Contents

[Theoretical Background and Concepts 3](#_Toc202861249)

[Unpacking Math Rounding Operations 3](#_Toc202861250)

[Unpacking math ceil() operation 3](#_Toc202861251)

[Unpacking math floor() operation 3](#_Toc202861252)

[Data Wrapping and Tokens 4](#_Toc202861253)

[AES-256 GCM encryption 4](#_Toc202861254)

[Seeded Byte Shuffling 4](#_Toc202861255)

[Reed Solomon Error Correction Codes 5](#_Toc202861256)

[Tokens 5](#_Toc202861257)

[Wrapping Length Equations 6](#_Toc202861258)

[LSB Steganography 8](#_Toc202861259)

[How Images are Represented in Bytes 8](#_Toc202861260)

[The Working Principle of LSB Steganography 8](#_Toc202861261)

[LSB Token Structure 9](#_Toc202861262)

[BPCS Steganography 9](#_Toc202861263)

[The CGC Binary System 9](#_Toc202861264)

[Bit Planes 9](#_Toc202861265)

[Bit plane complexity coefficients 10](#_Toc202861266)

[Conjugation of a bit plane 10](#_Toc202861267)

[The Working Principle of BPCS Steganography 11](#_Toc202861268)

[BPCS Token Structure 13](#_Toc202861269)

[LSB vs BPCS 13](#_Toc202861270)

[Speed 13](#_Toc202861271)

[Maximum Capacity 13](#_Toc202861272)

[Data Hiding Quality 14](#_Toc202861273)

[Overall 14](#_Toc202861274)

[Steganalysis Algorithms 15](#_Toc202861275)

[Bit Plane Slicing Algorithm 15](#_Toc202861276)

[Image Difference Calculation Algorithm 15](#_Toc202861277)

[The Combination of Both Algorithms 15](#_Toc202861278)

# Theoretical Background and Concepts

## Unpacking Math Rounding Operations

Over the course of the project, we will need to unpack the math ceil() and floor() operations so we can use them in algebraic equations for varying purposes.

### Unpacking math ceil() operation

The math.ceil() operation is used to round up non integers.

For every :

From this we know that the operation adds at most 1 to the original value. So, we can unpack the operation by overestimating and underestimating how much the operation added to the initial value depending on what is inside the operation and should we over/underestimate it.

### Unpacking math floor() operation

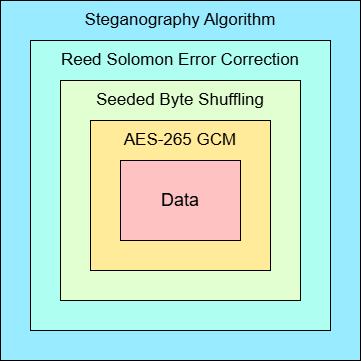
The math.floor() operation is used to round down non integers.

For every :

From this we know that the operation subtracts at most 1 from the original value. So, we can unpack the operation by overestimating and underestimating how much the operation subtracted from the initial value depending on what is inside the operation and should we over/underestimate it.

|  |  |  |
| --- | --- | --- |
|  | Overestimating | Underestimating |
|  |  |  |
|  |  |  |

## Data Wrapping and Tokens

This chapter will cover the varying stages of wrapping, unwrapping, error correction, data validation, encrypting, and decrypting that message data goes through before being embedded in an image and after being extracted from one. The following wrapping diagram describes the wrapping and steganography order:

Every wrapping operation outputs a token, this token contains all the encryption, error correction, and steganography parameters (if steganography is used). Meaning, for every steganography embedding operation there is a token that can be used to extract the data from the picture

### AES-256 GCM encryption

Galois/Counter Mode (GCM) is a mode of operation for symmetric-key cryptographic block ciphers which is widely adopted for its performance. GCM throughput rates for state-of-the-art, high-speed communication channels can be achieved with inexpensive hardware resources.

This wrapping layer encrypts the given data, and to decrypt it correctly, the following parameters all need to match those output / used in the encryption process.

