Information Flows and Formats of the Stegano System

April 28, 1997

This document describes the information flow and the used information formats in the Stegano system. With this document it is possible to build systems that are compatible with the Stegano system.

1 Definitions

Byte is a sequence of eight bits. The bits inside a byte are numbered so that the most significant bit is the bit number 7 and the least significant bit is the bit number 0. The numerical value of a byte is $\sum_{i=0}^{7} 2^i bit_i$. Values of bytes can be expressed both in hexadecimal and in decimal form. For example, $\mathrm{C5_H} = 197$.

In the graphical and in the bit string formats, bit 7 is the leftmost bit. For example, byte $\mathrm{C5_H}$ is equivalent to the bit string 11000101_2.

Normal C-like operators, like bitwise AND (&-operator), are defined for bytes.

Byte String is a sequence of bytes. Bytes in the byte string are numbered in left-to-right order, from 1 to $\|bytestring\|$, where $\|bytestring\|$ is the lenght of the byte string in bytes.

$bytestring_i$ means the ith byte of the byte string. For example, let $bs = \mathrm{00FC80_H}$ be a bytestring. Then $\|bs\| = 3$ and $bs_1 = \mathrm{00_H}$, $bs_2 = \mathrm{FC_H}$ and $bs_3 = \mathrm{80_H}$. The bit string presentation for bs is $000000001111110010000000_2$.

The numerical value of a byte string bs is $|bs| = \sum_{i=1}^{||bs||} 256^{||bs||-i} bs_i$. For example, $|00FC80_H|$ is $FC80_H$ in hexadecimal format and 64640 in decimal format.

The concatenation (with the $\circ$-operator) of two or more byte strings is a bytestring, whose length is the sum of the lengths of the concatenated byte strings. That is, if $bytestring\_a = bytestring\_b \circ bytestring\_c$, then $\|bytestring\_a\| = \|bytestring\_b\| + \|bytestring\_c\|$. Furthermore, $\forall i : 1 \leq i \leq \|bytestring\_b\| : bytestring\_a_i = bytestring\_b_i$ and $\forall i : 1 \leq i \leq \|bytestring\_c\| : bytestring\_a_{\|bytestring\_b\|+i} = bytestring\_c_i$.

Cover Bits are the bits of the cover file that might change in the hiding process.

Extraction Process is the procedure of finding and extracting a hidden message from the cover file.

Hiding Process is the procedure that hides the user-given message into the cover file.

Key Material is a byte string that contains key material that can be used in various algorithms in the hiding and extraction processes.

Non-Cover Bits are the bits of the cover file that cannot change in the hiding process, regardless of the used pass-phrase, message, filename or any other parameter.

Pass-phrase is an user-given password, that is used to calculate the needed key material. The pass-phrase is a byte string that may contain any byte values.

Plain Message is the original message that the user wants to hide before any processing. The plain message is a byte string that may contain any byte values.

2 The Hiding Phase Information Flow

Figure 1 describes the hiding phase information flow in the Stegano system.

