F5—A Steganographic Algorithm
High Capacity Despite Better Steganalysis

Andreas Westfeld

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

Abstract. Many steganographic systems are weak against visual and statistical attacks. Systems without these weaknesses offer only a relatively small capacity for steganographic messages. The newly developed algorithm F5 withstands visual and statistical attacks, yet it still offers a large steganographic capacity. F5 implements matrix encoding to improve the efficiency of embedding. Thus it reduces the number of necessary changes. F5 employs permutative straddling to uniformly spread out the changes over the whole steganogram.

1 Introduction

Secure steganographic algorithms hide confidential messages within other, more extensive data (carrier media). An attacker should not be able to find out, that something is embedded in the steganogram (i. e., a steganographically modified carrier medium) [8].[1]

Visual attacks on steganographic systems are based on essential information in the carrier medium that steganographic algorithms overwrite [5]. Adaptive techniques (that bring the embedding rate in line with the carrier content) prevent visual attacks, however, they also reduce the proportion of steganographic information in a carrier medium. Lossy compressed carrier media (JPEG, MP3, ...) are originally adaptive and immune against visual (and auditory respectively) attacks.

The steganographic tool Jsteg [4] embeds messages in lossy compressed JPEG files. It has a high capacity—e. g., 12 % of the steganogram's size—and, it is immune against visual attacks. However, a statistical attack discovers changes made by Jsteg [5].

MP3Stego [3] and IVS-Stego [6] also withstand auditory and visual attacks respectively. Appart from this, the extremely low embedding rate prevents all known statistical attacks. These two steganographic tools offer only a relatively small capacity for steganographic messages (less than 1 % of the steganogram's size).

[1] The steganographic techniques considered here are not intended for robust watermarking.

I. S. Moskowitz (Ed.): IH 2001, LNCS 2137, pp. 289–302, 2001.

2 JPEG File Interchange Format

The file format defined by the Joint Photographic Experts Group (JPEG) stores image data in lossy compressed form as quantised frequency coefficients. Fig. 1 shows the compressing steps performed. First, the JPEG compressor cuts the uncompressed bitmap image into parts of 8 by 8 pixels. The discrete cosine transformation (DCT) transfers 8×8 brightness values into 8×8 frequency coefficients (real numbers). After DCT, the quantisation suitably rounds the frequency coefficients to integers in the range $-2048 \ldots 2047$ (lossy step). The histogram in Fig. 2 shows the discrete distribution of the coefficient's frequency of occurrence.

If we look at the distribution in Fig. 2, we can recognise two characteristic properties:

1. The coefficient's frequency of occurrence decreases with increasing absolute value.
2. The decrease of the coefficient's frequency of occurrence decreases with increasing absolute value, i. e. the difference between two bars of the histogram in the middle is larger than on the margin.

We will see in Sect. 3 that these properties do not survive the Jsteg embedding process.

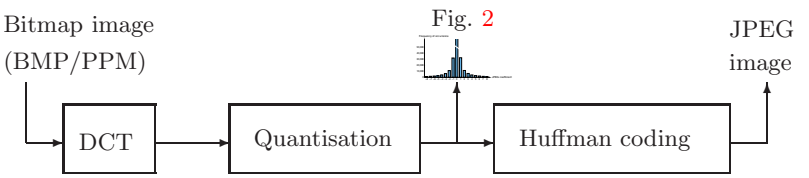

Fig. 1. The flow of information in the JPEG compressor

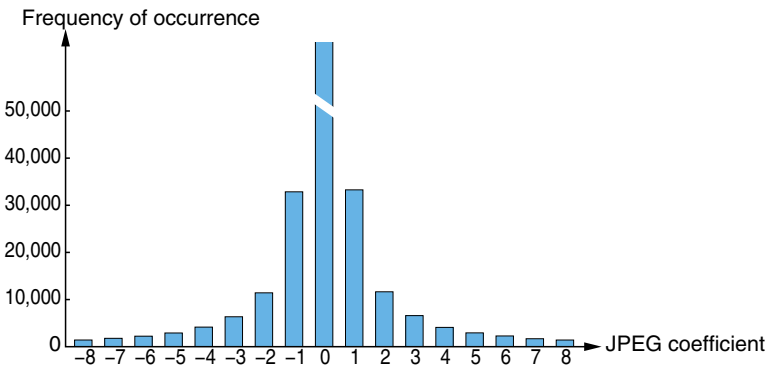

Fig. 2. Histogram for JPEG coefficients after quantisation

Fig. 3. Carrier medium (World Exhibition in Hanover 2000)