1. Symmetric encryption key
2. Verification tag
3. Update header
4. Nonce

If any of these parameters change in any way between the encryption and decryption, the decryption will fail, and the output data will be flagged as incorrect.

### Seeded Byte Shuffling

This wrapping layer shuffles the bytes of the encrypted data using a seeded shuffler from Python’s random module.

The seed for shuffling consists of 64 bytes (512 bits) of data and is determined pseudo-randomly by Python’s random.randbytes function.

The odds of unshuffling the output without the key successfully is one in if the length of the seed is known (which it isn’t for the user) or one in , where n is the number of bytes shuffled if the length of the key isn’t known.

Even if a threat actor manages to unshuffle the bytes successfully, they won’t know it. This is because the layer before it is also encrypted so they’ll have to get pass that too before being able to check if the unshuffling was correct.

### Reed Solomon Error Correction Codes

Reed–Solomon codes are a group of error-correcting codes that were introduced by Irving S. Reed and Gustave Solomon in 1960. They have many applications, including consumer technologies such as MiniDiscs, CDs, DVDs, Blu-ray discs, and QR codes.

Reed–Solomon codes operate on a block of data treated as a set of finite-field elements called symbols. Reed–Solomon codes can detect and correct multiple symbol errors.

This wrapping layer acts as a semi-effective failsafe in case the data embedded in the image via steganography was changed (up to a certain point depending on the chosen parameters).

The parameters required to use this error correction code:

1. Block size; signifies the number of bits that each block is comprised of. This value ranges from 0 to 255 (). When the complexity and computational cost goes up quadratically. So, we limit the maximum block size to 255.
2. Symbol num; signifies how many bits of the block are dedicated for error correction and not data storage. This value ranges from 0 to (. This is because when the symbols take up more than half of the total block size, we account for more errors that we can have.

### Tokens

As previously mentioned, every time data gets wrapped a matching token is generated, and this token contains all the relevant info to unwrap the previously wrapped data.

Every parameter but the key has a constant length due to its' algorithm’s restraints:

1. Steganography Algorithm ID: 1 byte.
2. Block size: 1 byte ().
3. Symbol num: 1 byte ().
4. Verification tag: 16 bytes.
5. Nonce: 16 bytes.
6. Update Header: 8 bytes.
7. Byte Shuffling Seed: 64 bytes.
8. Encryption Key: dynamic length - will be concatenated to the end of the token so it will be the remaining bytes.

The structure of the token that will be outputted when using the steganography algorithms will be different than the above structure, because each steganography algorithm also uses other parameters. The additional parameters will be inserted between the byte shuffling seed and the encryption key.

### Wrapping Length Equations

In future parts of the project, we will need to accurately estimate the unwrapped length of a message when given all the parameters and the wrapped length. We can do this using simple algebra:

The AES-GCM algorithm adds and AAD (Additional Authenticated Data) that has a size of 16 bytes. So, the total length of the message is the initial length + the AAD length.

In every RS ECC block, we have a certain number of symbols. The more symbols we have per block, the less data we can fit in it.

*All lengths and sizes are in bytes, not bits.*

: The initial length of the message.

: The AAD length.

: The size of each RS ECC block.

: The number of Symbols in each RS ECC block.

: The final length of the message.

Total length of the message after AES:

Number of data bytes per RS ECC block:

Number of RS ECC blocks used:

The resulting function is : the length of unwrapped data as a function of : the wrapped length of said data.

## LSB Steganography

LSB steganography is a commonly used technique for hiding secret messages within images. It is widely known for its simplicity and low runtime even for big images. This chapter will overview the principle of LSB steganography, and the token structure that matches this steganography algorithm.

### How Images are Represented in Bytes

When we look at an image, we see a wide range of color depth and brightness, but how can computers represent these images as simple collections of bytes?

Each pixel in an image can be represented by 3 independent values:

1. How much red is in the pixel
2. How much green is in the pixel
3. How much blue is in the pixel

These values range from 0 to 255, so they take up 3 bytes together (one byte each).

