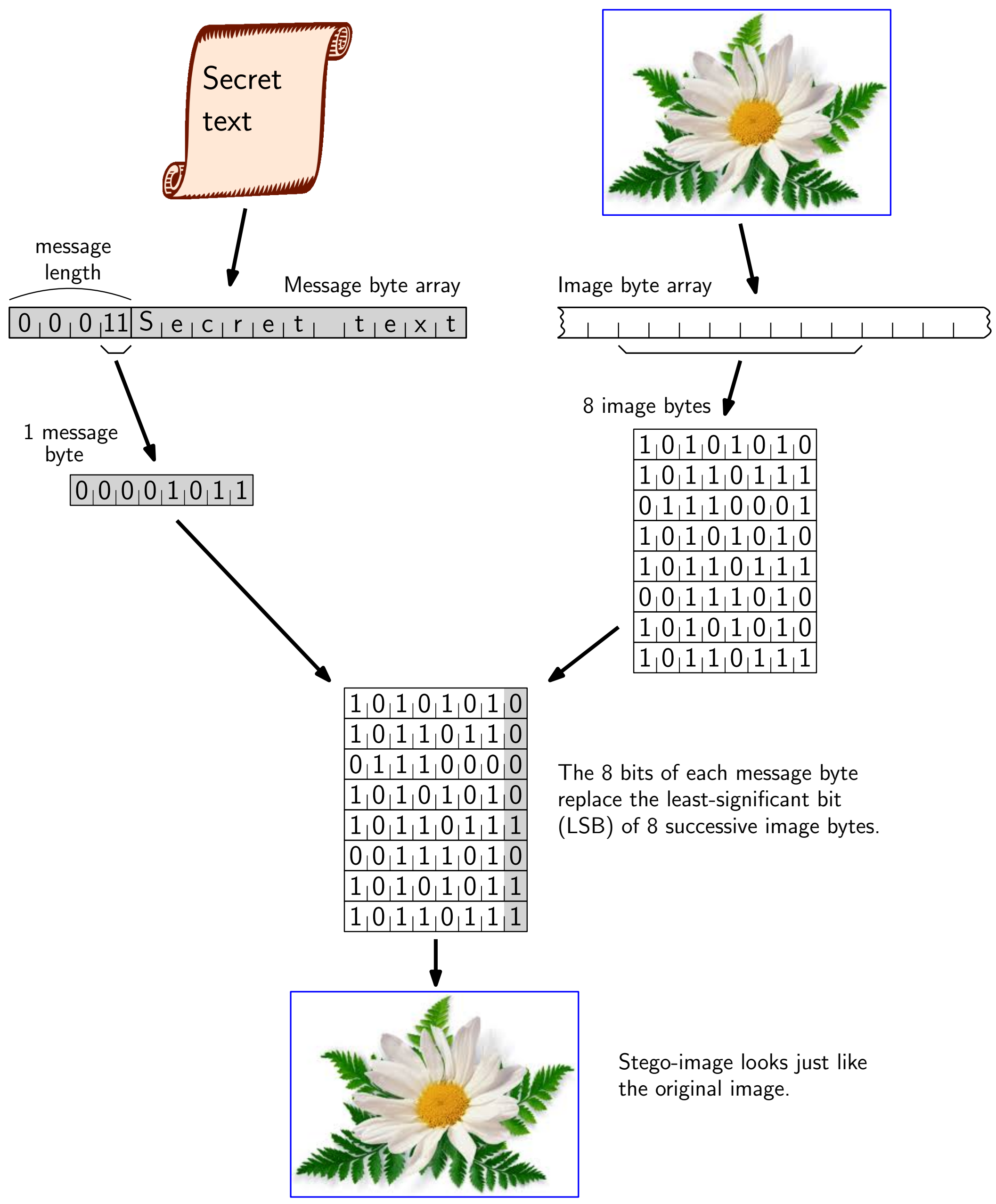
Practical 3 – Steganography

Steganography[1] is the technique of hiding a secret message in an innocuous-looking object such as an image or audio file [1]. Unlike cryptography, where observers are aware about the exchange of encrypted information, in steganography no-one can see the message being transmitted. This can allow people ruled by an oppressive regime to communicate undetected, and companies collaborating on new products to exchange data unknowingly to their competitors. Nefarious uses of the technique are widespread, e.g. there are suspicions that it was used by the terrorists involved in the 9/11 attacks [2]. Steganography is a type of covert channel.

In this practical you will experiment with a simple steganography method in which the secret message is copied, bit by bit, into the least-significant bit (LSB) of consecutive pixels from an image file. The colour differences between the original image and the resulting *stego-image* are imperceptible to the human eye (and the original image is destroyed to preclude such comparison). The main steps of LSB steganography are shown below.



[1]The term comes from the Greek words '*steganos*' (covered) and '*-grahia*' (writing).

Exercise 1

In this exercise you will develop message hiding and message extraction methods to complete the implementation of a LSB steganography tool.

