Steganography

The art of hiding information in plain sight.

Knowledge is a process of piling up facts; wisdom lies in their simplification.

Martin H Fischer

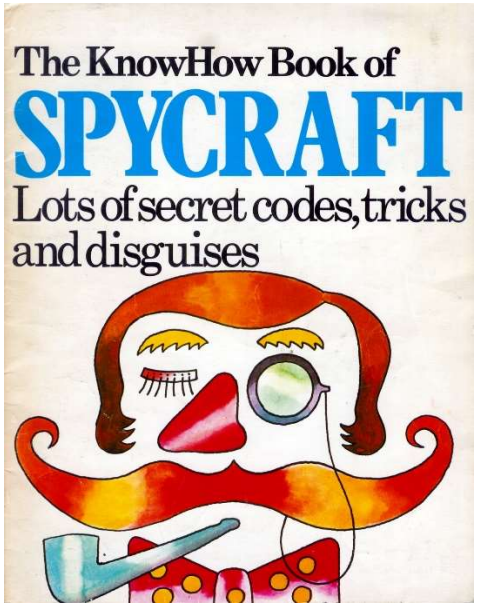
Modification Record

Issue	Date	Author	Changes (Including the change authority)
A	07 June 2021	ZizWiz	Original from various notes
B	10 June 2021	ZizWiz	Updated with help files for the Apps

Contents

1. Introduction

Years ago, as a child I found a book in a local bookshop sale that fascinated me. It was all about codes, tricks, and disguises. This book started me off on a long journey in how to encrypt and hide things in plain view. I still have the original book and it was later joined by an update that also had a CD with it.



I never knew there was a name for this disguising of items till much later in life. This document will describe just a very small part of my foray into Steganography - The art of hiding information in plain sight. My slant on this is to hide a passwords etc in plain site while making it hard for anyone to work it out. It is never impossible to work it out just need time and resources.

Here I will explain some code that I put together. I had read a few articles and seen some code samples. Here I am joining some of those sample ideas to form two apps. One to encode and one to decode.

Steganography is often used to hide copyright information in an image, movie, or audio file. If you pirate the file, then the owner can prove you did so by decoding it and pulling out their copyright information.

However, this is not fool proof e.g., the image you pirate is a PNG file. You open it inside a graphics program and change its dimensions very slightly and resave and the information needed to decode it may not be available anymore. You could also just resave the original image in a lossy format like JPG, and this will use compression to save the image which again may remove the information you need to decode your copyright information.

2. Citations

I read articles on this subject from the following Rod Stephens, Carlos Delgado and Hamzeh Soboh as well as many websites including Wikipedia and Stackoverflow. I give thanks to all

those that have taken the time to write and put information into the public domain. I in turn do the same so others can learn from it.

As a result of my learning here I have taken ideas from the above and various other sources and come up with some apps that allowed me to understand how this all works. I only created the apps as part of my own study and if you use them, you do so at your own risk. I do not guarantee them.

In the app where we hide images inside others, I took the code written by Rod Stephens and modified it to my own needs.

3. Important notes

- Below we will save images but please note when you save the combined image save it in a lossless format like BMP or PNG. If you save it as a JPG, the pixel values get modified to save space and even a tiny change in the pixels can completely destroy the steganography.
- Once saved do not alter the combined image in anyway as it may destroy the info needed to decode the information.

4. Hiding an image inside image

We start with hiding one image inside another. The images are the same size with one being totally black and the other totally white. Later we will look at complex photos. This method will hide information about the second picture in the least-significant colour bits of the first image. The magic here is that small changes to the least significant bit of a colour will not be noticeable by eye. Even a machine will have trouble spotting this. Note however that there will be some patterns that you could use that are more difficult than others to hide.

In an 8-bit binary number changing just the least significant bit will only contribute only one 256th possible colour changes. This means that there is only one colour shade change possible out of the many that could be possible. Probably this will not be seen with the naked eye.

We could change all 4 LSB bits and see what happens, there are many ways to do this but here we will show just one simple method.

We take a Black image that we will hide the white image inside which has values for Red, Green and Blue of 0. As an 8-bit binary this is 00000000

The White image has values of 255 or in 8-bit binary that is 11111111.

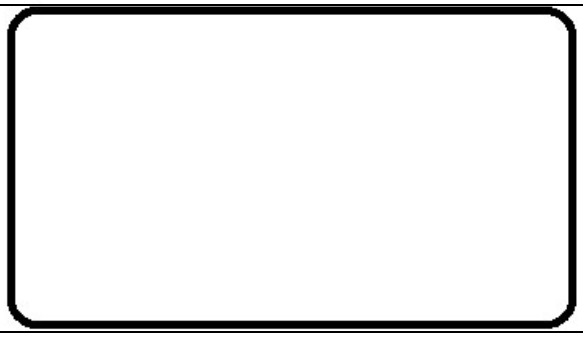
Now we take the 4 most significant bits from the black image 0000 and we concatenate them with the 4 least significant bits in the white image 1111

The result is we get a combined value of 00001111 = 15.

Now to recover the value we will split the value 00001111 into two 4-bit numbers. 0000 and 1111. We now take the hidden set which is 1111 and we concatenate this with 4 0bits in its

least significant area which results in 11110000 which is value 240 and remember we started out with 255 so we are very close to that colour. It will be a very slightly darker shade of white.

We see these colours below note the black border is only there to show where the colour is. The colour is inside the border.

	Colour Swatch	RGB value
Original image		0, 0, 0
Image to hide		255, 255, 255
Combined image		15, 15. 15
Recovered image		240, 240, 240

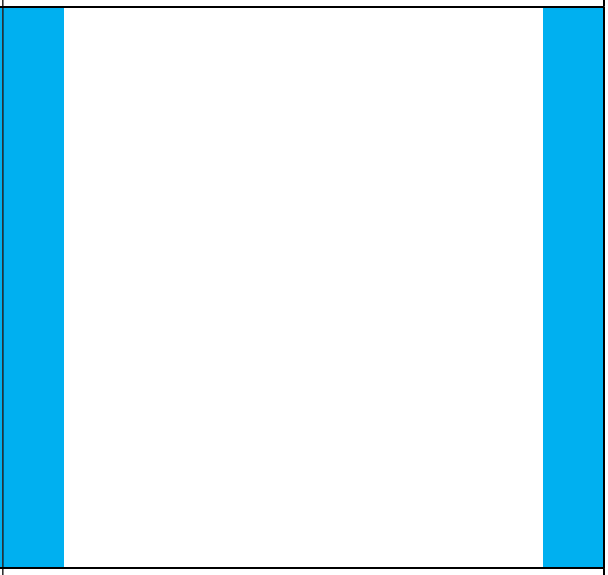
Above we used just two images of the same size and we can see the results.

If we look at an image saved as 24-bit BMP, that has a size 400x800pixels we see we have a total of 320 000 pixels and we know each pixel is 24 bit, 8 bits (1 byte) for each colour RGB, so there are 960 000 bytes or places where we can store info.

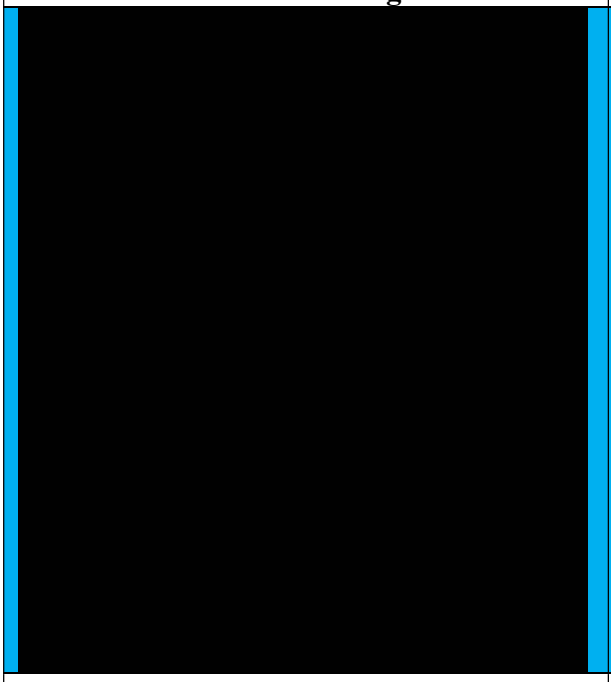
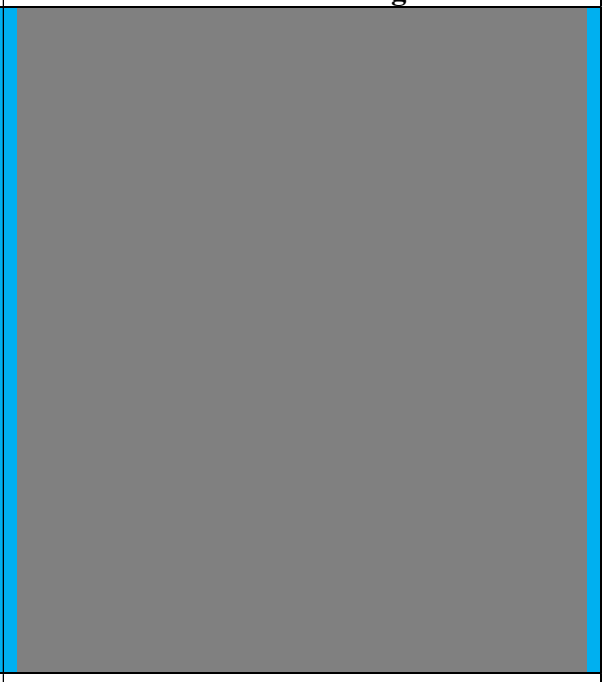
The size of image we can hide inside another depends on how many bits of the main picture we will use. The following experiment will show what happens when various numbers of bits are used.

