J., Langford, J., & von Ahn, L. (2002). Provably secure steganography. In *Advances in Cryptology: Proceedings of CRYPTO 2002*, volume 2242 of *Lecture Notes in Computer Science* (pp. 77–92). Springer. Available at `http://eprint.iacr.org/2002/137/`.

Kahn, D. (1996). The history of steganography. In *Proceedings of 1st Information Hiding Workshop*, volume 1174 of *Lecture Notes in Computer Science* (pp. 1–5). Springer. Available at `http://books.google.co.uk/books?id=gWdiyTlEBIIC&pg=PA1`.

Ker, A. D. (2004). Improved detection of LSB steganography in grayscale images. In *Proceedings of 6th Information Hiding Workshop*, volume 3200 of *Lecture Notes in Computer Science* (pp. 97–115). Springer. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK09D.pdf`.

Ker, A. D. (2005). Resampling and the detection of LSB matching in colour bitmaps. In *Security, Steganography, and Watermarking of of Multimedia Contents VII*, volume 5681 of *Proceedings of SPIE* (pp. 1–15). SPIE. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK11B.pdf`.

Knuth, D. E. (1998). *The Art of Computer Programming Volume 2 – Seminumerical Algorithms*. Addison-Wesley, 3rd edition.

Pevný, T. & Ker, A. D. (2014). Steganographic key leakage through payload metadata. In *Proceedings of 2nd ACM Workshop on Information Hiding and Multimedia Security* (pp. 109–114). ACM. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK62B.pdf`.

Sharp, T. (2001). An implementation of key-based digital signal steganography. In *Proceedings of 4th Information Hiding Workshop*, volume 2137 of *Lecture Notes in Computer Science* (pp. 13–26). Springer. Available at `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.5380`.

Simmons, G. J. (1983). The prisoners' problem and the subliminal channel. In *Advances in Cryptology, Proceedings of CRYPTO '83* (pp. 51–67). Plenum Press. Not freely downloadable. For students only, available at `https://www.cs.ox.ac.uk/teaching/materials13-14/advsec/ThePrisonersProblem.pdf`.

Westfeld, A. (2001). F5–a steganographic algorithm. In *Proceedings of 4th Information Hiding Workshop*, volume 2137 of *Lecture Notes in Computer Science* (pp. 289–302). Springer. Available at `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.115.3651`.

Chapter 2

Steganalysis

Reading (course text):	Fridrich, §5.1.1, 10.1–2, 11.1.3, parts of chapter 12
Alternatives & further depth:	Böhme, §2.3.2, 2.10.1–3 Schaathun, §2.3, 6.1, 8.2, 10.3, 14.1

Security is almost always defined in terms of the opponent: a system is secure if we prevent them achieving their attack goal. The enemy of the steganographer is the Warden, who is performing **steganalysis**, trying to detect the hidden data. In this chapter we take the role of the Warden and see how steganalysis can be achieved. As well as being an important topic in its own right, it allows us to understand better the weaknesses of stegosystems, and it will motivate the steganographic advances to be presented in chapter 3.

We must first clarify the attack model, which is the amount of information available to the Warden, and then review briefly the decision theory which covers the general topic of detection. Then we see examples of two types of steganalysis: specific methods which target an embedding weakness (in our example, spatial-domain LSBR), and general methods which apply machine learning techniques to train a detector to recognise stego objects by example. Genuine experimental results are included.

Finally, we discuss how the topic of steganalysis has recently widened to include scenarios more general than the classic Prisoners' Problem, and briefly mention the techniques used

to perform steganalysis when there are multiple actors, multiple images, and incomplete information about the cover sources.

2.1 The Warden's Knowledge

In other literature the Warden is also called a **steganalyst**, **attacker**, or **adversary**. Their aim is to uncover Alice and Bob's information hiding scheme, by determining the presence or absence of hidden data in stego/cover objects.

In the last chapter we mentioned Kerckhoffs' Principle, which (loosely summarised as "assume the enemy knows the system") is an acknowledged axiom in cryptography research because it makes a conservative assumption: the cryptographer cannot accidentally underestimate their opponent. There are more components to a stegosystem than just the embedding and extraction algorithms, and more realistic scenarios in which the eavesdropper knows less, and the literature has not always been clear about exactly what the Warden is supposed to know.

At the time of writing, I know of no published work identifying *all* the different possible scenarios for steganalysis. Here is a partial attempt to do that, first codifying the level of the Warden's knowledge about the embedding:

(A*) The Warden knows the content of the hidden payload and the embedding algorithm used.

(A) The Warden knows the length of the hidden payload, but not its content, and knows the embedding algorithm used.

(B) The Warden knows the embedding algorithm used, but nothing about the payload.

(C) The Warden does not know the embedding algorithm used.

and second about the source of covers:

(1) The Warden knows the exact characteristics of Alice's cover source.

(2) The Warden does not know the exact characteristics of Alice's cover source, but can learn about it by seeing examples.

(3) The Warden does not have information about Alice's cover source, but can learn about a similar one by seeing examples.

(4) The Warden knows nothing about Alice's cover source.

In addition, we might consider variations in what the Warden knows about the secret key generation algorithm, which is relevant to exhaustion attacks, but we will not do that here. We could identify finer distinctions in the above lists, as well. More importantly, there are new possibilities arising when there are multiple actors (in our analogy, prisoners) sending multiple communications; see subsection 2.5.

Steganography is usually measured against situation (A)(2): if the stegosystem is *not* reliably detectable even when an adversary knows the embedding and extraction algorithms, plus the length of payload they are searching for, and when they can learn about the cover source empirically, then the embedding is secure. We generally do not consider situation (A*) because, as we said in subsection 1.2, an embedder should avoid known (or chosen!) plaintexts attacks by encrypting prior to embedding. As for knowledge about the covers, this is a contemporary topic: much theoretical work, like that we shall see in chapter 4, assumes case (1), and most experimental work assumes case (2), but case (3) is probably the most likely.

Although a stegosystem which is reliably detectable in situation (A)(2) is not 'secure', this does not mean that it will never be used, and we still need detection techniques that work in more difficult scenarios. Even case (C)(4), in which the Warden seems to have no information on which to base a decision, is not entirely hopeless if Alice sends many communications, which the Warden can cluster into two identifiably different types of object (cover and stego). Dealing with unknown embedding algorithms (C), and so-called **mismatched covers** (3), is something of research interest at the moment.

Whatever the scenario, steganalysis in real-world objects (as opposed to theoretical constructions) is never perfect. We need to measure its accuracy, and such measurement comes from decision theory.

2.1.1 Receiver Operating Characteristic

Recall that the Warden seeks a decision function $D : \mathcal{C} \to \{\text{Positive}, \text{Negative}\}$ which classifies an object as cover or stego correctly, with as high a probability as possible. For technical reasons we should allow D to include randomness in its decision. The Warden can make two types of error:

- A **false positive** (type I error), $D(\boldsymbol{x}) = \text{True}$ when $\boldsymbol{x}$ is actually an innocent cover.
- A **false negative** (type II error), $D(\boldsymbol{x}) = \text{False}$ when $\boldsymbol{x}$ is actually a stego object.

Given a decision function, a **stochastic** (random) source of covers[1], and a particular embedding algorithm and payload size, we can measure the probability of these errors, respectively α and β. In machine learning literature the probability $1-\alpha$ is called **specificity**, and $1-\beta$ is called **sensitivity** or (in statistical terminology) **power**. Steganographers tend to stick to calling α the **false positive rate** and β the **false negative rate**.

The performance (accuracy) of the detector can be represented as a single point $(\alpha, \beta) \in [0,1]^2$. But almost all detectors can vary their sensitivity, by adjusting some parameter in the detection process (as we shall see next, adjusting a threshold is usually the simplest way to do this), whereby they can be tuned to be more sensitive – more false positives but fewer false negatives – or, conversely, less sensitive. Tracing out the set of all pairs (α, β) as the sensitivity is varied creates a curve called the **receiver operating characterstic** (ROC), a name using old-fashioned language which reflects its history in 1940s radar development. Conventionally, the false positive (α) rate is shown on the x-axis, and true positive/power ($1-\beta$) rate on the y-axis. Some examples are shown in Figure 2.1.

The ROC therefore describes the accuracy of a detector, with curves nearer to the top-left corner representing better detection, and a 45° line representing random guessing (one can achieve $\alpha = 1-\beta$ by ignoring the input and giving a positive detection at random, with probability α). Unfortunately, it is not always easy to say that one detector is "better than" another, when the curves cross: the application determines whether a higher false positive or false negative rate is preferred. Fridrich discusses some ways to reduce the ROC curve to a summary detector "score" in (Fridrich, 2010, §10.2.4). In steganalysis it is more complicated still, because the ROC depends on the size of the embedded payload: most likely the detector gets better (gives fewer false negatives for any given false positive rate) as the payload causes more or larger changes to the cover.

2.2 A Simple Example of Steganalysis

To illustrate the decision-theoretic concepts around steganalysis, we will start with an extremely simple example, perhaps the first example of statistical steganalysis to be published. Given a sequence $\boldsymbol{x}$, for example encoding RGB values of a raster image or

[1] This implies a probability distribution over all covers, which is tricky to define properly. A rough-and-ready definition is to say that the cover is an image taken at random from all those produced by a particular cover source.

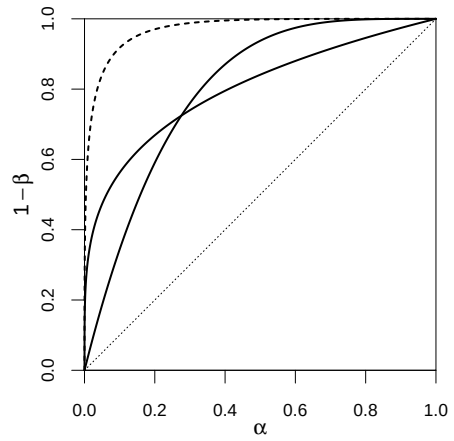


Figure 2.1: Some examples of ROC curves. α is the false positive rate and $1 - \beta$ the detection rate. The dotted line at 45° indicates a useless random guess. The dashed line represents a detector which is unquestionably better – with lower false positive and negative rates – than either of the solid lines, but either of the solid lines might represent a better choice than the other, depending on the application.

quantized coefficients in a JPEG, we can produce the **histogram** which we will denote $\boldsymbol{h}$,

$$\boldsymbol{h}[j] = \#\{i \,|\, \boldsymbol{x}[i] = j\}.$$

The elements of histogram are sometimes called **bins**. For the example cover image in Figure 1.1, we display the histogram of all its RGB values (pooled, though on other occasions one might want to separate the colour channels) in Figure 2.2. Below left, we zoom in on the bins 12–27. Below right, the partial histogram of the same image after embedding to maximum capacity (3 bpp) by LSBR. Below middle, where the image has only half of its pixels used for embedding, 1.5 bpp payload.

Right away, we can see a weakness of LSBR embedding. What has happened, for a reason which will be explained in section 2.3, is that pairs of histogram bins $\boldsymbol{h}[2i]$ and $\boldsymbol{h}[2i+1]$, are being pulled together by the embedding, until with maximum payload they are almost equal: this is called the "pairs of values" effect. The same thing happens, very reliably, in other images as well.

(Incidentally, the same can be seen in the DCT domain. Recall from subsection 1.5.2 that the JSteg embedding operation works like LSBR, but on the quantized DCT coefficients

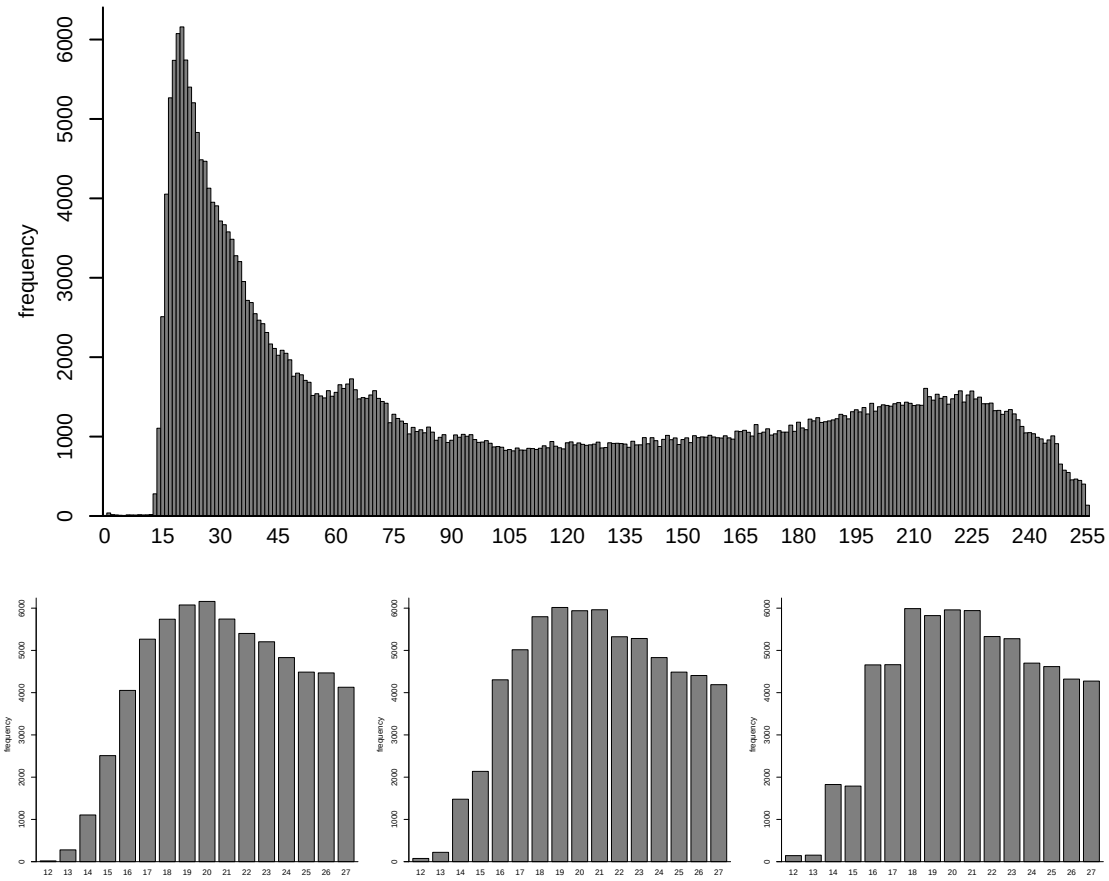


Figure 2.2: Above, complete histogram of the RGB values for the cover image in Figure 1.1. Below, a selection of 16 histogram bins: left, the cover image; middle, with payload of 1.5 bits per cover pixel (50% capacity); right, with a payload of 3 bits per cover pixel (100% capacity). The pairs of values effect is apparent.

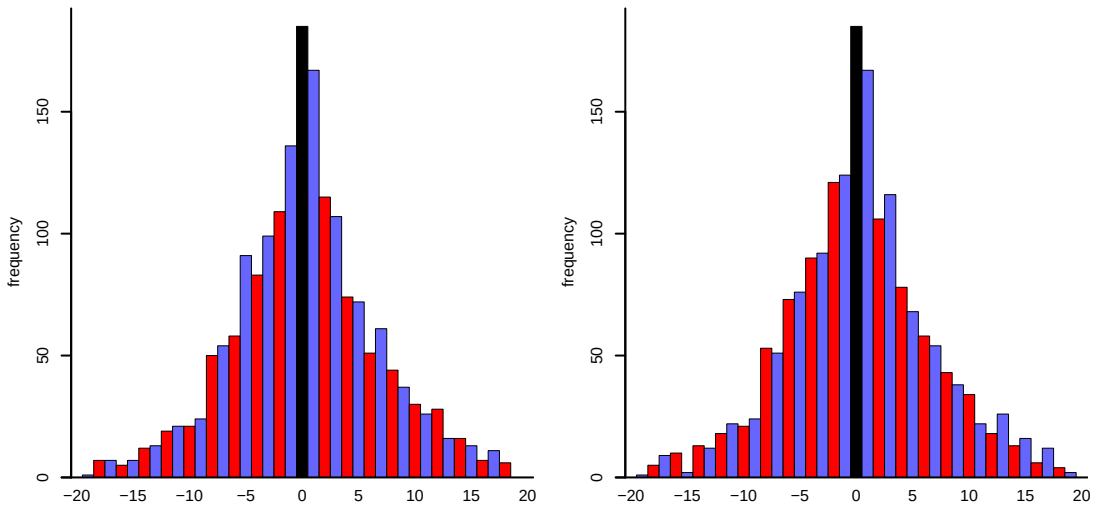


Figure 2.3: Left, histogram of the quantized DCT values (divided by their quantization level) for the $(0, 4)$-mode for the JPEG cover image. Right, the corresponding histogram when the JSteg embedding operation is simulated for a 100% capacity payload. The colours mark the parity of the histogram bins. As well as the pairs of values effect, the symmetry of the cover histogram is broken by the embedding.

of a JPEG file and omitting all coefficients with value 0 or 1. In Figure 2.3 we show the histogram of the $(0, 4)$-DCT mode (i.e. the coefficients $\boldsymbol{C}^{p,r}[0, 4]$ for all p and r) of the JPEG version of our cover image, and after simulating the JSteg embedding operation. We see the "pairs of values" effect, and also that the cover histogram is approximately symmetrical, with this symmetry broken by embedding.)

So even without knowing much about Alice's covers, we can already construct a steganography detector for LSBR in spatial- or transform-domain images. Given an image, we

extract its histogram and compute the so-called **chi-square** value[2]

$$T = \sum_{i=0}^{127} \frac{\big(\boldsymbol{h}[2i+1] - \boldsymbol{h}[2i]\big)^2}{\boldsymbol{h}[2i+1] + \boldsymbol{h}[2i]}. \tag{2.1}$$

The formula arises as a suitably scaled measurement of how far apart are all the pairs of histogram counts $\boldsymbol{h}[2i+1]$ and $\boldsymbol{h}[2i]$. If some histogram bins are empty (which sometimes happens for values near 0 or 255), terms with zero denominator should be excluded. In stego objects, according to our observation, T should be fairly close to zero, whereas in cover objects it should be significantly greater than zero. (You will prove the effect of LSBR payload on T, in the exercises.)

