Secure and Dependable Systems
Jacobs University Bremen
Dushan Terzikj

Course: CO21-320203
May 16, 2019

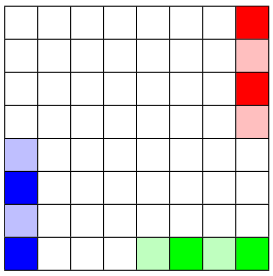
# Problem Sheet 6

## Problem 6.1

**Solution:**

(a) Please check *hide.c.*

(b) Please check the function *im_sign*, *im_verify* in the file *hide.c*. The algorithm is the following:



Let the above figure be an image. The colored pixels are so called ”signature zone”. If an image contains hidden text then the image will be ”signed”. The image is signed in the following way:

(1) The red-ish pixels are signed by modifying the red color of the pixel. The dark red pixels are modified such that the 2 LSB of the red value is set to 11, while the light red pixels are modified such that the 2 LSB of the red value is set to 00.
For example, if the red pixels in the figure have red values of 10010101, 10010101, 10010101, 10010101 after ”signing” they will have values of 10010111, 10010100, 10010111, 10010100.

(2) Same as (1), except this time the **green** value is modified.

(3) Same as (1), except this time the **blue** value is modified.

This is fairly reliable, since the edge pixels are usually the same color, or a very similar color. By changing the 2 last significant bits and making a pattern of twelve chunks of 2 bits (11, 00, 11, 00...) makes it ”probabilisticly” unique.

To wrap up, when a text is to be hidden in an image, the image is first signed (with *im_sign*) and then the text is encoded.
When the image is to be decoded, first it is verified (using *im_verify*) to check whether there is a hidden text. If there is, then it is decoded, if not the algorithm stops.

(c) An efficient way to minimize visual impact is to use the 2 LSB of the RGB values of the pixel. It is sufficient enough to store $r \cdot c \cdot 6$ bits of data in total, without noticeable changes for the human eye ($r$ and $c$ are rows and columns of pixels). The algorithms works in the following way:

(1) Take 2 bits from the message that needs to be hidden and put those 2 bits in the 2 LSB in the red value of the pixel.

(2) Take the next 2 bits from the message that needs to be hidden and put those 2 bits in the 2 LSB in the green value of the pixel.

(3) Take 2 bits from the message that needs to be hidden and put those 2 bits in the 2 LSB in the blue value of the pixel.

(4) Go to the next pixel and repeat step 1.

(5) If there are no more bits left, then make the next 4 values (whether red, green or blue) have 2 LSB of 00. Since 4 chunks of 00 makes 00000000 which in ASCII is the NUL termination character, we know that we have come to the end of our message.

Example: if we want to encode ”h” and the first 2 pixels have RGB values (in order) 10011000, 10011000, 10011000, 10011000, 10011000, 10011000. ”h” has binary value of 01101000. Divide 01101000 into chunks: 01—10—10—00 and append those chunks in order of the RGB values of the pixels. The pixels will now have RGB values of 10011001, 10011010, 10011010, 10011000, 10011000, 10011000.

This strategy is already implemented in the source code.