4.1. Hiding White image inside Black image

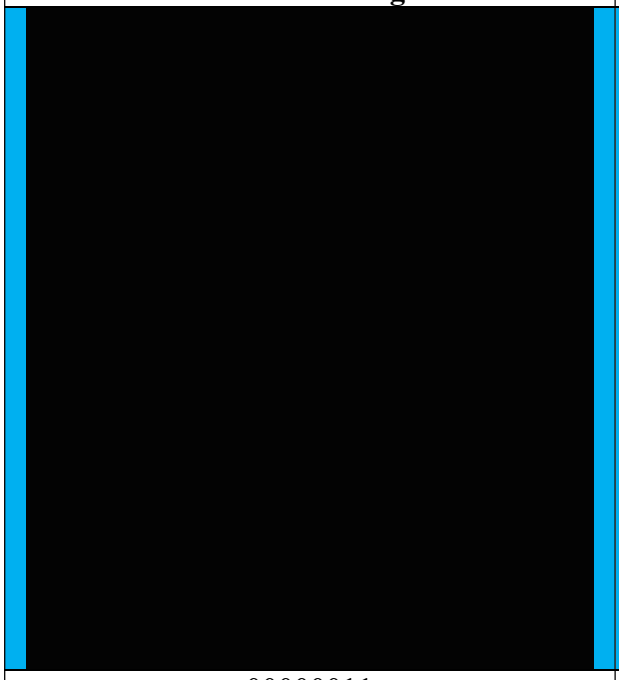
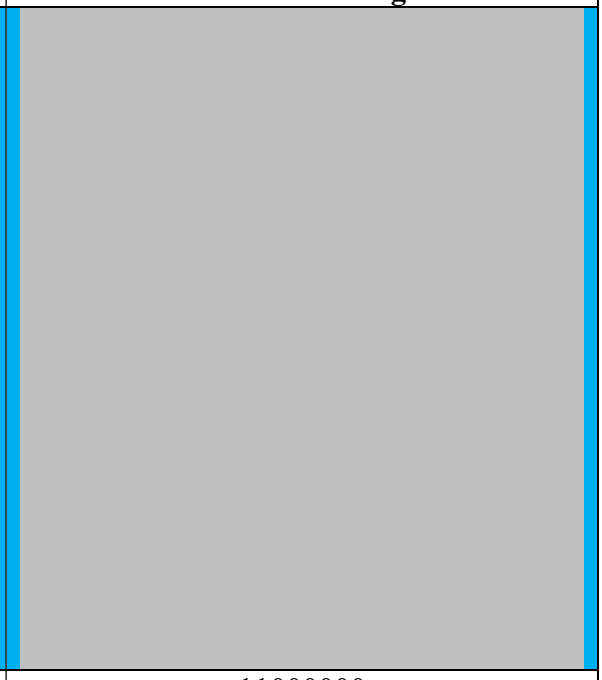
To show what happens in a worst-case scenario we take two images. One will be totally black and the other totally white both are the same size so same number of pixels. We will hide the White image inside the Black image and then recover it out of the combined image. We will use various "Bit hiding strengths" to hide and retrieve it so we can see the difference. As one it totally white we will add it inside a table which has a light blue background so we can see it.

Original Picture	Picture to be hidden
	
00000000	11111111
 Color|Solid Hue: 160 Sat: 0 Lum: 0 Red: 0 Green: 0 Blue: 0	 Color|Solid Hue: 160 Sat: 0 Lum: 240 Red: 255 Green: 255 Blue: 255

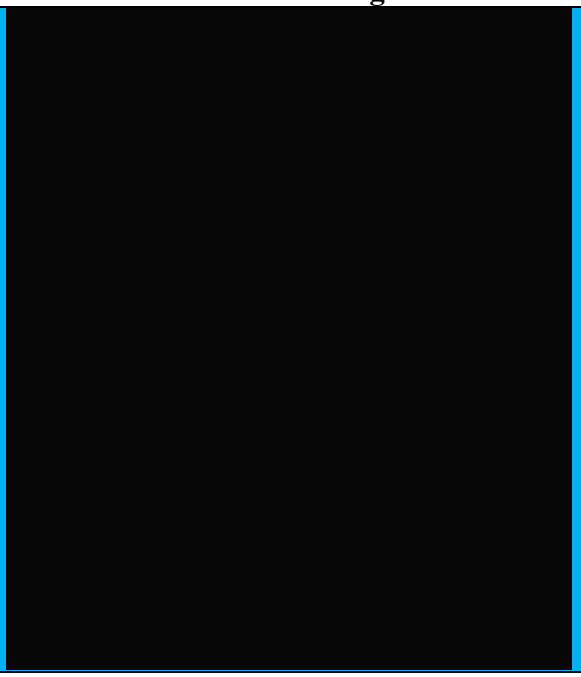
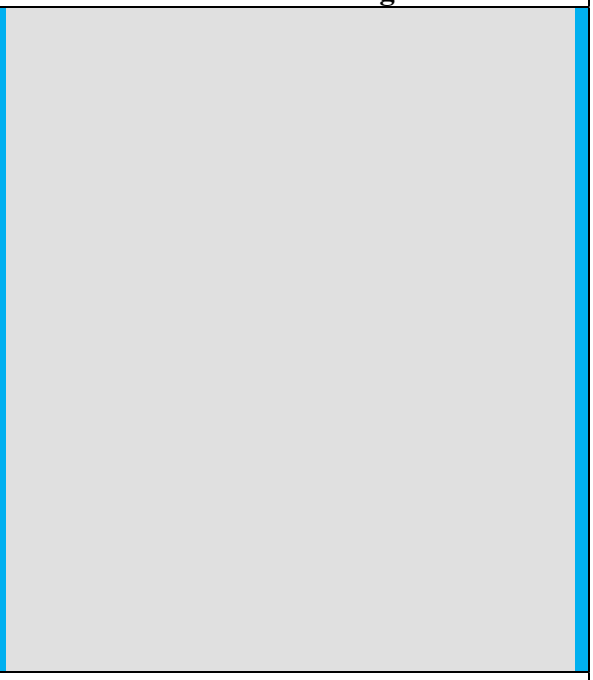
4.1.1. Using 1 Bit

Combined images	Recovered Image
	
00000001	10000000
 Color|Solid Hue: 160 Sat: 0 Lum: 1 Red: 1 Green: 1 Blue: 1	 Color|Solid Hue: 160 Sat: 0 Lum: 120 Red: 128 Green: 128 Blue: 128

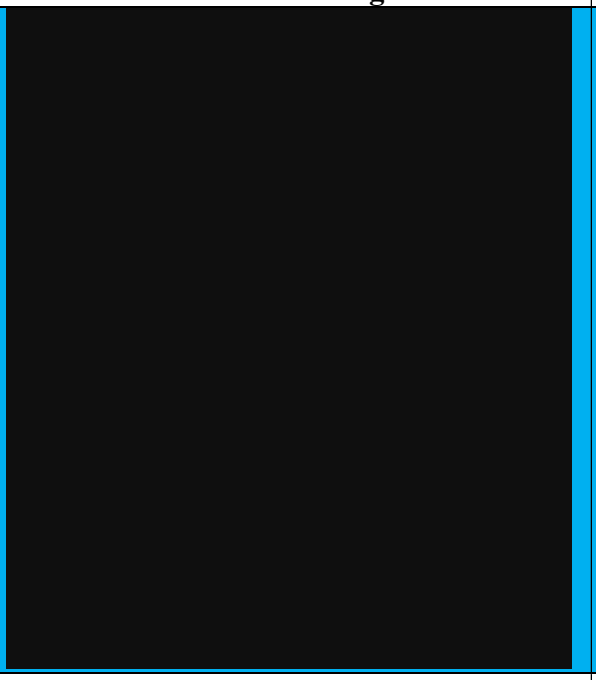
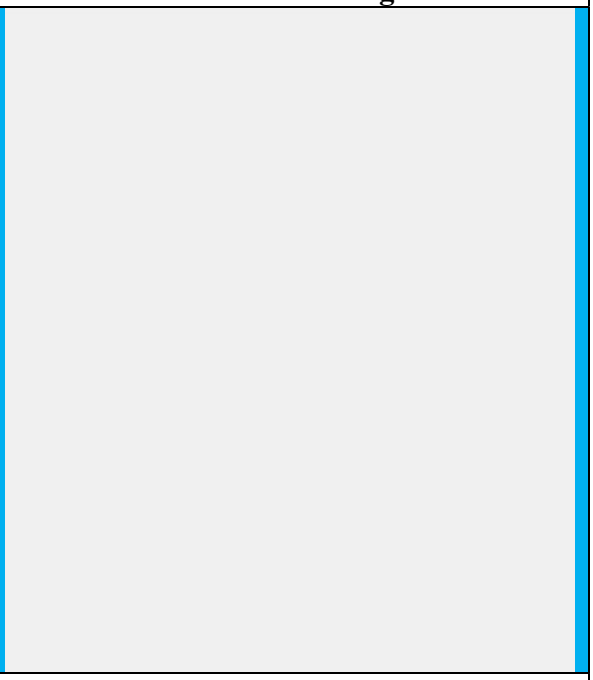
4.1.2. Using 2 Bits

Combined images	Recovered Image
	
00000011	11000000
 Color|Solid Hue: 160 Sat: 0 Lum: 3 Red: 3 Green: 3 Blue: 3	 Color|Solid Hue: 160 Sat: 0 Lum: 181 Red: 192 Green: 192 Blue: 192

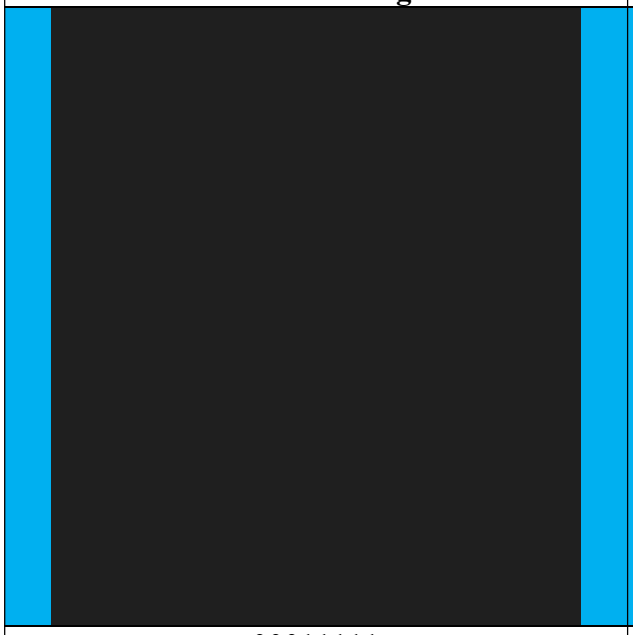
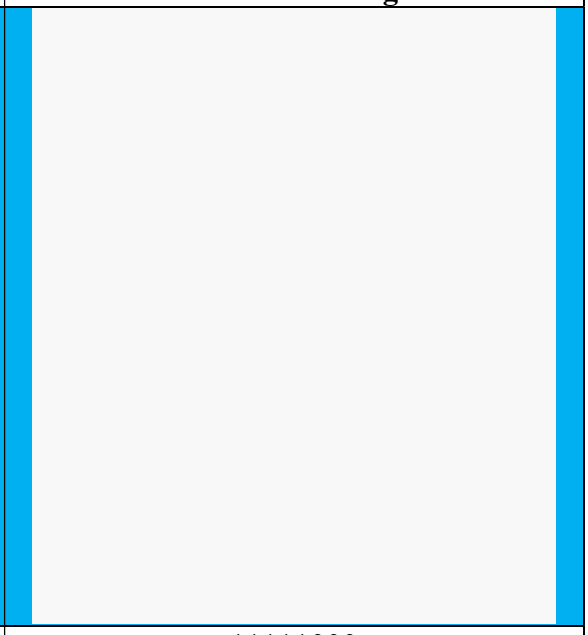
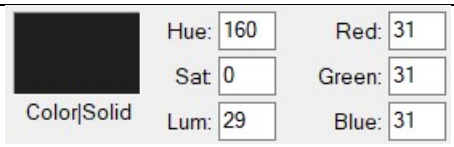
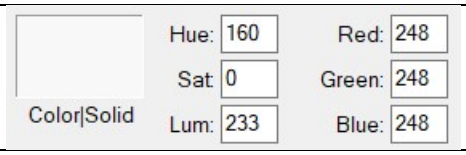
4.1.3. Using 3 Bits

