BITMAP MORPH

Secure Steganography

User Guide

Version 1.0 (Public)

Contents

1. Requirements

Bitmap Morph is written in Java and requires Java Runtime Environment (JRE) 1.6 or later; this can be downloaded form the Oracle website. A command line is needed to use Bitmap Morph as it does not have a graphical user interface.

Bitmap Morph is for educational purposes and has not been fully tested. It has been released for individuals who wish to gain a greater understanding of steganography and for those individuals to see the effect of encoding secret data into a bitmap image using bit replacement.

2. Introduction

Bitmap Morph is a Java application that uses steganography to hide files within a bitmap image. It uses bit replacement techniques to achieve this analysing the carrier and secret file. Bitmap Morph then decides how many bits to use in order to encode the secret file. As this is an application for educational purposes it can use the entire bit depth for encoding. For instance if the bit depth is 8 Bitmap Morph will be able to use up to 8 bits per RGB component making the carrier image unrecognisable. It is not recommended to use more than 3 bits per RGB component when encoding secret data.

Bitmap Morph also has built in encryption and noise insertion, it is possible to disable the encryption or noise insertion if required through the use of arguments. The encryption keys and initialisation vectors are randomly generated using a Cryptographically Secure Pseudo Random Number Generator then stored within the stagger code.

Bitmap Morph does not use passwords for authentication instead it uses stagger codes. Stagger codes are generated using Bitmap Morph and are required for both encoding and decoding secret data. They can be used to control how densely data is encoded into a bitmap. They also contain encryption keys. An initialisation vector is randomly generated at the time of encryption and inserted into a header which is then encrypted using a different key and initialisation vector (this IV is stored within the stagger code and randomly generated when the stagger code is created) before being appended to the start of the secret file. For more information about stagger codes go to section 7.

Bitmap Morph also includes tools to analyse stegograms illustrating the effects of hiding secret data within bitmap images. These tools are Peak Signal to Noise Ratio Calculator, Averaged Hamming Distance Calculator and LSB visual enhancement to highlight hidden data within an image.

3. Features

Bitmap Morph contains the following features;

1. Stagger Code Generator.
2. Stagger Code Analyser.
3. Insert random bits of noise up to 8 bits per RGB component.
4. Encode single files into a bitmap image (Single).
5. Encode multiple files into a bitmap image (Sequential).
6. Extract files from a bitmap image.
7. Enhance a bitmap image to show secret files.
8. Bitmap Analyser.
9. Peak Signal to noise calculator.
10. Hamming distance calculator.
11. Randomly generated encryption keys and initialization vectors using Java's built in CSPRNG.
12. AES 128 bit encryption using CBC mode.

4. Limitations

1. Stagger codes must be used for encoding / extraction.
2. Passwords can not be used to authenticate stagger codes. i.e. if your stagger code is intercepted and not encrypted then the attacker can extract your secret files from the cover image.
3. Filenames of secret files are lost during the encoding / extraction process however the secret files extension is retained.
4. Supports file extensions of less than 7 characters.
5. Stagger Codes are 512KB in size.
6. Supports windows bitmaps. O/S 2 bitmaps are not compatible.
7. No GUI.
8. No error checking when entering values via arguments
9. Max bitmap and secret file size is 1GB.
10. Limited entropy of Java's CSPRNG - this should not affect this program but I have mentioned it for completeness.
11. Uses AES 128 bit encryption using CBC mode. This is so the built in Java Crypto framework can be used instead of using other cryptography frameworks.
12. Requires large amounts of memory to run; up to 4G of runtime memory.

I would not recommend using Bitmap Morph in the wild. As stated before it is intended for educational purposes only. Although I believe it is secure I have not implemented any cipher checking after the secret data has been encrypted and for this reason I can not guarantee that the encryption used

here is 100% secure. I also believe that they're issues with AES 128 and would recommend AES 256 for applications such as this.