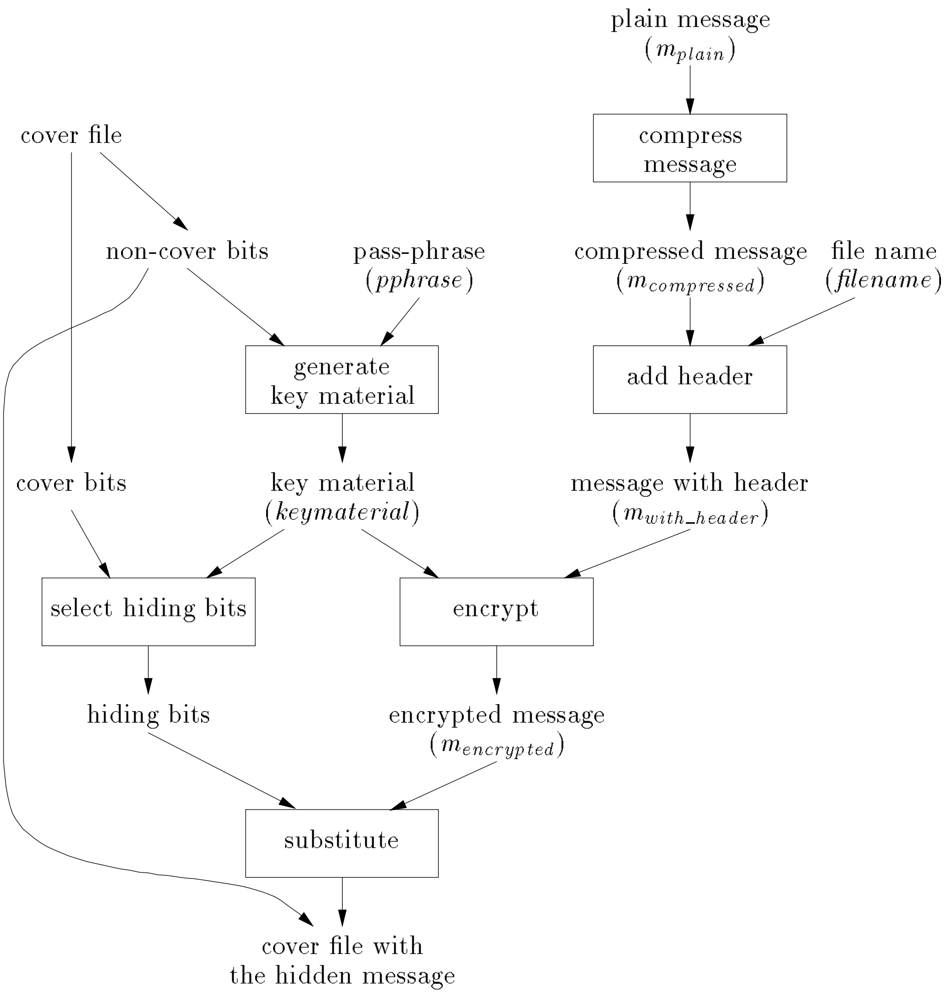

Figure 1: The hiding phase information flow in the Stegano system.

3 Short Descriptions of the Used Cover File Formats

3.1 The PGM Image Format

PGM images are gray-scale images. The values of pixels are in range $[0, 255]$. In this document, an N pixels wide and M pixels high PGM image is modelled as a 2-dimensional byte table pgm, whose elements $pgm_{n,m}$ represent the values of the corresponding pixels in the PGM image. $N, M \geq 1$, $1 \leq n \leq N$ and $1 \leq m \leq M$.

3.2 The JPEG Image Format

JPEG images used the Stegano system are JPEG files that use lossy compression. The hiding and extraction processes use the JPEG image after it has been DCT-transformed and the quantization has been done. In this phase the JPEG image consists of C components ($c \geq 1$), each of which has $N_c \times M_c$ blocks. N_c and M_c may have different values for different components, i.e. components can be of different size. Each block has 8×8 integers that are in range [-128, 127] i.e. the normalized quantization coefficients. See [2] for more details on JPEG.

The JPEG image jpg is modelled as a set of 6-tuples $\langle c, n, m, x, y, val\rangle$, each tuple representing the value of a single coefficient. For each coefficient in each block and in each component, there is a unique 6-tuple in the model. $1 \leq c \leq C$, $1 \leq n \leq N_c$, $1 \leq m \leq M_c$, $1 \leq x, y \leq 8$ and $val \in [-128, 127]$. $jpg_{c,n,m,x,y}$ is the value of the val field in the tuple $\langle c, n, m, x, y, val\rangle$.

The coefficients $\langle c, n, m, 1, 1, val\rangle$ are said to be AC-coefficients and the coefficients $\langle c, n, m, x, y, val\rangle$, where $1 < x, y \leq 8$, are said to be DC-coefficients.

