Detecting Low Embedding Rates*

Andreas Westfeld

Institute for System Architecture, Technische Universität Dresden
01062 Dresden, Germany
westfeld@inf.tu-dresden.de

Abstract. This paper shows three approaches for detecting steganograms with low change density. MP3Stego is a steganographic algorithm with a very low embedding rate. The attack presented here is a statistical analysis of block sizes. It is able to detect 0.001 % of steganographic payload in MP3 files. The second approach is the use of hash functions to combine sample categories for the chi-square attack. One of these hash functions enables us to detect about 0.2 bits per pixel in true colour images. Another algorithm (Hide) was presented at the last workshop and constructed to be secure against visual and statistical chi-square attacks. The detection method for Hide combines the three colour components of each pixel to recognise an increased number of "neighbour colours".

1 Introduction

Steganographic tools change bits in a carrier medium to embed a secret message. Whether these changes are noticeable to an attacker or not, depends on many different things. The embedding function must keep certain properties the attacker knows about carrier media. If an attacker has a better model for the carrier media, the person who implements the tool cannot be sure about the security of the algorithm.

There are two kinds of attacks: On the one hand there are attacks that prove the use of a steganographic tool without error, e. g. specially produced palettes that occur only with S-Tools resp. Mandelsteg and so on [5]. On the other hand, most statistical attacks have a probability of error larger than 0. If we embed less and spread the changes over the carrier medium we decrease the change density. The lower the change density the higher the probability of error.

A lower change density decreases the probability of detection, although this decreases the steganographic capacity as well. As we will see, the question is not how much data is embedded, but how much the carrier is changed. Sect. 2 gives an example of a tool with only limited steganographic payload (less than 0.1 %), and with surprisingly strong changes per embedded bit—although imperceptible by human ears. Maybe its low capacity kept away potential attackers. (Some years ago, I looked at this tool through the glasses of one specific vulnerability

* This work is supported by the German Federal Ministry of Economics and Technology (BMWi).

F.A.P. Petitcolas (Ed.): IH 2002, LNCS 2578, pp. 324–339, 2003.

that many simple tools have. But this attack did not match the embedding algorithm of MP3Stego.)

The main issue of Sect. 3 is the definition of categories for the chi-square attack. Building a direct histogram of samples will lead to a significant statement only if there are at least 97 % of the samples steganographically used. This is the case if the message was continuously embedded, or if we know the embedding places. It is necessary to guarantee one embedded bit per observed value.

Finally, Sect. 4 explains an attack on Hide, a steganographic tool presented by Sharp [11] at the last workshop. Hide uses an algorithm secure against statistical chi-square attacks [12]. It does not simply overwrite the least significant bits. Nevertheless, it is detectable.

2 MP3Stego

MP3Stego is a modified version of the 8HZ-mp3 [1] encoder. It reads Windows WAV files (RIFF-WAVE-MSPCM) and encodes them as MPEG Audio Layer-3. WAV files from audio CDs typically contain digital audio signals that consist of 16 bit samples recorded at a sampling rate of 44.1 kHz. So we end up with 2×705.6 kbits/s in these WAV files. Using the command

```
encode example.wav example.mp3
```

these sound data are reduced by a factor of 11. The resulting MPEG Layer-3 stream in the MP3 file still maintains the same sound quality with only 128 kbits/s. This is realised by perceptual coding techniques addressing the perception of sound waves by the human ear. Compared with other audio coding schemes, MP3 files achieve the highest sound quality for a given bit rate. Because of this, the MP3 file format is very popular and it is a great idea to use it for steganography. With MP3Stego [6] we can embed a file (e. g. hidden.txt) in the Layer-3 stream while encoding a WAV file. In a first step, the tool compresses the file to hide using zlib [9]. A passphrase (e. g. abc123) is used to encrypt the compressed message with triple-DES and to dilute the changes pseudo-randomly:

```
encode -E hidden.txt -P abc123 example.wav example.mp3
```

The heart of a Layer-3 encoder is a system of two nested iteration loops for quantisation and coding. The inner iteration loop (cf. Fig. 1) finds the optimal quantisation parameter (q_factor). If the number of bits resulting from the quantisation (block_length) exceeds the number of bits available to code a given block of data (max_length), this can be corrected by adjusting the global gain to result in a larger q_factor. The operation is repeated with increasing q_factor until the resulting block is smaller than max_length.

Without embedding, the iteration will end as soon as the block_length is not larger than the specified max_length. The parameter hidden_bit is 2 if a block should bypass the steganographic processing after finding this optimal size.

```
int inner_loop(int max_length, int *q_factor, int hidden_bit)
{
       int block_length;
       *q_factor -= 1;
       /* increase q_factor until block_length <= max_length */
       do {
              *q_factor += 1;
              block_length = quantize();
              switch (hidden_bit) {
              case 2:                       /* nothing to embed */
                     embed_rule = 0;
                     break;
              case 0:                       /* embed 0 */
              case 1:                       /* embed 1 */
                     embed_rule = (block_length % 2) != hidden_bit;
                     break;
              }
       } while ((block_length>max_length) || embed_rule);
       return block_length;
}
```

Fig. 1. The modified inner iteration loop of the Layer-3 encoder (simplified)

2.1 Embedding Algorithm

In case hidden_bit is 0 or 1, the inner iteration loop will continue until a q_factor is found that produces an even or odd block_length respectively. The final block_length is not larger than the specified max_length. (In rare cases this is an endless loop if the block_length is already 0 and hidden_bit is 1.) We should take into consideration that incrementing the q_factor by 1 does not automatically flip the least significant bit (LSB) of the block_length. In most cases the block_length will decrease by a value larger than one. So if we want to embed a hidden bit, the LSB of the block_length could remain the same for several iterations. The per track maximum of such unsuccessful series is 12. . . 18 (consecutive) iterations on an average CD.