5. Usage via the Command Line Interface

To load the CLI type the following in your command prompt;

```
Java -jar Bitmap_morph.jar cli
```

Please note that for this to work you must be in the folder that Bitmap Morph is located. For example if Bitmap Morph is located at c:\test then your screen should look like;

```
c:\test> java -jar bitmap_morph.jar cli
```

The CLI will give you 3 basic functions;

- Create stagger codes with default ranges for P1 max and P2 max
- Encode secret data into a bitmap image using single or sequential stagger codes
- Decode secret data from a bitmap with a specified stagger code

Before encoding secret data you need to generate a stagger code via the CLI.

To use these functions follow the on screen prompts.

WARNING: When using sequential stagger codes the original cover image is always overwritten with the stegogram.

NOTE: if you want to use the folder that you are already in for the destination of a stagger code instead of writing the folder location such as c:\test you can just enter a full stop "." or if the stagger code and image is in the same folder you can enter the name of the files instead of the whole file path for instance test.bmp instead of c:\test\test.bmp.

6. Usage via Arguments

WARNING: When using arguments entered values are not validated, this can cause the program to crash or behave unexpectedly. This mode is intended for advanced users only. Please note that spaces can not be in filenames or folder names when using these arguments. **All arguments are case sensitive.**

WARNING: When using sequential stagger codes the original cover image is always overwritten with the stegogram.

In the following pages a description of each argument that can be used with Bitmap Morph will be presented. Each variable for the arguments are separated with a space.

Bitmap Morph has **14 arguments** these are as follows;

Argument	1. help
Example	`Java -jar bitmap_morph.jar help`
Explanation	Returns a list of valid arguments.

Argument	2. bitmaploader <bitmap location>
Example	`Java -jar bitmap_morph.jar bitmaploader c:\test.bmp`
Explanation	This will print information about the bitmap including its size, pixel density and payload for LSB encoding - max payload is 8 times this number using 8 bits per RGB component.

Argument	3. expandlsb <bitmap location>
Example	`Java -jar bitmap_morph.jar expandlsb c:\test.bmp`
Explanation	This will enhance the image to show noise where secret files are encoded. This will turn the image into a 9 colour image by expanding the least significant bit for each colour component. As Bitmap Morph will always utilise the LSB it is a visual way of looking at stegograms. Once a stegogram is created use this method to enhance the stegogram and the original image then compare the two to see the difference.

Argument	4. bitmask <bitmap location> <number of bits>
Example	`Java -jar bitmap_morph.jar bitmask c:\test.bmp 2`
Explanation	This is used to demonstrate the effects of inserting random noise into a bitmap image. The second argument tells Bitmap Morph how many bits per RGB component to change; the max value is 8. This method will insert randomly generated noise into the amount of bits that you specify, for instance if 2 is entered then the LSB and bit 7 will be replaced with randomly generated noise.

Argument	5. generatecode <destination> <number of codes> <false>
Example	`Java -jar bitmap_morph.jar generatecode c:\s 20 false`
Explanation	This will generate 20 stagger codes and save them to the folder c:\s. The third argument should always be false unless you want to specify the parameters for the stagger code. Stagger codes will be given a randomly generated name and the file extension of .stg

Argument	6. generatecode <destination> <number of codes> <true> <P1 Max> <P2 Max>
Example	`Java -jar bitmap_morph.jar generatecode c:\s 20 true 20 50`
Explanation	This will generate stagger codes with user specified parameters. P1 max and P2 max has a maximum value of 255. By changing these values it is possible to change how densely the secret file is encoded into the cover image. The less dense the encoding the harder it will be to discover via steganalysis. The default value for P1 is and P2 is randomly selected when a stagger code is generated if the users does not specify it. For more information about stagger codes and P1, P2 values and what they are refer to chapter 6.

Argument	7. generateseq <destination> <number of secret files> <bitmap location>
Example	`Java -jar bitmap_morph.jar generateseq c:\s 3 c:\test.bmp`
Explanation	This will generate sequential stagger codes. For information about these refer to chapter 6. In order to generate sequential codes the method must be told which cover image is being used and how many secret files will be encoded into this cover image. This is what argument 3 and 2 are.

Argument	8. analysecode <stagger code location>
Example	`Java -jar bitmap_morph.jar analysecode c:\s\10922563.stg`
Explanation	This method returns information about a particular stagger code. For details of what these stats mean refer to chapter 6.