Combined images	Recovered Image
	
00000111	11100000
 Color|Solid Hue: 160 Sat: 0 Lum: 7 Red: 7 Green: 7 Blue: 7	Color|Solid Hue: 160 Sat: 0 Lum: 211 Red: 224 Green: 224 Blue: 224

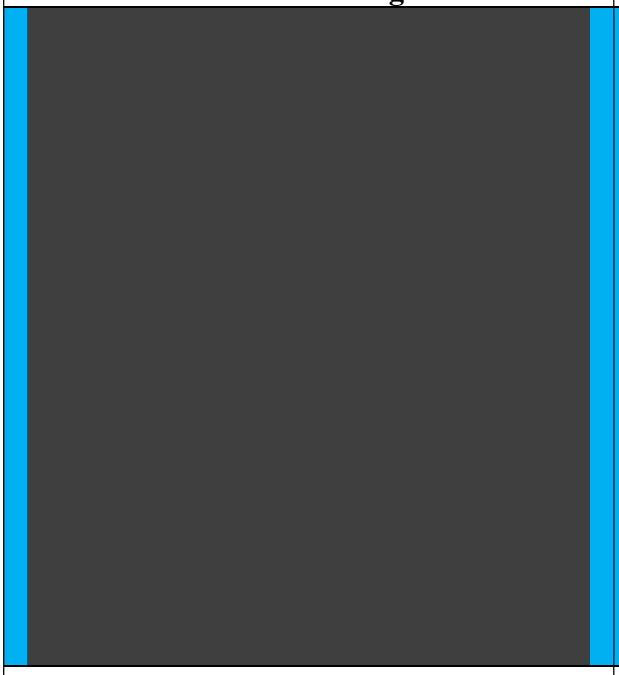
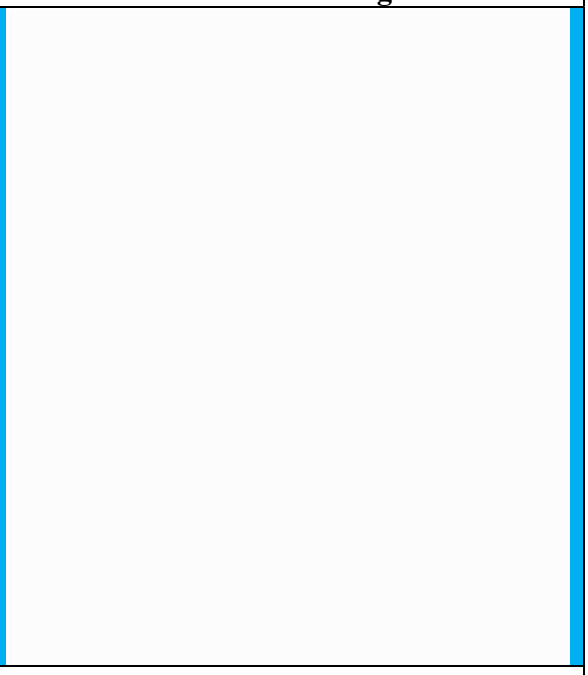
4.1.4. Using 4 Bits

Combined images	Recovered Image
	
00001111	11110000
 Color|Solid Hue: 160 Sat: 0 Lum: 14 Red: 15 Green: 15 Blue: 15	 Color|Solid Hue: 160 Sat: 0 Lum: 226 Red: 240 Green: 240 Blue: 240

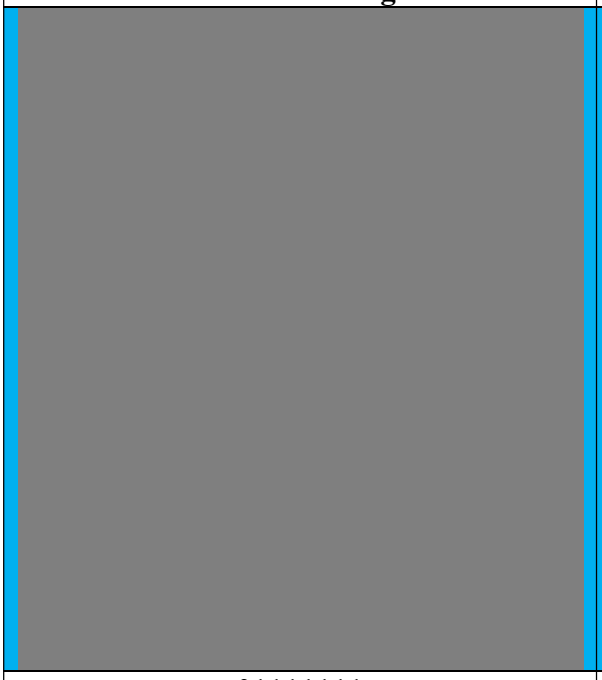
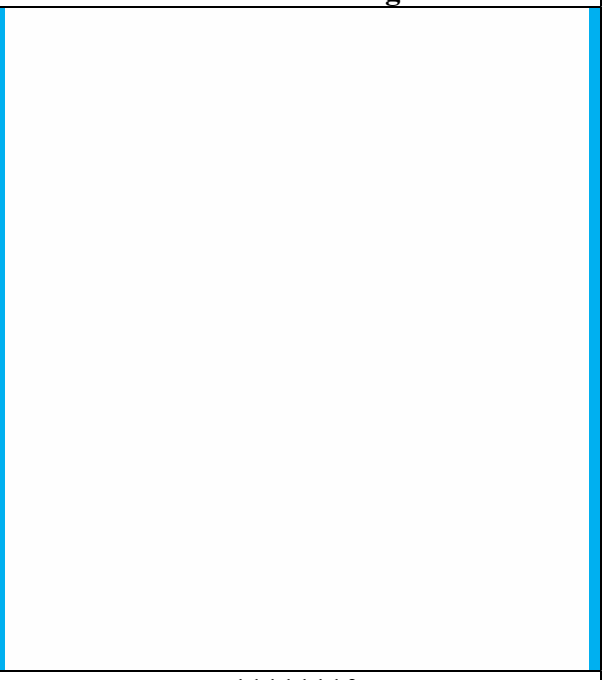
4.1.5. Using 5 Bits

Combined images	Recovered Image
	
00011111	11111000
	

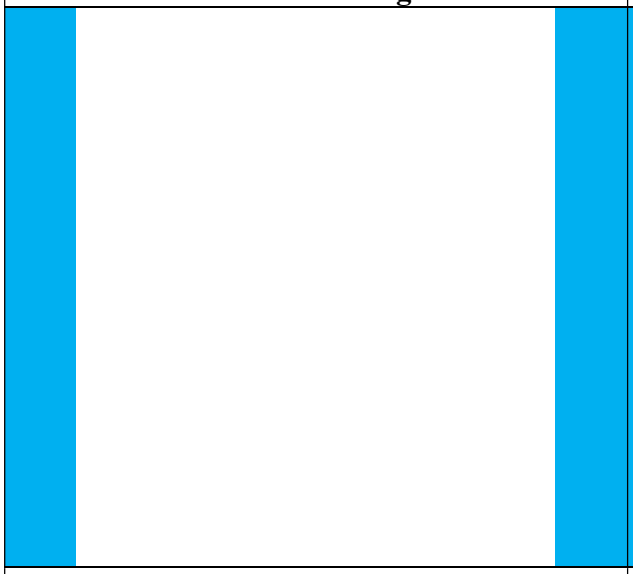
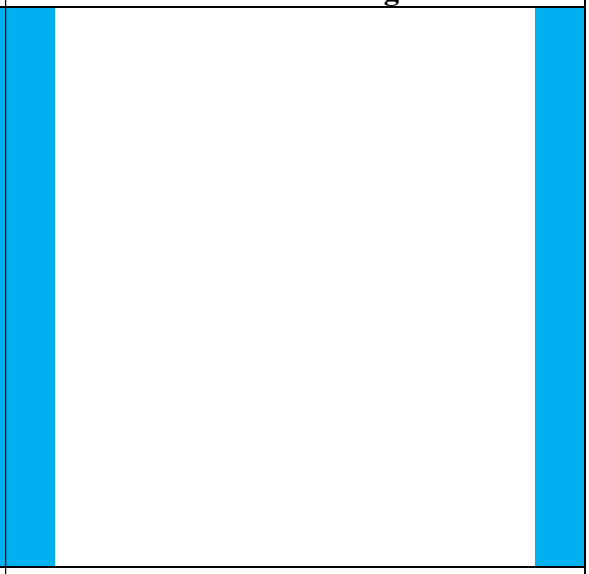
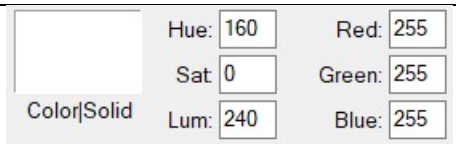
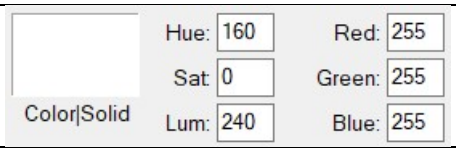
4.1.6. Using 6 Bits

Combined images	Recovered Image
	
00111111	11111100
	

4.1.7. Using 7 Bits

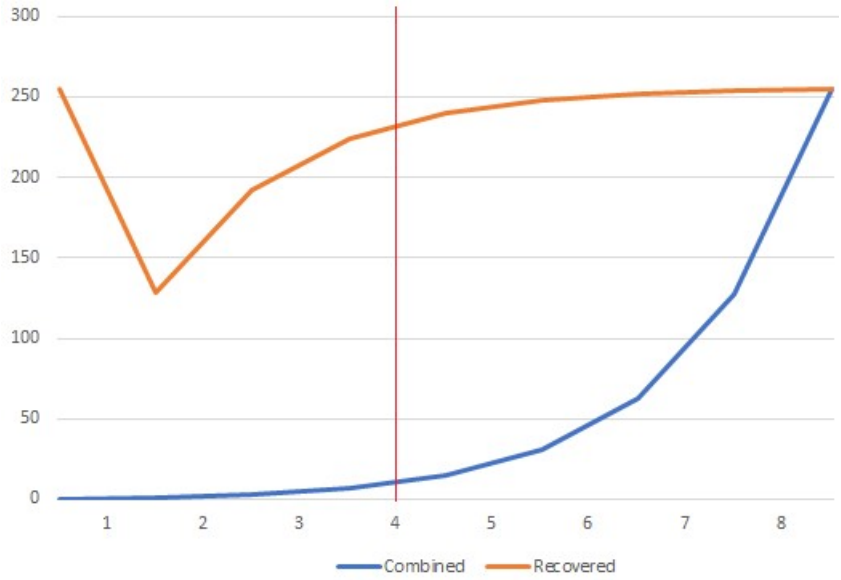
Combined images	Recovered Image
	
01111111	11111110
 Color|Solid Hue: 160 Sat: 0 Lum: 120 Red: 127 Green: 127 Blue: 127	 Color|Solid Hue: 160 Sat: 0 Lum: 239 Red: 254 Green: 254 Blue: 254

