Error-correcting Codes in Steganography

Jörg Keller Johannes Magauer

FernUniversität in Hagen
FB Informatik — LG Parallelität und VLSI
58084 Hagen, Germany
joerg.keller@fernuni-hagen.de

Abstract: We investigate image steganography under re-encoding of images. After a secret message or a digital watermark is embedded into a jpeg image, quite frequently the image is re-encoded with a different quality factor. We model this re-encoding as a noisy communication channel with respect to the embedded message, and derive the corresponding error model. We present an error-correcting code that allows complete reconstruction of the embedded message in most cases, and partial reconstruction in all experiments performed, over a wide range of differences in quality factors.

1 Introduction

In steganography, a secret message is embedded into another, innocent looking message, in order not only to conceal the secret messsage's content, but to conceal the sheer existence of the secret message. The secret message may also be a hidden watermark that is used to invisibly tag messages with a unique mark, in order to detect illegal copying. Typically, a text forming a bit string is embedded into an image file. Today's image coding formats often use lossy compression in the frequency domain. When such an image is re-encoded, a part of the embedded bit string may be lost. Re-encoding occurs quite frequently, e.g. using a lower quality factor than in the first encoding to reduce the size of the image. We investigate the use of error-correcting codes to be able to reconstruct the embedded bit string (at least partially) after the re-encoding. In order to do so, we model the re-encoding as a noisy communication channel over which the embedded message is sent. We derive the error model from observations of the changes in the bit string seen in experiments. The types of errors are quite unusual in that not only bits are changed, but also that bits can frequently be deleted and inserted. We present an error-correcting code suitable for the error model and validate its performance.

While there exists quite an amount of literature on image steganography, to our knowledge the re-encoding of images with embedded messages has not been dealt with. For secret messages, the research concentrates on new embeddings that cannot be detected by known statistical measures such as histogram tests, and on new methods to detect the existence of known embeddings. For digital watermarks, the research concentrates on new watermarks that are robust against operations on images such as rotation or scaling, and are undetectable or at least not removable. Yet, as re-encoding happens again and again, we feel that it deserves an investigation as well.

The remainder of the paper is organized as follows. Section 2 gives a brief introduction to steganography on jpeg images. In Section 3 we derive the error model, present an appropriate error-correcting code, and report on experiments. Section 4 gives a conclusion.

2 Image Steganography

To embed a bit string s into an image file, one can either choose the pixel domain or the frequency domain of the image. In the pixel domain, one uses the fact that slight changes to some of the pixel values[1] are normally not to be detected. Hence, one encrypts and randomizes the bit string by a bitwise exclusive or with a key string[2], and then embeds it by performing a bitwise exclusive or with the lowermost bit of selected pixel values. Only a small percentage of the pixels can be chosen (less than 5%), because otherwise the lowermost bits of the pixels would look much more random than in an ordinary image file, and the existence of an embedded message would be revealed (so-called LSB-Test). While this restricts the capacity of an image with respect to embedded messages' size, it has no other consequences for the methods to be described. For more on steganography, see e.g. [KP99].

One of todays wide-spread image formats is JPG, or more correctly the Joint Photographic Experts Group (JPEG) File Interchange Format (JFIF) [Mia99]. We refrain from giving a full description here and only sketch the most common use. The image is split into fragments of 8×8 pixels. Then each color in each fragment is transformed by a discrete cosine transform (DCT). On the resulting values, a quantization is performed, and a run-length and huffman encoding compacts the image. The loss of information during the coding results from the quantization, where the 64 frequency values are divided by entries in a list, and rounded to the next integer. The larger the entries in the list, the smaller are the quantized values (and the more zeroes result) and therefore the higher the compression factor. The size of the entries is controlled by a so-called quality factor, which is a percentage. Quality 100% means least compression, and fewest loss of information. If one wants to embed a bit string in the frequency domain, this is done after the quantization, by changing rounded frequency values. Normally, only values different from zero are used for the embedding. A popular steganographic algorithm is F5 [Wes01]. We base our investigation on this algorithm as it is well analyzed.