Argument	9. encode <stagger code location> <secret file location> <cover file location>
Example	`Java -jar bitmap_morph.jar encode c:\s\10922563.stg c:\secret.doc c:\test.bmp`
Explanation	Encodes the secret file to the cover image creating a stegogram using the specified stagger code. If the secret file does not use all of the bitstream noise will be added after the secret file so it cant be isolated by an attack. This will also encrypt the secret file using AES 128 bit encryption using a randomly generated key stored within the stagger code.

Argument	10. encodefalse <stagger code location> <secret file location> <cover file location>
Example	`Java -jar bitmap_morph.jar encodefalse c:\s\10922563.stg c:\secret.doc c:\test.bmp`
Explanation	Same as for the encode argument but noise is not added after the secret file is encoded. If the secret file does not utilise all of the bitstream then the rest of the bitstream is left unaltered.

Argument	11. encodenoenc <stagger code location> <secret file location> <cover file location>
Example	`Java -jar bitmap_morph.jar encodeseq c:\s\ 10922563.stg c:\secret.doc c:\test.bmp`
Explanation	Encodes the secret file without encrypting it. Noise is still added to the unused space. This is worth doing if you are encoding uncompressed files like text documents then running the expandlsb command on the stegogram to see the secret data. Notice the large amount of black stripes within the enhanced stegogram.

Argument	12. encodeseq <stagger code location> <secret file location> <cover file location> <number of secret files>
Example	`Java -jar bitmap_morph.jar encodeseq c:\s\ 10922563.stg c:\secret.doc c:\test.bmp 3`
Explanation	This argument is used when encoding secret files using sequential stagger codes. You need to encode each file one by one using this command and the stagger codes should be generated before encoding takes place. Each secret file will be encoded with a different stagger code generated using the CLI or the argument **generateseq**.

Argument	13. extract <stagger code location> <stegogarm>
Example	`Java -jar bitmap_morph.jar extract c:\s\95271393.stg c:\test\test.bmp`
Explanation	This argument will extract a secret file encoded into a bitmap. This will work for both sequential and single stagger codes. Please note that the secret file will be placed in the Bitmap Morph directory. It will be named with a random filename but it will retain the correct file extension.

Argument	14. stats <bitmap location 1> <bitmap location 2>
Example	`Java -jar bitmap_morph.jar stats c:\test\test.bmp c:\ test\test1.bmp`
Explanation	This will display the Peak Signal to Noise Ratio and Hamming distance for the selected bitmaps. It compares both bitmaps one should be the stegogram the other should be the original image. The smaller the hamming distance the harder it will be to detect the secret file and the higher the PSNR the harder it will be to detect the secret file. A PSNR of above 38 is undetectable to the human eye.

7. Stagger Code Possibilities

Stagger codes have been introduced to get away from the reliance on user generated passwords and to create a way of encoding secret data in a non linear way that removes the relationship between the secret file and the cover image. They can also be used to enable the user to control how the secret data is encoded which is useful when learning about steganography.

Stagger codes rely on number pairs that describes how to modulate the encoding process. These are referred to as P1 and P2. P1 and P2 are used as on and off descriptors. For instance P1 tells the encoder how many bits to use for encoding and P2 tells the encoder how many bits to ignore when encoding. A stagger code will contain 256,000 - 511,996 P1 and P2 values giving a huge amount of possible permutations. P1 and P2 max values are randomly calculated when a stagger code is generated. P1 and P2 max will lie between 128 and 255 unless stated otherwise by the user. By default P2 max will be P1 max / 2.

A value known as Carrier Efficiency (Carrier EFF) is used to calculate how much data can be encoded using a particular stagger code in respect to the amount of data that can be encoded if a stagger code is not used. The default Carrier EFF is 0.66 or 66%; this is maintained by the P2max = P1Max/2 constraint. This means that 66% of the bitstream can be used to encode secret data.

For instance if you have a bitmap that is 8MB in size you will be able to use about 1MB of that file to encode secret data using LSB replacement without using stagger codes. This means that you will take the least significant bit from each RGB component and use that to encode secret data. This will be extremely easy for an attacker to extract as the data will be encoded in a linear fashion.