4.1.8. Using 8 Bits

Combined images	Recovered Image
	
11111111	11111111
	

4.1.9. Conclusion

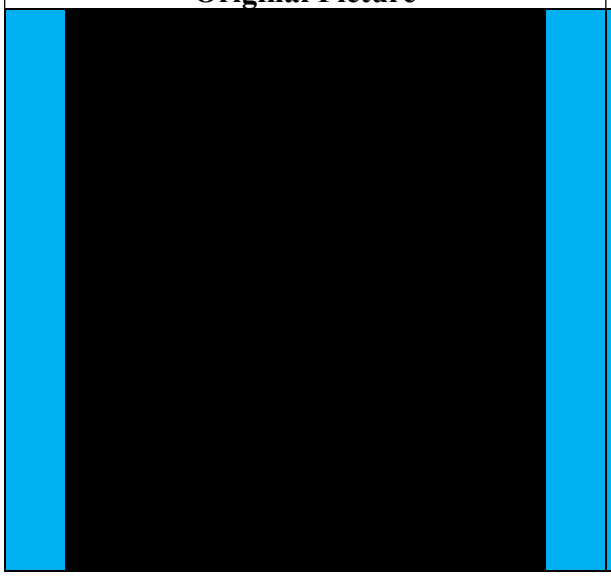
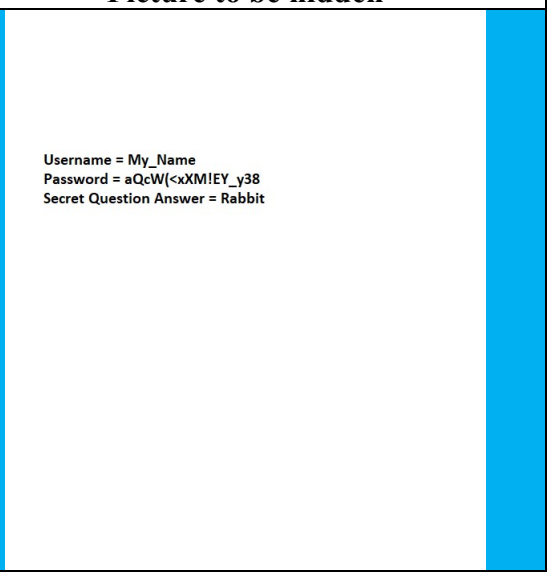
From the tests above we can graph the results and we see that changing 4 bits is probably the ideal max we should change. The red perpendicular line shows the 4 bit ideal set.



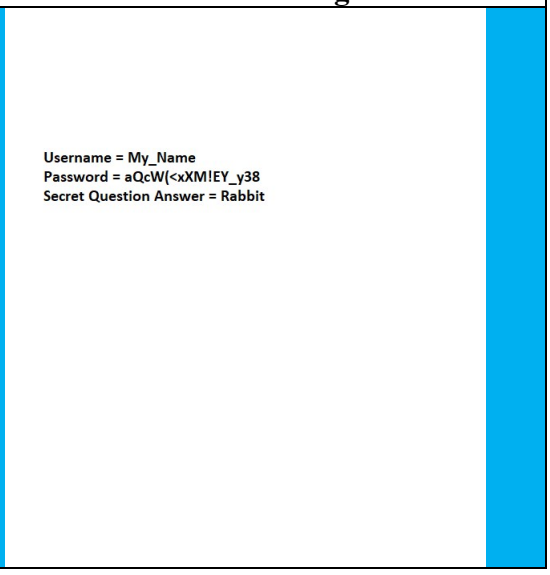
4.2. Hiding Image of text inside same size Black image

Now looking at this we may think it is not very good but what we need to think of is how will we use it. Let's think that we want to hide the username, password and secret question answer for a site login. We try to make the password random unguessable but because we do this it

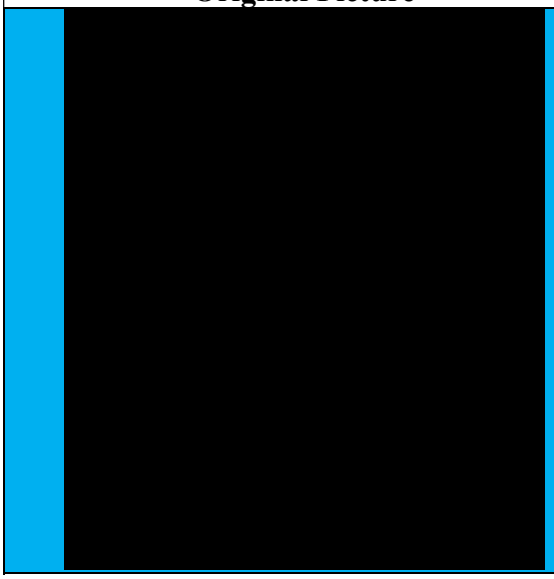
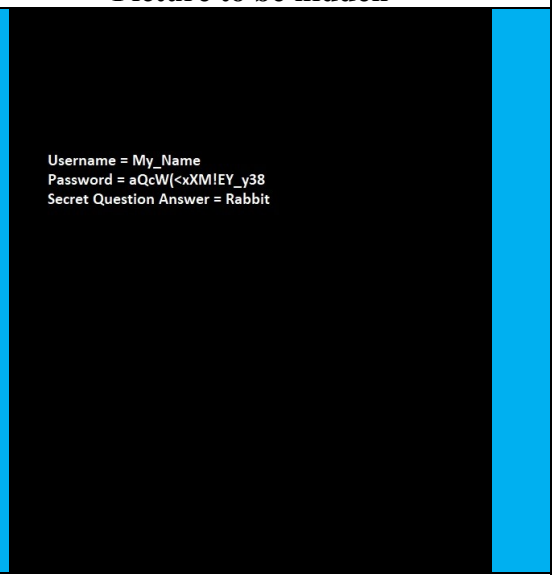
makes it harder to remember it. Using a picture to hide it in will be a lot easier. Here we could have the image we want to hide as black writing on a white background as below:

Original Picture	Picture to be hidden
	

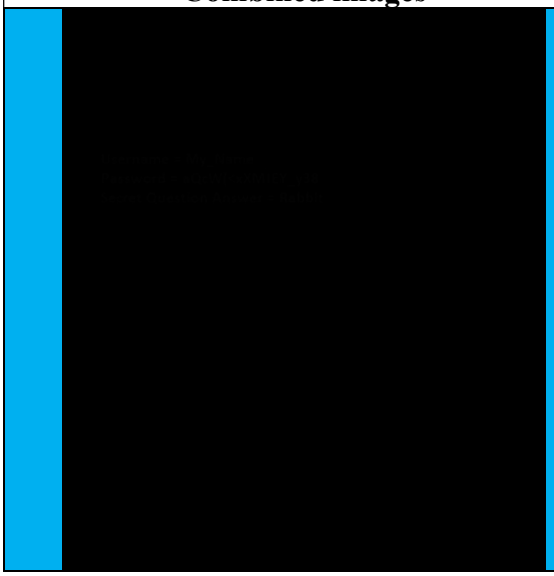
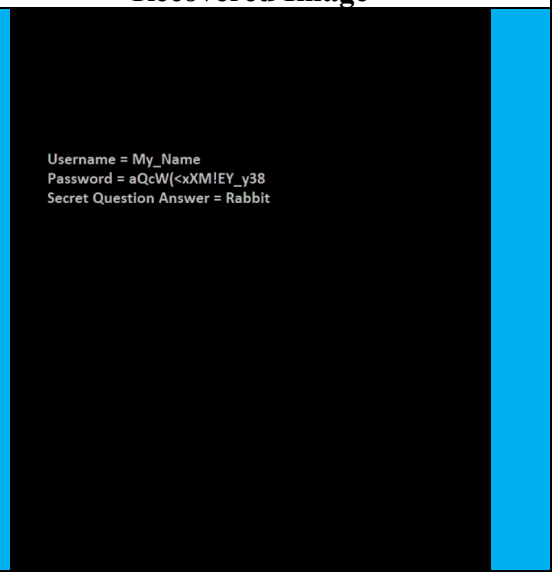
From the example above we had seen that using 4 LSB bits gives us a good picture so here now we show the combined image and the recovered image.

Combined images	Recovered Image
	

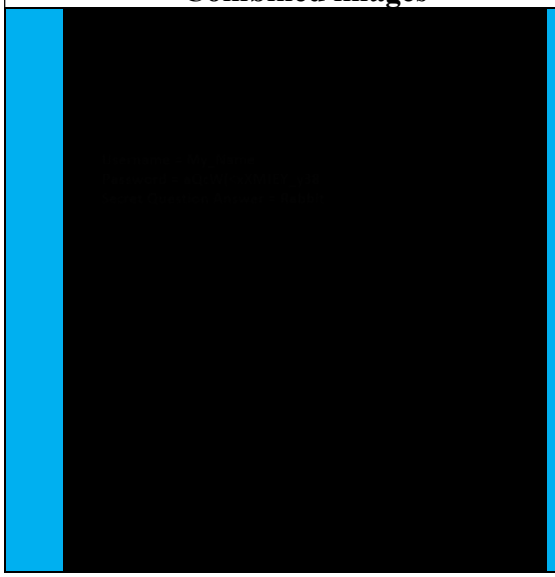
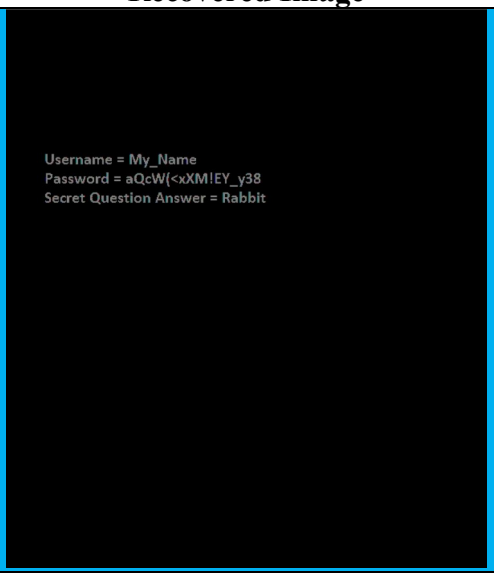
To make it slightly more difficult we could have white writing on a black background

Original Picture	Picture to be hidden
	

In this exercise we can see that the using just 2 bits the text is still very legible.

