Cicada 3301 - 2012 ”everywhere” images analysis

tweqx

December 2022

Abstract

TODO

1 Introduction

Back in 2012, during the first edition of the Cicada 3301 puzzle, solvers were given a list of coordinates after numerous puzzle steps. A QR code could be found in most locations, linking to an URL of the form:

`http://845145127.com/`*numbers*`.jpg`

where each QR code found had a different sequence ”*numbers*”.

Two different JPG images could be found at these URLs: they both consisted of an overexposed Cicada 3301 logo on a black background. The group name could be found beneath this logo, along with the omnious message ”everywhere” placed above.

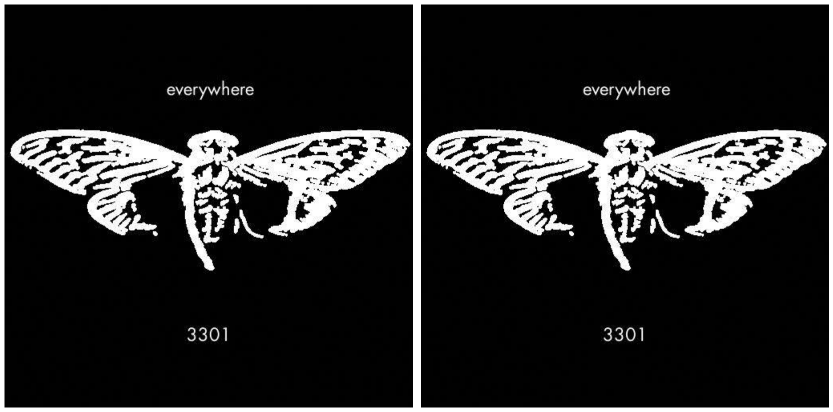

Figure 1: The two ”everywhere” images

While they looked exactly the same, a different PGP-signed message was steganographically hidden within each variant.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

A poem of fading death, named for a king
Meant to be read only once and vanish
Alas, it could not remain unseen.

1:5
152:24
the product of the first two primes
14:13
7:36
12:10
7:16
24:3
271:22
10:7
13:28
12:7
86:17
93:14
the product of the first two primes
16:7
96:4
19:13
47:2
71:22
75:9
77:4

You've shared too much to this point.  We want the best,
not the followers.  Thus, the first few there will receive
the prize.

Good luck.

3301
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.11 (GNU/Linux)

