February 28, 2021

Implementation Note

snote.eu.com exploits the F5 algorithm to hide user information in the DCT coefficients of a JPEG image. The algorithm proposed in the scientific literature has been slightly modified to increase the image steganography capacity and avoid that, in the face of a continuous reuse of the image, the quality of the image is deteriorated due to the average reduction of the coefficient's values.

1. F5 algorithm

The F5 algorithm [1,2] is defined by the couple of the embedding and extracting procedures. Both exploit the following **LSB$_{F5}$** function that modify the lower significant bit of the coefficient x.

$$LSB_{F5}(x) = \begin{cases} 1 - x \bmod 2 & for\ x < 0 \\ x \bmod 2 & otherwise \end{cases}$$

The embedding and extracting procedures perform as follow.

Embedding. Embedding message m∈{0,1} in JPEG image x∈X^n using the F5 algorithm (no matrix embedding employed).

```
// Initialize a PRNG using stego key (or passphrase)
//Input: message m∈{0,1}, quantizied DCT coefficients x∈X^n
Path = Perm(n); // Perm(n) is a pseudo-random permutation of {1,2,…n}
y=x;
i=1; j=1; // i message index, j coefficient index
while ( i ≤ m ) & ( j ≤ n ) {
      if ( x[Path[j]] ≠ 0 ) & ( x[Path[j]] is not DC term ) {
            if LSB_F5(x[Path[j]]) = m(i) {i=i+1;} // equal no change to y
            else { // flip y
                  y[Path[j]] = x[Path[j]] - sign([Path[j]])
                  if y[Path[j]] ≠ 0 {i=i+1;}    // zero do not embed m
                  // next coefficient
            }
      }
      j=j+1;
} // y are stego DCT coefficients coveying i message bits
```

Extracting. Extracting message $m \in \{0,1\}$ from JPEG stego image $y \in X^n$ embedded the F5 algorithm (no matrix embedding employed).

```
// Initialize a PRNG using stego key (or passphrase)
// Input: quantizied DCT coefficients $y \in X^n$
Path = Perm(n); // Perm(n) is a pseudo-random permutation of {1,2,…n}
i=1; j=1; // i message index, j coefficient index
while( $j \le n$ ) {
      if ( y[Path[j]] $\ne$ 0 ) & ( y[Path[j]] is not DC term ) {
            m[i] = $LSB_{F5}$(y[Path[j]]) {i=1+1;}
      }
      j=j+1;
}     // Read the header of extracted bits to find the message length and
      // truncate m.
```

As we can see the embedding operation reduce the coefficient size for which we have not

$LSB_{F5}(x[\mathbf{Path}[j]]) = \mathbf{m}(i)$

by 1, i.e.

$y[\mathbf{Path}[j]] = x[\mathbf{Path}[j]] - sign([\mathbf{Path}[j]])$

either the coefficient is positive or negative.

2. F5* algorithm

In the plausible hypothesis of falling into the "else" part of the embedding procedure for half the bits of the message, each writing action will correspond to a reduction of a number of coefficients equal to 50% of the length of the message. Let consider a message long as the number of available coefficients (i.e., a message of whom size corresponds to the steganography image capacity), at each iteration, namely at “save” click, the 50% of the coefficient will be decreased by |1| that corresponds to both an image quality degradation and a steganography capacity reduction.

In order to reduce the impact of the F5 steganography on the image quality (and capacity), snote.eu.com has slightly modified the F5 embedding procedure. The idea behind the change is to increase the absolute value of the odd coefficients and decrease that of the even ones, so for example the coefficient 1 (or -1) will become 2 (or -2) respectively, while the coefficient 2 (or -2) will become 1 (or -1). This choice significantly increases the image capacity as coefficients equal to ±1 are available for steganography as well.

First, the LSBF5 function is modified as follow (where the % operator Is the reminder that differ from the mod operator only for negative values):

$$LSB^{*}_{F5}(x) = \begin{cases} 1 - x \,\%\, 2 & for\ x < 0 \\ x \,\%\, 2 & otherwise \end{cases}$$

Second, the sign function of the embedding procedure is modified as follow

```
sign*(x) =  if (x ≥ 0) then {                     // positive values
                  if x % 2 = 0 then 1             // even
                  else -1                         // odd
            } else {                              // negative values (reverse)
                  if 1 + x % 2 = 0 then 1         // odd
                  else -1                         // even
            }
```

```
// Initialize a PRNG using stego key (or passphrase)
//Input: message m∈{0,1}, quantizied DCT coefficients x∈X^n
Path = Perm(n); // Perm(n) is a pseudo-random permutation of {1,2,…n}
y=x;
i=1; j=1; // i message index, j coefficient index
while ( i ≤ m ) & ( j ≤ n ) {
      if ( x[Path[j]] ≠ 0 ) & ( x[Path[j]] is not DC term ) {
            if LSB_F5(x[Path[j]]) = m(i) {i=i+1;} // equal no change to y
            else { // flip y
                  y[Path[j]] = x[Path[j]] - sign*([Path[j]])
                  if y[Path[j]] ≠ 0 {i=i+1;}    // zero do not embed m
                  // next coefficient
            }
      }
      j=j+1;
} // y are stego DCT coefficients coveying i message bits
```

For instance, we have

x	sign*(x)	next coeff
-3	1	-4
-2	-1	-1
-1	1	-2
1	-1	2
2	1	1
3	-1	4

The extraction procedure is not modified as the flipping method works either if you increase or decrease the coefficient.