Combined images	Recovered Image
	

We can also try it with 1 bit and we see we can still read it although the image is not so clear.

Combined images	Recovered Image
	

As a hacker you would not be interested in the image but in the bytes behind it that go to make up the image so even if you find all the changed bytes you may still be looking for text rather than an image.

4.3. Hiding busy image of inside same size image

In this example we can see that the result we get are not perfect and you can see it has changed but if you never had the originals, you would not know this, you would just think the image was slightly poor in quality. The software could be written to try and match better where it puts the hidden data but that is a task for a future app.

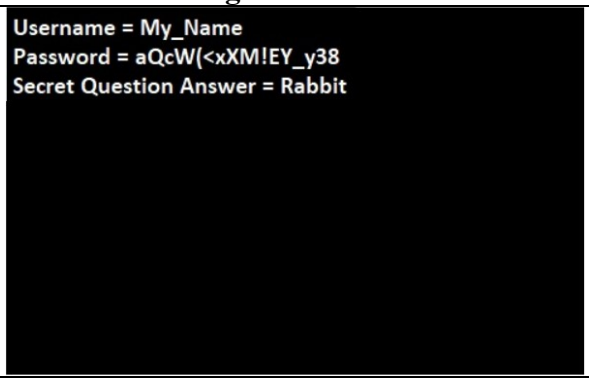
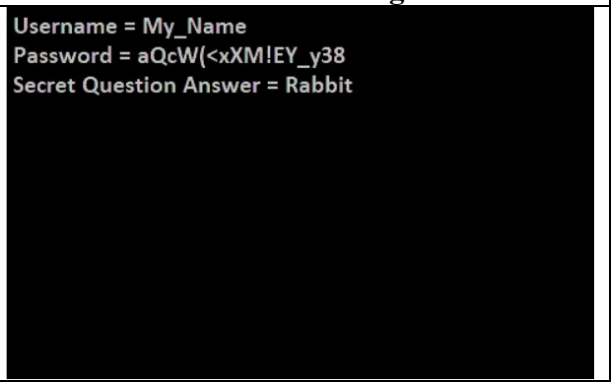
Original image	
Combined image	

Image to hide	
Recovered image	

4.4. Hiding Image of text inside same size image

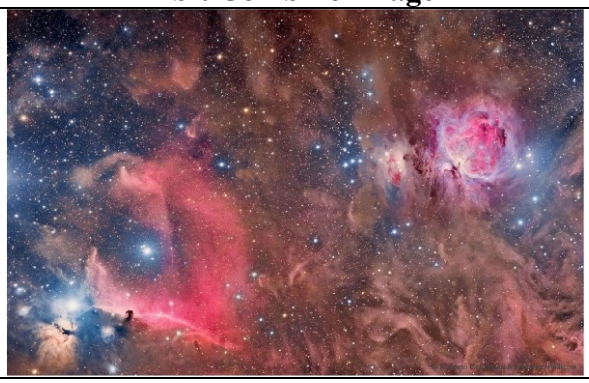
We could make the image we are hiding smaller than the image it is hid in. We could also use a very busy image to hide it in and this may make it even more difficult to find if you went through the images hex bytes.

To show how good this could be the image to hide is left large and is all black. When you look at the retrieved image you can see that the white is slightly off-white but the Original and the Combined images look the same.

Original Image	Image to be hidden
	
Combine Image	**Recovered Image**
	

4.5. Hiding Image of text inside larger sized image

If we make the image to hide smaller and then place its data randomly inside the original image, then it looks very similar but would make it harder to hack. Below we see the smaller image is hidden but each of the Combined images looks no different to the other. Here we only alter the pixels equal to the number we have in the image to hide. The rest are left as original. There are other things that can be done to make it even harder to see but I think you can see here that this method with some changes could be useful. Remember that if the combined image is changed or corrupted then you will not find it easy to recover the data.

Original Image	1bit Combine Image
	
3bits Combine Image	**4bits Combine Image**
	

Original Image to be hidden	1bit Recovered image
Username = My_Name Password = aQcW(<xXM!EY_y38 Secret Question Answer = Rabbit	Username = My_Name Password = aQcW(<xXM!EY_y38 Secret Question Answer = Rabbit
3bits Recovered image	**4 bits Recovered image**
Username = My_Name Password = aQcW(<xXM!EY_y38 Secret Question Answer = Rabbit	Username = My_Name Password = aQcW(<xXM!EY_y38 Secret Question Answer = Rabbit

It can also be made harder to see if we encrypt the image with a password before we hide it but how many times do you want to encrypt the data, I guess that depends on how valuable the data is.

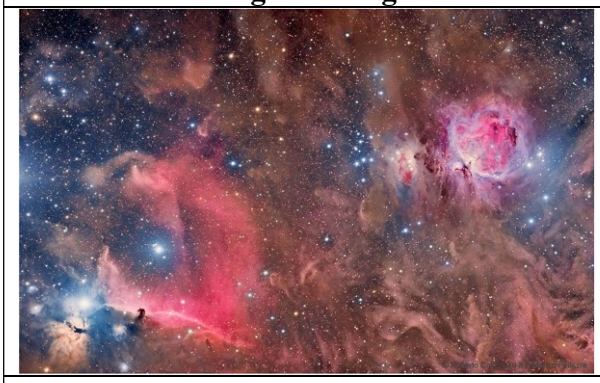
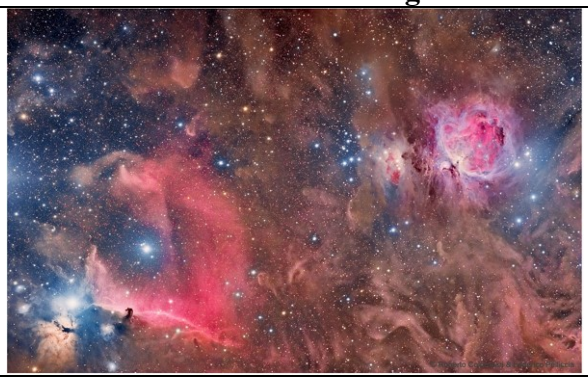
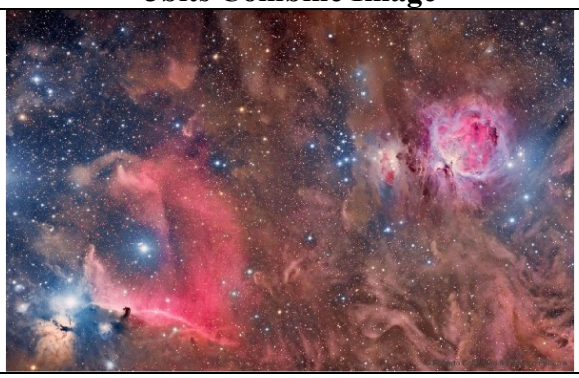
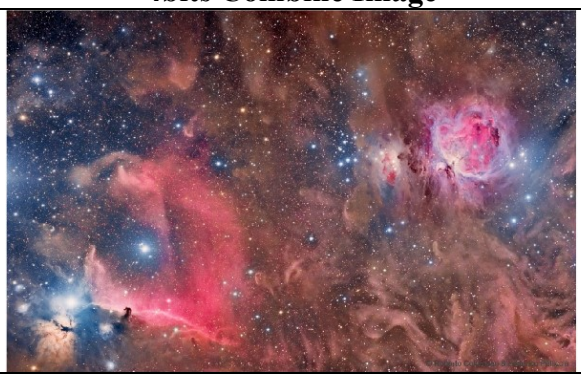
4.6. Hiding Image of QR Code inside larger image

A further experiment is to hide a 2D machine readable code inside the image. This will provide a further level of obfuscation to the data we are hiding in plain sight. Many of these 2D codes are themselves encrypted in such a way that you need special readers to decode them. They can also hold a lot of information and they themselves have error correction.

Here we use a 38x38 QR and encode the information into it.

```
Username = My_Name
Password = aQcW(<xXM!EY_y38
Secret Question Answer = Rabbit
```

We make a bitmap of the QR and get it size down to 84x84 pixels. The best I could have would be 38x38, but I do not have a reader that will read such a small code to hand. You can get them, but I just have standard phone apps available. I am not so worried by the recovered image quality, but rather can I read it with the lens in the phone app.

Original Image	1bit Combine Image
	
3bits Combine Image	**4bits Combine Image**
	

Original Image to be hidden	1bit Recovered image
	
3bits Recovered image	**4 bits Recovered image**
	

As I mention this method is very good to hide a lot of information in a very small space. Some QR codes are also secure codes meaning that a normal reader will not be able to read them as the data is encrypted. These are used by organisations like hospitals on the wristbands of patients. Secure code cannot be easily tampered with meaning that the patient should always get the correct treatment. I have written far more on the subject of Machine-Readable Codes than I can explain in a few lines here.

We can also look at taking the QR image to hide and using its hex bytes to hide as text in the picture. Let’s look at hiding text inside an image.

5. Hiding text inside image

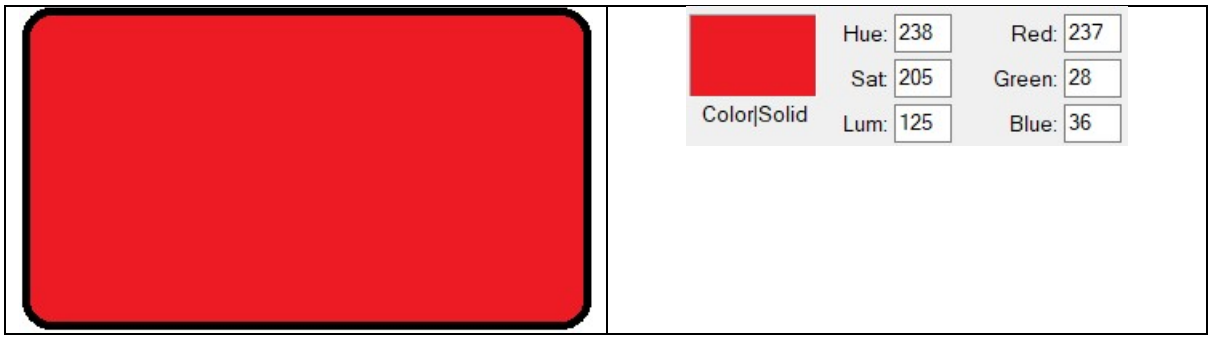
Here we take some plain text and show how to hide it inside an image. We already know that we can play with the least significant bit of the RGB colour.

1st we work out how many characters we can hide. If there is not enough space, then we need to get a picture with more pixels.