Bytes are comprised of 8 bits each, and each bit has a different significance in the total value of the byte. Some bits can change the total value by 1 or 2, and others can change it by a half of the maximum value the byte can represent. Due to this behavior, if we want to hide data in the byte, it is our best interest to use the bits with the least significance – to cause as little change as possible in the overall value.

### The Working Principle of LSB Steganography

LSB steganography works by replacing the least significant bits of the vessel image with the secret message bits. The least significant bits of an image are the least important bits of the color values of each pixel and changing them is unlikely to cause a noticeable change in the image.

Unlike LSB scripts on the internet, my implementation allows the user to choose how many bits of a byte to sacrifice for data storage. Apart from this difference, my LSB implementation is straightforward and semi-similar to those on the internet.

### LSB Token Structure

As mentioned previously, each token generated when using steganography algorithms has an extra parameter:

For LSB steganography, the user has an option to choose how many bits of one byte to ‘sacrifice’ for data storage, even though it only ranges from 1-8 and can be represented with only 3 bits, the module I use to write these tokens into a file can only write bytes, and not bits. So, its size is the minimal size one byte.

## BPCS Steganography

BPCS (Bit Plane Complexity Segmentation) steganography is a unique steganography method that can use a higher percentage of the vessel image’s data to hide data than other known steganography algorithms. This is because the principle of those techniques was either to replace a special part of the frequency components of the vessel image (JPEG DCT methods), or to replace all the least significant bits of a multivalued image with secret information (LSB methods).

This chapter covers the concepts that are the foundation of BPCS steganography, the working principles of BPCS steganography, and the token structure that matches my derivative of BPCS steganography.

### The CGC Binary System

The reflected binary code (RBC), also known as reflected binary (RB), Canonical Gray Code (CGC), or Gray Code after Frank Gray, is an ordering of the binary numeral system such that two successive values differ in only one bit (binary digit). This binary system can minimize the impact of Hamming Cliffs. This binary system is used in BPCS steganography.

For example, the representation of the decimal value "1" in binary would normally be "001", and "2" would be "010". In Gray code, these values are represented as "001" and "011". That way, incrementing the value from 1 to 2 requires only one bit to change, instead of two.

### Bit Planes

Bit planes (also known as binary images) are images of any size where each bit can be represented with only one bit. Black pixels are represented by a 0, and white pixels are represented by a 1.

Each pixel in a 24-bit image is made up of 24 independent bits (hence the name 24-bit image).

We can use this fact to split an image into 24 bit planes, we do this by slicing the first bit of each pixel and placing it in the same pixel coordinates in a binary image, then slice the second bit of each pixel and place it in the same pixel coordinate in another binary image, and so on – for each bit place in the image.

### Bit plane complexity coefficients

A core concept in one of the steganography algorithms this paper covers is measuring how complex a bit plane of finite size is. We can quantify how complex is an image with a “complexity coefficient” that is marked . The complexity coefficient can be thought of as a quantification of how informative a bit plane is. If the complexity coefficient is low, then the bit plane is informative, if it is high, then the bit plane is noisy and it’s harder for our eyes to detect differences between bit planes that have such complexity coefficients.

As stated in *Principle and applications of BPCS-Steganography*:

“The length of the black-and-white border in a binary image is a good measure for image complexity. If the border is long, the image is complex, otherwise it is simple. The total length of the black-and-white border equals to the summation of the number of color-changes along the rows and columns in an image. For example, a single black pixel surrounded by white background pixels has the boarder length of 4.”

B.W border length of a bit plane can be calculated by iterating through all the rows and columns and counting how many times the bit shifted value. The time complexity of this algorithm is for a bit plane of dimensions .

The maximum B.W border length of a bit plane of dimensions is .

We can calculate with the following formula:

### Conjugation of a bit plane

I find that the explanation in *Principle and applications of BPCS-Steganography* explains this section perfectly. So, this section will be almost identical to the one mentioned above, besides some small modifications that I’ll make to illustrate the points made better.