Using a default stagger code the useable bitstream will be reduced to about 675KB however the secret data will be staggered throughout the image in a non linear way and encrypted using a random 128 bit key and a random initialisation vector.

Stagger codes can be used to illustrate the effects of encoding secret data with differing densities. Stagger codes with a higher code EFF will utilise more of the bitstream decreasing the PSNR making it easier to detect the secret data. This is achieved by decreasing the P2max value when generating a stagger code. The following command will create a stagger code with a Carrier EFF of 90%;

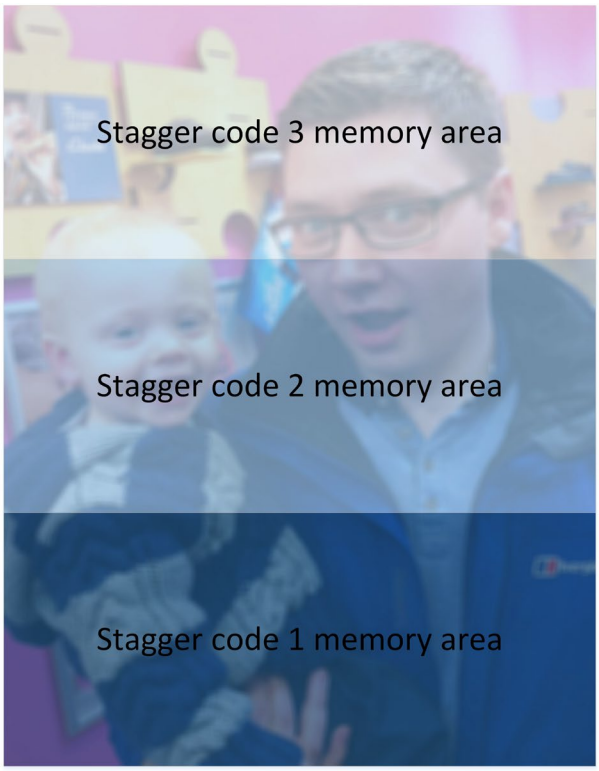
```
Java -jar bitmap_morph.jar generatecode c:\ 1 true 255 28
```

This will create one stagger code with a Carrier EFF of 0.9 or 90% and save it to the c:\ folder. What we are telling the stagger code generator to do here is create a stagger code where each P1 value will be between 1 and 255 and each P2 value will be between 1 and 28. Due to the large amount of P1 and P2 values the average P1 value throughout the dataset will be 128 and the average P2 value will be 14.

This means that on average for every 142 bits in the bitstream 128 will be used to encode data and 14 bits will be ignored. This allocation of used and ignored bits will be randomly assigned by the CSPRNG used to generate the P1 and P2 number pairs.

Sequential Stagger Codes

Sequential stagger codes are used when more than one secret file is required to be encoded into the same image. Multiple stagger codes can be used to encode multiple files into a single image. The below image demonstrates this;



This is useful when a single image is used to pass secret data to more than one receiver without the other receivers knowing that more secret data exists in the stegogram. Each receiver will only be able to gain access to the secret data that was encoded with their stagger code.

Sequential stagger codes need to be generated differently using the `generateseq` argument or through the CLI. Encoding is also different using the `encodeseq` argument or following the prompts via the CLI.

8. Steganalysis Tools

Bitmap Morph contains three tools to analyse stegograms and view the effects of hiding secret files within the carrier file. These tools are;

- Peak Signal to Noise Ratio Calculator (PSNR)
- Averaged Hamming Distance Calculator
- LSB Visual Enhancement

8.1 PSNR Calculator

PSNR is a method used to measure the noise in a digital image. For this calculation to take place the original image and the stegogram is needed. The procedure will compare both images and calculate the PSNR. The higher the PSNR the less noise present in the stegogram therefore stegograms with higher PSNR values will be harder to detect. A stegogram with a PSNR value of 38 or lower will contain noise that is detectable to the human eye. In order for a steganographic algorithm to be successful it needs to limit the amount of noise that is added to the stegogram via the encoding process. Bitmap Morph beats other applications in this area however due to the "dumb" stegnagraphic algorithm it can be greatly improved. The use of stagger codes especially sparse (low Code EFF)stagger codes help to give a higher (better) PSNR value.

