

Steganography Review

Binary numbers

We usually use base 10 (decimal numbers), which has 10 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. In base 10, the rightmost digit represents the 1s, the next digit to the left represents the 10s, the next digit to the left represents the 100s, and so on:

—	—	—	—	—
$\times 10^4$	$\times 10^3$	$\times 10^2$	$\times 10^1$	$\times 10^0$
$\times 10000$	$\times 1000$	$\times 100$	$\times 10$	$\times 1$

For example, the decimal number 237 is represented by $2 \times 100 + 3 \times 10 + 7 \times 1$, or $2 \times 10^2 + 3 \times 10^1 + 7 \times 10^0$:

$$\begin{aligned} & \qquad \qquad \qquad 2 \qquad \qquad 3 \qquad \qquad 7 \\ & = \qquad \qquad 2 \times 10^2 + \qquad 3 \times 10^1 + \qquad 7 \times 10^0 \\ & = \qquad \qquad 2 \times 100 + \qquad 3 \times 10 + \qquad 7 \times 1 \end{aligned}$$

Binary numbers are numbers represented in base 2. Base 2 has just two digits: 0, 1. Base 2 works the same way as base 10, but instead of representing a power of 10, each position represents a power of 2:

—	—	—	—	—	—	—	—
$\times 2^7$	$\times 2^6$	$\times 2^5$	$\times 2^4$	$\times 2^3$	$\times 2^2$	$\times 2^1$	$\times 2^0$
$\times 128$	$\times 64$	$\times 32$	$\times 16$	$\times 8$	$\times 4$	$\times 2$	$\times 1$

So the decimal number 237 is written in binary as:

$$\begin{aligned} & \qquad \qquad \qquad 1 \qquad \qquad 1 \qquad \qquad 1 \qquad \qquad 0 \qquad \qquad 1 \qquad \qquad 1 \qquad \qquad 0 \qquad \qquad 1 \\ & = \qquad \qquad 1 \times 2^7 + \qquad 1 \times 2^6 + \qquad 1 \times 2^5 + \qquad 0 \times 2^4 + \qquad 1 \times 2^3 + \qquad 1 \times 2^2 + \qquad 0 \times 2^1 + \qquad 1 \times 2^0 \\ & = \qquad \qquad 1 \times 128 + \qquad 1 \times 64 + \qquad 1 \times 32 + \qquad 0 \times 16 + \qquad 1 \times 8 + \qquad 1 \times 4 + \qquad 0 \times 2 + \qquad 1 \times 1 \\ & = \qquad \qquad 128 + \qquad 64 + \qquad 32 + \qquad 0 + \qquad 8 + \qquad 4 + \qquad 0 + \qquad 1 \end{aligned}$$

Everything in a computer is represented by a binary number—remember, everything is a number!

This is all you will need to know about binary numbers for this course, but if you would like more explanation or want to learn more, you can look at these resources:

- <https://www.mathsisfun.com/binary-number-system.html>
- <http://www.codeconquest.com/tutorials/binary/>
- <https://www.khanacademy.org/math/pre-algebra/applying-math-reasoning-topic/alternate-number-bases/v/number-systems-introduction>

Binary Numbers in Pixels

As you have learned, each pixel is represented as 3 components: a red value, a green value, and a blue value. Each of these can take a value from 0 to 255, inclusive—but this is 255 represented in base 10. Remember, to the computer, these are binary numbers. What is 255 represented in binary?

1

1

1

1

1

1

1

1

=

1x2⁷

+

1x2⁶

+

1x2⁵

+

1x2⁴

+

1x2³

+

1x2²

+

1x2¹

+

1x2⁰

=

128

+

64

+

32

+

16

+

8

+

4

+

1

+

1

=

255

So when a pixel has the RGB values (255, 255, 255), to the computer that's (1111 1111, 1111 1111, 1111 1111). Here are some more examples showing pixels' RGB values in binary:

Decimal RGB values	Binary RGB values	Color	Color name
0,0,0	00000000,00000000,00000000	<div></div>	black
0,255,0	00000000,11111111,00000000	<div></div>	green
0,139,139	00000000,10001011,10001011	<div></div>	dark cyan

As you can see, each red, green, and blue value has 8 binary digits, which we call 'bits'. This is why each value can be between 0 and 255: the smallest number represented in 8 binary digits is 0, and the largest is 1111 1111, which is 255 in base 10.