[1] For monochrome images, the pixel value is a single numerical value, representing shades from black to white. For color images, typically three values for red, green, and blue are used. Using 8 bits for each value is common practice.

[2] A bit string of the same length as the string s, which is generated by a stream generator from a secret key value.

3 Error Model and Error-Correcting Code

We restrict ourselves to using images in JPG format, as they are most common. First, we took 100 existing images given in pixel format, and encoded them into JPG format with quality 100%. Then we decoded the images again, and controlled how many pixels differed. On average, 36% of the pixels differed. For a quality of 70%, which is a common value, even 84% of the pixels differed. Hence, we completely abandoned the idea of embedding in the pixel domain, as the error rates seem too high for an error-correcting code to have any chance of success.

Then we took the 100 images from the previous test, in JPG format with quality 100%. We converted them to a pixel format, and then coded them again as JPG with quality 100%. We found that on average, less than 10% of the quantized values had changed. When the original image had a lower quality factor, the fraction of differing quantized values was even lower. Hence, we decided to proceed with embedding in the frequency range. Our first assumption of the error model was the symmetric binary channel. Therefore, we used first a $(12, 8)$-Hamming code and later a cyclic $(15, 7)$-code to embed a bit string of length 1416 (177 byte) into 10 JPG images, randomly chosen among the above images. Then we loaded the images into an image viewer (Irfan View) and re-encoded them by storing them again. Yet, the message could not be reconstructed from any of the re-encoded images. Therefore, we tested whether the assumption on the error model was correct. We found that the changes to the frequency values due to the embedding in general lead to a different second encoding, so that a number of quantized values that were zero in the first encoding now have a non-zero value, and vice versa. Looking at the embedded message, this results in a number of bits being changed, and moreover in a number of bits of the message that have been deleted, while some bits have been inserted into the message. Hence, our error model must be that of a bit-slip-channel [Fur81, Chap. 6.5].

When designing our code, we used the observation that if up to x bits can be deleted or inserted, then it is sufficient to code a 1-bit as $2x + 1$ 1s, and to code a 0-bit as $2x + 1$ 0s to ensure correct decoding[3]. If a deletion or insertion can happen in two successive, equal code words, then it is sufficient to separate the code words by x bits which differ from the code words, and to code the code words by $3x + 1$ bits. For example, if $x = 2$, then 00 would be encoded as 0000000110000000. While such a code reduces the capacity by a factor of 9 for $x = 2$, it was possible to decode all embedded messages in all test images. In order to avoid detection of the embedded message by the LSB-Test, we use a permutation on the bits before embedding them into the image. The permutation forms part of the key.

[3] There are several other, more complex codes ensuring correct decoding, which we found out by systematic generation of all possible codes up to a length 9.

4 Conclusion

We developed a code that allows to embed a message into a JPG image, and decode the message successfully after re-encoding of the JPG. This is achieved by using an error-correcting code based on the bit-slip channel, which we found experimentally to model the re-encoding situation quite well.

We were even able to refine our embedding to allow re-encoding with lower quality than the original image, and still be able to decode the embedded message. This however further reduced the capacity, and only worked if the quality factor in the first encoding was chosen above 70%.

References

[Fur81] F. J. Furrer. *Fehlerkorrigierende Block-Codierung für die Datenübertragung*. Birkhäuser, 1981.

[KP99] S. Katzenbeisser and F. Petitcolas. *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House, 1999.

[Mia99] John Miano. *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*. Addison-Wesley, 1999.

[Wes01] Andreas Westfeld. F5 – a Steganographic Algorithm: High Capacity Despite Better Steanalysis. In *Proc. 4th Int.l Workshop on Information Hiding*, pages 289–302, 2001.