3. DCT coefficients range, and flipping

The range of possible DCT coefficient values is obtained by the formula

$$S_{uv} = \frac{1}{4} C_u C_v \sum_{y=0}^{7} \sum_{x=0}^{7} s_{xy} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

The argument of the summation is of the magnitude of sxy times two values which are at most 1 (absolute value). So still 8 bits. You are summing 8×8=64 of those values, so 26×28=214. But you also divide this by 14 so the maximum value requires 12 bits. You also have to consider that this value is obtained only when u and v are 0, and in that case $C_u = C_v = \frac{1}{\sqrt{2}}$ so $C_u C_v = \frac{1}{2}$, that is another bit is unneeded. Some other checks are required when only u=0 or v=0, but overall, you will need 11 bits at most.

If you represent the coefficient as a short (i.e., a signed integer type in C99) your range is 211 = 2048 large and goes from -1024 to 1024.

And we have

x	sign*(x)	next coeff
-1024	-1	-1023
-1023	1	-1024
-1022	-1	-1021
1022	1	1021
1023	-1	1024
1024	1	1023

4. Histograms

Consider the classical “Lena” image (74KB 512x512 RGB)

we embed in the image a pseudo-random 1024-character string, the DCT coefficient histogram in the range [-25, 25] change as described in figure 1 and 2.

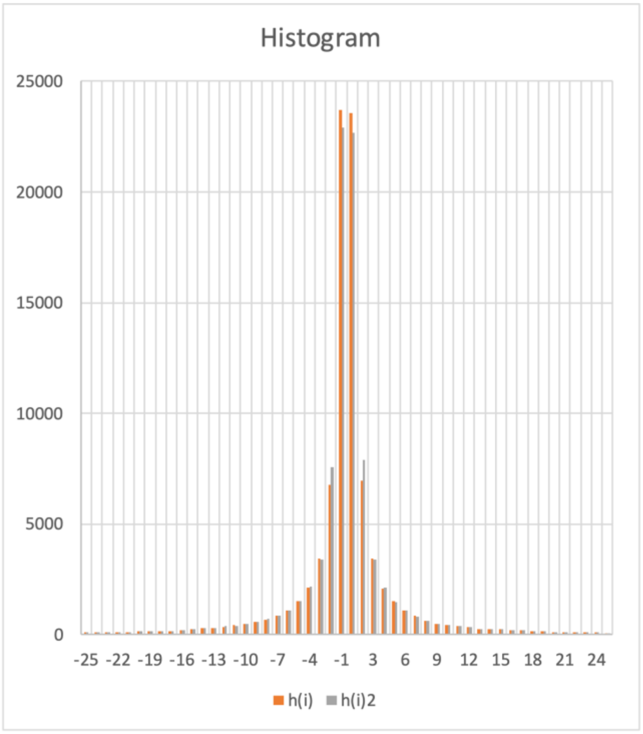

Figure 1

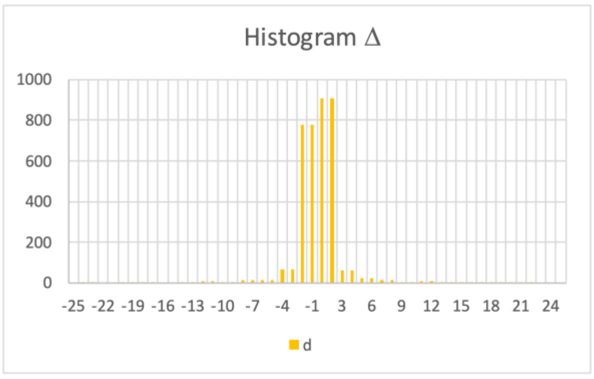

Figure 2

The image has 91776, no zero / no DC, DCT coefficients. F5* can hide in this image at most an 11,2 KB size message while F5 is able to mask only 5,4 KB size message (-51%). Our test message is 8192-bit length, and it affects 3856 coefficients, i.e., the 47% of those used to hide the message itself.

This result is just obtained on a single run. A larger test should be conducted to compare the properties of F5* with regard to the ones of F5. Nevertheless, this research activity is out of the scope of the snote.eu.com project.

Bibliography

[1] A. Westfled, F5—A Steganographic Algorithm High Capacity Despite Better Steganalysis, I. S. Moskowitz (Ed.): IH 2001, LNCS 2137, pp. 289–302, 2001.

[2] J. Fridrich, Steganography in the digital media, Cabridge University Press, pp. 119-122, 2010.