8.2 Hamming Distance Calculator

The hamming distance of two binary words describes the differential between those binary words. The hamming distance calculator will compare each pixel like for like from the stegogram and original image to give an averaged differential value for the whole image. This will be a value of how many bits per pixel have been changed regardless of bit depth. It also gives this value as a percentage.

To use the PSNR calculator and Hamming Distance Calculator use the `stats` argument. An example of this is;

```
Java -jar bitmap_morph.jar stats c:\test\test.bmp c:\test\
test1.bmp
```

8.3 LSB Visual Enhancement

This will analyse the stegorgram and enhance it by turning it into a 9 colour image and expanding the least significant bit to show hidden data that could be present in this part of the image. The secret data will appear as noise in the enhanced image. To see this clearly it is best to use good

sharp images that have been taken in good conditions. Images that have been taken in poor conditions such as at night time will contain alot of noise in this part of the image reducing the effect of this technique. It is also possible to gain an understanding of the secret file if it is not encrypted. For instance uncompressed files such as text documents will contain a lot of black in the noise spectrum whereas images that contain compressed data such as zip files, jpeg images or avi videos will have hardly any black stripes in them. To encode data without encrypting it use the `encodenoenc` argument. An example of the effectiveness of this technique is demonstrated in Annex A. To use this technique the `expandlsb` argument can be used.

9. Examples of Use

Scenario 1; You need to encode a single file (hide.txt) into a bitmap image (steg.bmp) to transmit over the internet.

Navigate to the folder that contains Bitmap Morph via the command prompt.

Using arguments type;

```
Java -jar bitmap_morph.jar generatecode . 1 false
```

This will generate a stagger code and put it in the folder where Bitmap Morph is located. Share this stagger code with the receiver via a secure channel. In an ideal world the receiver would already have the stagger code so transmission would not be necessary. The stagger code that is generated is called 5412369.stg. Next type to encode the secret file;

```
Java -jar bitmap_morph.jar encode 5412369.stg c:\hide.txt c:\
steg.bmp
```

The file ABGH8YDC0122.bmp will be created and have the file hide.txt encoded into it.

When the receiver gets the file move it to the same directory as Bitmap Morph and type;

```
Java -jar bitmap_morph.jar extract 5412369.stg ABGH8YDC0122.bmp
```

The secret file will be extracted in this directory and given a random number as a filename with the correct file extension.

Using the CLI type;

```
Java -jar bitmap_morph.jar cli
```

To generate a **STAGGER CODE** type; (when the on screen prompt appears)

1 and press enter, then	(to generate a stagger code)
si and press enter, then	(to generate a single stagger code)
. and press enter, then	(to save the stagger code in this directory)
1 and press enter.	(only generate 1 stagger code)

A stagger code has now been generated and saved in the same directory as Bitmap Morph. To **ENCODE** the secret file type;

2 and press enter, then	(to encode a secret file)
n and press enter, then	(to use single stagger code)
5412369.stg and press enter, then	(**choose** the stagger code)
hide.txt and press enter, then	(select the secret file)
stego.bmp and press enter.	(select the carrier image)

The secret file will now be encoded. Once encoded send to the receiver.

When the receiver gets the image put it with the stagger code in the same directory as Bitmap Morph and type;

```
Java -jar bitmap_morph.jar cli
```

To **EXTRACT** the secret data type;

3 and press enter, then	(to extract a secret file)
5412369.stg and press enter, then	(select the stagger code)
ABGH8YDC0122.bmp and press enter.	(select the stegogram)

The secret file will now be extracted and saved in the same directory.

Scenario 2; You need to encode two files and publish the stegogram on an internet site. the receivers will then save the stegogram and extract the secret files meant for them. **It is recommended that the CLI should be used when encoding files using sequential stagger codes.**

WARNING: When using sequential stagger codes the original cover image is ALWAYS overwritten with the stegogram.

Navigate to the folder that contains Bitmap Morph via the command prompt.