The set of modifiable AC-coeffiecients $jpgmod$ is an ordered set of 6-tuples $\langle c, n, m, x, y, val\rangle$ such that

$$\begin{aligned}\langle c, n, m, x, y, val\rangle \in jpgmod \quad \Leftrightarrow \quad & \langle c, n, m, x, y, val\rangle \in jpg \wedge \\ & 1 < x, y \leq 8 \wedge abs(val) \geq 4 \end{aligned} \qquad (1)$$

where abs is the absolute value function. In addition, it holds that

$$\forall i : \forall j : 1 \leq j < i \leq \|jpgmod\| : jpgmod_j \leq jpgmod_i \qquad (2)$$

, where $\|jpgmod\|$ is the number of 6-tuples in $jpgmod$ and

$$\begin{aligned} jpgmod_j \leq jpgmod_i \quad \Leftrightarrow \quad & jpgmod_{j,c} \leq jpgmod_{i,c} \wedge jpgmod_{j,n} \leq jpgmod_{i,n} \wedge \\ & jpgmod_{j,m} \leq jpgmod_{i,m} \wedge jpgmod_{j,x} \leq jpgmod_{i,x} \wedge \\ & jpgmod_{j,y} \leq jpgmod_{i,y}. \end{aligned} \qquad (3)$$

where $jpgmod_i$ means the ith 6-tuple in the ordered set $jpgmod$ and $jpgmod_{i,c}$ means the value of the c-field of the ith 6-tuple in the ordered set $jpgmod$.

3.3 The GIF Image Format

Hack the code.

4 Generating the Key Material

The key material byte string *keymaterial* is an output of an one-way hash function. The input to the one-way hash function generating the key material is the concatenation of the user defined pass-phrase *pphrase* and the non-cover bits of the cover file. This way the key material depends not only on the user defined pass-phrase, but also on the used cover file. The user defined pass-phrase is concatenated both in the beginning and in the end of the one-way hash function input. That is,

$$\begin{aligned} keymaterial &= H_x(pphrase \circ coverbits_{non\_cover} \circ pphrase) \quad \text{and} \qquad (4) \\ \|keymaterial\| &> 0 \qquad (5) \end{aligned}$$

where H_x is the used one-way hash function. $coverbits_{non\_cover}$ is a byte string composed of all or some of the non-cover bits of the cover file in a cover file format depended way. The following subsections describe how to calculate $coverbits_{non\_cover}$ from various cover file formats.

Currently supported one-way hash functions are:

- H_{SHA}. The SHA hash function. Produces 20 bytes of key material.

4.1 The PGM Image Format

Let *pgm* be the 2-dimensional byte table as described in Section 3.1.

The $coverbits_{non\_cover}$ byte string is the concatenation of all the bytes in the *pgm* table, excluding the least significant bit of each byte. The concatenation of bytes is done in left-to-right, top-down order. The exclusion of the least significant bit of a byte is done by ANDing the byte with constant $\mathrm{FE_H}$. That is,

$$\begin{aligned} coverbits_{non\_cover} &= pgm'_{1,1} \circ pgm'_{2,1} \circ \ldots \circ pgm'_{N,1} \circ \\ &\quad\; pgm'_{1,2} \circ pgm'_{2,2} \circ \ldots \circ pgm'_{N,2} \circ \\ &\quad\; \vdots \\ &\quad\; pgm'_{1,M} \circ pgm'_{2,M} \circ \ldots \circ pgm'_{N,M} \qquad (6) \\ \|coverbits_{non\_cover}\| &\geq 1 \qquad (7) \end{aligned}$$

where $pgm'_{n,m} = pgm_{n,m} \& \mathrm{FE_H}$.

4.2 The JPEG Image Format

Let jpg be a set of 6-tuples as described in Section 3.2.