After the lossy quantisation, the Huffman coding ensures the redundancy-free coding of the quantised coefficients. Reference [2] contains a more detailed description of the JPEG compression. The following sections mainly refer to the distribution in Fig. 2. Statements of file sizes and steganographic capacities relate to the true colour image *Expo* shown in Fig. 3.

3 Jsteg

This algorithm made by Derek Upham serves as a starting point for the contemplation here, because it is resistant against the visual attacks presented in [5], and nevertheless offers an admirable capacity for steganographic messages (e. g., 12.8 % of the steganogram's size). After quantisation, Jsteg replaces the least significant bits (LSB) of the frequency coefficients by the secret message.[2] The embedding mechanism skips all coefficients with the values 0 or 1. Fig. 4 shows Derek Upham's embedding function of Jsteg in C source code.

However, the statistical attack [5] on Jsteg reliably discovers the existence of embedded messages, because Jsteg replaces bits and, thus, it introduces a dependency between the value's frequency of occurrence, that only differ in this bit position (here: LSB). Jsteg influences pairs of the coefficient's frequency of occurrence, as Fig. 5 shows. Let c_i be the histogram of JPEG coefficients. The assumption for a modified image is that adjacent frequencies c_{2i} and c_{2i+1} are similar. We take the arithmetic mean

$$n_i^* = \frac{c_{2i} + c_{2i+1}}{2} \tag{1}$$

to determine the expected distribution and compare against the observed distribution

$$n_i = c_{2i}. \tag{2}$$

[2] Let us assume a uniformly distributed message. That not only simplifies the presentation, furthermore it is plausible if the message is compressed and encrypted.

```
short use_inject = 1;          /* set to 0 at end of message */

short inject(short inval)     /* inval is a JPEG coefficient */
{
    short inbit;
    if ((inval & 1) != inval)       /* don't embed in 0 or 1 */
        if (use_inject) {    /* still message bits to embed? */
            if ((inbit=bitgetbit()) != -1) { /* get next bit */
                inval &=~1;        /* overwrite the lsb ... */
                inval |= inbit;         /* ... with this bit */
            } else
                use_inject = 0;     /* full message embedded */
        }
    return inval;        /* return modified JPEG coefficient */
}
```

Fig. 4. Derek Upham's embedding function Jsteg (comments added)

The difference between the distributions n_i and n_i^* is given as

$$\chi^2 = \sum_{i=1}^{k} \frac{(n_i - n_i^*)^2}{n_i^*} \tag{3}$$

with $k-1$ degrees of freedom, which is the number of different categories in the histogram minus one.

Fig. 6 shows the statistical attack on a Jsteg steganogram (with 50 % of the capacity used, i. e. 7680 bytes). The diagram presents the probability of embedding

$$p = 1 - \frac{1}{2^{\frac{k-1}{2}}\Gamma\left(\frac{k-1}{2}\right)} \int_0^{\chi^2} \mathrm{e}^{-\frac{t}{2}} t^{\frac{k-1}{2}-1} \mathrm{d}t \tag{4}$$

as a function of an increasing sample: Initially, the sample comprises the first 1 % of the JPEG coefficients, then the first 2 %, 3 %, ... The probability is 1.00 up to 54 % and 0.45 at 56 %; A sample of 59 % and more contains enough unchanged coefficients to let the p-value drop to 0.00.

