# Exercise Assignments

**Work in Groups of 2 or 3 students!**

**Exercise 1** Analyse the system OpenPuff (`https://en.wikipedia.org/wiki/OpenPuff`).

1. Choose an image form the NASA image gallery (`https://www.nasa.gov/topics/`), e.g. `http://apod.nasa.gov/apod/ap150523.html`.
2. Embed different watermarks into some copies of this image (at least 5 copies).
3. Analyse the number of pixels that are modified by this embedding (use MatLab or Octave).
4. How many pixels are modified within one watermark work?
5. Reconstruct a new image from your copies of the different watermark works by determining pixel for pixel the majority/median/average values of the rgb-colours of the watermark works.
6. Did this process significantly reduces the difference of the reconstructed work from the original work?

**Exercise 2** Analyse the system OutGuess (`http://uncovering-cicada.wikia.com/wiki/OutGuess`).

1. Consider the example images given at the moodle platform of this course.
2. The images *StegoWork with OutGuess 01* to *StegoWork with OutGuess 05* are 5 copies of the same image with an added payload (3 different key are used).
3. One of the two images *Test OutGuess 01* or *Test OutGuess 02* is the original image whereas the other image is a further copy with an embedded payload. Decide which copy is the original image.
4. Apply different compression rates of the jpg-encoding (e.g. by using Octave and
   ```
   imwrite(image, filename.jpg, 'Quality', 100);,
   imwrite(image, filename.jpg, 'Quality', 85);,
   ```
   ...) and analyse the differences between *Test OutGuess 01* and *Test OutGuess 02* according to the different compression rates.

**Exercise 3** Test the influence of DCT coefficients on the quality of images:

1. Take the image *Image without digital watermark* from the moodle platform of this course.
2. Compute the DCT of a partition of the image into $8 \times 8$ blocks. To compute the DCT you can use any library for MatLab or Octave).
3. For each of the 4 quarters perform the following test:
   - Multiply the values of the quarters with some constants: 1, -1, 2, -2, ...
   - Compute the inverse DCT.
   - For which constant do you discover a significant observable change of the image.
4. For each of the 4 quarters perform the following second test:
   - Add some constant values to the values of the quarters: 50, -50, 100, -100, ...
   - Compute the inverse DCT.
   - For which constants do you discover a significant observable change of the image.

**Exercise 4** Apply a statistical test on modified images by the $\chi^2$-test.

1. For your tests use the png-image `Image for Collection 03` from our moodle page.
2. Simulate different stego-systems for embedding a message into the least significant bits of the pixel values of the cover image:
   (a) Since a secure stego-system has to encrypt the payload, you should generate your payload as a random bit sequence of 8000 bits (0's and 1's).
   (b) The first system should use the simple sequential selection rule (using the blue colour channel row by row). I.e. embedding should be done in every sequential pixel starting from the upper-left corner of the image.
   (c) The second system should simulate the pseudo random selection rule by using a sequence of uniformly chosen random positions from the blue colour channel.
   (d) The third system should use the pseudo random selection rule by using the seed `27183` at `rand('seed',27183)` and the positions generated by `unidrand(r)` and `unidrand(c)` where `r` and `c` denote the number of rows and columns of the image. Again, embed the random payload into the least significant bits of the blue colour channel. Ensure that no position is used multiple times.
   (e) Transform the png-image into jpeg-image and use OutGuess to embed a sequence of 1000 characters `0`.
3. Perform the $\chi^2$-test for correlated and uncorrelated value pairs ($(2i, 2i+1)$ and $(2i, 2i-1)$) for a partition of the image into windows of size $50 \times 50$ blocks of the blue colour channel.

Analyse now the histograms of images modified only by the system OutGuess.

1. Implement the jpg-transformation for YCbCr-images (i.e. implement a transformation for computing the quantized DCT coefficients (Lect. 4)).
2. Consider the example images given at the moodle platform of this course *Test OutGuess 01* and *Test OutGuess 02*. Compute the histograms of the quantized DCT coefficients of these images.
3. Perform the $\chi^2$-test for correlated and uncorrelated value pairs ($(2i, 2i+1)$ and $(2i, 2i-1)$) for a partition of the image into windows of size $40 \times 40$ blocks over the quantized DCT.
4. Can you determine the original cover work by your tests (*Test OutGuess 01* or *Test OutGuess 02*)?

**Exercise 5** Implementing your own embedding system:

1. First step in implementing you own embedding system:
   (a) Implement the PRNG of Blum, Blum, and Shup where the primes $p, q$ are
   $$p = 59999 \quad \text{and} \quad q = 60107$$
   - Consider that we have to deal with large numbers, thus you should declare integers as `uint64`.
   - To generate a random number (or a random bit-string) of $n \leq 64$ bits, implement a function $[\texttt{bits}, \texttt{xi}] = \texttt{getBits(}n\texttt{, xi, }M\texttt{)}$.

   (b) Determine a pseudo random bit sequence to represent a random walk through the image using you implementation the PRNG of Blum, Blum, and Shup where
   $$[\text{rand\_offset}, \text{next\_xi}] \quad = \quad \texttt{getBits}(64,\ \text{xi},\ 60091 \cdot 59971)$$
   where the initial value of xi is 20151208.