Although the quality of some frames is artificially decreased by messages embedded with MP3Stego, you probably need golden ears to notice that. Without the original music file it is difficult to distinguish between background noise and steganographic changes.

2.2 Detection by Block Length Analysis

The length of steganographically changed blocks is smaller than one quantisation step size below the upper bound max_length, i. e. smaller than necessary for the requested bit rate. If max_length were fixed, an MP3 file bearing a steganographic message would have a lower bit rate than a clean one. Then we could

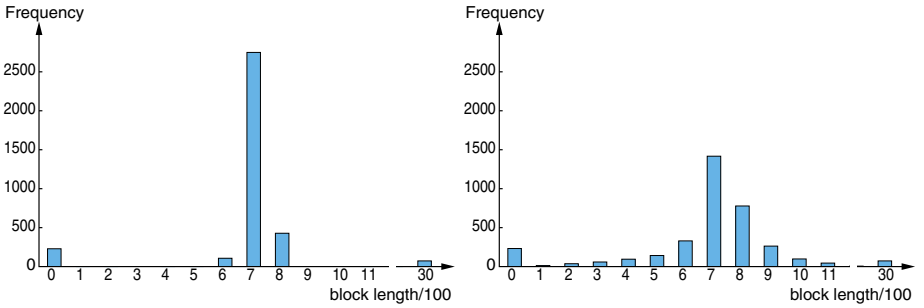


Fig. 2. Histogram of block length without steganography (left) and with the maximum of embedded data (right)

just calculate the bit rate (or the mean value of the block lengths) to detect steganographic changes. Unfortunately, max_length is adjusted from frame to frame by the rate control process to bring the bit rate of the blocks in line with the requested average (default 128 kbps). Every time the block length is steganographically decreased, the following blocks are larger to equalise the bit rate. At the end, the steganographic MP3 file and a clean version from the same WAV file have equal size.

Although the mean value is the same, the variance is increased. The histograms in Fig. 2 show that there is a peak at 7 (i. e. blocks with 700–799 bits), and two accumulations at 0 and 30 (0–99/3000–3099). There are some seconds of quietness between tracks. Each frame of digital silence contains one block of 3056 bits and three zero-length blocks. So the first and last accumulation in the histogram is caused by the pause at the end of the track. For detection of steganography we will only consider block lengths between 100 and 3000 bits, so that we get a unimodal distribution with an expected block length of 764 bits.

To calculate the variance s^2 we need the count of considered block lengths n, the sum of block lengths $\sum x$, and the sum of their squares $\sum x^2$:

$$s^2 = \frac{\sum x^2 - \frac{1}{n}(\sum x)^2}{n-1}$$

As mentioned earlier, the max_length is adjusted from block to block to get the requested average bit rate. The initial value of max_length is 764, which is the ideal block length for 128 kbits/s. Since max_length is only the upper limit for the blocks, the first frames of the MP3 file are shorter than the average. After getting too large, the value of max_length swings in to 802 bits (or some more in case something is embedded). This oscillation of max_length causes a stronger variance of the block length at the start of the MP3 file. Hence, the variance depends also on the file length.

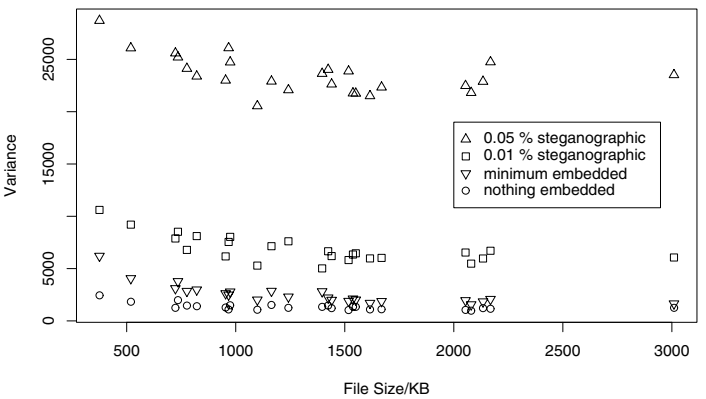
Figure 3a shows the result of four test series with 25 tracks from a mixed CD:

1. one without a message,
2. one with an empty message,
3. one with 0.01 % steganographic contents, and
4. one with 0.05 % relative to the length of the MP3 file.

All messages were pseudo-random. MP3Stego can embed 0 bytes as the shortest message. However, this does not mean that the MP3 file remains unchanged. Because every message is compressed using zlib to eliminate the redundancy before it is embedded, there are effectively more than 0 bits to embed.