As often happens, the Warden has produced a scalar value T which should be higher (or in this case lower) when steganography is present. They can create a decision function $D(\boldsymbol{x})$ by setting a threshold τ:

$$D(\boldsymbol{x}) = \begin{cases} \text{True}, & \text{if } T(\boldsymbol{x}) \leq \tau, \\ \text{False}, & \text{if } T(\boldsymbol{x}) > \tau. \end{cases} \tag{2.2}$$

(Or in other cases with the positive decision above rather than below the threshold.) Decreasing τ in (2.2) means that fewer positive decisions will be made, the probability of false positives reduces but the probability of false negatives increases. Conversely, increasing τ means more false positives and fewer false negatives. This is an example of how easily the Warden can trade false positives for false negatives by adjusting the sensitivity of their detector.

The chi-square detector is not very sensitive. To measure its accuracy in real images, we took a set of 1600 colour PNMs (all taken with the same camera, and subject to minimal processing to convert the raw camera CCD output to pixels), each sized 2000×1500 pixels. We measured the statistic (2.1) for each cover, and then again when randomly-generated relative payloads of $p = 0.2, 0.4, \ldots, 1$ bits per cover byte (1.8M, 3.6M, ..., 9M bits of payload) were embedded using LSBR. From this data we can estimate the false positive and false negative rates[3] as we vary the detection threshold τ, and draw the ROC for each payload size. This is shown in Figure 2.4.

[2]This detector is called the **chi-square** detector because there is a weak relationship between (2.1) and the chi-square distribution from statistics. Do not be fooled: most literature follows the original (Westfeld & Pfitzmann, 1999) in making this connection and then derives a so-called *probability of embedding* value from T, but this is comes from a misconception about the nature of conditional probability. The

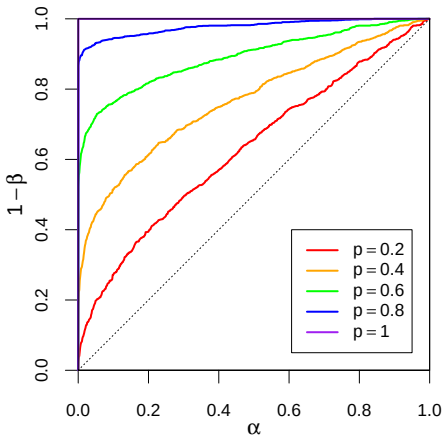


Figure 2.4: ROC curves for the chi-square detector as observed in 1600 images, 3 megapixel RGB, with payload p bits per cover byte embedded by LSBR.

Although the detector achieves perfect accuracy – no observed false positives or false negatives at all – when the payload is the maximum possible, the accuracy quickly falls back when the payload is smaller, to barely above random at $p = 0.2$ bits per cover byte. This is because we can say something useful about the value of (2.1) in stego images with full payload but cannot say much about it in cover images. In fact it varies a lot in cover images, and causes lots of false positives unless the detection threshold is very low.

Plotting ROC curves, for a range of payload sizes, obtained by calculating the detection value on a large library of real-world covers, is the correct way to measure the accuracy of any steganalysis technique. Scalar summaries of the ROC, as in (Fridrich, 2010, §10.2.4), can also be calculated. But we must take care not to over-interpret results for one set of covers, because they might be an artefact of the cover source not repeated in general, and modern reputable publications proposing new steganalysis methods should provide experimental results observed in a number of different sets of cover images. Over-interpretation of results observed in images from only one source, or in too few images for statistical rigour, is an easy trap to fall into.

---

chi-square quantile is not a measure of probability in this situation.

[3]It is an estimate – of the probabilities α and β when images are picked at random from the same source – because it comes from a finite set of samples. 1600 samples is enough for reasonable accuracy, but this data tell us less about the accuracy of the same detector in images from a different source, which might have quite different characteristics.

2.3 A Structural Attack on Spatial-Domain LSBR

In this section we analyze the parity structure of the LSBR embedding operation, which explains the "pairs of values" effect. On its own it does not lead to a powerful detector, but the same analysis in *pairs* of pixels can be extremely powerful. We give a simplified example of a detector called **couples**, based on pairs of pixels, which can even estimate the size of the hidden payload.

Detectors of this type appeared first in (Fridrich et al., 2001) and (Dumitrescu et al., 2002), and then many related techniques in a hodge podge of different notations. At first the relationships between the different techniques were unclear, and it was not until (Ker, 2005) that the so-called **structural** detectors were unified in a common framework and notation. The example which will be presented in this section is simpler than any which have been published, but it is reasonably easy to see how to extend it, and see (Fridrich, 2010, p. 233) for discussion of other structural detectors.

2.3.1 Parity Structure

Let us explain the "pairs of values" effect. First, we want to model the embedding stochastically (randomly). Suppose that a cover image $\boldsymbol{c}$ is turned into a stego image $\boldsymbol{s}$ with a payload which is proportion p of the maximum. (In a n-pixel RGB colour image, the payload is $3np$.) Take one random pixel in the cover: since the payload is spread into the cover by a pseudorandom permutation, there is probability p that this pixel is used for payload, and probability $1 - p$ that it is ignored by the embedding and extraction processes[4].

Let us re-word once more the formula for LSB replacement (1.1), taking each pixel at a time:

$$\boldsymbol{s}[i] = \begin{cases} \boldsymbol{c}[i], & \text{if location } i \text{ was not used for embedding,} \\ \boldsymbol{c}[i], & \text{if location } i \text{ was used for embedding but already had the correct LSB,} \\ \boldsymbol{c}[i] + 1, & \text{otherwise, if } \boldsymbol{c}[i] \text{ is even,} \\ \boldsymbol{c}[i] - 1, & \text{otherwise, if } \boldsymbol{c}[i] \text{ is odd.} \end{cases}$$

[4] A technicality for pedants only: these probabilities are not *quite* independent, because if pixel number one is used for embedding it makes it fractionally less likely that pixel number two is also used. But the effect of this tiny dependence can be ignored.

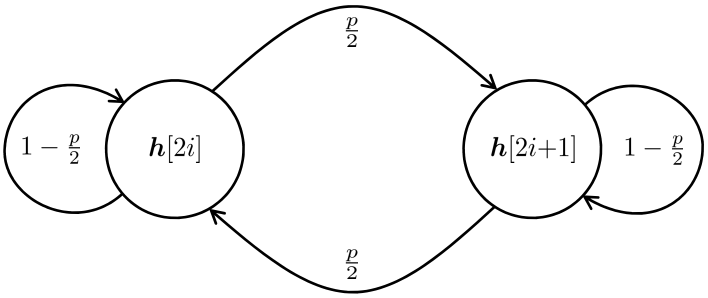


Figure 2.5: Graphical representation of the effect of LSB replacement on histograms.

Given an assumption of random payload bits (or at least bits uncorrelated with the cover) we can say that the probability of any location having the payload bit already is $\frac{1}{2}$. So putting this together we have

$$\boldsymbol{s}[i] = \begin{cases} \boldsymbol{c}[i], & \text{with probability } 1-\frac{p}{2}, \\ \boldsymbol{c}[i]+1, & \text{with probability } \frac{p}{2} \text{ if } \boldsymbol{c}[i] \text{ is even}, \\ \boldsymbol{c}[i]-1, & \text{with probability } \frac{p}{2} \text{ if } \boldsymbol{c}[i] \text{ is odd}. \end{cases} \tag{2.3}$$

We can represent this graphically in Figure 2.5: for a pixel which originally lies in histogram bin $\boldsymbol{h}[2i]$, there is probability $1-\frac{p}{2}$ that it remains there, probability $\frac{p}{2}$ that it moves into histogram bin $\boldsymbol{h}[2i+1]$, and conversely.

This is the **structural property** of LSBR: even-valued pixels might be left alone or incremented, but never decremented, and conversely for odd-valued pixels. It leads to predictable relationships between stego histograms and cover histograms.

For a fixed cover image, with histogram $\boldsymbol{h}$, let us model embedding with a random key and random payload of length $3np$: in this case, the histogram of the stego object $\boldsymbol{h'}$ is also random. But we can compute the expected (average) value of its counts, which comes from (2.3), and for large numbers of pixels we can assume that the observed stego object has histogram approximately equal to its expectation[5] giving us the linear equations

$$\begin{pmatrix} \boldsymbol{h'}[2i] \\ \boldsymbol{h'}[2i+1] \end{pmatrix} \approx \begin{pmatrix} 1-\frac{p}{2} & \frac{p}{2} \\ \frac{p}{2} & 1-\frac{p}{2} \end{pmatrix} \begin{pmatrix} \boldsymbol{h}[2i] \\ \boldsymbol{h}[2i+1] \end{pmatrix}. \tag{2.4}$$

[5]Technically we are using the fact that a binomial random variable $\mathrm{Bi}(m,q)$ has mean mq, and the law of large numbers.

This equation explains the histograms seen in Figure 2.2: when $p = 1$, we have $\boldsymbol{h'}[2i] \approx \frac{1}{2}(\boldsymbol{h}[2i] + \boldsymbol{h}[2i+1]) \approx \boldsymbol{h'}[2i+1]$, and when $0 < p < 1$ the pairs of values $\boldsymbol{h'}[2i]$ and $\boldsymbol{h'}[2i+1]$ are pulled close together in proportion to p.

However this equation is not terribly useful for steganalysis. Although it relates the cover to the stego image in an interesting way, it lacks an important component: the expected behaviour of histograms for natural cover images. Although histograms of natural cover images tend to be fairly "smooth", this is not true at peaks, and the smoothness varies widely depending on the source of the images and the image processing operations which were involved in acquiring it from a camera or scanner. That is why the chi-square statistic makes for a weak detector.

2.3.2 Structural Steganalysis

It only takes a small generalization, though, to create powerful steganalysis. Instead of a simple histogram, let us look at an **adjacency histogram** or **co-occurrence matrix**, which counts how often every *pair* of pixels intensities occur as neighbours.

$$\boldsymbol{h}[j,k] = \#\{i \,|\, \boldsymbol{x}[i] = j \text{ and } \boldsymbol{x}[i+1] = k\}.$$

Here we intend the order of data in the representation $\boldsymbol{x}$ to be such that neighbouring pixels have neighbouring values, for example by concatenating row scans of red, green, and blue bytes. One can include vertical as well as horizontal neighbouring pixels in the counts, and also symmetrise, but it makes no difference to the subsequent analysis (nor much to the outcomes).

The pairs of values effect also applies to adjacency histograms: a pair of pixels can change between bins $\boldsymbol{h}[2i, 2j]$, $\boldsymbol{h}[2i, 2j+1]$, $\boldsymbol{h}[2i+1, 2j]$, and $\boldsymbol{h}[2i+1, 2j+1]$, depending on whether the first and/or second pixel in the pair has its LSB flipped. Modelling the embedding as independent changes to pixels, as in the previous subsection, leads to the diagram in Figure 2.6.

In turn, we reach the following set of linear (approximate) equations

$$\begin{pmatrix} \boldsymbol{h'}[2i,2j] \\ \boldsymbol{h'}[2i,2j+1] \\ \boldsymbol{h'}[2i+1,2j] \\ \boldsymbol{h'}[2i+1,2j+1] \end{pmatrix} \approx \begin{pmatrix} (1-\frac{p}{2})^2 & \frac{p}{2}(1-\frac{p}{2}) & \frac{p}{2}(1-\frac{p}{2}) & (\frac{p}{2})^2 \\ \frac{p}{2}(1-\frac{p}{2}) & (1-\frac{p}{2})^2 & (\frac{p}{2})^2 & \frac{p}{2}(1-\frac{p}{2}) \\ \frac{p}{2}(1-\frac{p}{2}) & (\frac{p}{2})^2 & (1-\frac{p}{2})^2 & \frac{p}{2}(1-\frac{p}{2}) \\ (\frac{p}{2})^2 & \frac{p}{2}(1-\frac{p}{2}) & \frac{p}{2}(1-\frac{p}{2}) & (1-\frac{p}{2})^2 \end{pmatrix} \begin{pmatrix} \boldsymbol{h}[2i,2j] \\ \boldsymbol{h}[2i,2j+1] \\ \boldsymbol{h}[2i+1,2j] \\ \boldsymbol{h}[2i+1,2j+1] \end{pmatrix}. \quad (2.5)$$

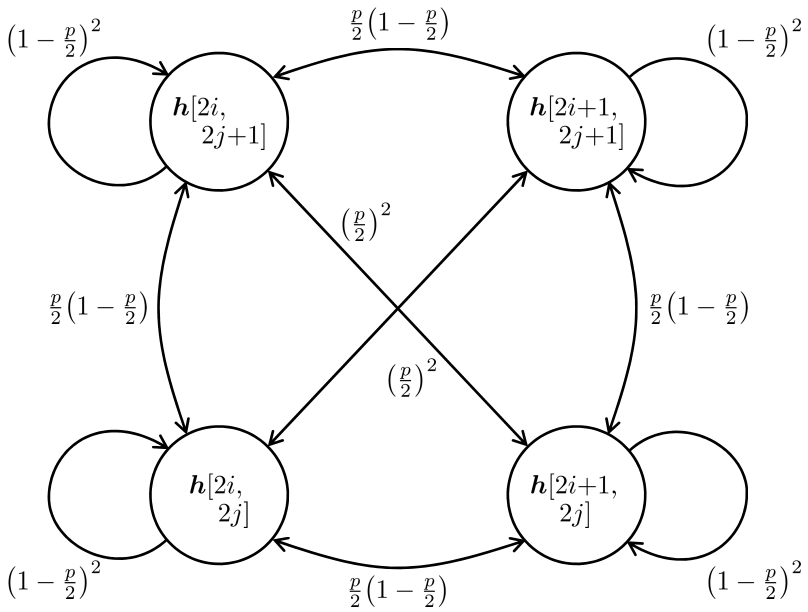


Figure 2.6: Graphical representation of the effect of LSB replacement on adjacency histograms.

Now we invert the system of equations, by inverting the matrix. It is easy to check that

$$\begin{pmatrix} \boldsymbol{h}[2i,2j] \\ \boldsymbol{h}[2i,2j{+}1] \\ \boldsymbol{h}[2i{+}1,2j] \\ \boldsymbol{h}[2i{+}1,2j{+}1] \end{pmatrix} \approx \frac{1}{(1-p)^2} \begin{pmatrix} (1-\frac{p}{2})^2 & -\frac{p}{2}(1-\frac{p}{2}) & -\frac{p}{2}(1-\frac{p}{2}) & (\frac{p}{2})^2 \\ -\frac{p}{2}(1-\frac{p}{2}) & (1-\frac{p}{2})^2 & (\frac{p}{2})^2 & -\frac{p}{2}(1-\frac{p}{2}) \\ -\frac{p}{2}(1-\frac{p}{2}) & (\frac{p}{2})^2 & (1-\frac{p}{2})^2 & -\frac{p}{2}(1-\frac{p}{2}) \\ (\frac{p}{2})^2 & -\frac{p}{2}(1-\frac{p}{2}) & -\frac{p}{2}(1-\frac{p}{2}) & (1-\frac{p}{2})^2 \end{pmatrix} \begin{pmatrix} \boldsymbol{h'}[2i,2j] \\ \boldsymbol{h'}[2i,2j{+}1] \\ \boldsymbol{h'}[2i{+}1,2j] \\ \boldsymbol{h'}[2i{+}1,2j{+}1] \end{pmatrix} \tag{2.6}$$

which expresses the histogram of the original cover in terms of the histogram of a stego image and the payload rate p.

In order to make a powerful detector, we need to find a property which is true of cover images and tends to fail for stego images (note that this is the reverse of the situation with the chi-square detector). It took researchers a while to come up with such a property, not helped by notation which obscured what was really going on, and it turns out that there are a number of possible options. For our detector we will use this following equation (sometimes called a **symmetry**):

$$\sum_i \boldsymbol{h}[2i, 2i+1] \approx \sum_i \boldsymbol{h}[2i+1, 2i+2]. \tag{2.7}$$

Why should this be true in covers? Both sides of the equation relate to the number of pairs of adjacent pixels differing by one. The LHS is the number of such pairs with the first pixel even, and the RHS is the number with the first pixel odd. It certainly makes sense that (2.7) should be approximately true in natural images, because even and odd values should not be related to subsequent pixel differences. It can be established empirically that it fails for stego images, and one could construct a statistic analogous to chi-square, but in fact we can use (2.7) to do something even more powerful.

Let us extract two equations from (2.6): with $j = i$,

$$\boldsymbol{h}[2i,2i{+}1] \approx \frac{1}{(1-p)^2}\Big(-\tfrac{p}{2}(1-\tfrac{p}{2})\boldsymbol{h'}[2i,2i] + (1-\tfrac{p}{2})^2\boldsymbol{h'}[2i,2i{+}1] + (\tfrac{p}{2})^2\boldsymbol{h'}[2i{+}1,2i] - \tfrac{p}{2}(1-\tfrac{p}{2})\boldsymbol{h'}[2i{+}1,2i{+}1]\Big) \tag{2.8}$$

and with $j = i{+}2$,

$$\boldsymbol{h}[2i{+}1, 2i{+}2] \approx \frac{1}{(1-p)^2}\Big(-\tfrac{p}{2}(1-\tfrac{p}{2})\boldsymbol{h'}[2i, 2i{+}2] + (\tfrac{p}{2})^2\boldsymbol{h'}[2i, 2i{+}3] \\ + (1-\tfrac{p}{2})^2\boldsymbol{h'}[2i{+}1, 2i{+}2] - \tfrac{p}{2}(1-\tfrac{p}{2})\boldsymbol{h'}[2i{+}1, 2i{+}3]\Big). \quad (2.9)$$

Now we can sum (2.8) and (2.9) over i, substitute into (2.7) – all of which is left as an exercise – and we reach a quadratic equation for p which depends only on $\boldsymbol{h'}$. In other words, we have found an approximate equation for the size of the unknown payload, in terms solely of the stego object and with no knowledge of the cover except for a belief in (2.7).

