University of Surrey
Semester 2 2022/2023

Dr. Saša Radomirović

COMM046
Multimedia Security and Digital Forensics
Coursework Assignment 1: Multimedia Security

Deadlines and Important Information

Deadline for submission: 4pm Monday, 27th March 2023. Pay attention to late submission policies of the University: late submissions will lead to a 10% penalty per day and any submission after 4pm on Wednesday, 29th March 2023 will receive a 100% penalty (i.e, will not be assessed and a zero mark will be automatically awarded).
Feedback: this will be provided in line with the university regulation of three semester weeks and before the second piece of coursework.
ECs: If you are unable to complete the coursework due to extenuating circumstances please inform the module convenor as soon as possible and follow the processes for EC requests.
Academic Misconduct: This is an **individual coursework assignment**. You must not discuss your work on this assignment with anyone except the lecturer nor use or solicit help from others to complete this assignment. You must submit your own work. Coursework is routinely checked for plagiarism, collusion and other forms of academic misconduct.

General Guidelines

This coursework amounts to 60% of the COMM046 module while the remaining 40% will be from the second coursework. The coursework assessment will include the following elements (100 marks in total):

- Solutions in form of images, commented MATLAB scripts and code as specified under Technical Tasks below [**90 marks**]
- Adhering to specified file, folder and function naming requirements [**10 marks**]

All files should be submitted as a single zip file named as follows: `COMM046_CW1_YourUsername.zip`. **The zip file must be submitted electronically on SurreyLearn.** The zip file must include **all** files necessary for the markers to properly test your work. You do not need to include the utility functions that were released in lab sessions.
The assessment of the MATLAB scripts and functions will be based on

- readability,
- functionality and
- correctness.

Readability means (1) commenting on the purpose of the methods, matlab commands or MATLAB code blocks that you use, (2) on the insights gained from the results of your methods and commands, and (3) avoiding unnecessary or unrelated commands.
I am not looking for excessive documentation, but rather for brief concise explanations. An example of two useful code comments is given in the following MATLAB code.

```
% create sequence of the first 7 Fibonacci numbers
A=[0,1]; for i = 1:5; A(i+2)=A(i)+A(i+1); end

B = A(end:-1:1); % reverse the order of elements in A
```

The first comment describes what the following line achieves, the second comment describes what the command to its left achieves. The spacing between the two lines helps the reader to understand that the first comment does not apply to what follows after the empty line.
Make sure to use the percent `%` character (as shown in the example above) when adding comments to your MATLAB files and code or else you will have errors in your MATLAB file.
Functionality means that your script or function does something useful. The code runs without inadvertent MATLAB errors (e.g., has the necessary leading `%` character for comments) and it takes input and provides output of the types specified in the task. For example, if the task is to write a function that embeds a given text in an image, then your function takes a text and an image as input and produces an image as output.
Correctness means that your script or function is free from bugs. The code computes correctly and provides the correct output. For example, if the task is to embed text in an image, then the function produces an image in which the input text is embedded. It does not embed a shorter, longer or different text.
Correctness also means that the commands you are using your MATLAB work files are appropriate to analyse the images and that you are drawing the correct conclusions from them.

Programming Language, External Tools and Libraries

All source code must be written in MATLAB and will be tested in MATLAB 2022a as installed on the lab computers.
The assigned tasks can and are expected to be solved using the utility functions and standard MATLAB functions that you have been provided and taught in the labs. If you **use or adapt third-party code** then you **must acknowledge** the source in a comment at the bottom of the submitted file and you must provide a link to the source. Third-party code means that it is neither your own original work nor code that is built into MATLAB or made available to you on SurreyLearn. The use or adaptation of third-party code may reduce the number of marks awarded. Unacknowledged use of third-party sources is considered as academic misconduct.

Technical Tasks

You have an individual zip file, named after your student URN, on SurreyLearn that you must download for the following tasks. **Do not use or download other students' zip files.**

1 Digital Watermarking [30 marks]

For this task you should use the `LSB_watermark_embed` and `LSB_watermark_extract` functions included in the `Task_1` folder in your zip archive. These functions are simplified versions of the `LSB_steg_embed` and `LSB_steg_extract` functions that you have worked with in the labs. The simplification is that the watermarks to be embedded are `uint8` vectors, i.e., one dimensional 8-bit number arrays which includes character strings, but not images which are 2-D or 3-D matrices.

a) Use `LSB_watermark_embed` to embed the SHA-512 hash of the string '**2023**' as the watermark into the image `original.png` and **your URN as the key**. You can find the image in the `Task_1` folder. Save the watermarked image as `watermarked.png` in the `Sol_1` folder.

Submission: Name the matlab file listing the commands you used to create the watermark `task1a.m`. You do not need to include comments in this file.

Save `watermarked.png` and the matlab file `task1a.m` in the `Sol_1` folder.

Marks: 10

5 marks for a correctly watermarked image, 5 marks for a correct and executable `task1a.m` file.

b) You have received 10 images in the `Task_1` folder. These images have been watermarked by embedding a SHA-512 hash of an unknown message and with an unknown key using the function `LSB_watermark_embed` from Task 1(a) above. The same watermark and key were used for all 10 messages.

Your task is to erase the watermark (and only the watermark) by replacing it with zeros. Change as few pixel values as possible.

Submission: Save the 10 images with erased watermarks in the `Sol_1` folder. Append '`-zero`' to their filenames to distinguish them from the original images.

Name your matlab work used to analyse and produce the erased watermarks `task1b.m` and save it in the `Sol_1` folder. Make sure to add comments to the file that briefly explain the purpose of the commands you used. To keep the file executable, make sure to precede every comment with the percent `%` character.