4 F3

The algorithm F3 serves as a tutorial example. It differs in double respects from Jsteg:

1. Instead of overwriting bits, it decrements the coefficient's absolute values in case their LSB does not match—except coefficients with the value zero, where we can not decrement the absolute value. Hence, we do not use zero coefficients steganographically. The LSB of nonzero coefficients match the

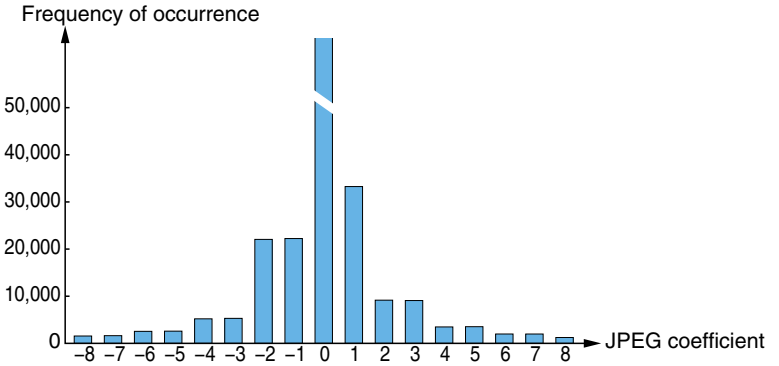

Fig. 5. Jsteg equalises pairs of coefficients

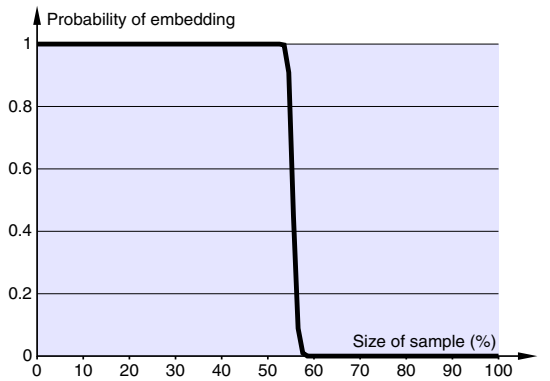

Fig. 6. Probability of embedding in a Jsteg steganogram (50 % of capacity used)

secret message after embedding, but we did not *overwrite* bits, because the Chi-square test can easily detect such changes [5]. So we can hope that no steps will occur in the distribution. In contrast to Jsteg, F3 uses coefficients with the value 1. The symmetry of 1 and −1 visible in Fig. 2 consequently remains.

2. Some embedded bits fall victim to shrinkage. Shrinkage accrues every time F3 decrements the absolute value of 1 and −1 producing a 0. The receiver cannot distinguish a zero coefficient, that is steganographically unused, from a 0 produced by shrinkage. It skips all zero coefficients. Therefore, the sender repeatedly embeds the affected bit since he notices when he produces a zero.

In comparison to Fig. 2, the histogram shows a relative surplus of even coefficients. This phenomenon results from the repeated embedding after shrinkage. Shrinkage occurs only if we embed a zero bit. The repetition of these zero bits shifts the (originally equalised) ratio of steganographic values in favour of the

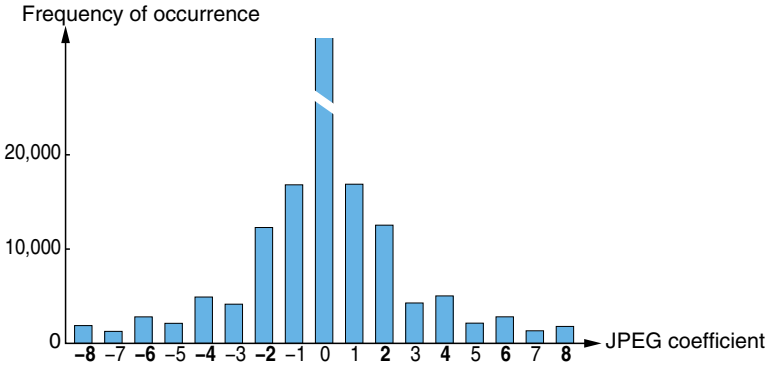

Fig. 7. F3 produces a superior number of even coefficients

steganographic zeroes. Hence, the F3 embedding process produces more even coefficients than odd. The steganographic interpretation of coefficients with the values 1 or −1 is 1 (because their LSB is 1). For this reason the embedding function keeps them unchanged when it embeds a 1. Fig. 7 shows the flashy frequency of occurrence for even and odd coefficients, which we can detect by statistical means.