2.3.3 Performance of Simplified Couples

An estimator of p is called a **quantitative** steganalyzer (or **estimator**), because it fulfils a forensic tool beyond simple detector: it allows the Warden to estimate the size of the payload sent by Alice[6]. The simplified couples method has created a quantitative steganalyzer, and it is just the tip of the iceberg when it comes to estimators based on structural analysis of LSBR.

For a start, we can make additional assumptions about covers – $\sum_i \boldsymbol{h}[2i, 2i+3] \approx \sum_i \boldsymbol{h}[2i+1, 2i+4]$, $\sum_i \boldsymbol{h}[2i, 2i+5] \approx \sum_i \boldsymbol{h}[2i+1, 2i+6]$, etc. – and need to work out how to combine all the assumptions. Then we could move to analysis of triplets of pixels or even quadruples, which become much more complicated, with the estimator for p being the result of a polynomial of degree 4, 6 or higher. Many of my papers in the period 2004–2008 were on this topic.

But even the simplified detector we just presented is quite powerful. First, we can use its output as a decision function (give positive detection if the payload estimator exceeds a threshold). We used the same set of images as in section 2.2 and plotted ROCs for detection of relative payloads $p = 0.05, 0.1, \ldots, 0.25$ bits per cover byte. This is shown in Figure 2.7, and compare it with the same experiment for the chi-square detector in Figure 2.4: the simplified couples detector maintains low false positive and false negative rates even when the payload is much smaller, and has practically perfect detection for

[6]More precisely, it estimates the *number of embedding* changes made by Alice. If Alice should happen to embed a large payload which requires few changes, the Warden cannot possibly hope to know: when a location carries payload without a change it is, by definition, not detectable.

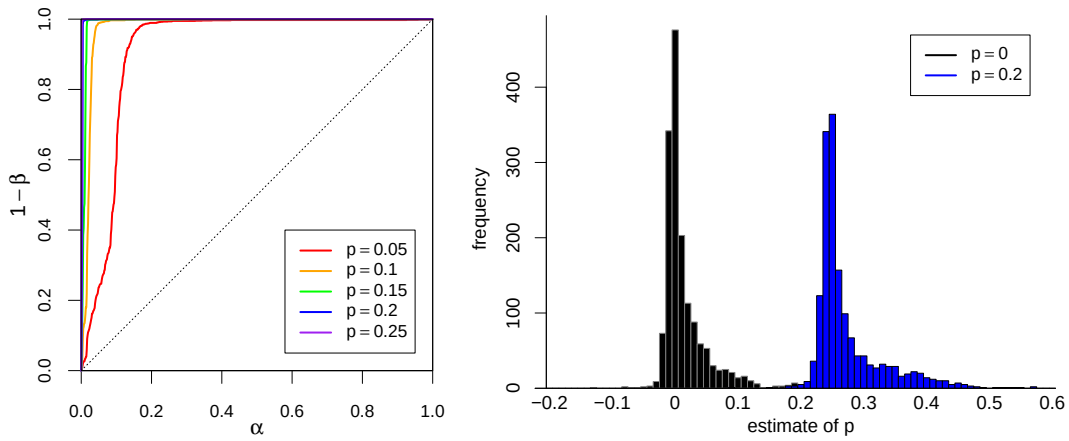


Figure 2.7: ROC curves for the simplified couples detector as observed in 1600 images, 3 megapixel RGB, with payload p bits per cover byte embedded by LSBR.

payloads as low as $p = 0.2$. More sophisticated structural detectors can reliably detect payloads as low as 0.01–0.02 bits per cover byte, but a lot depends on the cover source. (For example, we shall see in chapter 4 that one expects smaller *relative* payloads to be more detectable in larger images.)

We can also measure the accuracy of the forensic tools, estimating the size of hidden payload. We will not go into this, beyond looking at the results from the experiment on the right of Figure 2.7: when the payload is zero, the estimate is usually close to zero, and when the payload is 0.2 bits per cover byte, the estimate is usually close to 0.2. In this case the estimator is worst for high payloads, but this can be avoided by constructing more complicated estimates. In the literature the accuracy is often measured by the mean and variance of the estimate compared with the true payload, although this can be dangerous if the true variance is infinite (a theoretical possibility which can never be determined experimentally) so some have advocated robust measures of accuracy, such as median error and interquartile range.

The same structural methods, by the way, can also be applied to JPEG steganography schemes that use LSBR as their underlying embedding operation. JSteg is particularly vulnerable.

2.4 General Attacks using Machine Learning

Targeted steganalysis, of the type in previous section, is hard work. A researcher has to study the effect of embedding on histograms, or something similar, until they spot a statistic which is stable in cover images and changes when payload is added. A different statistic might be needed for every different embedding algorithm. For example, the chi-square and couples statistics are completely ineffective in detecting LSBM embedding (as we will see in Figure 2.8): by behaving consistently on cover pixels of different parity, LSBM avoids all the structural defects of LSBR and is much more difficult to detect. There is an example in (Fridrich, 2010, §11.4), which summarises some of the literature on this, and shows how poor the detectors are.

An alternative method for steganalysis rose to dominance over the period 2004-2010, gradually displacing most targeted attacks (apart from those which, like structural detectors, manage to exploit a particularly gross weakness). The idea is to use machine learning algorithms to do the work. There are three components to machine learning steganalysis:

(i) A set of **features**, a collection of statistics for each image. Often adjacency histograms, or similar counting statistics, are used.

(ii) A set of **training data**, which comes from a large library of innocent cover images (from the same source as Alice's, or a very similar source). Using the steganography algorithm, a matching set of stego images (usually all with the same sized payload) are also created.

(iii) A machine learning algorithm or **learner** which, having been fed the features from the training cover and stego data, is supposed to learn a general rule for separating cover images from stego images.

After being trained, the rule can be used to classify novel images as cover or stego.

This sets up the steganalysis problem as a supervised binary classification problem, which pins it firmly to the type (A)(2) we described in section 2.1. It is important that the training data comes from a source which is the same as, or very closely matched with, Alice's. Otherwise the learner might find rules which do not apply to the real data on which Warden wants to use it. The problem of mismatched training data in steganalysis is of current research interest.

2.4.1 Average Perceptron

Let us suppose that a feature extraction function $\phi : \mathcal{C} \to \mathbb{R}^k$ computes k real (or integer) features $\phi(\boldsymbol{x})$ from every object. We use $y = \pm 1$ for the **label**, which takes -1 when $\boldsymbol{x}$ is a cover image and $+1$ when $\boldsymbol{x}$ is a stego image. Many learners try to find a **linear** classifier

$$\hat{y} = \text{sign}(\boldsymbol{w} \cdot \phi(\boldsymbol{x})) \tag{2.10}$$

where $\hat{y}$ means the *estimated* label that corresponds to $\boldsymbol{x}$, and sign$(-)$ is the function which makes nonnegative numbers to 1 and negative numbers to -1. Training the learner amounts to finding **weights** $\boldsymbol{w}$[7], typically to minimize classification errors on training data. Nonlinear classifiers are more general but usually more difficult to find/train, and there are various tricks for reducing nonlinear problems to linear ones. See (Bishop, 2006) for more information about machine learning in general.

Given some labelled training data, $\{(\boldsymbol{x_i}, y_i) \,|\, i = 1, \ldots, n\}$, finding a good choice for $\boldsymbol{w}$ can be attempted in many ways. It is quite likely that no perfect $\boldsymbol{w}$ exists and "good choice" means that the classifier should work correctly – as often as possible – on examples *not* in the training data. One of the simplest algorithms for learning $\boldsymbol{w}$ is the **average perceptron**, the history of and justification for which we will not go into here; see (Bishop, 2006, §4.1.7) for more details. The perceptron algorithm is as follows:

(i) Set $\boldsymbol{w} = \boldsymbol{w_a} = \boldsymbol{0}$.
(ii) Pick a random element $(\boldsymbol{x}, y) \in \{(\boldsymbol{x_i}, y) \,|\, i = 1, \ldots, n\}$.
(iii) If $\text{sign}(\boldsymbol{w} \cdot \phi(\boldsymbol{x})) \neq y$ (i.e. training example i is misclassified) then $\boldsymbol{w}{=}\boldsymbol{w} + y\phi(\boldsymbol{x})$.
(iv) Set $\boldsymbol{w_a} = \boldsymbol{w_a} + \boldsymbol{w}$, return to step (ii).

The vector $\boldsymbol{w_a}$ is tracking the *running average* (the scalar constant is irrelevant) of the vector $\boldsymbol{w}$, which is being updated according to the perceptron rule. It may be necessary to repeat steps (ii)–(iv) many more times than n, so that each training point is visited multiple times, before $\boldsymbol{w_a}$ converges to a stable vector. It is this $\boldsymbol{w_a}$ which is used, after training, to classify novel objects by (2.10).

The perceptron does not usually make the most of its training data, though, and more complicated algorithms – particularly the kernelized support vector machine (KSVM)

[7] In order to allow some bias into the classifier, we can generalize a bit to $\hat{y} = \text{sign}(\boldsymbol{w} \cdot \phi(\boldsymbol{x}) + b)$ simply by adding a constant 1 to the features of every image; the weight element corresponding to this 1 works as b. See any book on machine learning for more explanation.

– dominated the steganalysis literature for the period roughly 2004–2011. But such complicated learning algorithms are slow to train, and this limits both the length of the feature vector (to a maximum of a few hundred features in practice) and the size of the training data (to a few thousand examples). Contemporary steganalysis is now using so-called **rich features**, first proposed in (Kodovksý & Fridrich, 2011), which involve much larger and rather generic counting features, typically 10000-50000 per image. They must use simpler classifiers such as perceptron or **Fisher linear discriminator** (FLD), often combining the results of many simple classifiers built on random subsets of features (a so-called **ensemble** classifier). This is a topic of current research.

2.4.2 SPAM Features

We will illustrate the paradigm of machine learning steganalysis using a simple feature set which we call **reduced SPAM**. They are inspired by the **SPAM**[8] **features** proposed in (Pevný et al., 2010), but the reduced features are simpler to describe and still sufficient to show how this type of steganalysis can work.

Good features should encode enough information about the presence of hidden data for the learning algorithm to find a good classification rule, but not be swamped by irrelevant information about the image content. Although early feature sets were inspired by the same sort of analysis as in section 2.3.2, researchers later realised that non-specialised feature sets would have better generality. So they usually boil down to elaborations of histograms, adjacency histograms, and so on.

The reduced SPAM features will be based on an adjacency histogram, but of a *filtered* image. Representing an image as an n-byte vector $\boldsymbol{x}$ where consecutive elements come from horizontal neighbours, we compute pixel differences:

$$\hat{\boldsymbol{x}}[i] = \boldsymbol{x}[i+1] - \boldsymbol{x}[i], \quad \text{for } i = 1, \ldots, n-1.$$

The purpose of the filter (which can be a lot more complex than the so-called Laplacian filter given here) is to reduce the effect of image content on the features, while leaving the effect of **stego noise** (alterations due to embedding changes) mostly unchanged. The elements of $\hat{\boldsymbol{x}}$ are integers, not necessarily positive.

Then we compute an adjacency histogram from the filtered image,

$$\boldsymbol{h}[j,k] = \#\left\{i \,|\, \hat{\boldsymbol{x}}[i] = j \text{ and } \hat{\boldsymbol{x}}[i+1] = k\right\}.$$

[8] Subtractive Pixel Adjacency Matrix

The reduced SPAM features will be normalized elements of $\boldsymbol{h}$. In order to keep down the dimensionality of the features (and hence the number weights to be learned by the classifier), we use only the central part of the adjacency histogram

$$\phi(\boldsymbol{x}) = \left(\frac{\boldsymbol{h}[j,k]}{\sum_{j,k=-T}^{T} \boldsymbol{h}[j,k]} \;\middle|\; j,k \in \{-T,\ldots,T\} \right) \tag{2.11}$$

where T is big enough to include useful data but small enough to keep the dimension $(2T+1)^2$ down. We will use $T = 4$, so that (2.11) gives 81 features from each image.

The real SPAM features differ in three important ways:

(i) They use filters in multiple directions: horizontal, vertical, and diagonal differences.

(ii) They are based on the frequency of triples rather than pairs, thus $\boldsymbol{h}[j,k,l] = \#\{i \,|\, \hat{\boldsymbol{x}}[i] = j \text{ and } \hat{\boldsymbol{x}}[i+1] = k \text{ and } \hat{\boldsymbol{x}}[i+2] = l\}$. This leads to higher dimensionality, and the standard SPAM features have dimension 686.

(iii) They normalize in a different way, computing empirical conditional probabilities $\mathrm{P}[\hat{\boldsymbol{x}}[i] = j \,|\, \hat{\boldsymbol{x}}[i+1] = k \text{ and } \hat{\boldsymbol{x}}[i+2] = l]$ instead of simple counts.

These changes make the features more powerful, more able to distinguish stego objects with smaller payloads from covers. And recent work has extended SPAM-style features still further, combining features extracted from different image filters (Fridrich & Kodovský, 2012). The dimensionality is a few thousand features per image, which necessitates efficient processing and speedy learners.

2.4.3 Detecting LSBM by Reduced SPAM Features

Let us see how well the reduced SPAM features work, in combination with the average perceptron learner. We will test against LSBM, where the specialised structural detectors are not effective.

We used the same set of 1600 images as in section 2.2, but split into 800 for training and 800 for testing (it is vital to test machine learning classifiers on data they have not seen before). 82 features – the reduced SPAM features plus a constant 1 – were extracted from the 800 training cover images, and the 800 corresponding stego images with maximum LSBM payload (1 bit per cover byte). These were fed into an average perceptron learner which produced a weight vector $\boldsymbol{w}$.

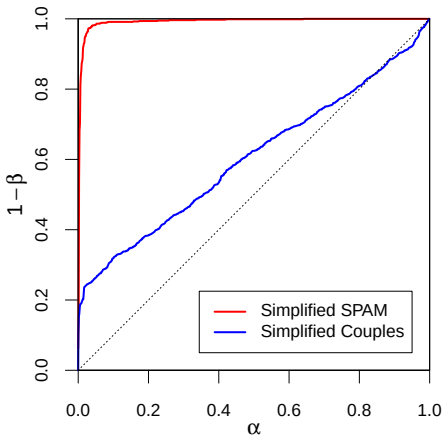


Figure 2.8: ROC curves for the detector created by training an average perceptron learner on reduced SPAM features, and the simplified couples detector (which does not work), when the payload is 3bpp embedded using LSBM.

The weight $\boldsymbol{w}$ was then used to test features from the other 800 covers, and corresponding stego images. In order to create a ROC curve, we set thresholds on the raw output of the decision function

$$D(\boldsymbol{x}) = \begin{cases} \text{True}, & \text{if } \boldsymbol{w} \cdot \phi(\boldsymbol{x}) \geq \tau, \\ \text{False}, & \text{if } \boldsymbol{w} \cdot \phi(\boldsymbol{x}) < \tau, \end{cases}$$

replacing the usual sign decision function by one whose sensitivity can be varied. Note that this is typically *not* the optimal way to alter the false positive/negative tendency of a machine learning classifier: instead, one should change the learner itself, penalizing one type of error over the other. But this is very slow to change, requiring a re-train for every point of the ROC, and is not used in steganalysis.

The ROC produced by this method, and that for the simplified couples detector, are shown in Figure 2.8. Observe that the couples detector barely works at all because LSBM does not have parity structure (that its performance is above random is due to some cover images which are over/underexposed; LSBM and LSBR behave identically on such pixels). Despite our SPAM features being simplified and reduced, and our use of a simple learner, we still have very good detection accuracy. It does fall off, however, for smaller payloads (experiments not included here) and LSBM payloads of around 0.1 bits per cover byte are difficult to detect accurately, even with state-of-art steganalysis.

2.4.4 Features for JPEG Steganography

Machine learning steganalysis was first used for JPEG images, against embedding methods like F5 and its contemporaries. The ideas are similar, using counts of the number of times different quantized DCT coefficients are observed. With DCT coefficients there are more different versions of "neighbour": the neighbouring coefficient from the same 8×8 block, or the corresponding coefficient in a neighbouring block. So the feature sets are a bit more complex to define, though the ideas are much the same as for spatial-domain features.

There are also different ways to filter a JPEG image. Coefficient differences are used, but an important method is called **calibration**. Here, the idea is to get a peek at the cover characteristics when (as usual) only the stego object is available, by decompressing the JPEG file, cropping it (by say 4 pixels on the top and left) to desynchronize the 8×8 boundaries, and recompressing. Calibration has proved to be a powerful addition to JPEG feature sets, although why it actually works can be a bit mysterious.

We will not say more about steganalysis of JPEG images here, save to mention that the F5 algorithm, which we saw in subsection 1.5.2, is basically detected because of the histogram changes of the embedding operation changing only towards zero. Shrinkage makes this effect much worse.

2.5 The Wider Picture

Our chapter has focused on case (A)(2) from section 2.1. This is appropriate for testing the security of a stegosystem, because it tests the undetectability of the embedding in a situation favourable to the detector. But practical steganalysis will not always be so favourable.

Quantitative steganalysis turns a classification problem into a regression problem, and can deal with case (B), where the Warden does not know the payload length. But such detectors are currently rare, except for LSB embedding (but see (Pevný et al., 2012) for use of the machine learning paradigm in quantitative steganalysis). Almost all literature applying binary classifiers to steganalysis neglect this case; one publication (Pevný, 2011) examines this situation and finds that training with random payloads works quite well.

Case (C) is called **blind steganalysis**[9]. It is difficult to define precisely because the

[9]The book (Fridrich, 2010) seems to conflate the idea of using machine learning with detecting unknown algorithms. In fact, there are plenty of targeted detectors built on machine learning algorithms.

