# **A Steganography Algorithm**

As we’ve learned in *Blown to Bits, the* software could be used to embed secret messages within images in such a way that the changes to the images cannot be detected with the naked eye. The reason is that modern image representation schemes, which use 24 bits per pixel, are capable of representing 224 different colors. That is 16,777,216 different colors, far more than the eye can distinguish. But how, exactly, is it done? How can a text message be hidden in an image? In this lesson, you'll learn how and get to try it yourself.

## **Bitmap Images**

A **bitmap** is an image representation scheme, similar to JPEG and PNG, but older and simpler. A bitmap image uses 3 bytes (24 bits) of data for each pixel in the image. In the RGB color scheme, that would be 1 byte to represent red, 1 byte for green and 1 byte for blue.

Suppose we are looking at one of those pixels with an RGB color value of (175,0,21). If we change this into binary we would get 10101111 00000000 00010101. This text is colored in (175,0,21). Now suppose we flipped the **rightmost bit** of each of these binary bytes from 0 to 1 or 1 to 0. That would give us 1010111**0** 0000000**1** 0001010**0** or (174,1,20). This text is colored in (174,1,20). As you can see, those slight changes in the bits do not change the color of that pixel in any noticeable way — the change is **imperceptible.** That's how steganography works.

## **Hiding the letter 'C'**

Now let's suppose we want to hide the letter 'C' in an image. Letters are represented using an 8-bit scheme known as the **ASCII code** (American Standard Code for Information Interchange). If you look up C's ASCII representation you'll see that it is 67 in decimal, which is 01000011 in binary.

To hide the letter 'C' in an image, we will place each of its 8 bits as the **rightmost bit of 8 consecutive bytes of image data**. In general, *even* decimal values, such as 0, 8 and 254, end with a '0' bit. *Odd* decimal values, such as 1, 5 and 255, end with a '1' bit.

**Encoding Algorithm**: If the image byte is an odd number and you need to hide a 0 bit in it, you will *subtract* 1 from its decimal value. Else, if the byte is an even number and you need to hide a 1 bit in it, you will *add* 1 to its decimal value. In all other cases, don’t change the bit.

Suppose an 8-byte chunk of our bitmap consists of the following values, represented in decimal. We will put the 8-bit ASCII code for 'C' under the bytes.

Image Bytes: 175 0 21 255 178 134 26 255  
Letter 'C' Bits: 0 1 0 0 0 0 1 1

If we follow the rule provided, we would want to change the bytes as follows, with the changed bytes shown in **bold**.

Image Bytes: 175 0 21 255 178 134 26 255  
Letter 'C' Bits: 0 1 0 0 0 0 1 1  
Altered Bytes: **174 1 20 254** 178 134 **27 255**

Changing the bytes in this way makes no perceptible difference in the color of these bytes. However, if you were to convert the altered bytes — **174 1 20 254 178 134 27 255** — such that each even byte equals '0' and each odd byte equals '1', you would derive **0 1 0 0 0 0 1 1**, which is C's ASCII code. Thus, you can see that the letter 'C' had been hidden in the image's pixel data.

## **Steganography Algorithm**

To summarize, our **algorithm** for hiding a message in an image is as follows:

**To Hide a Message in an Image:**  
Translate the letters in the message into their binary ASCII codes.  
Pick a block of image data, represented as bytes  
Start at the first byte in the image data.  
For each bit in the message,  
Alter the byte to represent the bit, by making it even or odd or leaving it as is.  
Go to the next byte

The algorithm for revealing the hidden message, is just the opposite.

**To Reveal the Message hidden in an Image:**

Start at the first byte in the image data.  
For each byte in the data block  
Record 0 for even bytes and 1 for odd bytes and add it to the message's bits. Translate the message's bits back to letters using in the ASCII code.

## **Let's Try It Ourselves**

The [following video](https://www.youtube.com/watch?v=BPN1phkNDnE) provides a demo of how we can use an editor to hide our initials in a bitmap image.