iQIcBAEBAgAGBQJPB/nmAAoJEBgfAeV6NQkPEnEQAKl5qtb3ZE5vs+c08KuzAi4a
tQEE71fvb65KQcX+PP5nHKGoLd0sQrZJw1c4VpMEgg9V27LSFQQ+3jSSyan7aIIg
SDqhmuAcliKwf5ELvHM3TQdyNb/OnL3R6UvavhfqdQwBXCDC9F0lwrPBu52MJqkA
ns93Q3zxec7kTrwKE6Gs3TDzjlu39YklwqzYcUSEusVzD07OVzhIEimsOVY+mW/C
X87vgXSlkQ69uN1XAZYp2ps8zl4LxoaBl5aVtIOA+T8ap439tTBToov19n0erusB
6VHS192m5NotfQLnuVT4EITfloTWYD6X7RfqspGt1ftb1q6Ub8Wt6qCIo6eqb9xm
q2uVzbRWu05b0izAXkHuqkHWV3vwuSfK7cZQryYA7pUnakhlpCHo3sjIkh1FPfDc
xRjWfnou7TevkmDqkfSxwHwP5IKo3r5KB87c7i0/tOPuQTqWRwCwcWOWMNOS7ivY
KQkoEYNmqD2Yz3Esymjt46M3rAuazxk/gGYUmgHImgcu1zzK7Aq/IozXI7EFdNdu
3EoRJ/UL9Y0l0/PJOG5urdeeTyE0b8bwgfC2Nk/c8ebaTkFbOnzXdAvKHB03KEeU
PtM6d6DngL/LnUPFhmSW7K0REMKv62h9KyP/sw5QHTNh7Pz+C63OO3BsFw+ZBdXL
hGqP6XptyZBsKvz2TLoX
=aXFt
-----END PGP SIGNATURE-----
```

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

In twenty-nine volumes, knowledge was once contained.
How many lines of the code remained when the Mabinogion paused?
Go that far in from the beginning and find my first name.

1:29
6:46
the product of the first two primes
2:37
14:41
17:3
27:40
the first prime
2:33
1:1
7:45
17:29
21:31
12:17
the product of the first two primes
22:42
15:18
24:33
27:46
12:29
25:66
7:47

You've shared too much to this point.  We want the best,
not the followers.  Thus, the first few there will receive
the prize.

Good luck.

3301
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.11 (GNU/Linux)

iQIcBAEBAgAGBQJPB1luAAoJEBgfAeV6NQkP9oAP+gLu+FsRDf3aRcJtBkCOU2MX
r/dagOTvCKWtuV+fedy0enWUZ+CbUjXOr98m9eq2z4iEGqKd3/MBXa+DM9f6YGUE
jPum4wHtQDSJlZMazuYqJOVZGw5XmF25+9mRM6fe3H9RCiNDZpuXl3MzwdivYhcG
B5hW14PcdHHteQf3eAUz+p+s06RDs+q1sNGa/rMQIx9QRe71EJwLMMkMfs81kfJC
tCt21+8ud0Xup4tjUBwul7QCcH9bqKG7cnR1XWsDgdFP6a4x9Jl2/IUvp1cfeT7B
YLS9W3lCM8thMemJr+ztQPZrpDlaLIitAT2L0B3f/k4co89v5X2I/toY8Z3Cdvoi
hk0AdWzMy/XLDgkPnpEef/aFmnls53mqqe9xKAUQPMrI73hiJ+5UZWuJdzCpvt+F
BjfQk15EJoUUW16K2+mBA1cSd+HJlnkslUTsjkq0E36XKChP+Cvbu/p6DLUMM2Xl
+n3iospCkkHR9QDcHzE4Rxg9A435yHqqJ/sL2MXG/CY8X4ec6U0/+UCIF9spuv8Y
7w66D05pI2u9M/081L7Br0i0Mpdf9fDbl0/6GksskccaPkMQ3MRtsL+p9o6Dnbir
6Z2wH2Kw1Bf0Gfx4VcpHBikoWJ5blCc6tfvT+qXjVOZjWAL7DvReavSEmW1/fubN
C3RWcjeI4QET2oKmV2NK
=LWeJ
-----END PGP SIGNATURE-----
```

Figure 2: The two PGP-signed messages found

The poems found in these messages hinted the book needed to complete the associated book code. For respectively each of the message above, they were "*Agrippa (A Book of the Dead)*" by William Gibson and the "*Encyclopædia Brittanica*".

When deciphered, each book code led to an Tor address, allowing for further progress.

Unfortunately, the two "Everywhere" variants weren't properly archived.

The furthest archive of the "Agrippa" variant can be found on the wiki, added by Lurker69 in 2014 : everywhere.

For many years, no copy of the "Brittanica" variant was known. One copy recently surfaced when optimisticninja's gitlab repository was made public[1]. No source that could justify where it originates from is provided. Furthermore, this

[1] `https://gitlab.com/optimisticninja/cicada3301/`, 2012/03.subreddit/04.qr_codes
Archive : `https://github.com/cicada-solvers/optimisticninja_cicada3301`

same repository is known to contain fake assets, such as a fake screenshot of the countdown found on `845145127.com`.

This analysis aims at investigating the two archives of the ”everywhere” to try to determine whether or not they are legitimate.

2 Goals and methodology

The upstream Outguess version in 2012 was Outguess v0.2. For this reason, we'll use its source code as the base of our analysis. Source code.