Warden is supposed to detect any kind of embedding, including methods they have never thought of, but not falsely flag unusual images which have been subject to innocent alteration such as image processing. There is some work applying anomaly detectors to steganalysis, usually replacing two-class supervised classification algorithms with one-class supervised machines. Unsupervised techniques can also be used.

Only recently has there been work on case (3), where the detector has to be trained on data which is not an exact match for Alice's cover source. Simpler classifiers tend to generalize a little better, and the technique of 'calibration' (sect. 2.4.4) seems to be important. But for realistic conditions we need to reconsider the Prisoners' Problem in two ways.

First, why did the Warden know which of Alice's image to look at? Mostly likely they would not, and Alice would send a stream of communications, only a few of which contain hidden payload. There are two interesting questions here. How should Alice allocate her payload amongst multiple images: put all of it into a few images, or spread it amongst all of them? And how can the Warden detect this, without being swamped by false positives if Alice sends many innocent images, or without missing all the little bits of evidence that should accumulate if Alice sends a little payload in every object? These problems were posed in (Ker, 2006) and there has been little success in answering the questions definitively.

Second, why did the Warden know that Alice, out of all the prisoners, is the one doing secret communication? In practice, a Warden would be a person or agency monitoring a network or social media site, scanning every image from every user without knowledge of which user might be a steganographer, or which of their images are the ones with payload. Some recent work looked at this as a problem of clustering or anomaly detection but on the level of actors (users/prisoners) rather than individual objects (Ker & Pevný, 2014). This even allows consideration of case (4), where the Warden knows nothing of Alice's source, because they calibrate their expectation of users by the behaviour of the majority: it is completely unsupervised.

These topics are the subject of a recent survey paper, which looks at making steganography and steganalysis research more applicable to real world problems (Ker et al., 2013).

Bibliography

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Dumitrescu, S., Wu, X., & Wang, Z. (2002). Detection of LSB steganography via Sample Pair Analysis. In *Proceedings of 5th Information Hiding Workshop*, volume 2578 of *Lecture Notes in Computer Science* (pp. 355–372). Springer. Apparently not freely downloadable, but subscribers can read it at SpringerLink.

Fridrich, J. (2010). *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge University Press.

Fridrich, J., Goljan, M., & Du, R. (2001). Reliable detection of LSB steganography in color and grayscale images. In *Proceedings of 3rd ACM Workshop on Multimedia and Security* (pp. 27–30). ACM. Available at `http://ws2.binghamton.edu/fridrich/Research/acmwrkshp_version.pdf`.

Fridrich, J. & Kodovský, J. (2012). Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*, 7(3), 868–882. Available at `http://dde.binghamton.edu/kodovsky/pdf/TIFS2012-SRM.pdf`.

Ker, A. D. (2005). A general framework for the structural steganalysis of LSB replacement. In *Proceedings of 7th Information Hiding Workshop*, volume 3727 of *Lecture Notes in Computer Science* (pp. 296–311). Springer. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK13D.pdf`.

Ker, A. D. (2006). Batch steganography and pooled steganalysis. In *Proceedings of 8th Information Hiding Workshop*, volume 4437 of *Lecture Notes in Computer Science* (pp. 265–281). Springer. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK18D.pdf`.

Ker, A. D., Bas, P., Böhme, R., Cogranne, R., Craver, S., Filler, T., Fridrich, J., & Pevný, T. (2013). Moving steganography and steganalysis from the laboratory into the real world. In *Proc. 1st ACM Workshop on Information Hiding and Multimedia Security* (pp. 45–58). ACM. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK57B.pdf`.

Ker, A. D. & Pevný, T. (2014). The steganographer is the outlier: Realistic large-scale steganalysis. *IEEE Transactions on Information Forensics and Security*, 9(9), 1424–1435. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK58C.pdf`.

Kodovksý, J. & Fridrich, J. (2011). Steganalysis in high dimensions: Fusing classifiers built on random subspaces. In *Proceedings of SPIE/IS&T Electronic Imaging: Media Watermarking, Security, and Forensics III*, volume 7880 (pp. 0L01–0L13). SPIE. Available at `http://dde.binghamton.edu/kodovsky/pdf/Kod11spie.pdf`.

Pevný, T. (2011). Detecting messages of unknown length. In *Proceedings of SPIE/IS&T Electronic Imaging: Media Watermarking, Security, and Forensics III*, volume 7880 (pp. 0T01–0T12). SPIE. Not freely available, but the conference presentation slides are at `http://www.researchgate.net/profile/Tomas_Pevny2/publication/253393301_Detecting_messages_of_unknown_length/links/00b7d537336d31d34e000000.pdf`.

Pevný, T., Bas, P., & Fridrich, J. (2010). Steganalysis by subtractive pixel adjacency matrix. *IEEE Transactions on Information Forensics and Security*, 5(2), 215–224. Available at `http://ws2.binghamton.edu/fridrich/Research/paper_6_dc.pdf`.

Pevný, T., Fridrich, J., & Ker, A. D. (2012). From blind to quantitative steganalysis. *IEEE Transactions on Information Forensics and Security*, 7(2), 445–454. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK41D.pdf`.

Westfeld, A. & Pfitzmann, A. (1999). Attacks on steganographic systems. In *Proceedings of 3rd Information Hiding Workshop*, volume 1768 of *Lecture Notes in Computer Science* (pp. 61–76). Springer. Available at `http://users.ece.cmu.edu/~adrian/487-s06/westfeld-pfitzmann-ihw99.pdf`.

Chapter 3

Countermeasures

Reading (course text):	Fridrich, §8.1–4, 9.1, 9.4
Alternatives & further depth:	Fridrich, rest of chapters 8 & 9, Appendix C Böhme, §2.8.2

Having seen the sorts of techniques with which the Warden can detect payload, we now return to the role of the steganographer. This chapter examines different ways to avoid detection. Naive ideas, such as trying to preserve the histogram when embedding, do not work. But when Alice's embedding uses **coding**, where the changes she makes to the cover are not simply to overwrite parts of it with her payload, steganography can be performed with fewer changes (making it harder to detect). Coding also, as a bonus, helps solve the non-shared selection channel problem which affects F5.

There are connections with classical coding theory, but we will try to avoid getting drawn into this subject. Chapters 8 and 9 of (Fridrich, 2010) give the connections in much more detail. We will use a minimum of coding terminology, and prove a simplified theorem to illustrate the bound on embedding efficiency.

Finally we outline how to perform "optimal" embedding, if not all embedding changes are equally dangerous, which minimizes changes to a distortion function. Recent research has found interesting connections with statistical physics.

3.1 Preserving Histograms

The first countermeasures to statistical steganalysis were based on a simple idea. If the Warden can detect us by some function of the image histogram, we should fix the embedding process to reduce the change to the histogram, or to ensure that there is no change at all to the counts in each histogram bin. For example, half of the pixels could be used to hide payload in the usual way, and the other half modified solely with the aim of restoring the original histogram.

Two notable embedding algorithms which try to preserve the histogram are OutGuess[1] and Steghide[2]. The algorithm used by OutGuess was described in (Provos, 2001): its main aim is to preserve not only the global histogram (in JPEGs, of quantized DCT coefficients) but also the histograms within local regions of the image, and it also contains a few other minor tweaks aiming to improve embedding efficiency. The Steghide algorithm comes from (Hetzl & Mutzel, 2005), which gives a graph-theoretic formulation of the following problem: try to exchange pixels, rather than change any of them. If pixels are exchanged, the histogram cannot be altered at all. Naturally, we must take care only to exchange pixels with similar values, else the result is visible.

There is a problem with these approaches. Although one might successfully preserve a histogram, what is to stop the opponent (the Warden) from making a decision based on an adjacency histogram? If one tries to preserve the entire adjacency histogram (not easy), the opponent could use 3rd-order data such as frequencies of pixel triplets. Indeed, trying to preserve the histogram of an image usually causes *more* changes to the adjacency histogram, and so on.

With modern steganalysis, which uses a wider range of features than simple histograms, embedding algorithms such as OutGuess and Steghide are comprehensively broken. Indeed, OutGuess is one of the most easily detectable steganography systems for JPEG files (Pevný & Fridrich, 2007). Of course it is easy, with hindsight, to see the flaws in their design, but it is the nature of research for progress to be made by breaking previous work.

[1] The original website `http://www.outguess.org/` has gone, but a version of OutGuess was re-released in 2014 at `http://www.outguess-rebirth.com/`. At the moment it appears to have the same poor security.

[2] `http://steghide.sourceforge.net/`

3.2 Improving Embedding Efficiency

If we cannot preserve all the possible features in the cover, we should try to change them as little as possible. That is, we want as few embedding changes as possible. Recall that the **embedding efficiency** is the reciprocal of the average number of changes per payload bit embedded; this section is about increasing the embedding efficiency.

We have already mentioned that the payload should be losslessly compressed prior to embedding. We take this as given. If the compressed payload is of maximum length (for example 1 bpnc if we hide in LSBs of nonzero DCT coefficients of a JPEG, or 3 bpp if we hide in raw colour images) then there is nothing we can do to improve efficiency. But if the payload is of less than maximum length then we have some flexibility: we can convey information in the *choice* of pixels changed, as well as their contents. When information is conveyed indirectly it is called **coding**. We then need a clever way for the recipient to extract that information, without knowledge of the original cover; the idea was first proposed by Ron Crandall on a steganography mailing list, but was not formally published until some time later.

To reason about coding, we need to decompose the embedding operation. Sadly, not all literature makes this decomposition explicit.

3.2.1 Decomposing the Stegosystem

Recall the definition of the embedding and extraction functions:

$$\begin{aligned} \text{Emb} &: \mathcal{C} \times \mathcal{K} \times \mathcal{M} \to \mathcal{C}, \\ \text{Ext} &: \mathcal{C} \times \mathcal{K} \to \mathcal{M}. \end{aligned}$$

We suppose that embedding can be broken down into three parts, which we next describe with reference to examples we have previously seen. We shall be a bit more concrete about the cover and message: let us say that the cover is n elements from an alphabet Σ (e.g. n quantized DCT coefficients, which are integers), and the message is $m < n$ symbols from an alphabet Γ (e.g. a binary string of length m). Thus $\mathcal{C} = \Sigma^n$, $\mathcal{M} = \Gamma^m$, and $\mathcal{K}$ can remain arbitrary.

At the lowest level, there is an operation which extracts one message symbol from one cover element:

$$\text{ExtSym} : \Sigma \to \Gamma.$$

For example, this might read the LSB from an integer, or its remainder (mod 3). Similarly, there is an operation to change a cover element to hold a specified message symbol

$$\text{EmbSym} : \Sigma \times \Gamma \to \Sigma.$$

This could be the LSBR or F5 embedding operation, and is also allowed to behave randomly (e.g. LSBM). The correctness criterion is that $\text{ExtSym}(\text{EmbSym}(c), m) = m$ for all $m \in \Gamma$ and $c \in \Sigma$. Note that these low-level operations are unkeyed and work on single elements at a time.

At the second level, there is an operation which converts a string of n message symbols into the true message, using the key:

$$\text{Decode} : \Gamma^n \times \mathcal{K} \to \Gamma^m,$$

and a corresponding function to turn one string of n message symbols into another which encodes a desired message:

$$\text{Code} : \Gamma^n \times \mathcal{K} \times \Gamma^m \to \Gamma^n.$$

In the examples we have seen, these might simply use the key to read or write m of the n symbols in a certain order, but we shall see how to improve on this. The correctness criterion is that $\text{Decode}(\text{Code}(\boldsymbol{b}, \boldsymbol{k}, \boldsymbol{m}), \boldsymbol{k}) = \boldsymbol{m}$.

Finally, the embedding and extraction functions are constructed by combining the encoding and decoding operation with the low-level operation, thus[3]

$$\begin{aligned} \text{Emb}(\boldsymbol{c}, \boldsymbol{k}, \boldsymbol{m}) &= \texttt{zipwith}\ \text{EmbSym}\ \boldsymbol{c}\ \text{Code}(\texttt{map}\ \text{ExtSym}\ \boldsymbol{c},\ \boldsymbol{k},\ \boldsymbol{m}), \\ \text{Ext}(\boldsymbol{c}, \boldsymbol{k}) &= \text{Decode}(\texttt{map}\ \text{ExtSym}\ \boldsymbol{c},\ \boldsymbol{k}). \end{aligned}$$

This presentation cannot directly describe stegosystems where certain cover elements are skipped depending on their value (like the F5 algorithm), but we return to that in section 3.3.

[3] We are using functional notation

$$\begin{aligned} \texttt{map}\ f\ (x_1, \ldots, x_n) &= (f(x_1), \ldots, f(x_n)), \text{ and} \\ \texttt{zipwith}\ g\ (x_1, \ldots, x_n)\ (y_1, \ldots, y_n) &= (g(x_1, y_1), \ldots, g(x_n, y_n)). \end{aligned}$$

3.2.2 Using Syndromes

We will illustrate improved-efficiency embedding with a simple example. Take a tiny cover with 7 elements (perhaps intensity bytes or quantized DCT coefficient integers), and suppose that the desired message length is 3 bits: $n = 7$, $m = 3$. We can write

$$\boldsymbol{c} = (c_1, \ldots, c_7).$$

We will use the LSBM operations for manipulating individual symbols:

$$\mathrm{ExtSym}(c) = c \,(\mathrm{mod}\ 2)$$

and

$$\mathrm{EmbSym}(c, b) = \begin{cases} c, & \text{if } \mathrm{ExtSym}(c) = b \text{ already,} \\ 1, & \text{if } \mathrm{ExtSym}(c) \neq b \text{ and } c = 0, \\ 254, & \text{if } \mathrm{ExtSym}(c) \neq b \text{ and } c = 255, \\ c \pm 1, & \text{otherwise, where } \pm 1 \text{ is picked at uniformly random.} \end{cases}$$

We will write

$$\boldsymbol{b} = (b_1, \ldots, b_7)$$

for the LSBs of the cover, $\boldsymbol{b} = \texttt{map}\ \mathrm{ExtSym}\ \boldsymbol{c}$.

Now define a binary matrix $\boldsymbol{H}$ by

$$\boldsymbol{H} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{3.1}$$

This matrix comes from coding theory. It is the parity-check matrix of a Hamming code. All we need to know about coding theory is that a matrix (over some field, probably binary) is called the **parity-check matrix** of an $\boldsymbol{[n, k]}$ **code** if it of dimension $(n - k) \times n$ and its rows are linearly independent. Note that our matrix algebra, including the definition of linear independence, must be over the right finite field: in this case, binary arithmetic.

Let us pretend that there is no embedding key for a moment (it simplifies the presentation, but in practice the key could be used to scramble the order of bits in the cover). Now using binary (mod 2) arithmetic, we can define

$$\begin{aligned} \mathrm{Decode}(\boldsymbol{b'}) &= \boldsymbol{H}\boldsymbol{b'}, \\ \mathrm{Code}(\boldsymbol{b}, \boldsymbol{m}) &= \boldsymbol{b} + \boldsymbol{H}^{-\mathbf{1}}(\boldsymbol{m} - \boldsymbol{H}\boldsymbol{b}). \end{aligned} \tag{3.2}$$

When $\boldsymbol{H}$ is a parity-check matrix for a code then $\boldsymbol{Hb}$ is called a **syndrome** of $\boldsymbol{b}$; the message $\boldsymbol{m}$ is communicated as the syndrome of the vector $\text{Code}(\boldsymbol{b}, \boldsymbol{m})$.

Now $\boldsymbol{H^{-1}}$ is not to be taken as a literal inverse because non-square matrices do not have inverses. $\boldsymbol{H^{-1}}(\boldsymbol{x})$ means "find some vector $\boldsymbol{e}$ such that $\boldsymbol{He} = \boldsymbol{x}$". A solution is guaranteed to exist if the rows of $\boldsymbol{H}$ are linearly independent. Then

$$\text{Decode}(\text{Code}(\boldsymbol{b}, \boldsymbol{m})) = \boldsymbol{H}\big(\boldsymbol{b} + \boldsymbol{H^{-1}}(\boldsymbol{m} - \boldsymbol{Hb})\big) = \boldsymbol{Hb} + \boldsymbol{m} - \boldsymbol{Hb} = \boldsymbol{m}.$$

In this case, because the columns of $\boldsymbol{H}$ in (3.1) are all the nonzero sequences of 3 bits, the solution to $\boldsymbol{He} = \boldsymbol{x}$ very easy to compute: if $\boldsymbol{x} = 0$ then $\boldsymbol{e} = 0$, otherwise $\boldsymbol{e} = (0, \ldots, 0, 1, 0, \ldots, 0)$ where the 1 is in position i if $\boldsymbol{x}$ is column i of $\boldsymbol{H}$.

This stegosystem communicates a relative payload (in this context denoted α) of $\alpha = 3/7$ bits per cover element. What is the embedding efficiency (denoted e)? As always, this is taken as an average over random messages:

With probability $\frac{1}{8}$, $\boldsymbol{m} = \boldsymbol{Hb}$ already: $\boldsymbol{e} = \boldsymbol{0}$. Zero changes.
With probability $\frac{7}{8}$, $\boldsymbol{m} \neq \boldsymbol{Hb}$ and so $\boldsymbol{e}$ contains one nonzero element: only one of the seven cover element needs to have its LSB flipped. One change.

For an average of $\frac{7}{8}$ changes it communicates 3 message bits, an embedding efficiency of $e = 3/\frac{7}{8} \approx 3.43$. This is considerably higher than the value 2 which would be obtained by choosing three fixed (given by the key) cover elements and changing their LSBs if necessary. The reason this is possible is because the embedder has a *choice* of which cover elements to change, hence communicating more information than flipping fixed elements. The recipient does not need to know the original cover, or to know exactly which elements were changed; they only need to compute the syndrome.