If we compress a file with 0 bytes using zlib we get 24 bytes. MP3Stego embeds these together with 4 extra bytes to store the length of the message. The resulting 28 bytes are about 2 % of the maximum capacity in a 3 MB MP3 file, or 0.001 % of the carrier file size. Even this low embedding rate is visually

a)



b)

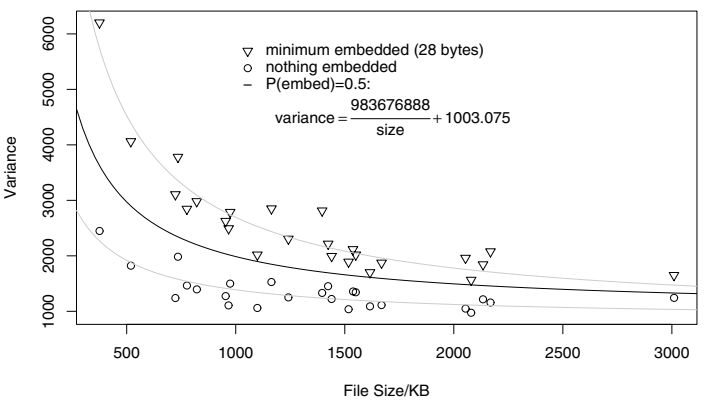


Fig. 3. The variance of the block length depends on the file size and payload

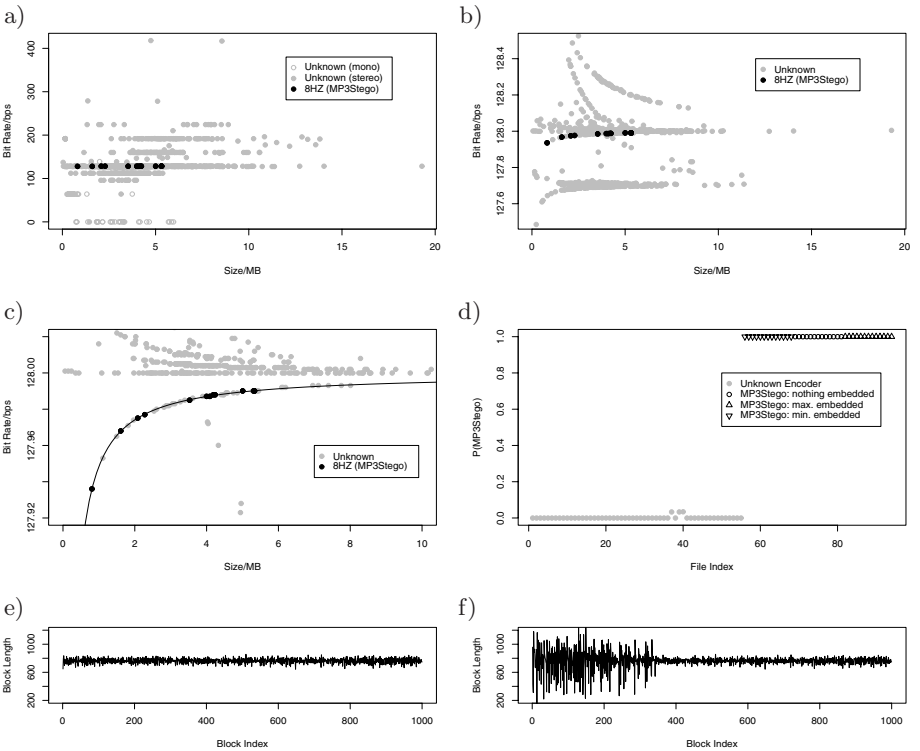


Fig. 4. a) Other encoders have different bit rates, or **b)** a characteristic rate control process, or **c)** produce the same bit rate like MP3Stego but are **d)** otherwise distinguishable by quadratic discriminance analysis. **e)** low variance in a clean MP3 file **f)** strong oscillation caused by 224 embedded bits (zlib-compressed zero-length file)

different in Fig. 3b. A curve of type "$A/\text{size} + B$" separates all cases correctly (a posteriori).

Now, the attack already has a good selectivity, especially if we restrict it to the first part of MP3 files, say 500 KB. However, for an attack under "real world circumstances" we have to recognise that a file was created using an 8HZ compatible application ([1], [6]) and not one of the many other implementations of MP3 encoders with their own characteristics of variance.

2.3 How To Distinguish Encoders

Figure 4a shows 1308 MP3 files of unknown origin classified by their bit rate and size. Most of these files are on the stroke at 128 kbits/s, together with the MP3Stego files (black bullets). Each black bullet actually stands for three MP3Stego files: The clean version is at the same position in the diagrams as

the versions with minimum and maximum payload. If we zoom to bit rates between 127.5 and 128.5 kbits/s (Fig. 4b) we discover that it is not just one stroke but many different curves. Probably every encoder has its own characteristic rate control process. The curve in Fig. 4c is the interpolated characteristic of MP3Stego. There is only a small subset (55 of 1308) with questionable files that could come from MP3Stego. But lets move from the macroscopic properties bit rate and file size to individual block lengths. The following autoregressive model explains one block size by its two predecessors.

$$\text{block}_i = \beta_0 + \beta_1 \cdot \text{block}_{i-1} + \beta_2 \cdot \text{block}_{i-2}$$

It is still possible to distinguish the questionable subset of unknown origin from files encoded with MP3Stego regardless whether there is something embedded. We apply the autoregressive model to the individual block lengths of a questionable file first. A quadratic discriminance analysis (QDA) with the coefficients β_0, β_1, and β_2 can tell us whether it matches the MP3Stego rate control process or not (Fig. 4d).