If we simply ignore the shrinkage, the superior number of even coefficients disappears. Unfortunately the receiver gets only fragments of the message in this case. The application of an error-correcting code could possibly solve the problem.

If we extract putative messages from unchanged carrier media with F3, these messages will have a distribution with more ones than zeroes. Therefore, if we embed more ones than zeroes (in a suitable ratio), the superior number in the histogram disappears as well. A more elegant solution of this problem (F4) makes use of the symmetry in Fig. 2.

5 F4

F3 has two weaknesses:

1. Because of the exclusive shrinkage of steganographic zeroes, F3 effectively embeds more zeroes than ones, and produces—as well as Jsteg, but in a different way—statistically detectable peculiarities in the histogram.
2. The histogram of JPEG files (Fig. 2) contains more odd than even coefficients (excluding 0). Therefore, *unchanged* carrier media contain (from Jsteg's or F3's perspective) more steganographic ones than zeroes.

The algorithm F4 eliminates these two weaknesses in one stroke by mapping negative coefficients to the inverted steganographic value: even negative coefficients represent a steganographic one, odd negative a zero; even positive represent a zero (as before with Jsteg and F3), and odd positive a one. In Fig. 8 each

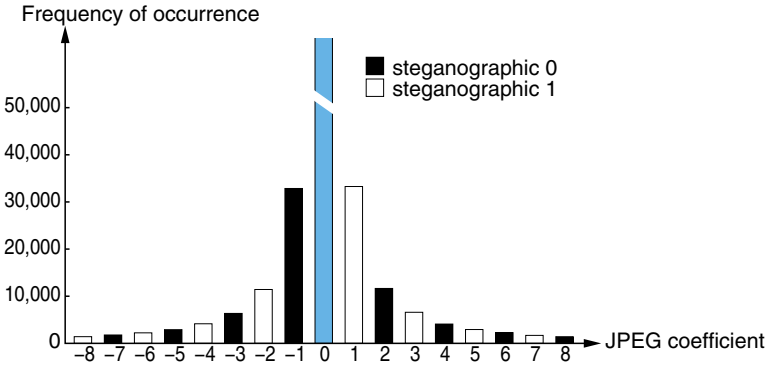

Fig. 8. Histogram for JPEG coefficients (Fig. 2) with F4's interpretation of steganographic values

two bars of the same height represent coefficients with inverse steganographic value (steganographic zeroes are black, steganographic ones white).

Fig. 9 shows the embedding loop of F4 in Java source code. The array coeff[] holds all the JPEG coefficients of the carrier medium.

Suppose we have two random variables X, Y for observed coefficients before and after F4 embeds a message. $P(X = x)$ denotes the probability for JPEG producing a coefficient with a given value x, and $P(Y = y)$ denotes the probability for F4 producing a coefficient with a given value y. We can write the two characteristic properties (cf. Sect. 2) for some coefficient values

$$P(X = 1) > P(X = 2) > P(X = 3) > P(X = 4) \tag{5}$$

$$P(X = 1) - P(X = 2) > P(X = 2) - P(X = 3) > P(X = 3) - P(X = 4) \tag{6}$$

If the message bits are uniformly distributed, we deduce

$$P(Y = 1) = \frac{1}{2}P(X = 1) + \frac{1}{2}P(X = 2) \tag{7}$$

$$P(Y = 2) = \frac{1}{2}P(X = 2) + \frac{1}{2}P(X = 3) \tag{8}$$

$$P(Y = 3) = \frac{1}{2}P(X = 3) + \frac{1}{2}P(X = 4) \tag{9}$$

We subtract (7) and (8) to get (10), as well as (8) and (9) to get (11).