When we try to hide our data in an image (as in our case), then we need enough budget of LSBs to hide our data in. These bits are located in the image pixels. Since each pixel has three elements (R, G, and B that represent the red, green, and blue elements of the pixel consecutively, assuming non-transparent image), each of these elements can have a value between 0 and 255. Now, assume that the image was 300 pixels width by 400 pixels height, then we'll have 300 x 400 x 3 = 360000 LSBs. Depending on how we encode the characters data will depend on how many characters we can hide. Example, hide one character between each pixel as 3 LSB bits of Red, 3 LSB of Green and 2 LSB bits of Blue then this means we can hide 120 000 characters in an image that is 300x400. You can work out other combinations. One thing we will need to do is add somewhere in the image how many characters we have hidden so we know how many to look for.

We can also distribute the character sat random throughout the image and we could encrypt the information before we hide it just to make it harder to find. Remember that as long as the combined image remains unaltered the integrity of the information will be correct. As soon as you change the combined image you will destroy the hidden information.

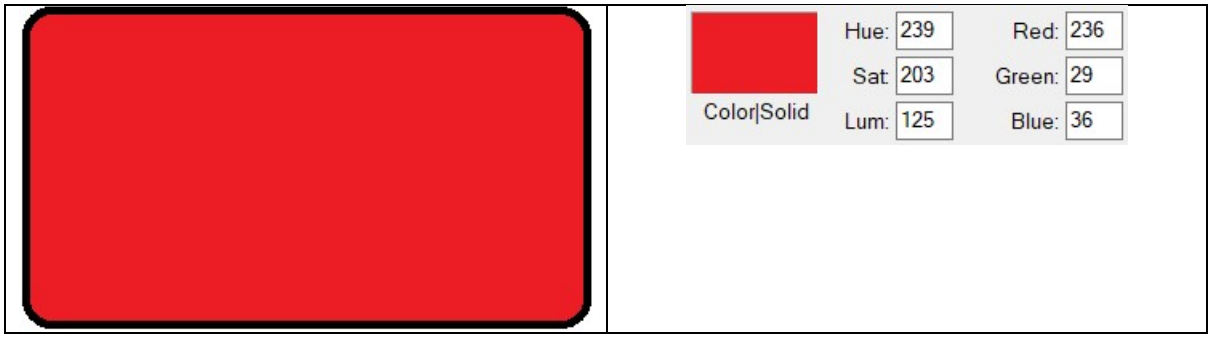
If we take the character 'A' we know this converts to the hex values of 41 and in turn this converts to the binary values of 01000001. Let's suppose we have a pixel as shown magnified below which has the RGB of 237 28 36. We convert this to binary, and we have 11101101 00011100 00100100



Now we hide the first 3 bytes of character A into the LSB of this pixel. We now geta new pixel of values

Character A = 01000001
Original Pixel = 11101101 00011100 00100100
New Pixel = 11101100 00011101 00100100

Now our new pixel looks like the following



I am sure you will agree it is difficult even when magnified to tell the difference between the original and the new pixel. Using this method means that one character is spread across 3 pixels. What if we want to spread the character across just one pixel which means we need to change the 3 LSB of R and G and 2 LSB of B (there are many more combinations you could use if you are creative in your coding)

Character A = 01000001
Original Pixel = 11101101 00011100 00100100
New Pixel = 11101010 00011000 00100101

Now our new pixel looks like the following

Again, we cannot tell the difference in this very magnified pixel. Imagine that this is in a picture of many thousand pixels, I think you will agree that it will be impossible to tell it has been altered.

If you place the text within random pixels it will be hard to get the letters out in the correct order that they went in as.

5.1. Examples used

5.1.1. Text to hide

```
Username = My_Name
Password = aQcW(<xXM!EY_y38
Secret Question Answer = Rabbit
```

5.1.2. Image being used to hide text in

The App I will use will take the text and hide it in a random place in the image. The text will also be encoded via an algorithm with the seed being a user provided password. I saw in another app that you can have a mode where the pixels your data is hidden at are highlighted. I have included that as well in my app. My app is built very much from what I have seen other apps do, I took what I thought were the best bits and mixed them together to produce my app.

The app comes in two parts, the encoder and a separate app to decode the information.

5.2. Hiding the text as text.

Here we take the raw text and hide it in the image. As the image is very busy and the data is short and stored at random places it is not easy to pick up with the naked eye in the combined image.

Original Image	
	
Combined Image	**Changed Pixels**
	

Encoded Text	Recovered Text
Username = My_Name Password = aQcW(<xXM!EY_y38 Secret Question Answer = Rabbit	

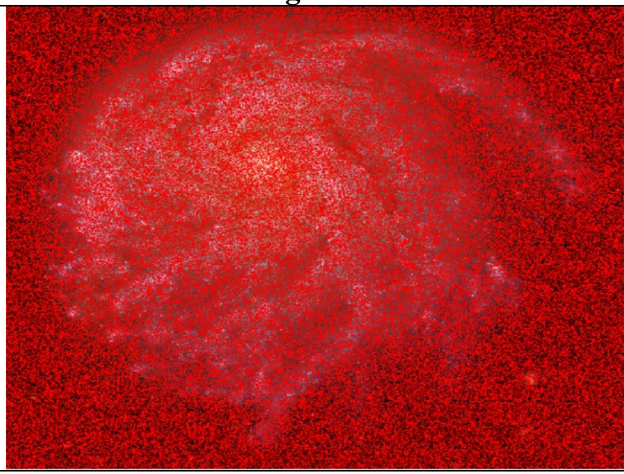
5.3. Hiding the QR as text.

Here we take the QR code bitmap and take the raw file and hide it in the image. The image here looks squashed, and this is just the app changing its aspect ratio. This is just an illustration and there are more and better ways to do this hiding in plain sight of the data.

Original Image

Combined Image

Changed Pixels



The text we hide and recovered is very large, so I only show a snippet here. The actual result of the image made from the recovered text shows it is exactly the same.

Original text snippet

BMæR......6...(...T...T...........
°R..t...t...ÿÿÿ........................ÿÿÿ.........ÿÿÿÿÿÿ...............ÿÿÿ.........ÿÿÿÿÿÿÿÿÿÿÿÿ.........ÿÿÿÿÿÿ.........ÿÿÿ.......
..ÿÿÿÿÿÿ.........ÿÿÿÿÿÿÿÿÿ.........ÿÿÿÿÿÿÿÿÿÿÿÿ.........ÿÿÿÿÿÿ...ÿÿÿ........................ÿÿÿ.........ÿÿÿÿÿÿ..........
.....ÿÿÿ.........ÿÿÿÿÿÿÿÿÿÿÿÿ.........ÿÿÿÿÿÿ.........ÿÿÿ.........ÿÿÿÿÿÿ.........ÿÿÿÿÿÿÿÿÿ.........ÿÿÿÿÿÿÿÿÿÿÿÿ.........ÿÿÿÿÿÿ..
........ÿÿÿ........................ÿÿÿ.........ÿÿÿÿÿÿ...........................ÿÿÿÿÿÿÿÿÿÿÿÿ.........ÿÿÿÿÿÿ.........ÿÿÿ.............................ÿÿÿ.........ÿÿÿÿÿÿÿÿÿÿÿÿ................
............................ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ.........ÿÿÿ........................ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ...............ÿÿ......
........................ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ...................ÿÿÿÿÿÿÿÿÿÿÿÿ.........ÿÿÿÿÿÿ.....................ÿÿÿÿÿÿ.........ÿÿÿ...ÿÿÿÿÿÿ....
...........ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ...ÿÿÿÿÿÿÿÿÿÿÿÿ....................................ÿÿÿÿÿÿÿÿÿÿÿÿ.........ÿÿÿÿÿÿ.....................ÿÿÿÿÿÿ.
........ÿÿÿ...............ÿÿÿÿÿÿ....................ÿÿ....................ÿÿÿÿÿÿ...............ÿÿÿÿÿÿÿÿÿÿÿÿ..........................ÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ.........ÿÿÿÿÿÿ.....................ÿÿÿÿÿÿ.........ÿÿÿ...ÿÿÿ.........ÿÿÿÿÿÿ...
..ÿÿÿÿÿÿ..ÿÿÿÿÿÿ....................ÿÿÿÿÿÿ.........ÿÿÿ...........................ÿÿÿ..........................ÿ
ÿÿ.........ÿÿÿÿÿÿ..........................ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ..............................ÿÿÿÿÿÿÿÿÿÿÿÿ..............................ÿÿÿÿÿÿ....................ÿÿÿÿÿÿ.
........ÿÿÿ........................ÿÿÿ........................ÿÿÿ.........ÿÿÿÿÿÿ..........................ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ..............................ÿÿÿÿÿÿÿÿÿÿÿÿ.

Recovered text snippet

Original Image	Recovered image
	
Encoded Text	**Recovered Decode to Text**
Username = My_Name Password = aQcW(<xXM!EY_y38 Secret Question Answer = Rabbit	Username = My_Name Password = aQcW(<xXM!EY_y38 Secret Question Answer = Rabbit

6. Passwords

A password is only secret if it is known by one person and never written down anywhere. Once you have a password and input it into a computer system then it is no longer secret. It is only your key to open a door. As we need lots of passwords on a daily basis and we try to make them all unique it is difficult to remember them all. Even if the password is random there are many ways in which it can be compromised. Many people will know the feeling of knowing someone has broken in with your password. This started me on a journey to look at a way in which I could make it more difficult to get hold of the password. As I said at the beginning as soon as the password is known to more than just you it is no longer secret.

Assume you had a password

```
mn$BK27nBjhY}v,.
```

and you place each character into a random picture pixel. If you knew which pixels had been altered which is highly unlikely if you used a unique image as you will have nothing to compare it against.