3 Inner workings of Outguess

3.1 General principles

To steganographically hide a message in an image, Outguess follows a serie of general steps. They have been highlighted in Outguess' `main` function, simplified below:

```
int
main(int argc, char **argv)
{
        // ...

        /* read command line arguments */
        while ((ch = getopt(argc, argv, "eErmftp:s:S:i:I:k:d:D:K:x:F:")) != -1)
                // ...
        }

        // `doretrieve` corresponds to the flag -r


        argc -= optind;
        argv += optind;

        // ...

        if (argc == 2) {
                srch = get_handler(argv[0]);
                if (srch == NULL) {
                        fprintf(stderr, "Unknown data type of %s\n", argv[0]);
                        exit (1);
                }
                if (!doretrieve) {
                        // expect the second argument to be the output image name

                        dsth = get_handler(argv[1]);
                        if (dsth == NULL) {
                                fprintf(stderr, "Unknown data type of %s\n",
                                argv[1]);
                                exit (1);
                        }
                }
                fin = fopen(argv[0], "rb");
                // ...
                fout = fopen(argv[1], "wb");
                // ...
        } else {
                // read/write from stdin/stdout
                // ...
        }

        /* Initialize Golay-Tables for 12->23 bit error correction */
        // ...

        fprintf(stderr, "Reading %s....\n", argv[0]);
        image = srch->read(fin);

        if (extractonly) {
                // Special mode, only available iif the outguess executable is named "extract"
                // Usage : extract image.jpg output_file.txt

                int bits;
                /* Wen extracting get the bitmap from the source handler */
                srch->get_bitmap(&bitmap, image, STEG_RETRIEVE);

                fprintf(stderr, "Writing %d bits\n", bitmap.bits);
                bits = htonl(bitmap.bits);
                fwrite(&bits, 1, sizeof(int), fout);
                fwrite(bitmap.bitmap, bitmap.bytes, sizeof(char), fout);
                exit (1);
        } else if (doretrieve)
                // -r

                /* Wen extracting get the bitmap from the source handler */
                srch->get_bitmap(&bitmap, image, STEG_RETRIEVE);
        else {
                /* Initialize destination data handler */
                dsth->init(param);
                /* When embedding the destination format determines the bits */
                dsth->get_bitmap(&bitmap, image, 0);
        }
        fprintf(stderr, "Extracting usable bits:   %d bits\n", bitmap.bits);

        // -e
        if (doerror)
                cfg1.flags |= STEG_ERROR;

        if (!doretrieve) {
                // hide message in image

                // -m
                if (mark)
                        cfg1.flags |= STEG_MARK;

                // -F ('on' by default)
                if (foil) {
                        dsth->preserve(&bitmap, -1);
                        if (bitmap.maxcorrect)
                                fprintf(stderr,
                                        "Correctable message size: %d bits, %0.2f%%\n",
                                        bitmap.maxcorrect,
                                        (float)100*bitmap.maxcorrect/bitmap.bits);
                }

                do_embed(&bitmap, data, key, strlen(key), &cfg1, &cumres);

                // outguess can embed up to two messages (a key has to be used for the second message though)
                if (key2 && data2) {
                        // ... embed the second file
                }

                if (foil) {
                        // ...

                        memset(steg_offset, 0, sizeof(steg_offset));
                        steg_foil = steg_foilfail = 0;

                        for (i = 0; i < bitmap.bits; i++) {
                                if (!TEST_BIT(plocked, i))
                                continue;

                                cbit = TEST_BIT(pbits, i) ? 1 : 0;

                                if (cbit == (data[i] & 0x01))
                                continue;

                                n = bitmap.preserve(&bitmap, i);
                                if (n > 0) {
                                        /* Actual modificaton */
                                        n = abs(n - i);
                                        if (n > MAX_SEEK)
                                        n = MAX_SEEK;

                                        steg_offset[n - 1]++;
                                }
                        }

                        /* Indicates that we are done with the image */
                        bitmap.preserve(&bitmap, bitmap.bits);

                        /* Calculate statistics */
                        // ... mean
                        // ... stddev

                        fprintf(stderr, "Foiling statistics: "
                                "corrections: %d, failed: %d, "
                                "offset: %f +- %f\n",
                                steg_foil, steg_foilfail,
                                mean, sqrt(dev / (count - 1)));
                }

                fprintf(stderr, "Total bits changed: %d (change %d + bias %d)\n",
                        cumres.changed + cumres.bias,
                        cumres.changed, cumres.bias);
```

```
                fprintf(stderr, "Storing bitmap into data...\n");

                dsth->put_bitmap (image, &bitmap, cfg1.flags);

                #ifdef FOURIER
                // -f
                if (dofourier)
                        fft_image(image->x, image->y, image->depth, image->img);
                #endif /* FOURIER */

                fprintf(stderr, "Writing %s....\n", argv[1]);
                dsth->write(fout, image);
        } else {
                /* Initialize random data stream */
                arc4_initkey(&as,  "Encryption", key, strlen(key));
                tas = as;

                iterator_init(&iter, &bitmap, key, strlen(key));

                encdata = steg_retrieve(&datalen, &bitmap, &iter, &as,
                cfg1.flags);

                data = decode_data(encdata, &datalen, &tas, cfg1.flags);
                free(encdata);

                fwrite(data, datalen, sizeof(u_char), fout);
                free(data);
        }

        // ...

        return 0;
}
```