The $coverbits_{non\_cover}$ byte string is the concatenation of all the DC coefficients of all the blocks in all the components in the JPG image. That is,

$$coverbits_{non\_cover} \quad = \quad \circ_{c=1}^{C}(\circ_{m=1}^{M_c}(\circ_{n=1}^{N_C} jpg_{c,n,m,1,1})) \tag{8}$$

where $\circ_?^?$ is the 'concatenation sum' operator, for example $\circ_{i=1}^{3} i = 01_{\mathrm{H}} \circ 02_{\mathrm{H}} \circ 03_{\mathrm{H}}$.

4.3 The GIF Image Format

Hack the code.

5 Compressing the Plain Message

In order to be able to hide longer messages, the plain message m_{plain} can be compressed. The output of a compression algorithm $COMP_x$ with input m_{plain} is the compressed message $m_{compressed}$. That is,

$$m_{compressed} = COMP_x(m_{plain}). \tag{9}$$

For each used compression algorithm $COMP_x$, there must exist an inverse function $DECOMP_x$ such that $\forall m : DECOMP_x(COMP_x(m)) = m$.

For cases where no compression is wanted, functions $COMP_{NONE}(m) = m$ and $DECOMP_{NONE}(m) = m$ are defined.

Currently supported compression algorithms are:

- $COMP_{NONE}$. No compression.
- $COMP_{GZIP}$. The GZIP compression algorithm.

6 Header Format

In order to be able to recognize the hidden message in the cover file and to be able to retrieve all the necessary information about the hidden message in the extraction phase, a header must be added into the message in the hiding phase. The header *header* is a byte string and it is added into the beginning of the message after compression. The result is the message with header, m_{with_header}. That is,

$$m_{with\_header} = header \circ m_{compressed} \tag{10}$$

The format of the header is shown in Figure 2.

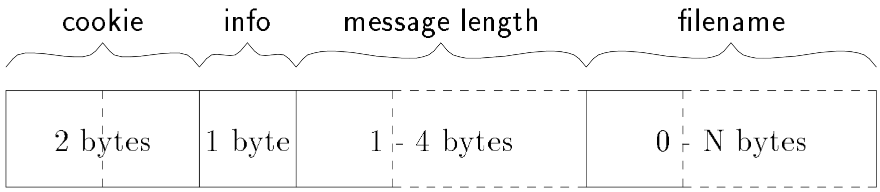

Figure 2: The format of the header.

The header can also be expressed as a concatenation of byte strings representing the fiels in the header:

$$\begin{aligned} header &= cookie \circ infobyte \circ mlength \circ filename & (11)\\ \|cookie\| &= 2 & (12)\\ \|infobyte\| &= 1 & (13)\\ 1 \leq \|mlength\| &\leq 4 & (14)\\ \|filename\| &\geq 1 & (15) \end{aligned}$$

The first two bytes of the header form the cookie field, that is used to recognize the hidden message in the cover file in the extraction phase. The value of the cookie is $\mathrm{FB73_H}$. In the byte string notation,

$$cookie = \mathrm{FB73_H} \tag{16}$$

The structure of the info field (*infobyte* byte string) is shown in Figure 3.

The four most significant bits (bits 4 – 7) are reserved for future use and must be 0s.

The compression algorithm bits (bits 2 and 3) identify the compression algorithm that is used to compress the message. Table 1 describes the association between the compression algorithm bits and the used compression algorithm (see also Section 5).