Of course, real covers are much larger than 7 elements. We could use this code by splitting the cover into groups of 7, embedding 3 bits of payload in each (perhaps in pseudorandom order specified by the secret key). Thus for any payload which is no greater than 3/7 times the total number of embedding locations, and with a low-level embedding operation which puts one bit into one cover element, we can use the code to achieve an embedding efficiency of $3/\frac{7}{8} \approx 3.43$. What about relative payloads of greater than 3/7? Well, we could embed by simple overwriting in some of the groups of 7, and use the code in others. This achieves an embedding efficiency of somewhere between 2 and 3.43. In general one can mix-and-match different codes over different groups of pixels.

3.2.3 Hamming Codes

More generally, embedding using syndromes has the following form. Given the parity-check matrix $\boldsymbol{H}$ of an $[n, k]$ code, we use a keyed version of (3.2) to perform encoding and decoding. The underlying payload symbols Γ should come from the same field as $\boldsymbol{H}$, so for example if we use ternary embedding then $\boldsymbol{H}$ is a matrix with entries (mod 3). Because $\boldsymbol{H}$ is $(n-k) \times n$, we will be embedding $n-k$ payload symbols in n cover locations, for a relative embedding rate of $\alpha = (n-k)/n$.

If

$$\text{Code}(\boldsymbol{b}, \boldsymbol{k}, \boldsymbol{m}) = \boldsymbol{b} + \boldsymbol{e}(\boldsymbol{b}, \boldsymbol{k}, \boldsymbol{m})$$

then nonzero elements of $\boldsymbol{e}$ are the locations of the embedding changes, and so we want $\boldsymbol{e}$ to have minimum **Hamming weight** (number of nonzeros). When we use a syndrome code with matrix $\boldsymbol{H}$, then $\boldsymbol{e}$ should be a minimal Hamming weight solution to $\boldsymbol{He} = \boldsymbol{x}$. This is called a **coset leader** and then efficient embedding involves finding matrices $\boldsymbol{H}$ such that

(i) coset leaders have small Hamming weight (if the average Hamming weight of a coset leader is w then the average embedding efficiency will be $(n-k)/w$), and

(ii) it is computationally feasible to find coset leaders.

If the matrix $\boldsymbol{H}$ is random and n is large then it can be infeasible to find a coset leader. It is not just a case of solving the simultaneous equations $\boldsymbol{He} = \boldsymbol{x}$, but of finding the solution $\boldsymbol{e}$ with minimum Hamming weight, and in general this is an NP-complete problem. But for **structured codes**, where $\boldsymbol{H}$ has been suitably designed, coset leaders can be found quickly.

The example in the previous subsection used a structured matrix $\boldsymbol{H}$, which is one of a family called the **Hamming codes**, and it is easy to see how to generalise it. For any integer $p \geq 2$, the parity-check matrix of the $[2^p - 1, 2^p - p - 1]$ binary Hamming code has columns which are all the different nonzero vectors of length p. The coset leaders are found in the same way as before. To solve $\boldsymbol{He} = \boldsymbol{x}$, if $\boldsymbol{x} = 0$ then $\boldsymbol{e} = 0$, otherwise $\boldsymbol{e} = (0, \ldots, 0, 1, 0, \ldots, 0)$ where the 1 is in position i if $\boldsymbol{x}$ is column i of $\boldsymbol{H}$. By arranging the columns in binary order, one can write down i immediately.

For general p, we store p payload bits in $2^p - 1$ cover locations, making zero changes with probability 2^{-p} and one change otherwise. So the overall relative payload is

$$\alpha = \frac{p}{2^p - 1}$$

and the embedding efficiency is

$$e = p\frac{2^p}{2^p - 1}.$$

Note that larger p forces smaller relative payloads, but at higher embedding efficiencies. So using $p = 2$, splitting a cover into blocks of size 3, allows a relative payload of up to $2/3$ and an embedding efficiency of $8/3$. When $p = 3$ we have the $[7, 4]$ binary Hamming code of the previous section, payloads of up to $3/7$ and embedding efficiency of $24/7$. Using $p = 4$, allows a relative payload of $4/15$ and embedding efficiency of $64/15$.

Suppose that we wanted to hide $21 \cdot 14 = 294$ bits of payload in the cover from Figure 1.1. Dividing its 360 000 embedding locations (RGB bytes) into 21 blocks of $2^{14} - 1$ (which leaves a few thousand unused) we can take $p = 14$ and use the $[2^{14} - 1, 2^{14} - 14 - 1]$ binary Hamming code to embed 14 bits of payload into each block, say using the LSBM operation, with an average of very nearly 21 embedding changes in total. Compared with simple LSBM the payload requires approximately 7 times fewer changes, which must surely be much less detectable. (We could do even better with codes over ternary alphabets and ternary embedding.)

As before, we could use different combinations of these codes to interpolate between the different relative payload sizes, with the embedding efficiency similarly interpolated. We have left open, however, the question of how the receiver knows exactly which codes to use to receive the payload; it could be part of the secret key, or stored in the first few (uncoded) bits of payload.

Hamming codes are not the only option for syndrome coding. It turns out that unstructured random matrices $\boldsymbol{H}$ work better, but they have to be **sparse** (having relatively few nonzeros). Sparsity allows for both high embedding efficiency and a reasonably quick search for coset leaders (Filler, 2007). For payloads near to maximum, random codes of small dimension are appropriate (Fridrich & Soukal, 2006). An advantage of using random matrices $\boldsymbol{H}$ is that they can be generated (according to some algorithm which gives the right sort of sparsity) from the secret key $\boldsymbol{k}$, removing the need for any further permutation of cover elements. The idea that the key tells the receiver *where* to find the payload no longer holds, instead it *decodes* the payload for the receiver.

3.2.4 Theoretical Bounds

How efficient is it possible to be? In this section we seek an upper bound on the embedding efficiency. This bound applies not only to syndrome coding (in the language

of coding theory, linear codes) but in fact to any stegosystem of the form described in subsection 3.2.1.

Let us fix $\Gamma = \{0, 1\}$. We bound the embedding efficiency in the following way: suppose that we have a cover of n elements and allow ourselves to make up to c changes; how big can the payload be? Simply by counting the number of different outcomes, we could only possibly convey

$$\sum_{i=0}^{c} \binom{n}{i} \tag{3.3}$$

different messages, because that is the number of different stego objects reachable from the cover in at most c changes. Our task is to bound (3.3).

Assume $c \leq \frac{n}{2}$. (There is no point using a binary coding function which makes more than $n/2$ changes on average: it could only achieve an embedding efficiency of less than 2, since the payload cannot be larger than n.) Then we have

$$\begin{aligned}
\sum_{i=0}^{c} \binom{n}{i} &= c^{-c}(n-c)^{-(n-c)} \sum_{i=0}^{c} \binom{n}{i} \Big(\frac{c}{n-c}\Big)^{c} (n-c)^{n} \\
&\overset{(a)}{\leq} c^{-c}(n-c)^{-(n-c)} \sum_{i=0}^{c} \binom{n}{i} \Big(\frac{c}{n-c}\Big)^{i} (n-c)^{n} \\
&\overset{(b)}{\leq} c^{-c}(n-c)^{-(n-c)} \sum_{i=0}^{n} \binom{n}{i} \Big(\frac{c}{n-c}\Big)^{i} (n-c)^{n} \\
&\overset{(c)}{=} c^{-c}(n-c)^{-(n-c)} n^{n} \\
&= \big(\tfrac{c}{n}\big)^{-c} \big(\tfrac{n-c}{n}\big)^{-(n-c)}
\end{aligned} \tag{3.4}$$

where the first equation is just multiplying by constants; (a) follows because $\frac{c}{n-c} < 1$, which follows from $c \leq \frac{n}{2}$; (b) is because the additional terms are all positive; (c) is just a binomial expansion of $(c + n - c)^n$.

In this notation, the relative payload is defined by

$$\alpha = \frac{\log_2 \# \mathcal{M}}{n}$$

and therefore (3.4) gives

$$\alpha \leq -\tfrac{c}{n} \log_2 \tfrac{c}{n} - \tfrac{n-c}{n} \log_2 \tfrac{n-c}{n} = H(\tfrac{c}{n}) \tag{3.5}$$

where H is the **binary entropy function**

$$H(x) = -x \log_2 x - (1-x) \log_2 (1-x). \tag{3.6}$$

This function has an important place in coding and communication theory. Because H is monotone increasing on the range $[0, 1/2)$, we can invert (3.5) to get

$$c \geq nH^{-1}(\alpha). \tag{3.7}$$

Now recall that the embedding efficiency is the number of bits of payload per change, i.e.

$$e = \frac{\alpha n}{c}.$$

We can use (3.7) to get the upper bound

$$e \leq \frac{\alpha}{H^{-1}(\alpha)}. \tag{3.8}$$

Now this is not actually the bound we want, because it applies to the worst-case message: we have shown that at least *some* messages must use at least $H^{-1}(\alpha)/\alpha$ changes per payload bit, but not necessarily all. This worst-case is called the **lower embedding efficiency**, and it is not quite the same as the embedding efficiency we defined using the *average* number of changes over random messages. However, it can be proved (way beyond our syllabus) that the two are equal asymptotically as $n \to \infty$. Therefore this bound should be considered an asymptotic limit on e for large covers.

A diagram of the relationship between α and e is displayed in Figure 3.1. Note that the x-axis is nonlinear. It shows the bound, and also the relative payload and embedding efficiency of the binary Hamming code family (recall that they can be interpolated by mixing blocks of different size using different p). The least efficient is simply LSB replacement or matching and for large relative payloads the Hamming codes are some way below the optimum. (It has been shown that sparse random codes come extremely close to the bound.) For small payloads, very efficient embedding is possible; we already know that small payloads were harder to detect, and boosted efficiency makes them even more difficult. Therefore the lesson is to keep payloads as small as possible.

Recall our discussion of **ternary embedding** in subsection 1.4.1: if one is going to make ± 1 changes to pixels/coefficients/whatever then we can reduce the number of changes by using $\Gamma = \{0, 1, 2\}$ and $\mathrm{ExtSym}(c) = c \,(\mathrm{mod}\, 3)$. For non-binary alphabets the bound (3.8) is higher; you will derive the bound as an exercise, and it is plotted in the same figure. In fact, there is a clever trick which allows efficiency very close to that of ternary embedding *without* having to re-code the payload in a ternary alphabet, which we will mention in the next section.

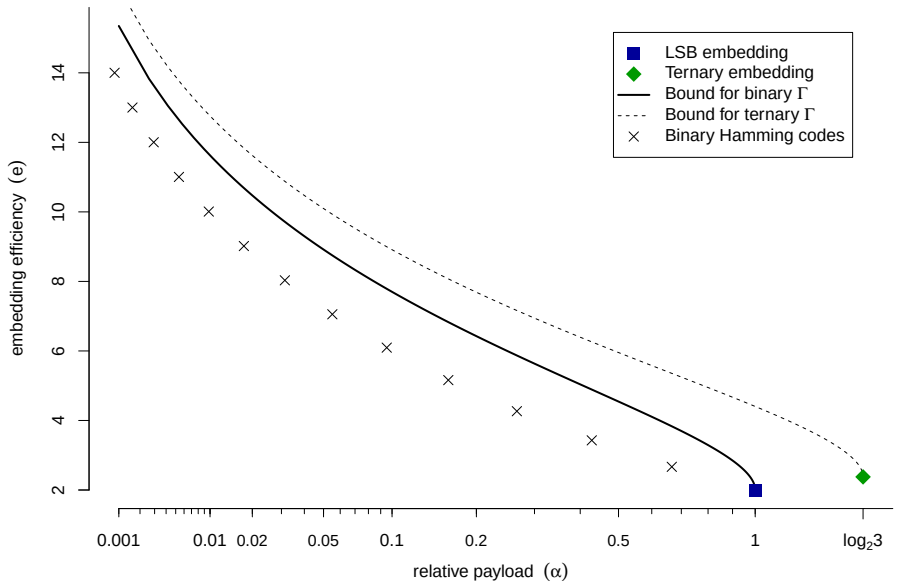


Figure 3.1: Asymptotic bounds on embedding efficiency (y-axis), as limited by the relative payload (x-axis, nonlinear scale). Bounds for binary and ternary alphabets are shown, as well as the values for the binary Hamming code family.

3.3 Non-Shared Selection Channel

Recall the problem which arose in our discussion of F5 embedding: if the embedder wants to skip all zero coefficients, but might create new zeros during the embedding, how does the receiver know which zeros were skipped and which carry payload? The list of elements in the object which carry payload is called the **selection channel** and F5 suffers from the **non-shared selection channel** problem. In general, what to do if certain part of the cover should not be used, but the recipient does not know which ones? This was a thorn in the side of early steganography schemes, but there is now an elegant solution.

We can formulate this problem using the same system as subsection 3.2.1. As well as the other functions, suppose that there is a predicate

$$\text{Wet} : \Sigma^n \to \mathbb{B}^n,$$

where $\mathbb{B} = \{\text{True}, \text{False}\}$, which informs the embedder that certain locations are "wet" and not for use[4]. We must take account of this during the coding procedure, but the "wet" locations are not known at the decoding stage:

$$\begin{aligned} \text{Decode} &: \Gamma^n \times \mathcal{K} \to \Gamma^m, \\ \text{Code} &: \Gamma^n \times \mathbb{B}^n \times \mathcal{K} \times \Gamma^m \to \Gamma^n. \end{aligned}$$

with the correctness condition $\text{Decode}(\text{Code}(\boldsymbol{b}, \boldsymbol{w}, \boldsymbol{k}, \boldsymbol{m}), \boldsymbol{k}) = \boldsymbol{m}$ and, for all i,

$$\boldsymbol{w}[i] \implies \text{Code}(\boldsymbol{b}, \boldsymbol{w}, \boldsymbol{k}, \boldsymbol{m})[i] = \boldsymbol{b}[i].$$

The stegosystem is combined in exactly the same way:

$$\begin{aligned} \text{Emb}(\boldsymbol{c}, \boldsymbol{k}, \boldsymbol{m}) &= \texttt{zipwith}\ \text{EmbSym}\ \boldsymbol{c}\ \text{Code}(\texttt{map}\ \text{ExtSym}\ \boldsymbol{c},\ \text{Wet}(\boldsymbol{c}),\ \boldsymbol{k},\ \boldsymbol{m}), \\ \text{Ext}(\boldsymbol{c}, \boldsymbol{k}) &= \text{Decode}(\texttt{map}\ \text{ExtSym}\ \boldsymbol{c},\ \boldsymbol{k}). \end{aligned}$$

The situation is very similar to the one we explored in the last section: with syndrome codes the receiver does not need to know which elements were changed, so they can solve the non-shared selection channel rather easily. In this context they are called **wet paper codes**.

[4]The terminology "wet" comes from the first steganography publication to propose this technique (Fridrich et al., 2005). In this analogy, the steganographer cannot write on wet parts of a piece of paper, but the paper dries before the recipient receives it.

Suppose that a cover has n elements (or is split into blocks of n elements). Let us once more pick a (binary, if $\Gamma = \{0, 1\}$) matrix $\boldsymbol{H}$ which is $m \times n$ and use $\text{Decode}(\boldsymbol{b'}) = \boldsymbol{H}\boldsymbol{b'}$. We are using n embedding locations to convey $m < n$ payload symbols. Suppose l of the n cover symbols are wet, and k are dry (so $n = l + k$). We want to solve

$$\boldsymbol{H}\boldsymbol{b'} = \boldsymbol{m} \tag{3.9}$$

without changing the wet elements of $\boldsymbol{b}$. This can be done by hand for small $\boldsymbol{H}$, but more systematically we can permute the columns of $\boldsymbol{H}$, and the corresponding elements of $\boldsymbol{b'}$, to put all the wet locations at the end. Then we have

$$\boldsymbol{H'} \begin{pmatrix} b_1 \\ \vdots \\ b_k \\ w_1 \\ \vdots \\ w_l \end{pmatrix} = \boldsymbol{m},$$

where $\boldsymbol{H'}$ is the column-permuted version of $\boldsymbol{H}$, $b_1, \ldots, b_k$ are the symbols in the dry cover locations, and $w_1, \ldots, w_l$ the symbols in the wet locations. Splitting $\boldsymbol{H'}$ into parts sized $m \times k$ and $m \times l$, we have

$$(\boldsymbol{H_1}\boldsymbol{H_2}) \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{w} \end{pmatrix} = \boldsymbol{m}$$

which can be rearranged to

$$\boldsymbol{H_1}\boldsymbol{b} = \boldsymbol{m} - \boldsymbol{H_2}\boldsymbol{w}. \tag{3.10}$$

Now it is simply a matter of solving the $m \times k$ linear equations. Will they have a solution? Usually not if $k < m$ (with fewer dry cover symbols than the size of the message, embedding is usually impossible). When $m \geq k$ it depends on whether the matrix $\boldsymbol{H_1}$ has full rank. For random matrices, this happens with probability $1 - O(2^{k-m})$, i.e. very likely if there are a few spare dry locations compared with the size of the payload. (You will investigate this in the exercises.) For large m and n, though, solving a system of $m \times k$ linear equations by Gaussian elimination can be very slow ($O(km^2)$) and researchers have investigated techniques for using particular types of sparsity for more efficient algorithms; one is described in (Fridrich, 2010, §9.2).

To make things more complicated, we would like to solve (3.9) with high embedding efficiency, i.e. by changing few elements of $\boldsymbol{b}$ in (3.10); it amounts to finding a coset leader (w.r.t. $\boldsymbol{H_1}$) for $\boldsymbol{m} - \boldsymbol{H_2 w} - \boldsymbol{H_1 b}$. However, we cannot hope that $\boldsymbol{H_1}$ is a usefully-structured matrix, because it is a selection of columns from $\boldsymbol{H}$ where we have no control over the selection (it is determined by the wet cover symbols). In the absence of structure, we must fall back to using sparse random codes or random codes with small codimension, and the algorithms involved are rather complex (Fridrich et al., 2006).

3.3.1 Applications

Wet paper codes are used in a number of ways. We briefly survey some applications.