To hide a message `data` in the image file `a` to the output file `b`, the steps are as follows:

- Detect the file format of `a` based on its file extension. As of Outguess v0.2, the following file formats are supported:
  - JPEG files : `.jpg`
  - PNM files : `.ppm`[2]

  Based on the detected file format, specialized function will be used. The `get_handler` is used to return a structure encapsulating the function pointers: `handler`. It is defined as (C++-style comments have been added for additional information):

  ```
  typedef struct _handler {
          // Used to detect the file format
          char *extension;        /* Extension name */
  ```

[2]Code exists to parse PNM files, but Outguess will not recognize the `.pnm` file extension. PNM stands for "Portable anymap", a superset of PPM files: "Portable pixmap". They will be noneless be always refered to as PNM files below.

```
        // Initialize the handler.
        // Will be called once for the handler used to output the file.
        // The parameter passed as argument can be set using -p
        //  (defaults to NULL)
        //
        // For the JPG handler, this parameter specifies the output
        //  quality. It's unused for the PNM handler.
        void (*init)(char *);
        // Parse the file handle and return an image object.
        image *(*read)(FILE *);
        // Write the image object to the file handle
        void (*write)(FILE *, image *);
        // Get the bitmap associated with the image.
        // The flag "STEG_RETRIEVE" will be passed as the last argument
        //  when the bitmap will be used to retrieve an embedded message.
        void (*get_bitmap)(bitmap *, image *, int);
        // Modify the image to match the changes made to its bitmap.
        // When -m is used, the flag STEG_MARK will be passed.
        void (*put_bitmap)(image *, bitmap *, int);
        // Callback to modify the file format handler that the bit at the
        //  offset given in the second parameter was flipped in the bitmap.
        // The purpose of this function is to preserve *some* statistics
        //  by modifying another part of the bitmap to cancel a given
        //  bitflip made.
        // It returns the index of the bit that was modified, if any, -1
        //  otherwise.
        //
        // Two offsets have a special meaning :
        //  - -1: the function is called once with this offset. It's used
        //      to signal to the statistics foiling logic to initialize.
        //
        //      The code is expected to configure the bitmap's `preserve`
        //       function pointer[3], and its `maxcorrect` value if possible.
        //  - any offset greater or equal to the number of bitmap bits:
        //     Used to signal to the statistics foiling code to finish
        //      processing as the image will soon be written to disk.
        int (*preserve)(bitmap *, int);
} handler;
```

The handlers for JPEG and PNM are respectively implemented in `jpg.c` and `pnm.c`.

Two important structures are used: `image` and `bitmap`. `image` is defined as (C++-style commented have been added for additional information)[4]:

```
typedef struct _image {
        // x: image width in pixels
        // y: image height in pixels
        // depth: number of color components
        // max: maximum value that a color component can be.
```

[3]The PNM handler's `preserve` is stubbed-out, meaning this file format does not support statistical foiling. However it does not initialize the `preserve` function pointer of the bitmap. This results will in a crash, meaning that to get Outguess to output an PNM file, statistical foiling had to be disabled.

[4]An additional unused `flags` field is also defined.

```
        //      (always 255 for JPEG)
        int x, y, depth, max;
        // pixels data. allocation of width * height * components bytes,
        //  filled with pixel colors components.
        u_char *img;

        // JPEG specific field: the bitmap of the image.
        // it is left uninitialized for PNM files.
        // this field exists only because of poor code design in my opinion,
        //  see the discussion below about STEG_RETRIEVE for more
        //  information
        bitmap *bitmap;
} image;
```

The `read` function pointer of the `handler` structure can be used to return an `image` structure from a file handle. `write` can be used to write an `image` structure to a file.

The `bitmap` structure represents the image data bits and is used during as an intermediary during the embedding process. It's defined as:

```
typedef struct _bitmap {
        // array where each element represents 8 bits of data.
        u_char *bitmap;   /* the bitmap */
        // array where each bit represents a boolean value indicating
        //  whether the corresponding  bit in `bitmap` is currently in
        //  use to store some message data.
        u_char *locked;   /* bits that may not be modified */
        // array where each bit represents a boolean value indicating
        //  whether the corresponding bit in `bitmap` is currently in
        //  use to foil statistics.
        u_char *metalock; /* bits that have been used for foil */
        // TODO
        char *detect;   /* relative detectability of changes */
        // array where the ith element corresponds to the original data byte
        //  from where the ith bit of the bitmap comes from.
        char *data;   /* data associated with the bit */
        // number of allocated bytes for the `bitmap`, `locked` and
        //  `metalock` fields. equals to (bits + 7) / 8
        int bytes;    /* allocated bytes */
        // number of bits stored in the bitmap.
        int bits;   /* number of bits in here */

        // function set by the file format handler. It is used exactly
        //  as the handler's `preserve` function.
        /* function to call for preserve stats */
        int (*preserve)(struct _bitmap *, int);
        // length of embdeddable message in bits. Stands for unlimited
        //  when 0.
        size_t maxcorrect;
} bitmap;
```