2.4 Estimating the Size of Embedded Text

In addition, we can use a plot of consecutive block lengths (Fig. 4e and f) to estimate the size of the embedded message. Although the steganographic changes are not dense—only up to 60 % of the blocks are used—the message bits are not uniformly spreaded over the whole MP3 file but randomly diluted with the ratio 3 : 2 (3 used, 2 skipped). We can use the following formula to estimate the length of the embedded message in bits:

$$\text{message length/bits} \approx 0.6 \cdot \text{last dirty block index}$$

3 Chi-square Attack Despite Straddling

The statistical chi-square attack [12] reliably discovers the existence of embedded messages that are embedded with tools which simply replace least significant bits (LSBs). However, if the embedded message is straddled over the carrier medium, and if less than 97 % of the carrier medium is used, a direct histogram of sample values will not lead to a satisfactory result. So we have to know either the embedding sequence (which we probably do not without a secret key), or change the categories of samples to guarantee one embedded bit per observed value.

After modifying these categories, the attack gives significant results even if only one third of the steganographic capacity is used. It can even detect a difference between clean and steganographic images with only 5 to 10 % of the capacity used.

There have been other attempts to generalise the chi-square attack to allow the detection of messages that are randomly scattered in the cover media. The most notable is the work of Provos and Honeyman [8], and Provos [7]. Instead of increasing the sample size and applying the test at a constant position, they

Table 1. The p-value (in %) depends on the part of the capacity that is used

	Exploitation of the steganographic capacity (%)									
`hash`	100	95	94	50	33	25	16	10	5	0
a_1 (w/o hash)	100	68.8	1.85	—	—	—	—	—	—	—
$a_1 + a_2$	100	99.9	99.8	99.5	38	4.5	0.6	—	—	—
$a_1 \oplus a_2$	100	100	100	99.9	1.0	0.1	—	—	—	—
$a_1 \oplus 3a_2$	100	100	100	2.3	—	—	—	—	—	—
$a_1 + a_2 + a_3$	100	100	100	100	100	100	100	100	100	100
$a_1 \oplus a_2 \oplus a_3$	100	100	100	100	100	100	100	100	100	100
$a_1 \oplus 3a_2 \oplus 5a_3$	100	100	100	100	90.6	91.7	66.1	37.7	12.6	2.5
$a_1 + 3a_2 + 5a_3$	100	100	100	99.9	76.1	33.9	7.4	1.1	—	—

use a constant sample size but slide the position where the samples are taken over the entire range of the image. Using the extended test they are able also to detect messages that are not continuously embedded but spread in the carrier. However, the resulting p-value of the chi-square test is not significant, i. e. most of the time it is jumping in the range between 0.05 and 0.95.

Here, we unify several observed values to one with the legitimate hope to get one steganographically used value on average. For example, if somebody uses 50 % of the steganographic capacity, only every second observed value is used for the secret message. In this case we have to combine two observed values to one, so that we can expect one steganographic bit in the combined sample. With the use of 33 % we have to combine three observed values for the same expectation.

3.1 Experiments

The experiments in Table 1 show that the resulting p-value all depends on *how* we unify the categories, i. e. which operation we use to combine the values. The table contains the results for 10 versions of a true colour image with different steganographic message sizes. 100 % exploitation of the steganographic capacity means that the steganographic algorithm replaced every LSB in all pixels with pseudorandom message bits (3 bits per pixel). 95 % means that the algorithm used only a subset of the LSBs and skipped 5 % of them. The column with 0 % exploitation contains the results for the carrier medium (without any embedded message). The 9 steganograms were created using S-Tools, although it does not matter which tool overwrites the LSBs in true colour images.

The a_i denote periodic sample values. The first line in Table 1 lists the results for the direct samples. Then there are three experiments for hashing each two consecutive samples, and four experiments for hashing each three samples. Let $b_1, b_2, \ldots, b_n$ be the n bytes of the image content, i. e. the observed values. The different hash functions combine the sample values as follows:

$$
\begin{aligned}
\boldsymbol{a_1}&: b_1,\ b_2,\ b_3,\ \ldots\\
\boldsymbol{a_1 + a_2}&: b_1 + b_2,\ b_3 + b_4,\ b_5 + b_6,\ \ldots\\
\boldsymbol{a_1 \oplus a_2}&: b_1 \oplus b_2,\ b_3 \oplus b_4,\ b_5 \oplus b_6,\ \ldots\\
\boldsymbol{a_1 \oplus 3a_2}&: b_1 \oplus 3b_2,\ b_3 \oplus 3b_4,\ b_5 \oplus 3b_6,\ \ldots\\
\boldsymbol{a_1 + a_2 + a_3}&: b_1 + b_2 + b_3,\ b_4 + b_5 + b_6,\ b_7 + b_8 + b_9,\ \ldots\\
\boldsymbol{a_1 \oplus a_2 \oplus a_3}&: b_1 \oplus b_2 \oplus b_3,\ b_4 \oplus b_5 \oplus b_6,\ b_7 \oplus b_8 \oplus b_9,\ \ldots\\
\boldsymbol{a_1 \oplus 3a_2 \oplus 5a_3}&: b_1 \oplus 3b_2 \oplus 5b_3,\ b_4 \oplus 3b_5 \oplus 5b_6,\ b_7 \oplus 3b_8 \oplus 5b_9,\ \ldots\\
\boldsymbol{a_1 + 3a_2 + 5a_3}&: b_1 + 3b_2 + 5b_3,\ b_4 + 3b_5 + 5b_6,\ b_7 + 3b_8 + 5b_9,\ \ldots
\end{aligned}
$$

3.2 Conclusions

It turns out that the hash values $a_1 + a_2 + a_3$ and $a_1 \oplus a_2 \oplus a_3$ do not distinguish anything. This wants to remind us of the "power of parity": Anderson and Petitcolas suggested not to embed each bit in a single pixel, but in a set of them, and embed the ciphertext bit as their parity [2]. If a bit of a_i is "1" with probability 0.6, then the probability that the same bit of $a_1 \oplus a_2$ will be 1 is 0.48; if we move to $a_1 \oplus a_2 \oplus a_3$, it is 1 with probability 0.504, and so on. The more observed values we combine, the more equalised our histogram will be. However, the chi-square attack works because the histogram of observed values in cover media is not equalised, but pairs in steganograms are. That's probably also the reason why all experiments to hash four values were not successful.