$$P(Y = 1) - P(Y = 2) = \frac{1}{2}P(X = 1) - \frac{1}{2}P(X = 3) \tag{10}$$

$$P(Y = 2) - P(Y = 3) = \frac{1}{2}P(X = 2) - \frac{1}{2}P(X = 4) \tag{11}$$

With (5) we know that the right hand sides of (10) and (11) are positive, so we find the first characteristic property for Y

$$P(Y = 1) > P(Y = 2) > P(Y = 3) \tag{12}$$

```
int nextBitToEmbed = embeddedData.readBit();
for (int i=0; i<coeff.length; i++) {
    if (i%64 == 0) continue; // skip DC coefficients
    if (coeff[i] == 0) continue; // skip zeroes
    if (coeff[i] > 0) {
        if ((coeff[i]&1) != nextBitToEmbed)
            coeff[i]--; // decrease absolute value
    } else {
        if ((coeff[i]&1) == nextBitToEmbed)
            coeff[i]++; // decrease absolute value
    }
    if (coeff[i] != 0) { // successfully embedded
        if (embeddedData.available()==0)
            break; // end of embeddedData
        nextBitToEmbed = embeddedData.readBit();
    }
}
```

Fig. 9. Java source code for the embedding function of F4 (simplified)

If we add $P(X = 2) - P(X = 3)$ to (6), we find

$$P(X = 1) - P(X = 3) > P(X = 2) - P(X = 4) \tag{13}$$

With (13) we see that the right hand side of (10) is greater than in (11). So the left hand sides give the second characteristic property for Y.

$$P(Y = 1) - P(Y = 2) > P(Y = 2) - P(Y = 3) \tag{14}$$

Similarly we can show these characteristic properties for other values modified by F4, i. e. decreasing occurrence with increasing absolute value (cf. (12)), and decreasing decrease with increasing absolute value (cf. (14)).

6 F5

Unlike stream media (like in video conferences), image files only provide a limited steganographic capacity. In many cases, an embedded message does not require the full capacity (if it fits). Therefore, a part of the file remains unused. Fig. 10 shows, that (with continuous embedding) the changes (×) concentrate on the start of the file, and the unused rest resides on the end.

To prevent attacks, the embedding function should use the carrier medium as regular as possible. The embedding density should be the same everywhere.

6.1 Permutative Straddling

Some well-known steganographic algorithms scatter the message over the whole carrier medium. Many of them have a bad time complexity. They get slower if

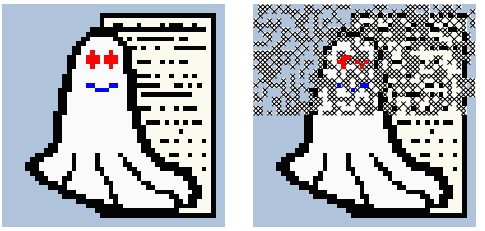

Fig. 10. Continuous embedding concentrates changes (×)

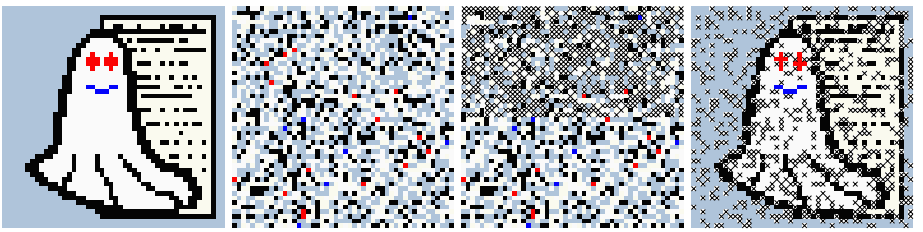

Fig. 11. Permutative embedding scatters the changes (×)

we try to exhaust the steganographic capacity completely. Straddling is easy, if the capacity of the carrier medium is known exactly. However, we can not predict the shrinkage for F4, because it depends on which bit is embedded in which position. We merely can estimate the expected capacity.

The straddling mechanism used with F5 shuffles all coefficients using a permutation first. Then, F5 embeds into the permuted sequence. The shrinkage does not change the number of coefficients (only their values). The permutation depends on a key derived from a password. F5 delivers the steganographically changed coefficients in its original sequence to the Huffman coder. With the correct key, the receiver is able to repeat the permutation. The permutation has linear time complexity $O(n)$. Fig. 11 shows the uniformly distributed changes over the whole image. Please treat the pixels as coefficients.