The handler function `get_bitmap` is used to return the `bitmap` structure associated with a passed `image` structure. Then, the bitmap will be modified to embed the data. Once modified, the image is to be modified back to match the bitmap changes through the `put_bitmap` function of the handler.

Both functions have an additional `flags` parameter to specify additional behaviours. The flags can be:

for `get_bitmap`: `STEG_RETRIEVE`: flag present when the bitmap will be used to retrieve some data, rather than for embedding a file.

This flag is unused for the PNM file format. For JPEG files, it is needed because the bitmap bits are based on DCT coefficients and not directly on pixels color components values. Thus, the bitmap to use when retrieving an embedded message is directly the bitmap encoded in the JPEG file. However, when producing a bitmap from an arbitrary `image` structure to be used to embed data, we don't have necessarily a bitmap of DCT coefficients to work with. Outguess solve this issue by always generating a bitmap from DCT coefficients by running the JPEG compression algorithm on the image pixel colors.

We wouldn't want to compress the image data into a bitmap when simply retrieving data from an JPEG image as this could destroy the embedded data.

for `put_bitmap`: `STEG_MARK`: flag present when the option `-p` is used. In theory this should mark visually pixels holding embedded data, however this feature is only implemented for the PNM format handler.

For this file format, it sets any pixel component having a bit changed to 255[5].

- Read the input file `a` into an `image` structure using its handler's `read` function.
- Initializes the output format handler by calling its `init` function.
- Get the output image bitmap, using image data of the input `a` we just generated. The `STEG_RETRIEVE` flag isn't used there.
- If statistical foiling is enabled (it is by default), the handler's statistical foiling system is initialized by calling its `preserve` function on the bitmap, passing in the offset -1.
- The first data file is embedded, with the user-supplied key (defaults to "`Default key`")
- If supplied, the second data file is embedded.
- Is statistical foiling is enabled, the handler `preserve` function will be called with the offset of all bits that have actually been flipped. To do so, Outguess loops over all bitmap bits, skipping not locked bits (since these

[5]regardless of the value of `max` – this is a bug. Furthermore, Outguess can produce PPM files having pixels whose value are greater than the maximum field of their header.

bits don't hold any information) and skipping bits left unchanged even though they are used to store information:

```
// statistics
memset(steg_offset, 0, sizeof(steg_offset));
steg_foil = steg_foilfail = 0;

for (i = 0; i < bitmap.bits; i++) {
  // skip bits that don't hold any message bits
  if (!TEST_BIT(plocked, i))
    continue;

  // skip unchanged bits
  cbit = TEST_BIT(pbits, i) ? 1 : 0;
  if (cbit == (data[i] & 0x01))
    continue;

  n = bitmap.preserve(&bitmap, i);

  // statistics
  if (n > 0) {
    /* Actual modificaton */
    n = abs(n - i);
    if (n > MAX_SEEK)
      n = MAX_SEEK;

    steg_offset[n - 1]++;
  }
}
```

The value returned by `preserve` in this case corresponds to the index of the bit that was changed to preserve the statistics, if any. If no bit was changed, a negative or null value is returned [6]. The `steg_offset` array is only used to display the mean/standard deviation of the distance of changed bits to foil statistics [7].

Finally, the statistical foiling logic is notified that the image finished being processed by calling the `preserve` function passing an offset greater or equal to the number of bits:

```
/* Indicates that we are done with the image */
bitmap.preserve(&bitmap, bitmap.bits);
```

During this entire process, the global variables are used to count the numbers of successful foils (`steg_foil`) and unsucessful foils (`steg_foilfail`). These variables are modified in handler-specific code. They are only used for informational purpose.

- The output `image` is synced with its bitmap. This takes into account all changes made to the bitmap.

[6] If the modified offset was 0, it wouldn't be counted in the foiling statistics. This is a bug.