Let's say you got lucky, and you extracted each character then you still need to work out the order they go into. If this was just 3 characters ABC then we can have the following combinations:

- ABC
- ACB
- CAB
- CBA
- BAC
- BCA

With 16 characters the number of combinations we can have is far greater and because the characters are random, we cannot check them against a dictionary and have to try each combination. I agree the first you try could work but then it may be the last one. Image if we had 128 Characters in the password. How many combinations will this give us?

```
3k<E5,{qdv3<@3"q%$A~3*y'*>ay!>pqP9R%x9/u?_t]Ph}Yeb3n,u*meu2AXv
&VkB}vx!e2uBtuUK2pKdctAmgT_k5>>WPM9Z"6-Se)<s&?h=]R4y>;HGyF,!A-
}$4[
```

Go even bigger to 2048 Characters now you will spend years just writing down each variation of the password that there could be even if we used a computer to do it. In years to come when we get faster Quantum computers maybe it will be easy to test each combination in a person's lifetime but with the rapid rate that the world changes by the time you find a working password to unlock whatever was locked it will be prehistoric and probably not of much worth to anyone anymore.

```
Wnz9BtZpyAg9yxC3xzKFahLzDuyMkkunrEUtYjM2wEHT2GNbnb3eyH9G882wyc
7zvu9FrUMDkrtZvgEP4fx8pQAXdpMB8VaRdeMVUgJFnGqxx8MLvN6xehXxHbBm
4t2qG5RStg4LkeMRZBwEz8mUjgw8W6wZ2VdvdD3anHkBqmN699986fCq2YJEhg
```

9xN4RAWj7ssBLVhwhjzwTjVbtQWqwpAmutv8nq4UYFtUKqajP67AY9c2m7YVsb
RTMBxmY9yU67pFxwhqzbD5UtHLmEVFgZcdLd49Pj5JA59bR8A6Jvf3DGwLeUPq
wDEmkxj2TArf6GkX9FrFrDLySF3LLBrJ5GrcyeDL3HqxVnN5DquSG7czFu8Svd
FSu6jCm53wmBf3QHqspwDT2VMaU4btcQFh4aSnfz3dzMCW2JYsA6vRVvn36nqB
wKhMcUZHyhmU3ARgAuLuJAhTNRy3NQkDZ39a57t4S2SBv9gb8QsxKqbMavJmJT
yYxVCFumGpVLyXbVChgFe9EmFSyJZZXYjftrfpXet5Xv4WANY9cjgUMXCqpxf3
9wDEjXxtBckcBnSp7rbTTLUFhuLAzYcuXv9UEMEmZcL767LK3wzE2h4HbugkUz
eGBgDtLPfL8wNga3XKGhPhtRn8j5mCrUTjaDXcmuVSv7CqJdUQtHhtNWjjRdDL
3Dnm7NXFAGFSZLspSaszJ8wKhsZbsKzeLF9hhDQWMZDbdyATeSTCwVCY25CbfK
bMyUEJ7T5CdC4ymgES54L5nhMxBp72fWwG2QCRENcDRQBRVDA5y36hcWkv62Nh
xMuPLSTAj9fKXYZpguAU75HQ6LvYWk5m2t84CNwjqmXd9RQz7fBjdtLK8Gdb9A
CVh3xF5MkWkXB9QXpfbbLTmAyvpfyADWC2nFfqVPA2WHCfcCpMVKaBu89kvc3x
6LJmmbtbTUzadqTL7MqwcW4NsXrDYckpnCJRfDMffBQY6y7F6caZKtMqeCPGGp
hRqHrg9Pj8fZwZEjJvtkAxAK5MNT5YQMFVLWxUaTqAULccun9WCPhBagehxWsk
jCxEENnrECkMVsUbGLXqxTA5dmnEp3puTHyy3G6JSE9cFCqQR9Wu7pVQUvUxKy
ZSA3DgABUwB4vj9DPR4AcWhUDTTbGTFbvVL9DW7qKgK2A3x4Sz8wcsChMKrAYC
jNhKYWWKjfvTwan26yTvL7fAMuPYuecBfqVZBE35ckjSgrUjpV3j5RGzgTvG3Z
MbTjycaTvzD3Gd6jh8FbdfwEVUsCsGFDVy6pL9EXM6338tMDNMNHAj7Q7ZTC7B
9bFQ43rtdKh3XUbhySRYxx6EYHQB3R73jwuuxAEA42FefqbHnFUmSVgH7T6WXX
Q8fjUVHUx6M6uTqycXE857jyAptkJbh4L6pzKtEm2xUPBQEwBgdzNdgPmrxWEJ
sAJnbnux3JHzEXgy8nepwkzfzyhWQKJ98VS7RHfN9khFswZpxWRKVzXDpUsa4n
nLYb8EzN3ABL7ULFh4ZvQ4jZYL9m5Wtug6KgsFHNrmGduNUVjZwPjBjEuRBfcf
BBQuny8skpTgBEhB8C5KVjeXjVNnQcLQBB8DAjg4F3h759EjTanmXtSBZ7gvh6
rTHDyrFZnCHwUcKV4wSK7r62tDsrXY36xAa5XZAPaC7TMawMunrTHkL4p5thJZ
bPc57PJ4kzQcEcCGJfxz8jQUfubSFcxrq668w534UEe25d6zZHuJRv6Y9WPE2d
8aa7SSqKJB6u6rMZfUEjcbBADzUCHdX2VQfZQZmkY4qAqKZCqmDBz4QkxK9aHU
yxTMtyRuhqThT22jGDcXJXHeBwawC88NFKq4J2ajgQbUdnKS4QuGNAhrTYebc9
hdvvp5H8TxubH9VgYjZaDa65nrrhy49pPKVTvZAKSbVtJYGgWJVRZT8WpzGkme
hKy2sa6GBhsp6GgEvhUa5ufPbb3pQcYnX6MXRZy8Ldpc5MHspUAueHKBa4Xeka
gWFkChJfE2U9bXBDKNPmjSH2tA6ApMavLczacKCbyWJEdWcAdvsbvRxU6cPTG4
rv

In some academic papers I read I see that some research is being done to work out if an image has been altered by trying to work out if the pixel it is looking at seems to fit with all those around it. At present this is not fool proof. If you do find what you think is a pixel that does not fit, you may need to work out if the camera or camera software caused the change unintentionally or if the user did change it. I think far more research would need to be done before this becomes a way to decrypt the hidden data.

6.1. Hiding an Encrypted password in image.

Here we just hide a password that we first encode and then restore it. We used two separate apps for this one to encode and one to decode. The key between the two is a password, I know we are talking about passwords and there are ways to pass key between the two without it being a password but to make it easy to show I am just using a plain password and the value is

```
password
```

The password we will hide will be 256 chars long

Original Image

Combined Image


Changed Pixels


If you look at the centre of the changed pixel image you will see the red dots that mark some of the pixels we have changed. If you look at the combined vs the original, then you cannot see any changes.

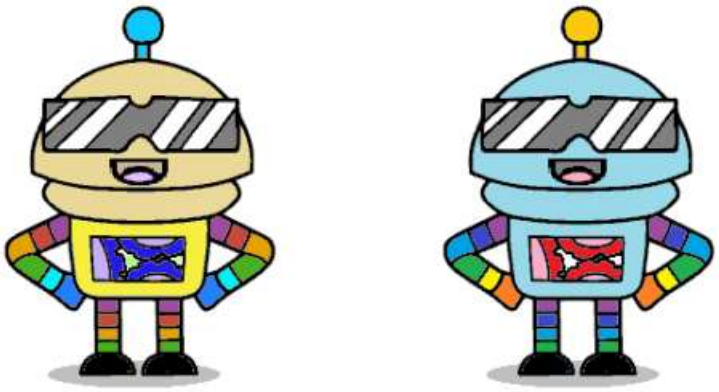
Hidden Password	Recovered Password
muCdv6tpC4SJkyedACPGUEdvrpU7tAZaWx68L2ReSxxbG4UvMJ23h67YjQRQRLUqYpcMeNnhE4cJUvdGYxnjVCg5AMNAf9nwThxMrVwgFGjgBbWY6mtJzHcdPp2JPw7b42UactkBazQe59xDa7qzLsBjSmCXEhxuFdd9NGJHb6ke5CVwx2SeD4xzyBfe8SyVHEcJFSsENwXeNZFEQaLncNhdLchZVuWytrXg3DkuRcT7fnmb8JMBHACBVtxZ5VYB	

7. Conclusion

We can see that we can hide images or text in plain sight inside another image. There are many uses for this type of work especially in helping us remember very long passwords and usernames to log into various systems. Care should be taken in what you hide because if you only keep one copy of the image and it gets corrupted then the data will potentially be lost forever.

8. Appendix – Hiding Text Apps Instructions

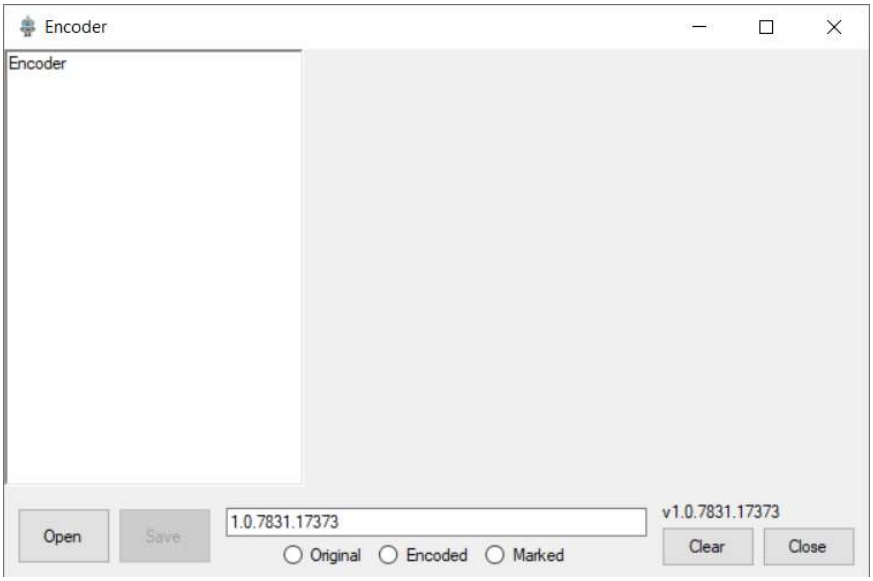
There are two apps you can use, one to encode and one to decode. These apps were built only for demonstration purposes and does incorporate some code in the public domain that has been written by others.



When the data is hidden it will get encrypted. The encryption can be changed but has to be the same in the encoder and decoder. The seed for the encryption and random placement of the hidden bits is the password.

8.1. Encoder

When you open the app you will get the main screen as shown below.



On the left is a text box with the words Encoder. This is the box where you put the text to be hidden.

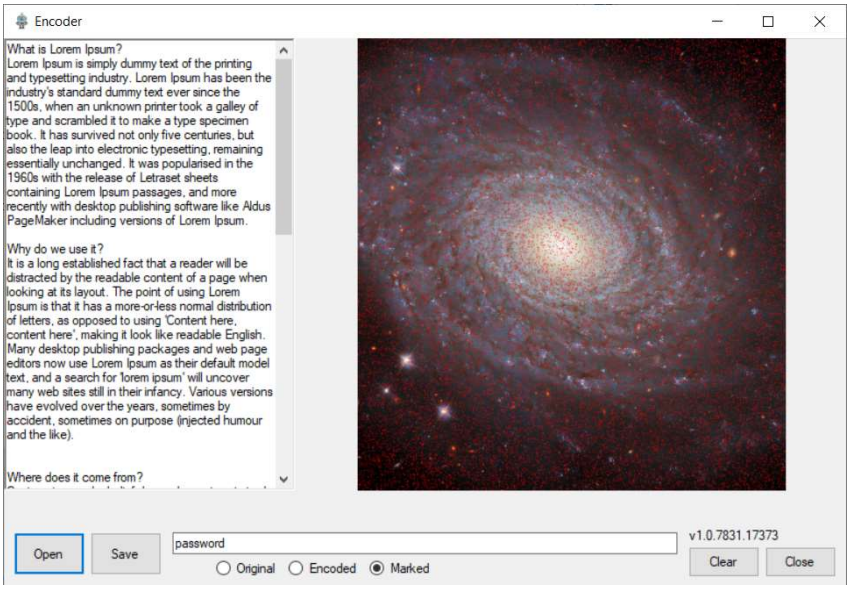
At the centre bottom you see a text box that contains a number. This is the box where you put the password. This password is the key used in the decoder to recover the text you hid.