We can deduce the following rules from the experiments:

1. The combination of observed values should not increase the number of categories too much. Otherwise they are underpopulated. Example: If the hash function simply concatenates the observed values (e. g., $256 \cdot a_1 + a_2$), we increase the number of categories by a factor of 256 (and divide the mean population by 256). The minimum theoretically expected frequency in a category of the chi-square test must be at least 5. This would require a population of 1280 or more for a_1 and a_2.
2. The unification should keep a lot of the entropy from the single values. A lossless unification means to keep all the bits, e. g., by concatenation. But a simple concatenation (cf. a), where we xor only the LSBs ($s = x \oplus y$), increases the number of categories and contradicts the first rule. So we need to reduce the information of the higher bits using a hash function (cf. b). The best hash function found is a linear combination with small odd factors:
   - The factors have to be **different** to equalise the entropy of the bits in the single values. Example: If bit 6 of the sample values has more information than bit 7, we lose less information if we combine bit 6 with bit 7, instead bit 6 with bit 6.
   - The factors have to be **small** to keep the number of categories small.
   - They have to be **odd** to project the sum of the LSBs into the LSB.

This distinguishes best between "low embedding rate" and "nothing embedded."

a) $A \mid x$ "$\cup_\circ$" $B \mid y \rightarrow A \mid B \mid s$

b) $A \mid x$ "$\cup_\oplus$" $B \mid y \rightarrow \text{hash}(A, B) \mid s$

3. It has a favourable effect if the observed values are locally close to each other. Because the colour and brightness of close pixels correlates stronger than that of more distant, less entropy is destroyed by combining them: Our hash function selects limited information from several values. If we consider one value, another value in the neighbourhood adds less information than one more distant. If we can only keep a limited amount of information, we discard less when we have less before. In true colour images it is better to combine the red and the green component of one pixel, rather than two red (or green) components of neighbouring pixels. An explanation of this might be, that they correlate stronger, because two colour components of one pixel have a local distance of 0.

The example in Fig. 5 illustrates the conversion of the most suitable variant $a_1 + 3a_2 + 5a_3$ into code that hashes all three colour components of a pixel for the histogram.

```
int histogram=new int[256];
for (int line=0; line<biHeight; line++) {
    for (int i=0; i<biWidth; i++) {
        int hash=0;
        for (int k=1; k<=3; k++)
            hash+=(2*k-1)*imageFile.readNextColourComponent();
        histogram[hash%256]++;
    }
}
```

Fig. 5. Hashing three colour components to one observed value for the histogram

4 Hide

Hide [11], a tool presented at the last workshop, uses an algorithm which is secure against the chi-square attack [12]. It does not simply overwrite the least significant bits (LSBs). Instead, the whole sample value is incremented or decremented (randomly selected) if the LSB does not match already. The LSB finally equals the next bit of the data to hide, pairs of values are not equalised, and the chi-square attack does not work.

Hide is also secure against the visual attacks [12], because

1. we may use JPEG files or an image taken with a digital camera that internally stores the photos using DCT based compression. However, the resulting steganogram may be saved only in the lossless formats BMP or PNG.
2. The algorithm is adaptive and excludes saturated samples from steganographic use. This way the significant parts of the visual correlation remains in the LSBs.

Fridrich et al. [3] presented a detection method for steganographic changes in images that were originally stored in the JPEG format. As a consequence of their steganalysis, they strongly recommend avoiding the use of images that have been stored in the JPEG format as carrier for spatial-domain steganography.

First, let us have a look at the key-based pseudo-random sequence generator, which is used to form the sample sequence. According to the specification [10], Hide uniformly modifies the signal over its length. Although the source code of Hide is not publicly available, this is easily checked using a pathologic carrier signal, e. g., a true colour bitmap where all pixels have 50 % luminance (grey.bmp). It is important to have no saturated colour components (0 or 255) in this test image, because Hide excludes them from embedding. After embedding into grey.bmp it looked still grey. Standard image manipulation programs (like Photoshop, Gimp, ...) have a function to maximise the contrast of an image. With one click on "AutoContrast" everybody can turn the steganographically changed grey into visibly changed pixels (cf. Fig. 6, right). With the amplified spots, it is obvious that the random number sequence is not balanced. This sequence is the only cryptographic measure taken in Hide.