6.2 Matrix Encoding

Ron Crandall [1] introduced matrix encoding as a new technique to improve the embedding efficiency. F5 possibly is the first implementation of matrix encoding. If most of the capacity is unused in a steganogram, matrix encoding decreases the necessary number of changes. Let us assume that we have a uniformly distributed secret message and uniformly distributed values at the positions to be changed. One half of the message causes changes, the other half does not. Without matrix encoding, we have an embedding efficiency of 2 bits per change. Because of the shrinkage produced by F4, the embedding efficiency is even a bit lower, e. g. 1.5 bits per change. (Shrinkage means to change without to embed sometimes, cf. Sect. 4.)

For example, if we embed a very short message comprising only 217 bytes (1736 bits), F4 changes 1157 places in the *Expo* image. F5 embeds the same message using matrix encoding with only 459 changes, i. e. with an embedding efficiency of 3.8 bits per change.

The following example shows what happened in detail. We want to embed two bits x_1, x_2 in three modifiable bit places a_1, a_2, a_3 changing one place at most. We may encounter these four cases:

$$\begin{aligned}
&x_1 = a_1 \oplus a_3,\ x_2 = a_2 \oplus a_3 \Rightarrow \text{change nothing}\\
&x_1 \neq a_1 \oplus a_3,\ x_2 = a_2 \oplus a_3 \Rightarrow \text{change } a_1\\
&x_1 = a_1 \oplus a_3,\ x_2 \neq a_2 \oplus a_3 \Rightarrow \text{change } a_2\\
&x_1 \neq a_1 \oplus a_3,\ x_2 \neq a_2 \oplus a_3 \Rightarrow \text{change } a_3.
\end{aligned}$$

In all four cases we do not change more than one bit. In general, we have a code word $\boldsymbol{a}$ with n modifiable bit places for k secret message bits $\boldsymbol{x}$. Let f be a hash function that extracts k bits from a code word. Matrix encoding enables us to find a suitable modified code word $\boldsymbol{a'}$ for every $\boldsymbol{a}$ and $\boldsymbol{x}$ with $\boldsymbol{x} = f(\boldsymbol{a'})$, such that the Hamming distance

$$d(\boldsymbol{a}, \boldsymbol{a'}) \leq d_{\max} \tag{15}$$

We denote this code by an ordered triple $(d_{\max}, n, k)$: a code word with n places will be changed in not more than $d_{\max}$ places to embed k bits.[3] F5 implements matrix encoding only for $d_{\max} = 1$. For $(1, n, k)$, the code words have the length $n = 2^k - 1$. Neglecting shrinkage, we get a change density

$$D(k) = \frac{1}{n+1} = \frac{1}{2^k} \tag{16}$$

and an embedding rate

$$R(k) = \frac{k}{n} = \frac{1}{n} \cdot \operatorname{ld}(n+1) = \frac{k}{2^k - 1} \tag{17}$$

Using the change density and the embedding rate we can define the embedding efficiency $W(k)$. It indicates the average number of bits we can embed per change:

$$W(k) = \frac{R(k)}{D(k)} = \frac{2^k}{2^k - 1} \cdot k \tag{18}$$

The embedding efficiency of the $(1, n, k)$ code is always larger than k. Table 1 shows that the rate decreases with increasing efficiency. Hence, we can achieve high efficiency with very short messages only.

Table 2 gives the dependencies between the message bits x_i and the changed bit places a'_j. We assign the dependencies with the "binary coding" of j to column a'_j. So we can determine the hash function very fast.

$$f(\boldsymbol{a}) = \bigoplus_{i=1}^{n} a_i \cdot i \tag{19}$$

[3] We denote our concrete example above by the triple $(1, 3, 2)$.