[](https://www.youtube.com/watch?v=BPN1phkNDnE)  
(Click the image of the video above to play it on YouTube.com)

## **Your Turn**

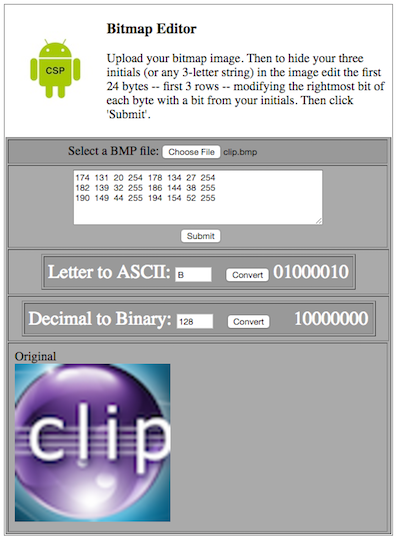
Ok. It's your turn to try your hand at digital steganography. You will use the [Bitmap Editor](https://appinventor.trincoll.edu/csp/webapps/stego/bitmap-editor.html) to embed your initials — or any 3-letter word, such as CAT or DOG — in a bitmap image. Here's what you need to do.

### **1. Find or create a bitmap image.**

A **bitmap** is an image file format. Bitmap files typically have the **.bmp** file extension. If you don't have a bitmap image, you can take an image in just about any other format and [use this online tool](http://image.online-convert.com/convert-to-bmp) to convert to a bitmap. You can also search online using Google’s [Advanced Image Search tool](https://www.google.com/advanced_image_search) to limit results to BMP files (select BMP files under the file type drop-down list). Note: other file types sometimes show up in the results (e.g. JPG) so make sure as you save the image, it is a bitmap one.

### **2. Upload your bitmap image.**

Click the Choose File button on the [**Bitmap Editor** app](https://appinventor.trincoll.edu/csp/webapps/stego/bitmap-editor.html) to upload your bitmap. After you upload the image, the web app will display 24 bytes of image data in its editor.

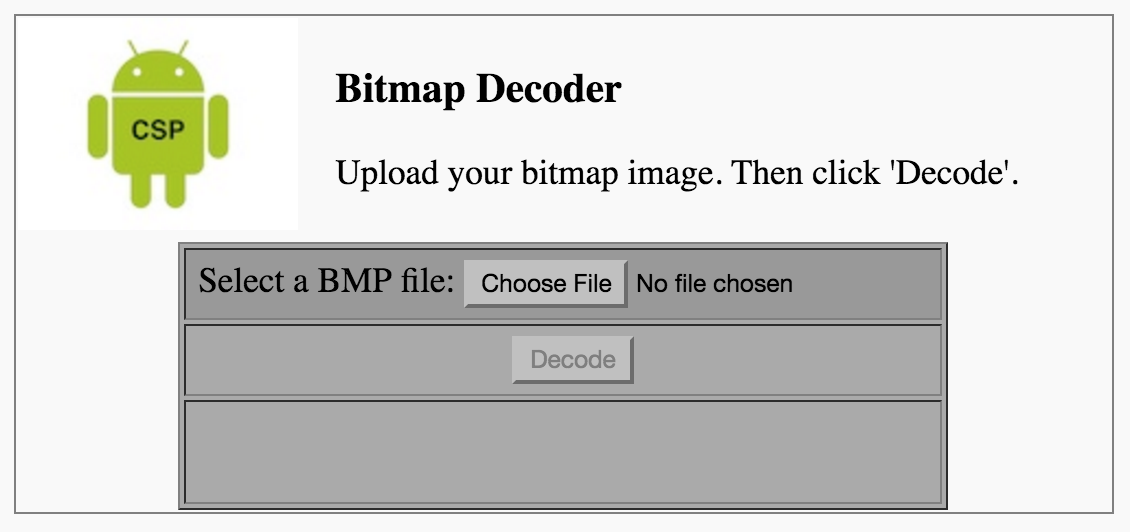
.[](http://appinventor.cs.trincoll.edu/csp/webapps/stego/bitmap-editor.html)

### **3. Edit the Bytes.**

Edit the 24 bytes to embed your 3-letter initials or word, following the algorithm described in the previous lesson. You can use the app's **helper** widgets to translate letters into their ASCII code and decimal numbers into their binary codes. Click the **Submit** button when you are done editing the data bytes. This will display the altered image.

### **4. Downlo**

To test whether you have successfully encoded your initials into the image, upload the altered image to the [Bitmap Decoder](https://appinventor.trincoll.edu/csp/webapps/stego/bmpdecoder.html) app and decode. If all went well the app should reveal the initials you hid in the image.

[](https://appinventor.trincoll.edu/csp/webapps/stego/bmpdecoder.html)

### Good work. Now you know how *steganography* works!