Let us now analyse the embedding function. It adjusts the LSB by incrementing or decrementing the sample value, or leaves it as it is. In case the LSB has to be changed, the random generator determines whether to increment or decrement. Because all saturated samples are excluded, there is no overflow from 255

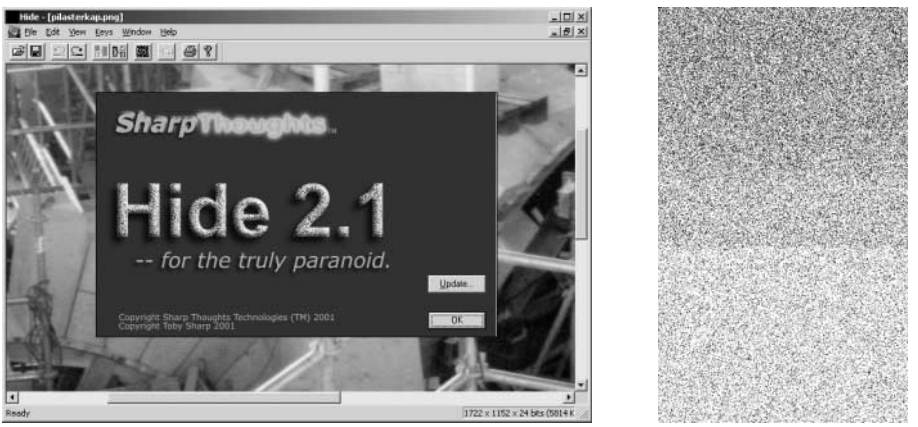


Fig. 6. Hide's user interface (left) and its unbalanced straddling (right)

$(r-1,\ g-1,\ b-1)$	$(r,\ g-1,\ b-1)$	$(r+1,\ g-1,\ b-1)$
$(r-1,\ g-1,\ b)$	$(r,\ g-1,\ b)$	$(r+1,\ g-1,\ b)$
$(r-1,\ g-1,\ b+1)$	$(r,\ g-1,\ b+1)$	$(r+1,\ g-1,\ b+1)$
$(r-1,\ g,\ b-1)$	$(r,\ g,\ b-1)$	$(r+1,\ g,\ b-1)$
$(r-1,\ g,\ b)$		$(r+1,\ g,\ b)$
$(r-1,\ g,\ b+1)$	$(r,\ g,\ b+1)$	$(r+1,\ g,\ b+1)$
$(r-1,\ g+1,\ b-1)$	$(r,\ g+1,\ b-1)$	$(r+1,\ g+1,\ b-1)$
$(r-1,\ g+1,\ b)$	$(r,\ g+1,\ b)$	$(r+1,\ g+1,\ b)$
$(r-1,\ g+1,\ b+1)$	$(r,\ g+1,\ b+1)$	$(r+1,\ g+1,\ b+1)$

Fig. 7. Hide produces up to 26 neighbours for each colour (r, g, b)

to 0 or vice versa. Most likely all the 256 possible 8-bit sample values are present in a carrier image. As described in [10], the effect of Hide on the histogram of the sample values is identical to filtering it with the low-pass filter $\{0.25, 0.5, 0.25\}$. Without access to the cover signal, it is difficult to recognise this filtering effect, especially when less data is embedded.

However, a true colour image does not contain all possible colours. This would require 16.7M pixels in the very unlikely case that all pixels have different colours, or even more pixels. Only a subset of the 2^{24} possible colours is used. On the other hand, the difference to the next colour will most likely be more than just ± 1 in all three components, although many pixels in an image have the same colour. In other words, Hide will generate a lot of new colours that are very close to the origin.

Fridrich et al. [4] considered colour pairs to detect LSB encoding in colour images. Two colours (r_1, g_1, b_1) and (r_2, g_2, b_2) are a *close colour pair*, if $|r_1 - r_2| \leq 1$, $|g_1 - g_2| \leq 1$, and $|b_1 - b_2| \leq 1$. If u is the number of unique colours, the number of all colour pairs is $\binom{u}{2}$. The detection algorithm consists of three steps:

1. Calculate the ratio between the number of close colour pairs and the number of all colour pairs.
2. Embed a test message in the LSBs of randomly selected pixels.
3. Calculate the ratio again for the new image.

Now, if the two ratios are almost the same, most probably the image already had a large message hidden inside.

The method to detect Hide does not embed any test message. Instead of colour pairs, it considers the count of neighbour colours. LSB encoding produces up to 7 neighbour colours for every colour in the image. Because Hide does not simply overwrite the LSBs, it produces up to 26 neighbour colours for each steganographically used one (cf. Fig. 7). A colour in a carrier medium has only 4 or 5 neighbours on average. Especially if we prevent visual attacks using JPEG files or images from a digital camera with DCT based compression, no colour will have more than 9 such neighbours. The histograms in Fig. 8 are created using a small C program. To speed up things, the program performs three steps:

1. insert all (non-saturated) colours of the image into a sorted tree,
2. walk through the colour tree and count the 0...26 neighbours (cf. Fig. 7) of each colour,
3. insert the colours according to their neighbour count into the histogram.

Without the tree, the program would have to consider all the non-existing colours too. It is clear that the pathologic carrier medium grey.bmp has only one colour (grey), and there are no other colours. So the histogram (cf. Table 2) contains only one colour without any neighbours. After embedding 56 KB (grey56k.bmp), this colour has 26 neighbours, and serves itself as their neighbour. The 26 newly generated colours split into 8 colours with 7 neighbours, 12 with 11 neighbours, and 6 with 17 neighbours.

After some small modifications this test worked also for greyscale images, where three consecutive grey values are processed like the three colour components of one pixel before. However, the distinction is very poor (cf. Fig. 9).

Table 2 contains 6 photos of varying resolution, scanned from paper or taken with digital cameras. Each photo comes with 4 versions: as carrier medium, with a one byte message embedded, with 100 bytes, and with 40 kilobytes embedded. Obviously, Hide stores some extra data. This helps us to detect even the shortest possible message (1 byte).