Hiding Information in Pixels

So what does this mean for hiding one image inside another? You have seen in the videos on **Hiding Data in Images: Steganography** that you can hide one image inside another without affecting the original image much, and that to do this you clear the pixel values of the start and hide images. Now we will look in more detail at why this works.

You know that each pixel's red, green, and blue values have 8 binary digits. In our start image, for each pixel, for each of its red, green, and blue values, we want to use 4 of these digits to hide information about another image in. (Note that we could use more or fewer than 4 digits to hide the second image in. We will talk about this more later in the course. For now, we will just use 4 digits to hide the image.) Which 4 digits should we use? We are going to put information about the hidden image in these digits, so we will lose the data about the start image in those digits. Which 4 digits are least important? Let's look at an example. Say we have a pixel with the following red value:

1

1

1

1

0

0

0

0

=

1x2⁷

+

1x2⁶

+

1x2⁵

+

1x2⁴

+

0x2³

+

0x2²

+

0x2¹

+

0x2⁰

=

128

+

64

+

32

+

16

+

0

+

0

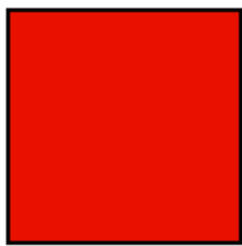
+

0

+

0

This is the number 240 in decimal. A pixel with a red value of 240 looks like this:

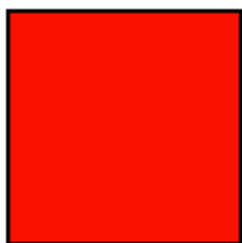


If we change the leftmost digit, a 1, to a 0, we are subtracting 128 (the binary digit is now 0111 0000). The new red value is 112 and looks like this:



That's a big difference, and we have only changed one digit! Imagine if we changed all 4 of the leftmost digits to end up with 0000 0000. Then we would have changed the original value by 240 ($128+64+32+16$) and the colour would be black. If we did this to every pixel in our image, it will change our original image significantly.

Now let's imagine we change the rightmost 4 digits, to end up with 1111 1111, or 255 in decimal. We have changed the original value by 15 and it looks like this:



Not too different from the original:



The point of this example is to show that the leftmost 4 digits are the most important, or most significant. If you change them, you could change the value by up to 240. The rightmost 4 digits are the least important. If you change them, the most you could change the value by is 15. We want to leave the leftmost digits and use the rightmost 4 digits to hide our other image in. This way we won't change our original image too much.

Which 4 digits of our hide image do we want to use? We only have 4 digits available in the start image, so we have to choose 4 digits from the hide image. We want to use the most important ones, so we use the leftmost 4. We need to store the leftmost 4 digits of our hide image inside the rightmost 4 digits of our start image. This is where the pixel clearing and shifting you saw in the video becomes necessary.

Pixel Clearing

As you saw in the lecture video, let's look at an example of clearing space in pixels using base 10 first, then think about doing the same thing using binary. Let's imagine we have a value of 4781 in the start image combining with 5236 in the hide image to give the result 4752 in the final image.

To clear the rightmost 2 digits of a 4-digit decimal number, you divide and multiply by 100, or 10^2 :

4 7 8 1 /100 =47 x100 = 4700

= $4 \times 10^3 + 7 \times 10^2 + 8 \times 10^1 + 1 \times 10^0$

10^2 10^2

clear these two digits

Let's try to apply the same logic to an 8-digit binary number. We can't use the value 4781 since the maximum value of an 8-digit binary number is 255 (1111 1111). Let's use the binary number 1010 1010, or 170 in decimal.

10101010

= 1x2⁷ + 0x2⁶ + 1x2⁵ + 0x2⁴ + 1x2³ + 0x2² + 1x2¹ + 0x2⁰

clear these four digits

It looks like the analogous value to divide and multiply by is 2^4, or 16. This makes sense: to clear 2 digits in a base 10 number, we divided and multiplied by 10^2; to clear 4 digits in a base 2 number we divide and multiply by 2^4.