First, we can solve the shrinkage problem of F5. All zero DCT coefficients are marked as wet, and the embedder uses a wet paper code (typically a keyed pseudorandom linear code with matrix $\boldsymbol{H}$ size $18 \times \lceil 18/\alpha \rceil$, which gives a good efficiency-complexity compromise) to embed the payload. The embedding operation is the plain F5 operation, and the payload symbols are the LSBs of the coefficients. The receiver uses the key to reconstruct the embedding matrix and recover the syndromes. By using a clever algorithm and some exhaustive searching, the embedding can be done with coset leaders, substantially increasing the embedding efficiency if the relative payload is small. This algorithm is called **no-shrinkage F5**, usually abbreviated **nsF5**.

A similar application would be for hiding in raw images, to avoid saturated (0 or 255) pixels. Many amateur camera images have areas of solid black or white due to over- or under-exposure, and we have seen that to embed in these areas can present obvious weaknesses. By marking all saturated areas as wet (and splitting the cover to try to distribute the wet areas fairly evenly between code blocks) they can be avoided.

A powerful embedding method becomes possible if embedding is done *at the same time as* JPEG compression. Recall that JPEG compression means quantization of DCT coefficients, so that a coefficient x becomes $q[x/q]$. Now if we embed in that quantized coefficient, it will become $q([x/q] + e)$ where e is the change, usually ± 1. For some values of x, $q([x/q] + e)$ will be closer to the original x than others (i.e. suppose quantization increases x, then an embedding change takes it back down a step, there is much less overall change than if quantization increases x and embedding increases it further). We can use wet paper codes to ensure that embedding changes which compound on quantization change are not used. This is called **perturbed quantization** (PQ), and there are many ways of applying it (to any kind of quantization, not just JPEG), which are beyond

the scope of this course. They have more power than other embedding schemes because they have some "side information", which is knowledge of the original unquantized cover. Such knowledge is not always available, particularly given that most digital cameras and phones take JPEG images almost exclusively.

Finally, we mention a particularly attractive application of wet paper codes. Suppose that we wish to get the efficiency of ternary embedding without the difficulty of transcoding a payload into ternary alphabet. Instead, we can use the following algorithm:

(i) Using an LSB embedding algorithm, perhaps along with a syndrome code, embed a binary payload of length m into a cover, remembering the locations which are changed. Suppose that m/e embedding changes were made (the embedding efficiency for this message was e).

(ii) Each of these m/e locations (apart from saturated ones) could be changed from c to $c+1$ or $c-1$ and still contain the correct LSB. So mark all unchanged or saturated locations as wet, and use a wet paper code to convey additional information in the approximately m/e changed locations by the value of their 2nd-least significant bit (i.e. element x maps to $\lfloor x/2 \rfloor \pmod 2$).

In other words, we are able to select whether to change by $+1$ or -1 to convey one additional bit of information for every change in the first stage. We have attained an embedding efficiency of $e+1$, which is approximately what one would have achieved by using a comparable ternary code to the binary code in step (i).

Other applications, and more details on these, are explained in §9.4 of (Fridrich, 2010).

3.4 Minimizing Distortion

The push to increase embedding efficiency implicitly assumes that all changes are equally detectable, from which it follows that minimizing the number of changes is the best option. This is sensible if the embedder has no idea how detectable each change might be, but if they either have information about the possible detectors used against them, or have built up a model of images which gives them information about which changes are more detectable, then they may want to take this into account when embedding. Coding that does so is called **adaptive embedding**.

The literature on this topic is still new, but it relies on some complicated ideas from information and coding theory, so we present here only the simplest model for so-called

optimal embedding, where the changes are independent. It is a different approach from those of sections 3.2 and 3.3.

We will not talk about the cover and stego objects directly. Instead, suppose that there is a given fixed cover with n locations (perhaps cover elements) which *could* be changed when the cover is turned into a stego object, and that the **distortion** associated with changing element i is $\boldsymbol{d}[i]$. We will say that distortion approximates detection risk, or something related to it, and we are supposing that the overall risk of the embedding is the average sum of the distortion of all the changes that were made. Think of $\boldsymbol{d}[i]$ as the **cost** of performing change i. We are assuming that these changes are binary (that they either happen or do not happen), and that they do not interact at all (so that one change does not affect the distortion associated with any other). No payload is carried except by the choice of embedding changes.

The question arises: how should the embedder choose their changes to minimize total distortion but maximize the payload carried? In practice, for a given size payload how do they minimize the distortion? An obvious solution is to convey $m < n$ bits of payload by making or not making (each carrying one bit) the changes associated with the lowest m values of $\boldsymbol{d}$, but this is not optimal. From a practical point of view it would not be sensible because it allows the enemy to focus their attention on a subset of the cover (we must assume that the enemy knows the distortion function), and it is incorrect from a theoretical point of view because it makes the m-th-least damaging change with the same likelihood as the least damaging change. It also denies us the embedding efficiency boost from coding, when the payload is smaller than the cover.

As usual, let us model the payload as random. We can think of an embedding algorithm as one which makes change i with probability $\boldsymbol{p}[i]$. To turn optimal embedding into a proper optimization problem, we will need to quote, in a rough-and-ready way, some results from the information theory literature:

Fact 3.1 An upper bound on the total information (in bits) which can be conveyed by potential changes, occurring independently with probabilities given by $\boldsymbol{p}$, is the total **entropy**

$$\sum_i H(\boldsymbol{p}[i])$$

where $H(-)$ is the binary entropy function (3.6).

(Equation (3.5) is the special case when all probabilities are equal.)

Fact 3.2 There exists suitable coding to allow the recipient to receive information very close to this bound, without access to the original cover.

(Random codes will approach the bound asymptotically as $n \to \infty$, for fixed m/n. That is not to say that they are efficient for the embedder.)

Given these, we want to choose the probabilities $\boldsymbol{p}$ so that the total entropy is big enough for a message of length $m < n$, but that the *average* total distortion is minimal. The optimization problem becomes

$$\text{minimize } \sum_i \boldsymbol{p}[i]\boldsymbol{d}[i] \text{ subject to } \sum_i H(\boldsymbol{p}[i]) = m. \tag{3.11}$$

Theorem 3.3 The optimal solution to (3.11) is given by

$$\boldsymbol{p}[i] = \tfrac{1}{1+2^{\lambda \boldsymbol{d}[i]}}$$

where λ is some positive constant which can be determined from $\sum_i H(\boldsymbol{p}[i]) = m$.

Proof We can solve this by use of a Lagrange multiplier. Let

$$\Lambda = \sum_i \boldsymbol{p}[i]\boldsymbol{d}[i] + \mu\Big(\sum_i H(\boldsymbol{p}[i]) - m\Big).$$

Recall that $H(x) = -x\log_2 x - (1-x)\log_2(1-x)$ so that $H'(x) = -\log_2(\frac{x}{1-x})$ (this is called the **logit function**). Then

$$\frac{\partial \Lambda}{\partial \boldsymbol{p}[i]} = \boldsymbol{d}[i] - \mu \log_2(\tfrac{\boldsymbol{p}[i]}{1-\boldsymbol{p}[i]})$$

which is zero if

$$\frac{\boldsymbol{p}[i]}{1-\boldsymbol{p}[i]} = 2^{\lambda \boldsymbol{d}[i]}$$

where $\lambda = 1/\mu$. The result follows; λ must be positive in order for the stationary point to be a minimum. ■

Note that the optimal embedding involves potentially making *any* of the changes in the cover[5] with positive probability, but the probability is lower for locations with higher distortion. Note that the probabilistic model of embedding is determined by optimality with

[5]Unless we decide to model "wet" embedding locations by assigning them a distortion of ∞.

respect to distortion, which then must be turned into a particular embedding algorithm (in this course we have worked the other way around).

So what sort of coding can take account of distortion? Actually, syndrome coding can still be used, but it is generally an NP-complete problem to find $\boldsymbol{H}\boldsymbol{e} = \boldsymbol{x}$ where the total distortion $\boldsymbol{e} \cdot \boldsymbol{d}$ is minimal. The introduction of **syndrome trellis codes** (STCs) (Filler et al., 2011) solves this problem: the parity-check matrix $\boldsymbol{H}$ is designed with the following structure:

$$\boldsymbol{H} = \begin{pmatrix} \boxed{\hat{\boldsymbol{H}}} & & & 0 \\ & \boxed{\hat{\boldsymbol{H}}} & & \\ & & \boxed{\hat{\boldsymbol{H}}} & \\ 0 & & & \ddots \end{pmatrix}$$

where $\hat{\boldsymbol{H}}$ is a small random binary matrix, typically with height around 10 and width approximately $1/\alpha$ so that the overall size of $\boldsymbol{H}$ is $m \times n$ (where m is size of payload and n size of cover). Because each payload bit depends on only a few cover elements, and using some dynamic programming tricks, solving $\boldsymbol{H}\boldsymbol{e} = \boldsymbol{x}$ with minimum total distortion is tractable. Furthermore, STCs come close to theoretical bounds called rate-distortion curves.

There are many challenges in improving these stegosystems. First, the model ought to be extended to allow embedding changes to interact. This is possible, and the general solution parallels something from theoretical physics: $\boldsymbol{p}[i]$ forms a Markov Random Field (MRF), and λ is like the inverse temperature of a physical system (Filler, 2010). MRFs are notoriously difficult to work with, and realistically the distortion function needs to have a particular form for the solution to be calculated. We then need a code that allows us to embed with the specified probabilities. Because of the difficulties, almost all adaptive embedding still uses purely additive distortion.

Most importantly, how does the embedder justify the choice of a particular distortion function? We can think of this problem as a generalization of that in section 3.1, and the same caveat applies: if a detector is looking at a different type of distortion from the one minimized by the embedder, the embedding might be very detectable. The good news is that STCs allow millions of dimensions of distortion to be measured, which is more than the number of features feasibly used by a machine-learning detector.

The first adaptive embedding using STCs was known as HUGO (Pevný et al., 2010);

it works in the spatial domain using an cost-sensitive form of LSBM. The current state-of-the-art is a family of embedding algorithms called UNIWARD (Holub et al., 2014) and they measure distortion of each possible change by its effect on a *wavelet-transformed* version of the image, because such transform captures dependencies in multiple directions. There is a spatial domain (raw image) version called S-UNIWARD, a JPEG version called J-UNIWARD, and a side-informed JPEG version that makes use of the uncompressed image from which the JPEG is derived (similarly to the PQ method that we mentioned in subsection 3.3.1) called SI-UNIWARD.

But very recent research has started exploiting the distortion measure itself, focusing steganalysis on the most-used parts of images and contrasting their features with those in less-used parts. Countermeasures to the countermeasures are developing.

Bibliography

Filler, T. (2007). Minimizing embedding impact in steganography using low density codes. Master's thesis, Czech Technical University in Prague. Available at `http://dde.binghamton.edu/filler/pdf/Tomas_Filler_master_thesis.pdf`.

Filler, T. (2010). Gibbs construction in steganography. *IEEE Transactions on Information Forensics and Security*, 5(4), 705–720. Available at `http://dde.binghamton.edu/FILLER/pdf/Fill10tifs-gibs-journal.pdf`.

Filler, T., Judas, J., & Fridrich, J. (2011). Minimizing additive distortion in steganography using syndrome-trellis codes. *IEEE Transactions on Information Forensics and Security*, 6(3), 920–935. Available at `http://dde.binghamton.edu/filler/pdf/Fill10tifs-stc.pdf`.

Fridrich, J. (2010). *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge University Press.

Fridrich, J., Goljan, M., Lisoněk, P., & Soukal, D. (2005). Writing on wet paper. *IEEE Transactions on Signal Processing*, 53(10), 3923–3935. Available at `http://ws2.binghamton.edu/fridrich/Research/WPC_TransactionsJournal1.pdf`.

Fridrich, J., Goljan, M., & Soukal, D. (2006). Wet paper codes with improved embedding efficiency. *IEEE Transactions on Information Forensics and Security*, 1(1), 102–110. Available at `http://ws2.binghamton.edu/fridrich/Research/wpc_with_improved_embedding_efficiency-ieee.pdf`.

Fridrich, J. & Soukal, D. (2006). Matrix embedding for large payloads. *IEEE Transactions on Information Forensics and Security*, 1(3), 390–395. Available at http://ws2.binghamton.edu/fridrich/Research/large_payloads-ieee_revised.pdf.

Hetzl, S. & Mutzel, P. (2005). A graph-theoretic approach to steganography. In *Proceedings of 9th International on Conference, Communications and Multimedia Security*, volume 3677 of *Lecture Notes in Computer Science* (pp. 119–128). Springer. Available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.85.3146.

Holub, V., Fridrich, J., & Denemark, T. (2014). Universal distortion function for steganography in an arbitrary domain. *EURASIP Journal on Information Security*, 2014(1). Available at http://jis.eurasipjournals.com/content/2014/1/1.

Pevný, T., Filler, T., & Bas, P. (2010). Using high-dimensional image models to perform highly undetectable steganography. In *Proceedings of 12th Information Hiding Conference*, volume 6387 of *Lecture Notes in Computer Science* (pp. 161–177). Springer. Available at http://hal.archives-ouvertes.fr/hal-00541353/PDF/HUGO-electronic_pre_proc.pdf.

Pevný, T. & Fridrich, J. (2007). Merging Markov and DCT features for multi-class JPEG steganalysis. In *Proceedings of SPIE/IS&T Electronic Imaging: Security, Steganography, and Watermarking of Multimedia Contents IX*, volume 6505 (pp. 3–14). SPIE. Available at http://dde.binghamton.edu/tomas/pdfs/Pev07-SPIE.pdf.

Provos, N. (2001). Defending against statistical steganalysis. In *Proceedings of 10th USENIX Security Symposium* (pp. 323–335). Available at http://www.citi.umich.edu/u/provos/papers/defending.ps.

Chapter 4

Theory

Reading (course text): Fridrich, §13.2, Appendix B.2

Alternatives & further depth: Böhme, §3.1, 3.2, 3.4

In the final chapter, we turn to the theory of steganography. We know, from chapter 2, that larger payloads cause more distortion to a cover, and are therefore more reliably detectable. The fundamental question is: given an acceptable level of risk for the embedding, how large a payload is too large? What is the **capacity**? An **acceptable risk** would be one where the detector is guaranteed a minimum false positive and/or false negative rate, so that the embedder does not feel that their risk of detection is too high.

We will not usually be able to answer the question directly, but we can infer some asymptotic results, particularly the **square root law** which relates secure capacity to the size of the cover. The difficulty is that we must reason about *every* possible detector, including those which have not yet been invented; we have to use some information theoretic arguments. Accordingly, we need abstract models of covers and of embedding, and the former is difficult to make realistic.

The theory of steganographic capacity is new, and there are many interesting results still to prove, and probably some surprises to uncover. It is an area of active research.

4.1 Probabilistic Models

In order to prove theorems about detectability, we need to model stochastically both the effect of embedding and the covers themselves. The former can be done with a high level of realism. The latter is bound to be difficult, otherwise image processing would be a lot easier a discipline. The state-of-art still involves abstract cover models which cannot capture completely the interdependencies in digital media. Nonetheless we still reach interesting theoretical results which hold up well when tested on real data.

4.1.1 Independent Embedding

We begin with the embedding. As in section 2.3, we assume that the payload is (uniformly) random, and we continue with the assumption of **mutually independent embedding**, which means that each cover location should be affected independently of the others[1]. With a suitable formulation we can cover a wide class of embedding operations including LSBR, LSBM, Ternary embedding, and F5.

Recall the decomposition of a stegosystem from subsection 3.2.1. At the lowest level, it is the EmbSym operation which causes changes to covers. The effect of these changes can be described by the matrix

$$\boldsymbol{Q}[j,k] = \mathrm{P}[\mathrm{EmbSym}(j,m) = k], \quad \text{when } m \text{ is chosen uniformly from } \Gamma.$$

Here j and k index elements of Σ. The assumption that the inserted symbols are uniformly random follows from the random payload assumption. For example, if the covers are bytes, then for LSBR

$$\boldsymbol{Q}[2j,2j] = \boldsymbol{Q}[2j,2j+1] = \boldsymbol{Q}[2j+1,2j] = \boldsymbol{Q}[2j+1,2j+1] = \frac{1}{2}, \text{ all other } \boldsymbol{Q}[j,k] = 0,$$

and for F5

$$\boldsymbol{Q}[j,j-1] = \boldsymbol{Q}[-j,-j+1] = \frac{1}{2} \text{ for } j > 0; \text{ all other } \boldsymbol{Q}[j,k] = 0.$$

The mutually independent embedding model is that every cover symbol is affected, independently of the value of that cover symbol and independently of whether other cover

[1] We previously noted that this is not quite correct, for fixed-length messages, but it is very close to reality.

symbols are used, with some probability γ called the **embedding rate** which is determined by the payload size[2]. In simple embedding schemes, γ will be proportional to the payload size m, for example $\gamma = m/n$ in simple LSBR and LSBM, but would be slightly superlinear (in m, if n is fixed) when improved embedding efficiency coding is used. By focusing on γ, rather than the actual payload size, we are separating improvements due to coding from the effect of steganography on the cover.

Suppose that a single cover symbol X can take values $\Sigma = \{1, \ldots, l\}$ and does so with probabilities $p_1, \ldots, p_l$. Under this embedding model, the distribution of the corresponding stego symbol Y must be

$$\begin{aligned}
\mathrm{P}[Y = j] &= \mathrm{P}[X \text{ not used for embedding } \wedge\ X = j] \\
&\quad + \sum_{i=1}^{l} \mathrm{P}[X \text{ used for embedding } \wedge\ X = i \ \wedge\ X \text{ changed to } j] \\
&= \mathrm{P}[X \text{ not used for embedding}]\,\mathrm{P}[X = j] \\
&\quad + \sum_{i=1}^{l} \mathrm{P}[X \text{ used for embedding}]\,\mathrm{P}[X = i]\,\mathrm{P}[X \text{ changed to } j] \\
&= (1 - \gamma)p_j + \gamma \sum_{i=1}^{l} p_i \boldsymbol{Q}[i, j]. \qquad (4.1)
\end{aligned}$$

4.1.2 Cover Models

Covers are more difficult to model, and we cannot hope to make a very realistic model at this stage. In this course we will use the **iid cover** model, which is simply that the cover is a random vector $\langle X_1, \ldots, X_n \rangle$, with n samples which we shall call "pixels", where each pixel is independent of every other and all pixels have the same distribution. ("iid" is the standard abbreviation for **independent identically distributed**).