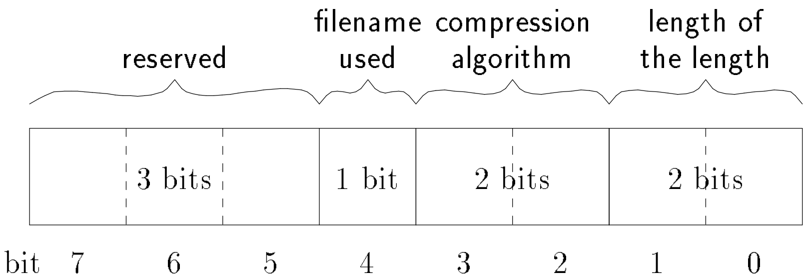

Figure 3: The structure of the info byte.

bit 3	bit 2	Compression function
0	0	$COMP_{NONE}$
0	1	$COMP_{GZIP}$
1	0	Reserved for future use
1	1	Reserved for future use

Table 1: Meaning of the compression algorithm bits.

The length of the length bits of the *infobyte* describe the length of the message length field (*mlength* byte string) in the header. Table 2 describes the association between the length of the length bits and the length of the message length field.

bit1	bit 0	Length of the message length field $\|\|mlength\|\| =$	Value range of the message length field $\|mlength\| \in$
0	0	1	$[0, 255]$
0	1	2	$[0, 65535]$
1	0	3	$[0, 2^{24} - 1]$
1	1	4	$[0, 2^{32} - 1]$

Table 2: Meaning of the length of the length bits.

The message length field describes the length of the hidden message after compression, that is:

$$|mlength| = \|m_{compressed}\| \tag{17}$$

The filename used bit of the *infobyte* indicates, whether the filename field in the header is used or not. If the bit is 0, then the filename field is not used. If the bit is 1, then filename field is used and the filename field is a 00_H-terminated byte string *filename*, that describes the file name of the hidden message. The file name may contain all the other byte values except 00_{H}, that is reserved to indicate the end of the **filename** field. 00_{H}. More formally,

$$\textsf{filename used bit is 0} \Rightarrow \|\mathit{filename}\| = 0 \tag{18}$$
$$\textsf{filename used bit is 1} \Rightarrow \|\mathit{filename}\| > 0 \tag{19}$$
$$\textsf{filename used bit is 1} \Rightarrow \mathit{filename}_{\|\mathit{filename}\|} = 00_{\mathrm{H}} \tag{20}$$
$$\textsf{filename used bit is 1} \Rightarrow \forall i : 1 \leq i \leq \|\mathit{filename}\| - 1 : \mathit{filename}_i \neq 00_{\mathrm{H}} \tag{21}$$

It is up to the extracting software to check that the file name is a valid file name in the used system architecture and that no security risks are involved with the file name.

7 Encrypting the Message

After the header addition, the message m_{with_header} is encrypted with an encryption algorithm. The result is the encrypted message $m_{encrypted}$.

Encryption algorithms that encrypt fixed sized blocks, like DES and IDEA, are used in the electronic code book (ECB) mode. Let E_x be the used block encryption algorithm, whose block size in bytes is b. Then the input blocks $(block_1, \ldots, block_{\lceil \|m_{with\_header}\|/b \rceil})$ to the E_x are byte strings defined by

$$\begin{aligned}\forall i : 1 \leq i \leq \lceil \|m_{with\_header}\|/b \rceil : block_i \quad = \quad & m_{with\_header,(i-1)b+1} \circ \\ & m_{with\_header,(i-1)b+1} \circ \\ & \vdots \\ & m_{with\_header,(i-1)b+b}\end{aligned} \tag{22}$$

If the block size of the encryption algorithm is b and $\|m_{with\_header}\|$ modulo $b \neq 0$, then the last encryption block is augmented with 00_{H}s to fill the block.

The encrypted message $m_{encrypted}$ is the concatenation of the encrypted blocks

$$m_{encrypted} = E_x(block_1) \circ E_x(block_2) \circ \ldots \circ E_x(block_{\lceil \|m_{with\_header}\|/b \rceil}) \tag{23}$$

The key for the encryption algorithm is taken from the beginning of the key material (*keymaterial* byte string). That is, if the encryption algorithm requires that the key K is $\|K\|$ bytes long, then

$$\forall i : 1 \leq i \leq \|K\| : K_i = keymaterial_i. \tag{24}$$

If $\|keymaterial\| < \|K\|$, then the used key material generation function and the encryption algorithm are incompatible and cannot be used together.