Your work files must be valid MATLAB `.m` files (not Word docx, pdf, or any other format files).

Marks:

- 10 marks for the correctly zeroed watermarks.
- 10 marks for the readability and correctness of the `task1b.m` file.

2 Steganalysis [30 marks]

Analyse the 6 images `Picture_1.png` through `Picture_6.png` in the `Task_2` folder and provide the results of your analysis by filling in the form at https://forms.office.com/e/wfujVS4wBr and including it in your `Sol_2` folder.

Your analysis should aim at finding out which of the images are stego images, whether any embedded information is encrypted, and what the information is. See the form for the detailed questions.

Hints:

- The full capacity of the cover image has been used whenever another image was embedded. Clean, non-stego images are included in the `Task_2/clean_images` folder for reference.
- Stego images containing plaintext images were generated with `LSB_steg_embed(cover,plaintext,key)`. Encryptions were generated with `Encipher(plaintext,key,'AES')`.
- All the embedded plaintext images have the same structure. They all consist of three words arranged in the same way. Only the words differ.
- If keys are used in the creation of a stego image, then they consist of three lowercase 5-letter words and are separated by dashes. For example `house-mouse-youth` would be a valid key. The keys were randomly generated from a list of words in the file `keylist.txt` which is in your `Task_2` folder.
- If the production of one stego image required more than one key, then the same key was used for each step.

Marks:

- 10 marks for readability and correctness of the MATLAB work files.
- 20 marks for online form answers.

Submission: Submit your answers to these questions by filling in the form at https://forms.office.com/e/wfujVS4wBr. **Make sure to save the answers of this form** (you will be given an opportunity to save them) and place the saved pdf file in the `Sol_2` folder. You may resubmit the form, but only the last submission will be marked, so do **not** split your answers across several submissions. The same deadline applies to the form submission and the saved pdf of your last form submission must be included in your zip folder.

You must also show your MATLAB work that led to your answers. Name the file containing your MATLAB work `task2.m`. If you prefer to split the work across multiple files, name them in the order that they should be read, i.e., `task2_1.m`, `task2_2.m`, etc. If you develop MATLAB functions to

support your analysis, then make sure to include them in the `Sol_2` folder, too. You may name your supporting functions as you like.

Your work files must be valid MATLAB `.m` files (not Word docx, pdf, or any other format files).

3 LSB Steganography [30 marks]

For this problem you can use your own steganography functions from Lab 2, Task 2(c) or the `LSB_steg_embed.m` and `LSB_steg_extract.m` functions provided as a solution to Lab 2 Task 2(c) on SurreyLearn.

Extend the functions by implementing as many of the features (a) – (d) as possible. All embedding features must be implemented in the same embedding function and all extraction features must be implemented in the same extraction function. (I.e., do not submit more than one embedding function and do not submit more than one extraction function. Only one function of each will be marked.)

Your embedding function's matlab file **must be called** `LSB_embed.m` and the function's signature must be `s = LSB_embed(c, p, k)`, where `c` is the cover image, `p` is the plaintext, `k` is the stego key and `s` is the stego image.

Your extracting function's matlab file **must be called** `LSB_extract.m` and the function's signature must be `p = LSB_extract(s, k, p_size, p_class)`, where `s` is the stego image, `k` is the stego key, `p_size` is the size of the plaintext, and `p_class` is the class of the plaintext.

a) Extend the embedding function so that it embeds a plaintext in a true colour or gray scale cover image using the LSB embedding method **with the two** (instead of 1) **least significant bitplanes**

- using random paths based on the stego key supplied,
- using sequential embedding when no stego key is supplied,
- and return `[]` giving appropriate error messages using `disp` when unsuitable inputs are provided.

b) Extend the extraction function so that it extracts LSB embedded plaintext from the two least significant bitplanes from true colour or grayscale stego images

- along random paths based on the stego key supplied,
- from a sequential embedding when no stego key is supplied,
- and return `[]` giving appropriate error messages using `disp` when unsuitable inputs are provided.

c) Extend the embedding function so that the LSB embedding itself contains sufficient information for the receiver to correctly extract the plaintext even if the receiver does not know the size of the embedding. The embedding should work

- along random paths based on the stego key supplied,
- along a sequential embedding when no stego key is supplied,
- and return `[]` giving appropriate error messages using `disp` when unsuitable inputs are provided.

d) Extend the extraction function so that it recovers an embedded plaintext from a stegotext produced with your embedding function implementing feature **(c)** above. The function should do this when the size parameter is not provided and when it is provided but set to 0.

The extraction should work

- along random paths based on the stego key supplied,
- along a sequential embedding when no stego key is supplied,
- and return `[]` giving appropriate error messages using `disp` when unsuitable inputs are provided.

Submission: Place the two functions in the folder called `Sol_3`. You may create helper functions that are called from your embedding and/or extraction functions. If you do, make sure to also copy those functions into the `Sol_3` folder.

Marks: Functionality and correctness are assessed by testing the functions on a set of different input texts and images.

Each of the four features is awarded up to 6 marks for functionality and correctness.

Up to 6 marks are awarded for readability based on the following criteria:

- Code blocks and complex individual commands are explained with comments.
- The method used to embed/extract plaintext from 2 bit planes is well-documented with comments in the code.
- The method used to embed/extract the size of the plaintext is well-documented with comments in the code.

Coursework submission: The `Sol_1`, `Sol_2` and `Sol_3` folders must be submitted together as a single zip file named `COMM046_CW1_YourUsername.zip`. **The zip file must be submitted electronically on SurreyLearn.**