This detection method works reliably, if the carrier image is read in from a JPEG file (which is recommended by the author of Hide). If the carrier is

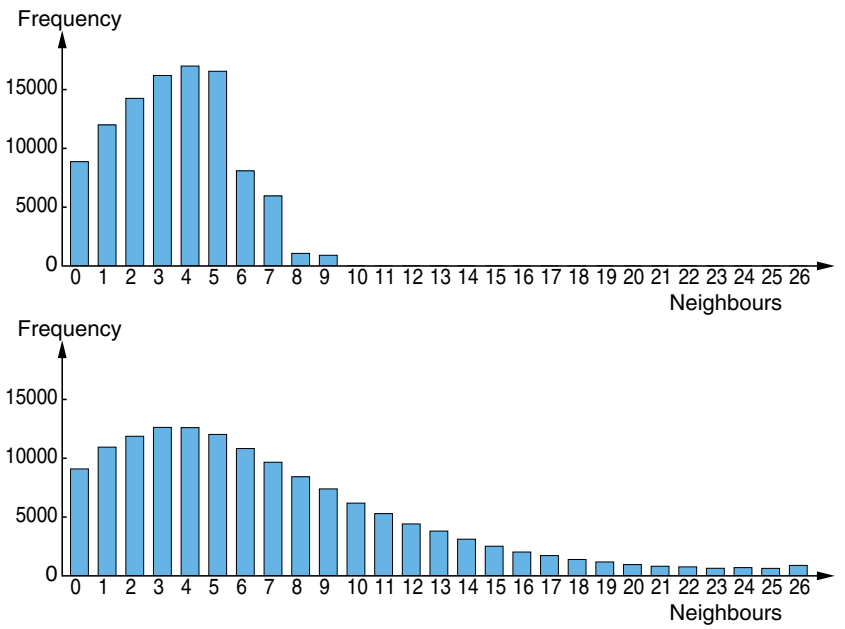


Fig. 8. Neighbourhood histogram of a carrier medium (top) and steganogram with 40 KB embedded (bottom)

Table 2. Hide's effect on the neighbourhood histogram

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
grey.bmp	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
grey56k.bmp	0	0	0	0	0	0	0	8	0	0	0	12	0	0	0	0	6	0	0	0	0	0	0	0	0	0	1
bicycle.bmp	5616	7073	8012	8780	11437	20318	9677	10594	1241	2125	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
with 1 byte	5616	7073	8011	8775	11430	20096	9864	10503	1467	2079	66	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
with 100 bytes	5615	7066	8013	8759	11361	19587	10264	10472	1881	2053	196	34	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0
with 40 KB	5658	6534	6858	6656	6896	6935	6959	7120	6981	6846	6570	6486	6204	5896	5465	5211	4783	4445	4263	4086	3998	3821	3684	3299	3002	2445	1858
dixieland.bmp	8870	11996	14254	16197	17002	16557	8090	5957	1072	911	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
with 1 byte	8869	11989	14256	16187	16999	16486	8136	5967	1143	907	27	7	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0
with 100 bytes	8872	11983	14237	16150	16965	16334	8306	5969	1331	929	75	12	12	10	2	0	1	1	1	0	0	0	0	0	0	0	0
with 40 KB	9093	10945	11869	12624	12605	12026	10827	9660	8425	7394	6185	5293	4416	3807	3119	2519	2026	1722	1391	1183	957	817	765	645	700	637	889
evening.bmp	3503	3727	4223	5061	7435	15994	7386	8460	941	1874	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
with 1 byte	3503	3726	4224	5058	7415	15816	7495	8459	1061	1903	38	6	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0
with 100 bytes	3503	3726	4221	5054	7360	15394	7746	8356	1455	1875	189	52	30	25	19	8	4	2	0	0	0	0	0	0	0	0	0
with 40 KB	3422	3533	3719	3943	4340	4799	5234	5459	5317	5282	5071	5013	4845	4382	4045	3691	3176	2863	2610	2329	2137	1897	1756	1657	1553	1710	2273
freiberg.bmp	7089	7148	8019	8358	8821	8603	4343	3126	596	411	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
with 1 byte	7088	7148	8002	8349	8803	8555	4422	3131	657	430	15	3	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0
with 100 bytes	7082	7141	7982	8348	8712	8444	4604	3129	859	465	62	14	11	3	1	0	0	0	0	0	0	0	0	0	0	0	0
with 40 KB	6982	6410	6038	6127	6128	5746	5347	5102	4764	4328	3951	3633	3508	3291	3161	2960	2598	2383	2186	1858	1665	1492	1222	1012	807	664	500
hippo.bmp	3631	4175	4455	4944	5264	5366	2839	1933	377	320	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
with 1 byte	3631	4168	4459	4926	5236	5316	2877	1971	438	338	19	6	4	3	0	0	0	0	0	0	0	0	0	0	0	0	0
with 100 bytes	3632	4167	4419	4931	5170	5117	3066	2059	545	377	56	33	17	13	13	3	6	0	0	0	0	0	0	0	0	0	0
with 40 KB	3665	3751	3456	3298	3149	3054	3099	2934	2729	2553	2307	2262	2037	2038	1917	1742	1608	1564	1462	1550	1468	1471	1444	1309	1143	966	826
kitchen.bmp	8317	9306	8669	8411	8536	13902	6411	7364	935	1343	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
with 1 byte	8317	9299	8669	8414	8514	13741	6561	7329	1083	1331	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
with 100 bytes	8311	9298	8654	8377	8470	13310	6864	7240	1572	1329	187	37	6	1	0	0	0	0	0	0	0	0	0	0	0	0	0
with 40 KB	8254	8569	7246	5959	5433	5027	5004	4875	4802	4759	4772	4805	4693	4705	4619	4599	4447	4158	3851	3703	3565	3366	3157	2993	2905	2724	2228