Currently supported encryption algorithms are:

- *DES*.
- *IDEA*.

8 Array and Table Permutation Algorithms

These algorithms are needed in the hiding phase to select the hiding bits that are actually used to hide the message.

8.1 Pseudorandom Permutation

These algorithms are based on the algorithm presented in [1].

8.1.1 The 2-Dimensional Version

The following code calculates the ith permuted index of a $X \times Y$ sized 2-dimensional array, where $1 \leq X, Y \leq 2^{31} - 1$, $X \cdot Y \leq 2^{31} - 1$, $1 \leq x \leq X$, $1 \leq y \leq Y$ and $1 \leq i \leq X \cdot Y$:

function $PRP_{2D}(i, keymaterial, X, Y)$
  $y = (i-1) \text{ div } X;$
  $x = (i-1) \bmod X;$
  $y = (y + f_H(x)) \bmod Y;$
  $x = (x + f_H(y)) \bmod X;$
  $y = (y + f_H(x)) \bmod Y;$
  return $< x+1, y+1 >;$

where $f_H(z)$ is defined to be

$$f_H(z) = |hash_1 \circ hash_2 \circ hash_3 \circ hash_4| \ \& \ 7\mathrm{FFFFFFF_H} \tag{25}$$
$$hash = H_{SHA}(keymaterial \circ zbs) \tag{26}$$

where H_{SHA} is the SHA hash function and zbs is the byte string such that

$$|zbs| = z \tag{27}$$

$$\|zbs\| = 4 \tag{28}$$

8.1.2 The 1-Dimensional Version

The following code calculates the ith permuted index of a I sized 1-dimensional table, where $1 \leq X \leq 2^{31} - 1$, $1 \leq i \leq X$

function $PRP_{1D}(i, keymaterial, X)$
  x, y are integers in range $[1, 2^{31} - 1]$

  do {
    $x, y = PRP_{2D}(i, keymaterial, X', X')$;
  } while $((y - 1) \cdot X' + x > X)$;
  return $(y - 1) \cdot X' + x$;

where X' is the smallest integer such that

$$X' \cdot X' \geq X \tag{29}$$
$$X' = 2^k, \text{ for some integer } k \tag{30}$$

8.2 Simple Random Index

The efficiency (and security?) of these algorithms is not good. Do not use them.

8.2.1 The 1-Dimensional Version

This algorithm permutes 1-dimensional arrays.

The following code calculates the ith permuted index of a X sized 1-dimensional table, where $1 \leq X \leq 2^{31} - 1$, $1 \leq i \leq X$

function $SRI_{1D}(i, keymaterial, X)$
  $used$ is a set of integers in range $[1, X]$, initially $\emptyset$
  j is an integer $\in [0, 2^{31} - 1]$, initially 0
  $index$ is an integer $\in [0, 2^{31} - 1]$
  $pseudoIndex$ is an integer $\in [0, 2^{31} - 1]$, initially 0

  do {
    do {

$index = f_H(pseudoIndex) + 1;$
$pseudoIndex = pseudoIndex + 1;$
$\}$ while($(index > X) \vee (index \in used)$);
$used = used \cup index;$
$j = j + 1;$
$\}$ while($j < i$);
return $index$;

where $f_H(z)$ is defined to be

$$f_H(z) = |hash_1 \circ hash_2 \circ hash_3 \circ hash_4| \ \& \ 7\mathrm{FFFFFFF_H} \& mask \quad (31)$$
$$hash = H_{SHA}(keymaterial \circ zbs) \quad (32)$$

where H_{SHA} is the SHA hash function and zbs is the byte string such that

$$|zbs| = z \quad (33)$$
$$\|zbs\| = 4 \quad (34)$$

and $mask$ is the smallest integer such that

$$mask \geq X - 1 \quad (35)$$
$$mask = 2^i - 1, \text{ for some integer } i \quad (36)$$

8.2.2 The 2-Dimensional Version

This algorithm permutes 2-dimensional tables.