Table 1. Connection between change density and embedding rate

k	n	change density	embedding rate	embedding efficiency
1	1	50.00 %	100.00 %	2
2	3	25.00 %	66.67 %	2.67
3	7	12.50 %	42.86 %	3.43
4	15	6.25 %	26.67 %	4.27
5	31	3.12 %	16.13 %	5.16
6	63	1.56 %	9.52 %	6.09
7	127	0.78 %	5.51 %	7.06
8	255	0.39 %	3.14 %	8.03
9	511	0.20 %	1.76 %	9.02

Table 2. Dependency ($\times$) between message bits x_i and code word bits a'_j

$f(\boldsymbol{a}')$	a'_1	a'_2	a'_3
x_1	$\times$		$\times$
x_2		$\times$	$\times$

$f(\boldsymbol{a}')$	a'_1	a'_2	a'_3	a'_4	a'_5	a'_6	a'_7
x_1	$\times$		$\times$		$\times$		$\times$
x_2		$\times$	$\times$			$\times$	$\times$
x_3				$\times$	$\times$	$\times$	$\times$

We find the bit place

$$s = \boldsymbol{x} \oplus f(\boldsymbol{a}) \tag{20}$$

that we have to change.[4] The changed code word results in

$$\boldsymbol{a'} = \begin{cases} \boldsymbol{a}, & \text{if } s = 0 \ (\Leftrightarrow \boldsymbol{x} = f(\boldsymbol{a})) \\ (a_1, a_2, \ldots, \neg a_s, \ldots, a_n) & \text{otherwise} \end{cases} \tag{21}$$

We can find an optimal parameter k for every message to embed and every carrier medium providing sufficient capacity, so that the message just fits into the carrier medium. For instance, if we want to embed a message with 1000 bits into a carrier medium with a capacity of 50000 bits, then the necessary embedding rate is $R = 1000 : 50000 = 2\,\%$. This value is between $R(k = 8)$ and $R(k = 9)$ in Table 1. We choose $k = 8$, and are able to embed $50000 : 255 = 196$ code words with a length $n = 255$. The $(1, 255, 8)$ code could embed $196 \cdot 8 = 1568$ bits. If we chose $k = 9$ instead, we could not embed the message completely.

6.3 Preserving Characteristic Properties

To prove the security of a steganographic algorithm, it would be necessary to formalise *perceptibility*. That is much more as for cryptography, where we can establish information-theoretic relations. Let us try to prove the resistance against *known* attacks instead.

The statistical attacks presented in [5] can reveal the presence of a hidden message, if the steganographic algorithm overwrites least significant bits. This is

[4] We interpret the resulting bit vector as an integer.

no longer the case with F4/F5. F4 preserves characteristic properties and does not equalise frequencies (cf. Sect. 5). We can show that F5 preserves the same characteristic properties: Let $0 \leq \alpha \leq 1$ be the fraction of coefficients used for steganography.[5] If we adopt (7) ... (9), the proof works for F5 too:

$$P(Y=1) = \left(1-\frac{\alpha}{2}\right) P(X=1) + \frac{\alpha}{2} P(X=2) \tag{22}$$

$$P(Y=2) = \left(1-\frac{\alpha}{2}\right) P(X=2) + \frac{\alpha}{2} P(X=3) \tag{23}$$

$$P(Y=3) = \left(1-\frac{\alpha}{2}\right) P(X=3) + \frac{\alpha}{2} P(X=4) \tag{24}$$

We subtract (22) and (23) to get (25), as well as (23) and (24) to get (26).

$$P(Y=1) - P(Y=2) = \left(1-\frac{\alpha}{2}\right)\big(P(X=1) - P(X=2)\big) + \frac{\alpha}{2} P(X=3) \tag{25}$$

$$P(Y=2) - P(Y=3) = \left(1-\frac{\alpha}{2}\right)\big(P(X=2) - P(X=3)\big) + \frac{\alpha}{2} P(X=4) \tag{26}$$

With (5) (cf. Sect. 5) we know that the right hand sides of (25) and (26) are positive, so we find the first characteristic property for Y:

$$P(Y=1) > P(Y=2) > P(Y=3) \tag{27}$$

With the characteristic properties of X (cf. (5) and (6))

$$P(X=1) - P(X=2) > P(X=2) - P(X=3)$$
$$P(X=3) > P(X=4)$$

we see that the right hand side of (25) is greater than in (26). So the left hand sides give the second characteristic property for Y:

$$P(Y=1) - P(Y=2) > P(Y=2) - P(Y=3) \tag{28}$$

Similarly we can show these characteristic properties for other values modified by F5.