Of course, pixels in real digital media are not independent: dependencies are introduced by smoothing and demosaicing (converting the output of a camera CCD, which samples

[2]In some literature the payload is alternatively represented by a **change rate** β, but we prefer to separate the effect of the EmbSym operation from the code which uses it; in our model a cover symbol can be *used* but not actually *changed*. Unfortunately, the coding still affects the embedding because the likelihood of changing a used pixel does depend on the code.

only one of red, green, or blue at each pixel, into a full-colour image) in a digital camera, by camera optics, and by the content of the picture itself. (Nonetheless, it is often the case that we can learn interesting facts or make accurate deductions from very coarse models, as computational linguistics has often found.) Therefore the literature has considered more general models of images. State of the art is work on Markov chains, where each pixel may be influenced by one neighbour (this can quickly be generalized to k-th order Markov chains where each pixel may be influenced by a fixed number of "previous" pixels); a big part of the technical difficulty is that independent embedding causes the stego object not to be a Markov chain any more. One day there will be cover models based on Markov Random Fields, which allow arbitrary dependence, and it will be interesting to see what conditions are required for the Square Root Law (coming up in section 4.4) to hold.

4.2 A Concept from Information Theory

We need one more abstract concept, this time from information theory, called **Kullback-Leibler divergence** (KL divergence or just KLD) and sometimes **relative entropy**. Think of it as a measure of distance between two distributions or two random variables. In this course we only need to define it for finite-valued random variables, which makes it a little simpler than the fully-general version you might read about in information theory literature.

Suppose that X and Y are discrete random variables taking the same finite set of values $\{x_1, \ldots, x_n\}$, where the distributions are given by

$$\mathrm{P}[X = x_i] = p_i, \quad \mathrm{P}[Y = x_i] = q_i.$$

Thus p_i and q_i define the distributions of X and Y; of course $\sum p_i = \sum q_i = 1$. Think of X as representing a cover pixel, and Y a stego pixel. We want to measure how much difference the embedding caused, by how much it affected the cover distribution. There are a number of distances between random distributions, for example **total variation**, $\frac{1}{2}\max_i |p_i - q_i|$ or **Hellinger distance** $\sqrt{\frac{1}{2}\sum_i (\sqrt{p_i} - \sqrt{q_i})^2}$, but it is KL divergence which has the right properties to make it work for us here.

The KL divergence is given by

$$D_{\mathrm{KL}}(X \parallel Y) = \sum_{i=1}^{n} p_i \log \frac{p_i}{q_i}$$

and you can see that the actual values $\{x_1, \ldots, x_n\}$ themselves are irrelevant to it[3]. For the definition to make sense, $q_i > 0$ for exactly the same i where $p_i > 0$ ($p_i = q_i = 0$ is permissible if we use the convention that $0 \log 0/0 = 0$). For our purposes it will be convenient to restrict attention to cases when none of the p_i or q_i is zero.

The first crucial property is that KL divergence is additive for independent random variables. Suppose that X and Y take values $x_1, \ldots, x_m$ with probabilities $p_1, \ldots, p_m$ and $q_1, \ldots, q_m$, and that X' and Y' take values $x'_1, \ldots, x'_n$ with probabilities $p'_1, \ldots, p'_n$ and $q'_1, \ldots, q'_n$. If X is independent of X' (i.e. $\mathrm{P}[X = x_i \wedge X' = x'_j] = p_i p'_j$) and Y is independent of Y' then

$$\begin{aligned} D_{\mathrm{KL}}\big(\langle X, X' \rangle \parallel \langle Y, Y' \rangle\big) &= \sum_{i=1}^{m} \sum_{j=1}^{n} p_i p'_j \log \frac{p_i p'_j}{q_i q'_j} \\ &= \sum_{i=1}^{m} \sum_{j=1}^{n} p_i p'_j \log \frac{p_i}{q_i} + \log \frac{p'_j}{q'_j} \\ &= \sum_{i=1}^{m} p_i \log \frac{p_i}{q_i} (\textstyle\sum_j p'_j) + \sum_{j=1}^{n} p'_j \log \frac{p'_j}{q'_j} (\textstyle\sum_i p_i) \\ &= D_{\mathrm{KL}}(X \parallel Y) + D_{\mathrm{KL}}(X' \parallel Y') \end{aligned}$$

from which it follows (by induction, or by adapting the above proof), that if $\langle X_1, \ldots, X_n \rangle$ are independent and $\langle Y_1, \ldots, Y_n \rangle$ are independent then

$$D_{\mathrm{KL}}\big(\langle X_1, \ldots, X_n \rangle \parallel \langle Y_1, \ldots, Y_n \rangle\big) = \sum_{i=1}^{n} D_{\mathrm{KL}}(X_i \parallel Y_i). \tag{4.2}$$

When objects consist of many independent samples (e.g. pixels in the iid cover model) then it is easy to calculate the KL divergence caused by embedding, by summing it over the different samples.

A note of caution: KL divergence is **not symmetric**, so in general $D_{\mathrm{KL}}(X \parallel Y) \neq D_{\mathrm{KL}}(Y \parallel X)$.

[3] We will use the natural logarithm in our definition, and all future mentions of log will assume natural base, in which case the unit of KL divergence is the **nat**; if the logarithm is to base 2 then this multiplies the value by a constant and the unit is the **bit**.

4.2.1 Example

We give an example of the calculation of KL divergence, and its interpretation.

Suppose just 2 pixels $\langle X_1, X_2\rangle$ which can only take two values $\{0,1\}$, iid covers where $\mathrm{P}[X_i = 0] = p_0 > \mathrm{P}[X_i = 1] = p_1$ (i.e. cover pixel are unequally distributed between the two values; of course $p_0 + p_1 = 1$). Suppose independent embedding where $\gamma = \frac{1}{2}$ and $\boldsymbol{Q} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$.

By (4.1), covers $\langle Y_1, Y_2\rangle$ have independent pixels where $\mathrm{P}[Y_i = 0] = q_0 = \frac{3}{4}p_0 + \frac{1}{4}p_1$ and $\mathrm{P}[Y_i = 1] = q_1 = \frac{3}{4}p_1 + \frac{1}{4}p_0$. By (4.2) and then using the definition of KL divergence we compute

$$\begin{aligned} D_{\mathrm{KL}}\big(\langle X_1, X_2\rangle \parallel \langle Y_1, Y_2\rangle\big) &= 2D_{\mathrm{KL}}(X_1 \parallel Y_1) \\ &= 2p_0 \log\Big(\tfrac{p_0}{q_0}\Big) + 2p_1 \log\Big(\tfrac{p_1}{q_1}\Big) \\ &= 2p_0 \log\Big(\tfrac{4p_0}{3p_0+p_1}\Big) + 2p_1 \log\Big(\tfrac{4p_1}{3p_1+p_0}\Big). \end{aligned} \tag{4.3}$$

Now suppose that, instead of using each pixel with probability $\frac{1}{2}$, the embedding uses exactly one pixel, randomly selected. This is not independent embedding, and computing the KL divergence is not as simple, but is instructive to compare the answer with the previous one. By taking each possible cover and stego combination, we compute

$$\begin{aligned} \mathrm{P}\big[\langle Y_1,Y_2\rangle{=}\langle 0,0\rangle\big] &= \tfrac{1}{2}\mathrm{P}\big[\langle X_1,X_2\rangle{=}\langle 0,0\rangle\big] + \tfrac{1}{4}\mathrm{P}\big[\langle X_1,X_2\rangle{=}\langle 1,0\rangle\big] + \tfrac{1}{4}\mathrm{P}\big[\langle X_1,X_2\rangle{=}\langle 0,1\rangle\big] \\ &= \tfrac{1}{2}p_0^2 + \tfrac{1}{2}p_0p_1 = \tfrac{1}{2}p_0. \\ \mathrm{P}\big[\langle Y_1,Y_2\rangle{=}\langle 0,1\rangle\big] & \\ = \mathrm{P}\big[\langle Y_1,Y_2\rangle{=}\langle 1,0\rangle\big] &= \tfrac{1}{4}\mathrm{P}\big[\langle X_1,X_2\rangle{=}\langle 0,0\rangle\big] + \tfrac{1}{4}\mathrm{P}\big[\langle X_1,X_2\rangle{=}\langle 1,0\rangle\big] + \tfrac{1}{4}\mathrm{P}\big[\langle X_1,X_2\rangle{=}\langle 0,1\rangle\big] + \tfrac{1}{4}\mathrm{P}\big[\langle X_1,X_2\rangle{=}\langle 1,1\rangle\big] \\ &= \tfrac{1}{4}p_0^2 + \tfrac{1}{2}p_0p_1 + \tfrac{1}{4}p_1^2 = \tfrac{1}{4}. \\ \mathrm{P}\big[\langle Y_1,Y_2\rangle{=}\langle 1,1\rangle\big] &= \tfrac{1}{4}\mathrm{P}\big[\langle X_1,X_2\rangle{=}\langle 1,0\rangle\big] + \tfrac{1}{4}\mathrm{P}\big[\langle X_1,X_2\rangle{=}\langle 0,1\rangle\big] + \tfrac{1}{2}\mathrm{P}\big[\langle X_1,X_2\rangle{=}\langle 1,1\rangle\big] \\ &= \tfrac{1}{2}p_0p_1 + \tfrac{1}{2}p_1^2 = \tfrac{1}{2}p_1. \end{aligned}$$

and plugging into the definition of KL divergence we have

$$\begin{aligned} D_{\mathrm{KL}}\big(\langle X_1, X_2\rangle \parallel \langle Y_1, Y_2\rangle\big) &= p_0^2 \log\left(\frac{p_0^2}{\frac{1}{2}p_0}\right) + 2p_0p_1 \log\left(\frac{p_0p_1}{\frac{1}{4}}\right) + p_1^2 \log\left(\frac{p_1^2}{\frac{1}{2}p_1}\right) \\ &= p_0^2 \log(2p_0) + 2p_0p_1 \log(4p_0p_1) + p_1^2 \log(2p_1). \end{aligned} \tag{4.4}$$

It is quite interesting to compare (4.3) and (4.4). It can be shown (we won't try to do it here) that the latter is always a bit greater than the former. What does this mean? If KL divergence measures a *distance* between cover and stego objects, and more distance equals more detectability, it tells us that embedding in exactly one pixel is, in principle, more detectable than embedding in both pixels each with probability $\frac{1}{2}$. This is because the detector has more information: they know that exactly one pixel has been used, never zero or two.

This illustrates how information about the location of payload can influence (and complicate!) the calculation of KLD.

4.3 The Data Processing Theorem

The second key property of KL divergence is that it cannot be increased by processing (applying any function to the random variables concerned), which is known as the **data processing theorem** or **data processing inequality**[4]. This is a key element of the theory of steganography, and will give the possibility of bounding the performance of *any* detector.

We only need the special case of the theorem applying to finite random variables, which can be proved by elementary methods. First, we will prove a famous mathematical inequality.

Lemma 4.1 (The Log-Sum Inequality) For $p_1, \ldots, p_n$ and $q_1, \ldots, q_n$ all > 0,

$$(\sum p_i) \log \frac{\sum p_i}{\sum q_i} \leq \sum p_i \log \frac{p_i}{q_i} \tag{4.5}$$

Proof Take the function $f : (0, \infty) \to (-\infty, \infty)$, $f(x) = x \log x$. We have $f''(x) = \frac{1}{x} > 0$, so f is convex.

Given any n positive numbers $x_1, \ldots, x_n$, consider the polygon P whose outline is given by the points $\big(x = x_i,\ y = f(x_i)\big)$. For any $\alpha_1, \ldots, \alpha_n$ such that all $\alpha_i \in (0, 1)$ and $\sum \alpha_i = 1$,

$$\left(x = \sum \alpha_i x_i,\quad y = \sum \alpha_i f(x_i) \right)$$

[4]There is another data processing inequality, which one finds in the information theory literature, for mutual information rather than KLD. It has a similar flavour.

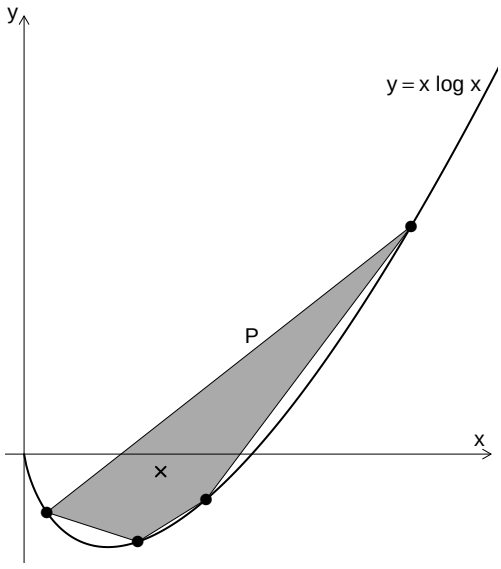


Figure 4.1: Because $y = x \log x$ is convex, every point inside the polygon P, whose vertices are $(x_i,\ x_i \log x_i)$, must be above the curve.

defines a point inside the polygon P. Because f is convex, P lies entirely *above* the curve $y = f(x)$ (see Figure 4.1)[5], so that

$$\big(\sum \alpha_i x_i\big) \log \sum \alpha_i x_i \leq \sum \alpha_i x_i \log x_i. \tag{4.6}$$

Now substitute $x_i = \frac{p_i}{q_i}$ and $\alpha_i = \frac{q_i}{\sum q_i}$ into (4.6). We get

$$\frac{\sum p_i}{\sum q_i} \log \frac{\sum p_i}{\sum q_i} \leq \frac{1}{\sum q_i} \sum p_i \log \frac{p_i}{q_i}. \tag{4.7}$$

Multiplying (4.7) by $\sum q_i$ gives (4.5). ■

(We can adapt the proof to cover cases when some p_i or q_i are zero, but there is no need for our application.)

For any distributions, $\sum p_i = \sum q_i = 1$, which substituting into (4.5) immediately tells us

Lemma 4.2 (Non-negativity of KL divergence) For any random variables X and Y,

$$D_{\mathrm{KL}}(X \parallel Y) \geq 0.$$

We are now in a position to prove the KL divergence version of the data processing theorem. If KL divergence measures the distance between two distributions, the theorem tells us that the distance cannot be increased by applying any function.

Theorem 4.3 (The Data Processing Theorem) For any random variables X and Y, and any function h,

$$D_{\mathrm{KL}}(X \parallel Y) \geq D_{\mathrm{KL}}\big(h(X) \parallel h(Y)\big). \tag{4.8}$$

Proof Let $\{x_1, \ldots, x_n\}$ be the possible values of X and Y and $\{y_1, \ldots, y_m\}$ its image

[5]Readers familiar with probability will realise that this is simply an application of Jensen's inequality.

under h. If h is not 1-1, $m < n$. Write $p_i = \mathrm{P}[X = x_i]$, $q_i = \mathrm{P}[Y = x_i]$, $P_i = \mathrm{P}[h(X) = y_i]$, and $Q_i = \mathrm{P}[h(Y) = y_i]$. By taking the preimage of y_i under h we simply have

$$P_i = \sum_{j\,:\,h(x_j)=y_i} p_j \quad \text{and} \quad Q_i = \sum_{j\,:\,h(x_j)=y_i} q_j.$$

By (4.5),

$$P_i \log \frac{P_i}{Q_i} \leq \sum_{j:h(x_j)=y_i} p_j \log \frac{p_j}{q_j}.$$

Then we simply substitute in,

$$\begin{aligned} D_{\mathrm{KL}}\big(h(X) \parallel h(Y)\big) &= \sum_{i=1}^{m} P_i \log \frac{P_i}{Q_i} \\ &\leq \sum_{i=1}^{m} \sum_{j:h(x_j)=y_i} p_j \log \frac{p_j}{q_j} \\ &= \sum_{j=1}^{n} p_j \log \frac{p_j}{q_j} \\ &= D_{\mathrm{KL}}(X \parallel Y) \end{aligned}$$

since the double sum covers all members of $\{x_1, \ldots, x_n\}$ once each. ■

4.3.1 Detector Bound

Now we can apply the data processing theorem to the problem of detection. Suppose that $\boldsymbol{X}$ is a random variable or vector which represents the distribution of covers in $\mathcal{C}$, and $\boldsymbol{Y}$ represents the distribution of stego objects in $\mathcal{C}$. The embedding algorithm and payload size used to create $\boldsymbol{Y}$ can be fixed or random, but in our applications they will typically be fixed.