The following code calculates the ith permuted index of a $X \times Y$ sized 2-dimensional array, where $1 \leq X, Y \leq 2^{31} - 1$, $X \cdot Y \leq 2^{31} - 1$, $1 \leq x \leq X$, $1 \leq y \leq Y$ and $1 \leq i \leq X \cdot Y$:

function $SRI_{2D}(i, keymaterial, X, Y)$
$index$ is an integer $\in [1, 2^{31} - 1]$
x, y are integers $\in [0, 2^{31} - 1]$

$index = SRI_{1D}(i, keymaterial, X \cdot Y) - 1;$
$y = index$ div X;
$x = index$ mod X;
return $< x + 1, y + 1 >$;

9 Hiding the Encrypted Message into the Cover File

In the last phase of the hiding process, the encrypted message $m_{encrypted}$ is hidden into the cover file. This phase is cover file format depended, and is described in the following subsections.

9.1 The PGM Image Format

Let pgm be the 2-dimensional byte table as described in Section 3.1.

procedure $hide_{PGM}$
  i, j are integers
  n, m are integers

  for($i = 1$; $i \leq \|m_{encrypted}\|$; $i{+}{+}$) {
    for($j = 0$; $j \leq 7$; $j{+}{+}$) {
      $< n, m >= P((i-1) \cdot 8 + j + 1, keymaterial, N, M)$;
      if bit $7 - j$ of $m_{encrypted_i}$ is set then {
        set bit 0 of $pgm_{n,m}$; }
      else {
        clear bit 0 of $pgm_{n,m}$; }
    }
  }

where P is the used 2-dimensional permutation algorithm.

Currently supported permutation algorithms are

- PRP_{2D}
- SRI_{2D}

9.2 The JPEG Image Format

Let jpg and $jpgmod$ be sets of 6-tuples as described in Section 3.1.

procedure $hide_{PGM}$
  i, j are integers

$index$ is an integer
c, n, m, x, y, v are integers

for($i = 1$; $i \leq \|m_{encrypted}\|$; $i{+}{+}$) {
  for($j = 0$; $j \leq 7$; $j{+}{+}$) {
    $index = P((i-1) \cdot 8 + j + 1, keymaterial, \|jpgmod\|)$;
    $\langle c, n, m, x, y, v \rangle = jpgmod_{index}$;
    if $v \geq 0$ then
      $jpg_{c,n,m,x,y,v} = \langle c, n, m, x, y, ((v \& \mathrm{FE_H}) | (\text{value of bit } 7{-}j \text{ of } {m_{encrypted}}_i)) \rangle$;
    if $v < 0$ then
      $jpg_{c,n,m,x,y,v} = \langle c, n, m, x, y, -((-v \& \mathrm{FE_H}) | (\text{value of bit } 7{-}j \text{ of } {m_{encrypted}}_i)) \rangle$;
  }
}

where P is the used 1-dimensional permutation algorithm.

Currently supported permutation algorithms are

- PRP_{1D}
- SRI_{1D}

9.3 The GIF Image Format

Hack the code.

Currently supported permutation algorithms are

- foo

10 Extracting Hidden Messages from the Cover File

The extraction process is the inverse of the hiding process. The only difference is that if the used encryption and permutation algorithms are not known, all the combinations of them that are applicape to the used cover file format must be tested in order to find the hidden message.

References

[1] Tuomas Aura. Practical invisibility in digital communication, Workshop on Information Hiding, University of Cambridge, May 1995.

[2] Gregory K. Wallace. The JPEG Still Picture Compressin Standard, Communications of the ACM, April 1991.