This means that if we divide the binary number 1010 1010 by the decimal number 16 we should get the binary number 1010 0000 (the left 4 digits are the same as in our original number, the right 4 digits are all 0s). Let's check this by hand by doing the math in base 10. The binary number 1010 1010 is 170 in base 10. If we divide and then multiply 170 by 16 and then convert the answer to binary, we should get 1010 0000.

170 / 16 = 10

10 x 16 = 160

160 in binary:

10100000

= 1x2⁷ + 0x2⁶ + 1x2⁵ + 0x2⁴ + 0x2³ + 0x2² + 0x2¹ + 0x2⁰

= 128 + 0 + 32 + 0 + 0 + 0 + 0 + 0

160 in binary is 1010 0000, which is what we expected.


You can also show that it works using code. We have written and run some JavaScript code to demonstrate this example. Below is the code and the output. You might find it helpful to run and play with the following code to help convince yourself that it works! Try doing some examples by hand as we did above and then running this code with those numbers.

Code It

```
1 var image = new SimpleImage(1,1); //creates an image of one black pixel
2 var pix = image.getPixel(0,0); //gets the only pixel in the image
3 pix.setRed(170); //sets its red value to 170
4 print (image) //see that the image is now 1 red pixel
5 var r = pix.getRed();
6 print (r); //prints out the red value of pix, see that it is 170
7 r = Math.floor(r/16)*16 //divide and multiply by 16
8 print (r) //see that it is now 160
```

Run Code

See It



170
160

It is important to remember that although the computer is storing binary numbers, you are writing decimal numbers in your code and see decimal numbers printed out. This is why you divide and multiply by the decimal number 16 and not the binary number 10000 (if you wrote 10000 instead of 16, the program would assume you mean the decimal number ten thousand, rather than the binary representation of the number sixteen). You can actually do this assignment without knowing anything about binary numbers as long as you know that the number to divide and multiply by is 16. However, if you want to check that the pixel values are being transformed and combined correctly, it will be helpful to understand binary numbers. It is also helpful to understand binary numbers if later you want to change the number of bits to hide the image in.

Now we have seen why you divide and multiply by 16 to clear the rightmost 4 digits in an 8-digit binary number RGB value. This is what you will do to pixels in the start image. What about pixels in the hide image? For those we want to move the leftmost 4 digits to the rightmost 4 digits and make the leftmost 4 digits 0s. You saw that with a 4-digit decimal number you can divide by 100, or 10^2:

5236 / 100 = 52

This means that with an 8-digit binary number, you can divide by 16 (2^4). So if we divide the binary number 1010 1111 by 16 we should end up with the binary number 0000 1010.

Let's check this by hand by doing the math in base 10. The binary number 1010 1111 is 175 in base 10. If we divide 175 by 16 and then convert the answer to binary, we should get 0000 1010.

175 / 16 = 10

10 in binary:

0

0

0

0

1

0

1

0

=

0x2⁷ +

0x2⁶ +

0x2⁵ +

0x2⁴ +

1x2³ +

0x2² +

1x2¹ +

0x2⁰

=

0 +

0 +

0 +

0 +

8 +

0 +

2 +

0

Hiding Images

We have walked through how to modify one of the RGB values of one pixel in a start or hide image. How do you hide a whole image in another image? This is explained in the lecture video Steganography: Part III. Here is the code you are shown near the end of the video:

```
1 var start = new SimpleImage("usain.jpg");
2 var hide = new SimpleImage("skyline.jpg");
3
4 start = chop2hide(start);
5 hide = shift(hide);
6 var stego = combine(start,hide);
7 print(stego);
```

The basic outline of the algorithm is:

- For each pixel in the start image, replace each of its red, green, and blue values with the cleared pixel value (keep the 4 leftmost digits the same, replace the 4 rightmost with 0s). This is what the line **start = chop2hide(start)** does. The code for the **chop2hide()** function is shown in the video.
- For each pixel in the hide image, replace each of its red, green, and blue values by moving the 4 leftmost digits to the right and replacing the 4 leftmost digits with 0s. This is what the line **hide = shift(hide)** does.
- Calculate the pixel RGB values in the final image by adding the values of the pixels in the modified start and modified hide images. This is what the line **var stego = combine(start,hide)** does.

标记为完成