Recall that a detector is a decision function $D : \mathcal{C} \to \{\text{Positive}, \text{Negative}\}$. We can use the data processing theorem to prove:

Theorem 4.4 For any detector D, with false positive rate α and false negative rate β,

$$\alpha \log\Big(\frac{\alpha}{1-\beta}\Big) + (1-\alpha) \log\Big(\frac{1-\alpha}{\beta}\Big) \leq D_{\mathrm{KL}}(\boldsymbol{X} \parallel \boldsymbol{Y}).$$

Proof In this setting, a false positive happens when $D(\boldsymbol{X}) =$ Positive and a false

negative when $D(\boldsymbol{Y}) = \text{Negative}$; therefore

$$\begin{array}{rclrcl} P[D(\boldsymbol{X}) = \text{Positive}] & = & \alpha & P[D(\boldsymbol{Y}) = \text{Positive}] & = & 1-\beta \\ P[D(\boldsymbol{X}) = \text{Negative}] & = & 1-\alpha & P[D(\boldsymbol{Y}) = \text{Negative}] & = & \beta. \end{array}$$

Now D is an example of a function like h in (4.8), so we can apply the data processing theorem to get

$$D_{\text{KL}}(D(\boldsymbol{X}) \parallel D(\boldsymbol{Y})) \leq D_{\text{KL}}(\boldsymbol{X} \parallel \boldsymbol{Y}).$$

But the left hand side just involves random variables over two values {Positive, Negative} and the KL divergence can be calculated by hand as

$$\alpha \log\Big(\frac{\alpha}{1-\beta}\Big) + (1-\alpha)\log\Big(\frac{1-\alpha}{\beta}\Big),$$

proving the result. ■

This is a powerful result, but the form of the function $\alpha \log\big(\frac{\alpha}{1-\beta}\big) + (1-\alpha)\log\big(\frac{1-\alpha}{\beta}\big)$ is difficult to work with. All we will need to know is that it is an upper bound on detector performance: for any given α it sets a minimum on β, and *vice versa*. To show this, we can use a simple inequality.

Lemma 4.5 $\alpha + \beta \geq 1 - \sqrt{\frac{1}{2}\Big(\alpha \log\big(\frac{\alpha}{1-\beta}\big) + (1-\alpha)\log\big(\frac{1-\alpha}{\beta}\big)\Big)}$.

Proof Define

$$f(x) = \alpha \log\Big(\frac{\alpha}{1-x}\Big) + (1-\alpha)\log\Big(\frac{1-\alpha}{x}\Big) - 2(\alpha + x - 1)^2.$$

Then

$$f'(x) = \tfrac{\alpha}{1-x} - \tfrac{1-\alpha}{x} - 4(\alpha + x - 1) = (\alpha + x - 1)\Big(\tfrac{1}{x(1-x)} - 4\Big).$$

Observe that the derivative is zero when $x = 1-\alpha$, negative when $x < 1-\alpha$, and positive when $x > 1-\alpha$, so f is minimum at $x = 1-\alpha$. And $f(1-\alpha) = 0$, therefore $f(\beta) \geq 0$ for all β. Rearranging gives the required result. ■

(Lemma 4.5 is a special case of **Pinsker's Inequality**.) Since $\alpha + \beta = 1$ corresponds to a detector with random output, this shows how low KL divergence guarantees that any detector must have near-random performance[6].

[6] There is a partial converse, in the sense that high KL divergence means that certain high-performance detectors exist in some cases, but in general we can only use the results of this section as an upper bound on detector performance.

4.4 The Square Root Law for IID Covers

Finally, we come to a result relating secure steganography capacity to the size of the cover. This version is for the iid cover model only, but it was an important advance in the theory of steganography. We will state the theorem here and then prove its three parts separately.

Theorem 4.6 Suppose that covers are distributed as iid pixels with a finite range of values which we (without loss of generality) call $\{1, 2, \ldots, l\}$. So the independent random variables $\langle X_1, \ldots, X_n \rangle$, where n is the size of the cover, are defined by $\mathrm{P}[X_i = j] = p_j$ for $j = 1, \ldots, l$. Suppose mutually independent embedding defined by the $l \times l$ matrix $\boldsymbol{Q}$, and embedding rate γ which depends on the cover size n.

(a) If $\sum_i p_i \boldsymbol{Q}[i, j] = p_j$, for all j, then the embedding is undetectable.

(b) If there exists j such that $\sum_i p_i \boldsymbol{Q}[i, j] \neq p_j$, then

  (i) If $\gamma^2 n \to \infty$ as $n \to \infty$ then there exists a detector which is asymptotically perfect, in the sense that α can be arbitrarily small and $\beta \to 0$.

  (ii) If $\gamma^2 n \to 0$ as $n \to \infty$ then *any* detector must be asymptotically useless, in the sense that $\alpha + \beta \to 1$.

4.4.1 Proof of (a)

From here on, let us write $q_j = \sum_i p_i \boldsymbol{Q}[i, j]$. In stego objects the probability of a pixel taking value j is, according to (4.1), $(1 - \gamma)p_j + \gamma q_j$. If $q_j = p_j$ for all j then the stego probabilities are exactly equal to the cover probabilities, and under the iid cover model with independent embedding there can be no dependencies between pixels in either cover or stego object. Therefore cover objects and stego objects have exactly the same distribution and cannot be distinguished (their KL divergence is zero; every detector is purely random).

4.4.2 Proof of (b)(i)

Pick j such that $q_j > p_j$ (at least one such j exists). An asymptotically perfect detector can be constructed by counting the occurrences of j in objects, so let $\boldsymbol{h}$ be the histogram of the observed object $\boldsymbol{x}$. Then define

$$D(\boldsymbol{x}) = \begin{cases} \text{True}, & \text{if } \boldsymbol{h}[j] > np_j + c\sqrt{n}, \\ \text{False}, & \text{if } \boldsymbol{h}[j] \leq np_j + c\sqrt{n}, \end{cases}$$

where c is a large constant to be chosen later.

There are many ways to prove that this is asymptotically perfect. Let us write $T = \boldsymbol{h}[j]$.

In cover objects, under the iid model, $T \sim \mathrm{Bi}(n, p_j)$. By (0.1), $\mathrm{Var}[T] \le \frac{n}{4}$. A false positive occurs if $T > np_j + c\sqrt{n}$. According to Chebyschev's inequality,

$$\alpha = \mathrm{P}\big[T > np_j + c\sqrt{n}\big] \le \frac{\mathrm{Var}[T]}{c^2 n} \le \frac{1}{4c^2},$$

which can be made arbitrarily small with suitably large choice of c.

Recall that, in stego objects, the probability of a pixel taking value j is $(1-\gamma)p_j + \gamma q_j = p_j + \gamma\epsilon$ where $\epsilon = q_j - p_j > 0$. Then

$$\begin{aligned}\beta = \mathrm{P}\big[T \le np_j + c\sqrt{n}\big] &= \mathrm{P}\big[T \le n(p_j + \gamma\epsilon) + (c\sqrt{n} - \gamma\epsilon n)\big] \\ &\le \frac{\mathrm{Var}[T]}{(c\sqrt{n} - \gamma\epsilon n)^2} \\ &\le \frac{1}{4(c^2 - 2c\gamma\epsilon\sqrt{n} + \gamma^2\epsilon^2 n)} \to 0,\end{aligned}$$

no matter what the value of c or ϵ, if $\gamma^2 n \to \infty$.

4.4.3 Proof of (b)(ii)

Let us compute the KL divergence between *one* cover pixel and *one* stego pixel, and bound it above. According to the definition,

$$\begin{aligned}D_{\mathrm{KL}}(X_i \parallel Y_i) &= \sum_{j=1}^{l} p_j \log \tfrac{p_j}{(1-\gamma)p_j + \gamma q_j} \\ &= \sum_{j=1}^{l} -p_j \log\Big(1 + \gamma\big(\tfrac{q_j - p_j}{p_j}\big)\Big) \\ &\overset{(a)}{\le} \sum_{j=1}^{l} -p_j\gamma\big(\tfrac{q_j - p_j}{p_j}\big) + p_j\gamma^2\big(\tfrac{q_j - p_j}{p_j}\big)^2 \\ &= \gamma\big(\textstyle\sum p_j - \sum q_j\big) + \gamma^2 \sum \frac{(q_j - p_j)^2}{p_j} \\ &\overset{(b)}{=} \gamma^2 \sum \tfrac{(q_j - p_j)^2}{p_j}\end{aligned}$$

where (a) follows from $\log(1+x) \geq x - x^2$ for x sufficiently close to zero[7], and (b) from $\sum p_j = \sum q_j = 1$, since both are distributions of random variables. Now recall that all cover pixels are independent, and all stego pixels are independent because of mutually independent embedding. Therefore by (4.2),

$$0 \leq D_{\mathrm{KL}}(\boldsymbol{X} \parallel \boldsymbol{Y}) = \sum_{i=1}^{n} D_{\mathrm{KL}}(X_i \parallel Y_i) \leq cn\gamma^2$$

for some constant c. It follows that $D_{\mathrm{KL}}(\boldsymbol{X} \parallel \boldsymbol{Y}) \to 0$ if $\gamma^2 n \to 0$, and by applying Theorem 4.4 and Lemma 4.5 that $\alpha + \beta \to 1$ for any detector.

4.4.4 Discussion

Let us interpret the meaning of Theorem 4.6. Suppose that we decide our maximum supposedly "secure" payload size m as a function of a cover size n; for example, we might embed at b bits per pixel or per nonzero DCT coefficient. Suppose for a moment that we use a simple embedding scheme such as LSBR or F5, where the embedding rate is equal to, or proportional to, m/n. Then Theorem 4.6 says that the key value is $\gamma^2 n \propto m^2/n$. If m grows asymptotically faster than $\sqrt{n}$, then for larger and larger covers the risk of detection tends to one, but when m grows asymptotically slower than $\sqrt{n}$ then, at least for sufficiently large covers, the risk of detection tends to zero. Hence the name **square root law**, saying that *the payload size is limited to grow as the square root of the cover size.*

This is a surprising result. Following signal-processing theory, it was expected that there would be a communication **rate**, hence the popularity of measures such as **bits per pixel** and **bits per nonzero coefficient**. Indeed, for perfect steganography, which is case (a) of Theorem 4.6, the entropy rate of the covers is the maximum communication rate of the channel. But for imperfect steganography, case (b) of Theorem 4.6, where the embedding does not quite manage to preserve the cover distribution, the only rate is zero because capacity is sublinear.

A survey paper (Ker et al., 2008) tested contemporary detectors against contemporary steganography schemes, measuring accuracy for payloads in cropped-down images so that the payload was constant, proportional to cover size, or to its square root. In all cases a close adherence to a square root law was observed. Figure 4.2 is an example: you can see that accuracy decreases with cover size if the payload is fixed, increases with the

[7] Note that $\gamma^2 n \to 0$ implies that $\gamma \to 0$, so our x *is* sufficiently close to zero for n sufficiently large.

cover size if the payload is proportional, and remains roughly constant when the payload scales with the square root of the cover size. The conclusion of (KER et al., 2008) is that the square root law is quite robust in practice. There are some complications, though, in *creating* covers of different sizes: one cannot simply resize, because the resizing introduces sampling artefacts which affect detectability, and when cropping JPEGs care must be taken to crop a "representative" region rather than picking, for example, a particularly flat area like sky.

Note that γ does *not* have to be strictly proportional to the payload size m, if sophisticated coding is used. We saw in chapter 3 that as the payload size becomes smaller the use of better (more efficient) codes becomes possible. As the embedding efficiency rises, so γ decreases. This means that m can grow slightly faster than $\sqrt{n}$, if such coding is used. The exercises explore the net effect on the capacity law, when using a code like the Hamming codes.

Finally, the square root law seems to lack a vital piece of information. If (given fixed coding) m scales with $\sqrt{n}$, what is the constant? For a given level of detectability, and a given cover source, can we state the asymptotic relationship $m \sim c\sqrt{n}$? For artificial cover sources the answer is yes, and it takes us into some old statistical theory about Kullback Leibler divergence and **Fisher information**. But to determine Fisher information from real covers, such as a source of images, is much more difficult and this is a topic of current research.

4.5 Extensions

The simplest square root law, in section 4.4, has been extended in many ways:

1. The combination of statistical attacks and key exhaustion attacks was examined in (KER, 2009). It was shown that the number of keys must scale with the payload size.

2. This was resolved in (KER, 2010a), where the use of coding was formally included in the theorem. This turned out to be rather difficult, because one must assume that the enemy is aware of the code being used, and therefore gets a small amount of information about the likely locations of changes. It was shown that Hamming codes introduce no asymptotically-significant weakness.

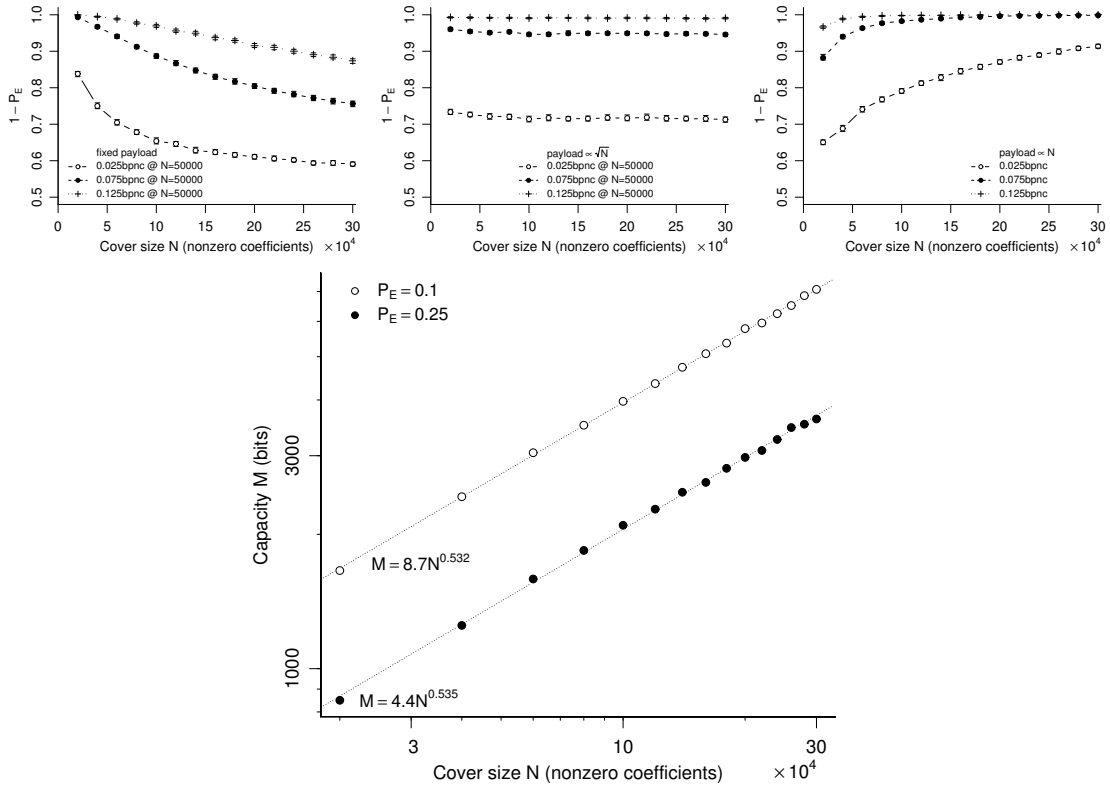


Figure 4.2: Taken from (Ker et al., 2008). Above, shows detector accuracy (y-axis, defined as the accuracy at the point of the ROC curve to minimize sum of false positive and false negative) as cover size varies. When the payload is of fixed length, it becomes less detectable in larger covers; when the payload length is proportional to the cover size, it becomes more detectable in larger covers; when the payload length scales with the square root of the cover size, detectability is roughly constant. Below, measuring the smallest payload which can be detected with given accuracy, as the cover size varies. We observe a close accordance with a square root law.

In each case the embedding algorithm was nsF5, and the detector was a Support Vector Machine using 274 features.

3. The cover model was widened to include dependency between pixels, with a Markov chain model (Filler et al., 2009). Bounding the KL divergence is a significant analytical hurdle.

4. A very interesting situation arises when, in the context of Theorem 4.6, the detector does not know the exact distribution of the cover source, but instead learns it by seeing examples of genuine covers (Ker, 2010b). This mimics the most usual real-world situation of training a machine learning classifier. It turns out that the detector requires (up to a constant multiple) as many genuine covers to learn from as the suspect objects they test, if they want to keep the embedder down to a square root law. This means that detectors cannot rest in learning a cover model, and they must constantly re-train in order to refine their information about covers.

Similar theorems by Fridrich investigate how steganographic capacity scales when covers are quantized and resampled. There is also a version of the Square Root Law for communicating by radio, in noisy channels (where the signals are continuous). Our understanding of asymptotic capacity laws is widening.

4.6 Conclusions

The theory of steganography is still new. Even apart from the difficulty of modelling covers faithfully, it is hard to reason about cases where the detector has less than full knowledge of the cover source, or more than zero knowledge about the (likely) location of embedding changes. KL divergence is a difficult beast to control, and even for the simple Markov chain model the necessary analysis can become fearsome.

Theory also has to rise to the challenges of steganalysis of multiple actors. Clearly any capacity law has to include the number of innocent actors, whose data can blind the detector in false positives. One still expects proportionate capacity to decline as the cover size increases, but to increase as the number of actors increases.

But already the theory has taught us a few important lessons about steganography: capacity is not linear, so the covert communication channel gets thinner over time (how to manage this is a topic not well researched), and the detector should continue its training forever.

Bibliography

Filler, T., Ker, A. D., & Fridrich, J. (2009). The square root law of steganographic capacity for Markov covers. In *Proceedings of SPIE/IS&T Electronic Imaging: Media Forensics and Security XI*, volume 7254 of *Proc. SPIE* (pp. 0801–0811). SPIE. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK36B.pdf`.

Ker, A. D. (2009). The Square Root Law requires a linear key. In *Proceedings of 11th ACM Workshop on Multimedia and Security* (pp. 85–92). ACM. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK40B.pdf`.

Ker, A. D. (2010a). The Square Root Law does not require a linear key. In *Proceedings of 12th ACM Workshop on Multimedia and Security* (pp. 213–223). ACM. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK43B.pdf`.

Ker, A. D. (2010b). The Square Root Law in stegosystems with imperfect information. In *Proceedings of 12th Information Hiding Conference*, volume 6387 of *Lecture Notes in Computer Science* (pp. 145–160). Springer. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK42B.pdf`.

Ker, A. D., Pevný, T., Kodovský, J., & Fridrich, J. (2008). The square root law of steganographic capacity. In *Proceedings of 10th ACM Workshop on Multimedia and Security* (pp. 107–116). ACM. Available at `http://www.cs.ox.ac.uk/andrew.ker/docs/ADK32B.pdf`.

Index