Using arguments type;

```
Java -jar bitmap_morph.jar generateseq . 2 stego.bmp
```

*This will generate 2 stagger codes to be used with stego.bmp. These now need to be shared with the receivers. The first receiver is known as Bob and the second receiver is known as Dave. Bob is given stagger code 111111111.stg and Dave is given stagger code 222222222.stg. The file meant for Bob is called BobsSecret.txt and the file meant for Dave is called DavesSecret.txt. Stego.bmp is the carrier file. These stagger codes are paired with stego.bmp and should **not** be used for a different carrier file.*

Now encode the secret files by typing;

```
Java -jar bitmap_morph.jar encodeseq 111111111.stg BobsSecret.txt
stego.bmp 2
```

then type;

```
Java -jar bitmap_morph.jar encodeseq 222222222.stg DavesSecret.txt
stego.bmp 2
```

Both files are now encoded into the stegogram stego.bmp. This image can now be published to the website. Please note that the number 2 tells the encoder that 2 files will be encoded.

When Bob and Dave download the stegogram from the website they move it and the stagger code to the same folder as Bitmap Morph.

Bob types;

```
Java -jar bitmap_morph.jar extract 111111111.stg stego.bmp
```

Dave types;

```
Java -jar bitmap_morph.jar extract 222222222.stg stego.bmp
```

Both files are now extracted Bob has his file and Dave has his. Neither Bob or Dave knows that each other exist and it is impossible to see more than one secret file in the stegogram through steganayalsis.

Using CLI type;

```
Java -jar bitmap_morph.jar cli
```

When the on screen prompt appears type;

1 and press enter, then	(to generate a stagger code)
se and press enter, then	(to generate sequential stagger codes)
. and press enter, then	(to save the stagger code in this directory)
2 and press enter.	(generate 2 stagger codes)
stego.bmp press enter	(selects the carrier file)

Two sequential stagger codes are now generated. These stagger codes should be used with the selected carrier image only. If they are used with other images you must make sure that the image file size is identical otherwise errors will occur.

Once the codes are generated type;

2 and press enter, then	(select encode)
y and press enter, then	(type y to select sequential codes)
2 and press enter, then	(2 secret files are being encoded)
47227268.stg and press enter, then	(the first stagger code)
DavesSecret.txt and press enter, then	(the first secret file)
52198168.stg and press enter, then	(the second stagger code)
BobsSecret.txt and press enter, then	(the second secret file)
stego.bmp and press enter, then	(the cover image)

Both secret files are now encoded into the steogram which is ready to send to Bob and Dave. Once Bob and Dave receives the steogram they can decode their file using their stagger code.

When Dave receives the stegogram he puts it in the Bitmap Morph folder and types;

```
Java -jar bitmap_morph.jar cli
```

When the prompt appears he then types;

3 and press enter, then	(select extract)
47227268.stg and press enter, then	(select stagger code)
stego.bmp and press enter, then	(select stegogram)

The file DavesSecret.txt is now decoded and saved in the Bitmap Morph directory.

When Bob receives the stegogram he puts it in the Bitmap Morph folder and types;

```
Java -jar bitmap_morph.jar cli
```

3 and press enter, then	(select extract)
52198168.stg and press enter, then	(select stagger code)
stego.bmp and press enter, then	(select stegogram)

The file BobsSecret.txt is now decoded and saved in the Bitmap Morph directory.

10. Troubleshooting

The following error might occur if your java heap file is less than the memory needed to run Bitmap Morph.

```
"Exception in thread "main" java.lang.OutOfMemoryError: Java heap
space"
```

This will occur if you are working with large bitmaps or large secret data. To solve it you need to use the Xmx and Xms commands when running Bitmap Morph to increase the size of your heap file. For example if you type;

```
Java -jar -Xmx4G -Xms4G bitmap_morph.jar extract 222222222.stg
stego.bmp
```

This will set the heap size to 4GB which should handle bitmaps of sizes up to 1GB.

If errors still occur after typing the above command you may need to increase the size of your virtual memory (Windows) or page file (Unix) to a value larger than 4GB.

If you experience any other errors email the details to BitmapMorph@L33T.uk