Download all the Java classes attached to these practical notes on the module VLE. These files implement a simple GUI tool for LSB steganography, and are a slightly modified version of a proof-of-concept steganography tool taken from [3].

In Eclipse (or other Java application development environment of your choice) set up a new ‘Java Application’ project and add to it all the Java files for the practical. Your task is to complete the implementation of two Steganography.java methods:

1) `void insert_hidden_text(byte[] image, String text)` – This method hides the text received as its second parameter within the image (byte array) received as its first parameter.

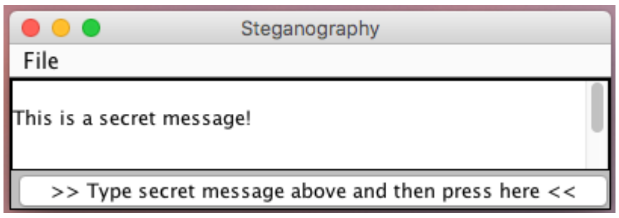
   As shown in the diagram from the previous page, the method must first create a “message byte array” containing the length of the text (in the first four bytes) followed by the actual text. This part of the implementation is done for you, but you should have a look at the implementation of the method `text2byteArray` to remind yourself (or to become familiar) with Java bit manipulation. Ask if you cannot understand how this method works.

   The second part of the method (left for you to implement) must iterate through each byte of the message byte array, and through every bit of this message byte, placing it in the least-significant bit of an image byte. Table 1 summarises several Java bit manipulation operators you may want to use.

2) `byte[] extract_hidden_text(byte[] image)` – This method returns the secret message extracted from a stego-image produced by `insert_hidden_text`.

   The method needs to first obtain the `length` of the hidden text, i.e. the four-byte integer whose 32 bits are “hidden” in the LSB of the first 32 bytes of the stego-image. Once this `length` is known, the method must iterate over every byte of the hidden text, extracting it from the LSB of eight consecutive image bytes.

Run the tool, type in your “secret” message, press the button underneath to insert it into a PNG or JPEG image of your choice (you must make sure that the image is not offensive), and then retrieve the message.



This is a secret message!

Table 1. Useful Java bitwise and shift operators

Operator	Example	Description
& (bitwise and)	a = b & 0x01	a’s bits will all be 0 except the LSB, which is taken from b
| (bitwise or)	a = b | 0x03	a’s bits will be identical to b’s except the two least-significant bits, which will both be 1
>>> (unsigned right shift)	a = b >>> 4	b’s are shifted 4 places to the right and the result is assigned to a; the first four bits are set to 0
<< (shift left)	a = b << 1	b’s bits are shifted left by one position and the result is assigned to a; the most significant bit is lost, and the least significant bit is 0

Exercise 2

Examine a stego-image produced by your tool to convince yourself that it is visually indistinguishable from the original image. (NB: Use a stego-image that contains a large "secret" text, e.g. a few thousand characters – otherwise, only a few pixels will be modified in the stego-image!)

Next, change your implementation from Exercise 1 so that the two least-significant bits from each image byte are used to hide the secret message. This doubles the size of a secret message that can be concealed inside an image file! Are the changes between your original image and the stego-image noticeable this time? Again, you should use a large secret text for this analysis.

What about using four LSBs of the eight bits of each image byte to "hide" the secret message – is the modified image easy to spot this time? Would the stego-image raise the suspicion of an observer who does not have access to the original image?

Exercise 3

Steganography is a well-known technique, and thus frequent exchanges of large stego-images (and other stego-objects) is likely to raise suspicion. To make sure that the "secret" text from your stego-images are not accessible to hackers who try to extract the message stored in the LSB of their pixels, use a stream one-time pad cipher to encrypt your text before inserting it in the image. You can use, for instance, the PDF file containing these practical notes as your unique encryption/decryption key.

If you do not remember how the one-time pad cipher works, look it up!

Further study

You can learn about more sophisticated steganography techniques and attacks of them (i.e. *steganalysis*) from [1,4,5].

References

[1] Niels Provos, and Peter Honeyman. Hide and seek: an introduction to steganography. IEEE Security & Privacy **1**(3):32-44. http://dl.acm.org/citation.cfm?id=859096.
[2] Tom Kellen. Hiding in Plain View: Could Steganography be a Terrorist Tool? SANS Institute White Paper, 2001. https://www.sans.org/reading-room/whitepapers/stenganography/hiding-plain-view-steganography-terrorist-tool-551.
[3] William Wilson, Dream.In.Code Java Tutorial on Steganography, May 2007. http://www.dreamincode.net/forums/topic/27950-steganography/.
[4] Abbas Cheddad, Joan Condell, Kevin Curran, and Paul Mc Kevitt. Digital image steganography: Survey and analysis of current methods. Signal Processing 90(3):727-752, 2010. http://www.sciencedirect.com/science/article/pii/S0165168409003648.
[5] Jessica Fridrich, Miroslav Goljan, and Rui Du. Detecting LSB steganography in color, and gray-scale images. IEEE MultiMedia 8(4):22-28, 2001. http://ieeexplore.ieee.org/abstract/document/959097/.