   (c) Generate some images by embedding random payloads at the LSBs of the blue color values of *Test OurGuess 01* along your generated random walk. Use random walks of different length.

   (d) Run the $\chi^2$-test for correlated value pairs at your modified images.

2. Now do some improvements of you embedding system:
   (a) For generating some payload sequences implement the Fibonacci LFSR from the lecture and use it to generate 100 input sequences of 15000 bits (by using different initial).

   (b) Count the number of required bit-flips for using several different seeds for your embedding algorithm of the first part of the exercise.

   (c) Run the $\chi^2$-test for correlated value pairs at the modified images with the lowest number of modifications.

**Exercise 6** Recall that in Exercise 5 you have used a PRNG to determine a pseudo-random walk through the blue color values of *Test OurGuess 01* to embedding random payloads at the LSBs. Now you should add to this embedding the advantages of matrix embedding:

1. Implement a function
$$\mathbf{e} \;=\; \texttt{cosetLeader}(\mathbf{m}, \mathbf{x})$$
that determines the changes within a given bit sequence $\mathbf{x}$ which should be used to embed the bit message sequence $\mathbf{m}$ such that the number of altered bits is as small as possible. For this function we assume that the number of bits of $\mathbf{m}$ match the number of rows of the predefined parity check matrix $H$ and that the number of bits of $\mathbf{x}$ match the number of columns of $H$.

2. Determine the minimum number of changes for matrix embedding if the following parity check matrix $H_1, H_2, H_3$, and $H_4$ are used:
$$H_1 \;=\; \begin{pmatrix} 0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\ 0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\ 0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\ 1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \end{pmatrix}$$
$$H_2 \;=\; \begin{pmatrix} 1&0&0&1&1&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&1&0&1&0&0&0&0&0&0&0&0&0 \\ 1&1&0&1&0&0&1&0&0&0&0&0&0&0&0 \\ 1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&1&1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&1&0&1&0&0&0&0&0 \\ 1&1&0&0&0&0&0&1&0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0 \\ 1&0&0&1&0&0&0&1&0&0&0&0&1&0&0 \\ 0&1&0&1&0&0&0&1&0&0&0&0&0&1&0 \\ 1&1&0&1&0&0&0&1&0&0&0&0&0&0&1 \end{pmatrix}$$
$$H_3 \;=\; \begin{pmatrix} 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\ 0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\ 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\ 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \end{pmatrix}$$
$$H_4 \;=\; \begin{pmatrix} 1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0 \\ 1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0 \\ 0&1&1&0&1&0&0&1&0&0&0&0&0&0&0&0 \\ 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0 \\ 0&1&1&0&0&0&0&0&1&0&0&1&0&0&0&0 \\ 1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0 \\ 0&1&0&0&1&0&0&0&1&0&0&0&0&1&0&0 \\ 0&0&1&0&1&0&0&0&1&0&0&0&0&0&1&0 \\ 1&1&1&0&1&0&0&0&1&0&0&0&0&0&0&1 \end{pmatrix}$$

3. Give some advantages and disadvantages of the four parity check matrices.

**Exercise 7** Simulate F5 on modified blue color values of *Test OurGuess 01*:

1. Use for the embedding a embedding path generated by you implementation of the PRNG of Blum, Blum, and Shup of Exercise 4.1.
2. Use for the payload a sequence of 4000 bits generated by your implementation of the Fibonacci LFSR.
3. Use matrix embedding by using $H_1$ of Exercise 5.1.
4. Is you have to embed an bit $b$ into the color value $x \in \{0, \ldots, 255\}$, perform the following 3 steps:
   (a) Normalize $b$, i.e. determine $x' = x - 128$.
   (b) Perform the F5 embedding step of $b$ in $x'$. Let $y'$ be the result.
   (c) Do the inverse of the first step, i.e. let the new color value of this pixel be $y = y' + 128$.
5. Run the $\chi^2$-test for *correlated value pairs* at the modified images and the unmodified image.

Discuss how you could apply calibration to attack our embedding system

**Exercise 8** Investigate the robustness of digital watermarks. Use the two systems

- OpenPuff, e.g. `https://www.filecroco.com/download-openpuff/`
- OpenStego, e.g. `https://sourceforge.net/projects/openstego/` or `https://www.heise.de/download/product/openstego-57363`

and the test image *Test OutGuess 01*. Generate at least 5 copies of *Test OutGuess 01* with different embedded messages. Run the following robustness tests for each of the watermarked images:

1. Rotation 90°, 2-times 90°, 3-times 90°, and 4-times 90°; store and read the image after each rotation by 90°.
2. Cut 10%, 20%,. . . 90% off from the image starting from the bottom line of the picture.
3. Perform different jpeg-compressions for storing the images to a file: Use the quality of $100\%, 90\%, \ldots, 10\%$.
4. Look for some image filters available for MatLab or Octave (or some other image editing tool) and apply this filters on your watermarked images (e.g. noise reduction, sharpening, change color depth, contrast change, gamma correction, . . . ).

Is it still possible to detect some embedded message after the transformations?