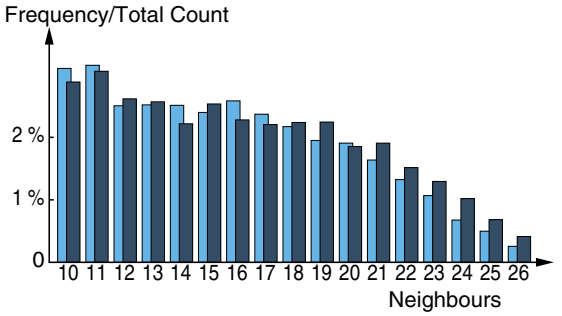


Fig. 9. Neighbourhood histogram of a greyscale carrier image (light) and steganogram with 5 KB (70 % of capacity) embedded (dark)

a high quality scan in BMP format the attack works less reliably, and only very large messages are detected. With BMP carriers, the visual attack [12] might be useful, because the saturated areas in BMPs and JPEGs have a different shape.

5 Summary and Further Work

A lower change density leads us to less noticeable changes. If an embedding function uses only a (pseudo-)randomly selected subset of changeable places in cover media, and if there are no traces that may serve as a "fragile watermark" (e. g. JPEG compatibility [3]), the steganographic changes are less detectable than without dilution. As we have seen, it depends also on their strength, whether diluted changes are detectable or not. Steganographic algorithms with low embedding rates should not naturally believed to be more secure than others.

The attack on MP3Stego depends also on the variance of the block lengths that other encoders leave in an MP3 file. To answer the question, whether files coded with other encoders cause false alarms or not, a collection of 1308 MP3 files of unknown origin was used. It was possible to distinguish all these files from files encoded with an 8HZ-based [1] decoder like MP3Stego. Many of these files had a bit rate that MP3Stego cannot produce. But the deciding point for separating different classes of MP3 encoders is the way they control the bit rate and equalise variations.

The variance measured in the MP3 files is influenced by the bit rate. The examples svega.mp3 and svega_stego.mp3 that come with the original software [6] have twice the variance because the bit rate of 128 kbits/s is used for only one channel (mono).

The success of the chi-square attack depends on how we define the categories of the histogram. It is necessary to guarantee one embedded bit per observed value. The empirically derived hash function allows the detection of one third of about 0.2 bits per pixel in true colour images.

The attack on Hide might be improved, when we determine the frequency of the 26 neighbours. Presently, the detection algorithm considers only the existence

of neighbours. But to judge the exploitation of the steganographic capacity it is necessary to consider also the ratio between the frequency of neighbour colours.

Acknowledgements

I thank Rainer Böhme for the collection of MP3 files and his introduction to the R language and environment.

References

[1] 8Hz Productions: MPEG Audio Layer III Encoder. http://www.8hz.com/mp3 325, 329, 338
[2] Ross J. Anderson, Fabien A. P. Petitcolas: On the Limits of Steganography. IEEE Journal on Selected Areas in Communications, Vol. 16, No. 4, May 1998. 332
[3] Jessica Fridrich, Miroslav Goljan, Rui Du: Steganalysis Based on JPEG Compatibility, in Multimedia Systems and Applications IV, Andrew G. Tescher, Bhaskaran Vasudev, V. Michael Bove, Jr. (Eds.), Proceedings of SPIE, Multimedia Systems and Applications IV, Denver, CO, 2001. 334, 338
[4] J. Fridrich, Rui Du, Meng Long: Steganalysis of LSB Encoding in Color Images, in Proc. ICME 2000, New York City, NY, 2000. 335
[5] Neil F. Johnson, Sushil Jajodia: Steganalysis of Images Created Using Current Steganography Software, in David Aucsmith (Ed.): Information Hiding. Second International Workshop, LNCS 1525, Springer-Verlag Berlin Heidelberg 1998. pp. 273–289. 324
[6] Fabien A. P. Petitcolas: MP3Stego 1.1.16, 1998/2002. http://www.cl.cam.ac.uk/~fapp2/steganography/mp3stego 325, 329, 338
[7] Niels Provos: Defending Against Statistical Steganalysis, 10th USENIX Security Symposium. Washington, DC, August 2001. 330
[8] Niels Provos, Peter Honeyman: Detecting Steganographic Content on the Internet, ISOC NDSS'02, San Diego, CA, February 2002. 330
[9] Greg Roelofs: Zlib Home Site. http://www.zlib.org 325
[10] Toby Sharp: An Implementation of Key-Based Digital Signal Steganography, in Ira S. Moskowitz (Ed.): Information Hiding. 4th International Workshop, LNCS 2137, Springer-Verlag Berlin Heidelberg 2001. pp. 13–26. 334, 335
[11] Toby Sharp: Hide 2.1. 2001. http://www.sharpthoughts.org 325, 333
[12] Andreas Westfeld, Andreas Pfitzmann: Attacks on Steganographic Systems, in Andreas Pfitzmann (Ed.): Information Hiding. Third International Workshop, LNCS 1768, Springer-Verlag Berlin Heidelberg 2000. pp. 61–76. 325, 330, 333, 334, 338