6.4 Implementation

The algorithm F5 has the following coarse structure:

1. Start JPEG compression. Stop after the quantisation of coefficients.
2. Initialise a cryptographically strong random number generator with the key derived from the password.
3. Instantiate a permutation (two parameters: random generator and number of coefficients[6]).

[5] F4 is the special case $\alpha = 1$

[6] including zero coefficients

4. Determine the parameter k from the capacity of the carrier medium, and the length of the secret message.
5. Calculate the code word length $n = 2^k - 1$.
6. Embed the secret message with $(1, n, k)$ matrix encoding.
   (a) Fill a buffer with n nonzero coefficients.
   (b) Hash this buffer (generate a hash value with k bit-places). (cf. (19))
   (c) Add the next k bits of the message to the hash value (bit by bit, xor). (cf. (20))
   (d) If the sum is 0, the buffer is left unchanged. Otherwise the sum is the buffer's index $1 \ldots n$, the absolute value of its element has to be decremented. (cf. (21))
   (e) Test for shrinkage, i. e. whether we produced a zero. If so, adjust the buffer (eliminate the 0 by reading one more nonzero coefficient, i. e. repeat step 6a beginning from the same coefficient). If no shrinkage occurred, advance to new coefficients behind the actual buffer. If there is still message data continue with step 6a.
7. Continue JPEG compression (Huffman coding etc.).

7 Conclusion

Many steganographic algorithms offer a high capacity for hidden messages, but are weak against visual and statistical attacks. Tools withstanding these attacks provide only a very small capacity. The algorithm F4 combines both preferences: resistance against visual and statistical attacks as well as high capacity. Matrix encoding and permutative straddling enable the user to decrease the necessary number of steganographic changes and to equalise the embedding rate in the steganogram. F5 accomplishes a steganographic proportion that exceeds 13 % of the JPEG file size (cf. Table 3). Please understand this result as a friendly provocation for security analysts. On the other hand F5 is able to decrease the embedding rate arbitrarily. The software with its source code is public [7].

Acknowledgements. I would like to thank Fabien Petitcolas for helpful comments.

Table 3. Comparison of several JPEG files created with F5

File name	File size (bytes)	Embedded size (bytes)	Ratio embedded to steganogram size	Embedding efficiency	Quantiser quality
expo.bmp	1,562,030	0	(carrier medium)	—	—
expo80.jpg	129,879	0	—	—	80 %
ministeg.jpg	129,760	213	0.2 %	3.8	80 %
maxisteg.jpg	115,685	15,480	13.4 %	1.5	80 %
expo75.jpg	114,712	0	—	—	75 %

References

1. Ron Crandall: Some Notes on Steganography. Posted on Steganography Mailing List, 1998. http://os.inf.tu-dresden.de/~westfeld/crandall.pdf 297
2. Andy C. Hung: PVRG-JPEG Codec 1.1, Stanford University, 1993. http://archiv.leo.org/pub/comp/os/unix/graphics/jpeg/PVRG 291
3. Fabien Petitcolas: MP3Stego, 1998. http://www.cl.cam.ac.uk/~fapp2/steganography/mp3stego 289
4. Derek Upham: Jsteg, 1997, e. g. http://www.tiac.net/users/korejwa/jsteg.htm 289
5. 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. 289, 291, 293, 299
6. Andreas Westfeld, Gritta Wolf: Steganography in a Video Conferencing System, in David Aucsmith (Ed.): Information Hiding, LNCS 1525, Springer-Verlag Berlin Heidelberg 1998. pp. 32–47. 289
7. Andreas Westfeld: The Steganographic Algorithm F5, 1999. http://wwwrn.inf.tu-dresden.de/~westfeld/f5.html 301
8. Jan Zöllner, Hannes Federrath, Herbert Klimant, Andreas Pfitzmann, Rudi Piotraschke, Andreas Westfeld, Guntram Wicke, Gritta Wolf: Modeling the Security of Steganographic Systems, in David Aucsmith (Ed.): Information Hiding, LNCS 1525, Springer-Verlag Berlin Heidelberg 1998. pp. 344–354. 289