[7] The final call to `preserve` will also try to modify some bits to preserve the distribution of DCT coefficients that failed to be preserved in previous calls. But these final modifications don't be taken into account in the displayed statistics. This is a bug.

- If the option `-f` has been compiled in Outguess (it's not by default) and if the user used it (it's not on by default), the image is outputed in frequency-space. The conversion is handled by the function `fft_visible` implemented in `fourier.c`.
- The image is written to disk.

3.2 JPG implementation

For JPEG files, the generation of the bitmap is quite complicated. Outguess uses a patched version of the JPEG 6b library. This patched version[8] hooks into the library internals and proxies reading/writing of the DCT coefficients.

The function that serves as the proxy can be found in `jpg.c`: `steg_use_bit`. It gets called whenever the JPEG library needs to store a DCT coefficient for a JPEG block. Changes to `jpeg-6b-steg/jcdctmgr.c`:

```
        // ...

        // replaced
        output_ptr[i] = (JCOEF) temp;
        // with
        output_ptr[i] = steg_use_bit(temp);

        // ...
```

And to `jpeg-6b-steg/jdcoefct.c`:

```
        // ...

        // added
        /* Retrieve LSB from DCT coefficient */
        JBLOCKROW block = coef->MCU_buffer[blkn + xindex];
        int k;
        for (k = 0; k < DCTSIZE2; k++)
                steg_use_bit((JCOEF) (*block)[k]);

        // ...
```

The `steg_use_bit` function is as follows (C++-style comments have been added for additional information, some unimportant lines have been omitted):

```
short
steg_use_bit (unsigned short temp)
{
        // only select coefficients where changing the LSB won't be too noticeable.
        // this skips DCT coefficients equal to 0 or 1
        if ((temp & 0x1) == temp)
                goto steg_end;

        switch (jpeg_state) {
                case JPEG_READING: // we're decoding a JPEG file
                        // fill the bitmap
```

[8]Note that the `.diff` file for the JPEG 6b library found in the Outguess source doesn't correspond to the patched source code.

```
                        WRITE_BIT(tbitmap.bitmap, off, temp & 0x1);
                        tbitmap.data[off] = temp;

                        // ...

                        off++;

                        if (off >= tbitmap.bits) {
                                // ... reallocate larger buffers
                        }
                        break;

                default: // JPEG_WRITING, we're writing a JPEG file out to disk
                        // overwrite LSB of the coefficient
                        temp = (temp & ~0x1) | (TEST_BIT(tbitmap.bitmap, off) ? 1 : 0);
                        off++;

                        break;
        }

  steg_end:
        // ...

        return temp;
}
```

3.3 Statistical foiling

Outguess version 0.2 introduced new methods that made it undetectable to statistical tests known when it was released[2]. These methods are known as ”statistical foiling” in Outguess' source code.

Most statistics to detect the use of steganography in JPEG images relied on the analysis of the distribution of JPEG DCT coefficients. As most methods to steganographically hide a message in a JPEG image do so in the least-significant bits (LSBs) of DCT coefficients, it can be assumed that the frequency of adjacent coefficients will be more similar than in an unmodified image. Thus, attacks analysed properties of the distribution of DCT coefficients across an image [3].

To make Outguess v0.2 undetectable using these methods, it now made sure to keep the distribution of DCT coefficients intact when embedding a message [2]. A new correction step would now run after the message was embedded. Its purpose was to modify some DCT coefficients that haven't yet been used to match the original DCT distribution.

3.3.1 Maximum message length

Because some bits would now be needed to correct the DCT distribution, computing the maximum embeddable message length isn't as straightforward [2][1]. Consider a JPEG image. Its DCT coefficients can range from -127 to 128. Let h_i be the number of DCT coefficients with the value i ($-127 \leq i \leq 128$). In other words, $(h_i)_{-127 \leq i \leq 128}$ is the histogram of the DCT coefficients. Let

$P = \sum_{i=-127}^{128} h_i - h_0 - h_1$ be the number of coefficients considered. We'll denote $f_i = \frac{h_i}{P}$ as the frequency of coefficient with the value i.

Consider a message of m bits. Embedding it in the image will modify the LSB of m DCT coefficients at random. However, the LSB of a given coefficient will sometimes already be the corresponding bit in the message m: on average, this will happen 1 times out of 2. That is to say, $\frac{m}{2}$ DCT coefficients will on average be actually modified after embedding the message.