When you press open you can choose the picture to hide the text into. This will then be show in the picture box on righthand panel.

The three radio buttons at the bottom will show you the original image and the encoded image. It will also highlight where it hid the text if you click on “Marked”.

When you click on save you will save the actual image inside the picture box which you had chosen via the radio buttons.

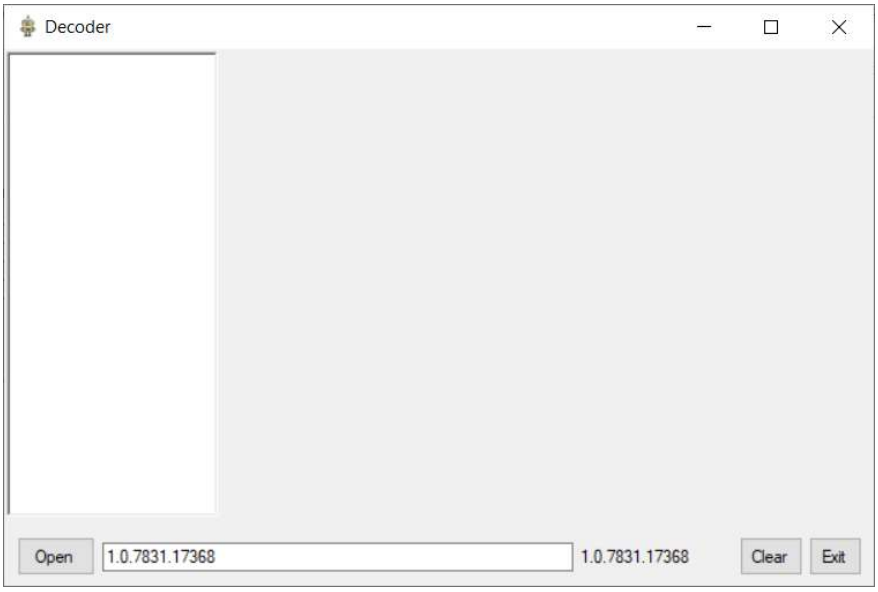
Below we see the text to be hidden and the image marked up where the text is hidden. We can also see the password which just happens to be “password”.



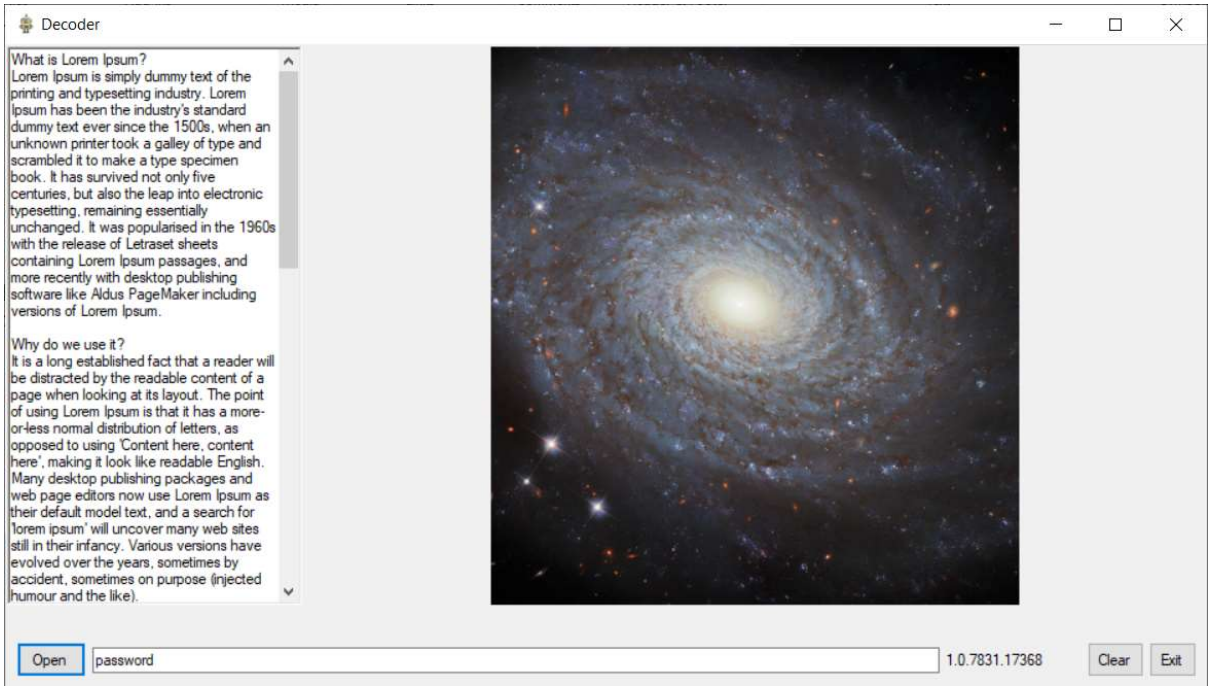
The amount of text that can be hidden depends on the size of the image and number of characters. It is best to choose the biggest image you can use as this will distribute the hidden text the best.

8.2. Decoder

At the bottom of the app is a text box. This is where we put the password that we used in the encoder. You can only recover the text that is hidden in the image if the password is correct.



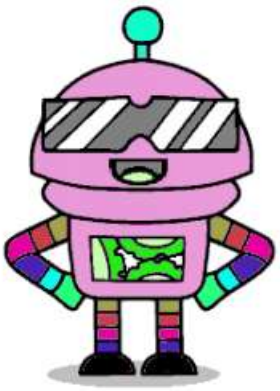
Then click on open button and find the image you want to decode. When the image opens the text will immediately be recovered and placed in the textbox on the left. The image is shown in the picture box on the righthand side.



If the text cannot be recovered, then an alert message will be displayed.

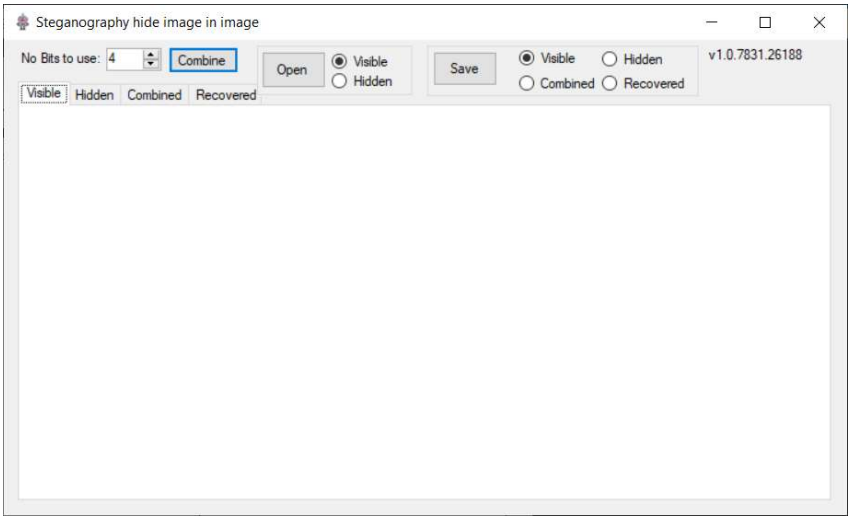
9. Appendix – Hiding Image App Instructions

This was developed using code in the public domain and then heavily modified by me to my needs. It is thanks to others who discuss their work in the public domain that I was able to learn how to do hide images inside other images.



Steganography_hide_image_in_image.exe

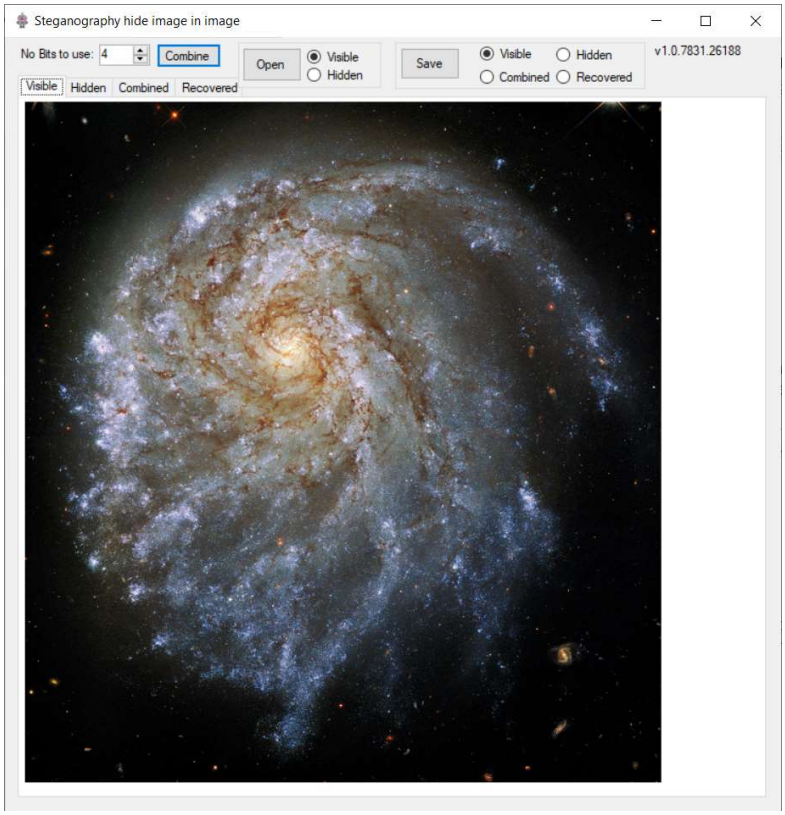
When you start the app you will get the following interface.



The first thing you will need to do is open the image you want to hide the other image in. This image you hide it will need to be the same size or larger than the image you wish to hide in it.

Next you will want to open the image to hide. Then select the number of bits you want to change. Usually, it will be 4 or less but this depends very much on the image you are trying to hide.

You can save each image if you want to later compare them.



My main aim was to hide secret information like passwords and usernames in images, so I did not develop this app very much further than to get my own understanding of technics to use when hiding one image inside another.

This app could be further advanced and could include things like

- Decode previous saved images
- Encrypt the image data before combining
- Distribute the bytes to hide randomly in main image.