Let $i \in \{-127, \ldots, 128\}$. If a coefficient with value i is modified, its new value will be $i \oplus 1$, that is to say $2k+1$ if $i = 2k$, $2k$ otherwise. After embedding the message, the updated frequencies for the values $2k$ and $2k+1$ are:

$$\begin{aligned} f^\star_{2k} = \frac{h^\star_{2k}}{P} &= \frac{h_{2k} - \frac{m}{2} h_{2k} + \frac{m}{2} h_{2k+1}}{P} \\ &= f_{2k} - \frac{1}{2}\frac{m}{P}\left(f_{2k} - f_{2k+1}\right) \end{aligned}$$

$$\begin{aligned} f^\star_{2k+1} = \frac{h^\star_{2k+1}}{P} &= \frac{h_{2k+1} - \frac{m}{2} h_{2k+1} + \frac{m}{2} h_{2k}}{P} \\ &= f_{2k+1} + \frac{1}{2}\frac{m}{P}\left(f_{2k} - f_{2k+1}\right) \end{aligned}$$

How many coefficients with the value i are holding data of the message? mf_i are (even though on average for half of these coefficients their LSB stayed the same, simply because they already matched the bits of the message). We will thus have $h_i - mf_i$ coefficients with the value i remaining for the correction step.

In the correction step, we want to use some of those bits to change the modified frequencies $f^\star$ back to their original values f. The (possibly negative) number of DCT coefficients with value $2k$ to purposely change to $2k+1$ is the average of the difference in distribution between values $2k$ and $2k+1$:

$$\begin{aligned} \delta_k &= \frac{(h_{2k} - h_{2k+1}) - (h^\star_{2k} - h^\star_{2k+1})}{2} \\ &= P \times \frac{(f_{2k} - f_{2k+1}) - (f^\star_{2k} - f^\star_{2k+1})}{2} \\ &= P \times \frac{1}{2}\frac{2m}{2P}\left(f_{2k} - f_{2k+1}\right) \\ &= P \times \frac{1}{2}\frac{m}{P}\left(f_{2k} - f_{2k+1}\right) \end{aligned}$$

Thus, at least $|\delta_k|$ bits are needed for the coefficient $2k$ if $f_{2k} > f_{2k+1}$, for the coefficient $2k+1$ otherwise. This results in the following inequality:

$$\forall k, |\delta_k| \leq \min(h_{2k} - mf_{2k}, h_{2k+1} - mf_{2k+1})$$

Or

$$P \times \frac{1}{2}\frac{m}{P}\left|f_{2k} - f_{2k+1}\right| \leq P \times \left(1 - \frac{m}{P}\right)\min(f_{2k}, f_{2k+1})$$

$$\frac{1}{2}m\,|f_{2k} - f_{2k+1}| \leq (P - m)\min(f_{2k}, f_{2k+1})$$

Finally, we can get an inegality on the message size m :

$$m \leq \frac{P \times \min(f_{2k}, f_{2k+1})}{\frac{1}{2}\,|f_{2k} - f_{2k+1}| + \min(f_{2k}, f_{2k+1})}$$

With $f_k = \max(f_{2k}, f_{2k+1})$ and $\overline{f}_k = \min(f_{2k}, f_{2k+1})$, this simplifies to :

$$m \leq 2P \times \frac{\overline{f}_k}{f_k + \overline{f}_k}$$

This relation should hold for all k different than 0 (as DCT coefficients 0 and 1 are ignored).

Code-wise, this computation happens in the first call of the JPEG handler's `preserve` function:

```
preserve_jpg(bitmap *bitmap, int off)
{
        char coeff;
        int i, a, b;
        char *data = bitmap->data;

        // Initialization call
        if (off == -1) {
                int res;

                // ...

                memset(dctfreq, 0, sizeof(dctfreq));

                // ...

                /* Calculate coefficent frequencies */
                for (i = 0; i < bitmap->bits; i++) {
                        dctfreq[data[i] + 127]++;
                }

                a = dctfreq[-1 + 127];
                b = dctfreq[-2 + 127];

                if (a < b) {
                        fprintf(stderr, "Can not calculate estimate\n");
                        res = -1;
                } else
                        res = 2*bitmap->bits*b/(a + b);

                // ...

                bitmap->maxcorrect = res;
                return (res);
        } else if (off >= bitmap->bits) {
                // ...

        //...
```

Interestingly enough, Outguess does not check that this relation holds for all pairs of DCT coefficients, but only for the pair $(-1, -2)$. Furthermore, it will exit when $h_{-2} > h_{-1}$ as it fails to "calculate an estimate". TODO: Why?

3.3.2 Outguess' statistical foiling algorithm

Furthermore, this piece of logic in `preserve_jpg` also initializes the statistical foiling logic (omitted in the previous code snippet – C++-style comments have been added to give additional information).

The JPEG statistical foiling implemented in Outguess is a variation of Algorithm 1 from Provos' [2].

The global variable `dctfreq` is an array of threshold frequencies for each coefficient value. From [2]: "The threshold indicates how many errors in the histogram we are willing to tolerate for a specific DCT coefficient. [...]. When the number of errors for a coefficient exceeds its threshold, we modify the image to preserve the statistics for that coefficient." In [2] it is named N^*.

The global variable `dctadjust` keeps track of the number of DCT coefficients that have to be changed for each coefficient value in order to keep the coefficient distribution intact. In [2], this array is named N_{error}. [9]

```
preserve_jpg(bitmap *bitmap, int off)
{
        char coeff;
        int i, a, b;
        char *data = bitmap->data;

        // Initialization call
        if (off == -1) {
                int res;

                // ... debug output

                // initialize the bitmap `preserve` function, as required
                bitmap->preserve = preserve_jpg;
                // so far, no bits have been used for statistics foiling.
                memset(bitmap->metalock, 0, bitmap->bytes);

                // no coefficients have to be adjusted for now
                memset(dctadjust, 0, sizeof(dctadjust));

                // threshold frequencies computation: step 1 of Algorithm 1
                memset(dctfreq, 0, sizeof(dctfreq));
                // ...

                /* Calculate coefficent frequencies */
                for (i = 0; i < bitmap->bits; i++) {
                        dctfreq[data[i] + 127]++;
                }

                // ...
```

[9] A third global variable `dctpending` keeps track of the total number of pending coefficient changes. However it stands unused.

```
        /* Pending threshold based on frequencies */
        for (i = 0; i < DCTENTRIES; i++) {
                // step 2. of Algorithm 1
                dctfreq[i] = dctfreq[i] /
                        ((float)bitmap->bits / DCTFREQRANGE);
                dctfreq[i] /= DCTFREQREDUCE;

                // differents from Algorithm 1
                if (dctfreq[i] < DCTFREQMIN)
                        dctfreq[i] = DCTFREQMIN;

                // ... debug output
        }

        // ...
        return (res);
} else if (off >= bitmap->bits) {
        // ...

//...
```

Note the presence of an additional step, where the threshold frequencies are capped at a minimum.

The logic ran when the image finishes processing is a direct implementation of Algorithm 1's last step [2]:

```
/* Reached end of image */
for (i = 0; i < DCTENTRIES; i++) {
        while (dctadjust[i]) {
                dctadjust[i]--;

                coeff = i - 127;

                if (preserve_single(bitmap, bitmap->bits - 1,
                                coeff) != -1)
                        steg_foil++;
                else
                        steg_foilfail++;
        }
}
```

The same can be said for the third case as to why `preserve_jpg` might be called: to indicate that a bit was modified.

```
// step 3
/* We need to find this coefficient, and change it to data[off] */
coeff = data[off] ^ 0x01;

// step 4
if (dctadjust[data[off] + 127]) {
        /* But we are still missing compensation for the opposite */
        dctadjust[data[off] + 127]--;
        // ...
        return (0);
}

// step 5
if (dctadjust[coeff + 127] < dctfreq[coeff + 127]) {
        dctadjust[coeff + 127]++;
        // ...
        return (0);
}

// call to exchDCT
i = preserve_single(bitmap, off, coeff);

if (i != -1) {
        steg_foil++;
        return (i);
}

/* We have one too many of this */
dctadjust[coeff + 127]++;
// ...

return (-1);
```

Algorithm 2, *exchDCT* is exactly implemented by the function `preserve_single`.

References

[1] Jessica Fridrich, Miroslav Goljan, and Dorin Hogea. Attacking the outguess. In *Proceedings of the ACM Workshop on Multimedia and Security*, volume 2002, 2002.

[2] Niels Provos. Defending against statistical steganalysis. In *10th USENIX Security Symposium (USENIX Security 01)*, 2001.

[3] Niels Provos and Peter Honeyman. Detecting steganographic content on the internet. Technical report, Center for Information Technology Integration, 2001.