A black and white squares

AI-generated content may be incorrect.Let P be a by size black-and-white image with black as the foreground area and white as the background area. W and B denote all-white and all-black patterns, respectively. We introduce two checkerboard patterns Wc and Bc, where Wc has a white pixel at the upper-left position, and Bc is its complement, i.e., the upper-left pixel is black. We regard black and white pixels as having a logical value of “1” and “0”, respectively.

P is interpreted as follows. Pixels in the foreground area have the B pattern, while pixels in the background area have the W pattern. Now we define P\* as the conjugate of P which satisfies:

1. The foreground area shape is the same as P.
2. The foreground area has the Bc pattern.
3. The background area has the Wc pattern.

The following properties hold true and are easily proved for such conjugation operation. “” designates the exclusive OR operation.

1. P\* = P Wc
2. (P\*)\* = P
3. P\* P

The most important property about conjugation is the following.

1. Let (P) be the complexity of a given binary image P, then we have; (P) = 1 – (P\*)

It is evident that the combination of each local conjugation (e.g., 8 by 8 area) makes an overall conjugation (e.g., 512 by 512 area).

1. states that every binary image pattern P has its counterpart P\*. The complexity value of P\* is always symmetrical against P regarding a = 0.5. For example, if P has a complexity of 0.7, then P\* has a complexity of 0.3.

### The Working Principle of BPCS Steganography

As the name suggests, BPCS steganography is a method to separate (or segment) bit planes by their complexity. Unlike other steganography algorithms that look onto each byte as an independent unit of data – this principle considers structured blocks of bits as the main unit of data. This working principle of this algorithm exploits how the human eye processes random binary patterns to hide information.

A close-up of a black and white pattern

AI-generated content may be incorrect.Consider the following example:

These images are of two bit planes that I generated at random using a program I found online.

If I’d ask you to describe how these bit planes appear as accurately as possible, it’ll be hard. That’s because these bit planes appear to us as noise, we can’t identify them individually because we can’t extract any identifying information from them. We can’t differentiate between them as effectively compared to how we differentiate images of faces. This is because when we look at a face, we can extract data and attributes from it, but when we try to do the same with patterns that appear to us as random, we can’t.

BPCS steganography works by replacing bit planes that appear to us as uninformative and random with other bit planes that contain the data we want to hide. Now all we need is to find a method to quantify how much information and significance each bit plane has to an image.

As explained previously, every bit plane has a complexity coefficient. Using this new attribute, we can now quantify how informative a bit plane is – and by that; how important it is to an image. The size for every bit plane that we replace is 8x8, this size was chosen because it balances the number of segmented bit planes in each image with the number of different options we have for their complexity coefficients. Because the segmented bit planes are of a constant size and they make up the image, I will refer to them as blocks.

To get all the less informative (or noisy) blocks in an image we must select a minimum complexity coefficient that will be used as a threshold value for filtering blocks, this value is marked as . If the complexity coefficient of a block is below , then it is too informative and cannot be used for embedding data, if it is equal or higher than , the block will be marked as accepted and will be able to be used for embedding data.

The minimum complexity coefficient is used not only for filtering the accepted blocks when embedding data, but also for filtering the accepted blocks that might have data hidden in them when extracting data from the image. This means that every time we alter an accepted block, we need to make sure that the complexity coefficient after modifying also qualifies as acceptable by .

The issue that arises when embedding data is that we can’t fine tune the complexity coefficients of bit planes. We can solve this problem by using the conjugation operation that was shown before.

In a case that the complexity coefficient of a block after modification is less than the determined , we can conjugate the block and the complexity coefficient after conjugation will be 1 – (P); as mentioned in the section explaining the conjugation operation. When we conjugate a block, we record it in a conjugation map so that when we extract the embedded data, we know which blocks were conjugated.

Because of the nature of the conjugation principle, must be in the range . If it’s not, then a range of complexity coefficients that won’t qualify as accepted even after conjugation is formed.

The embedding process of BPCS steganography is as follows:

1. Transform the vessel image from PBC to CGC system.
2. Segment each bit-plane of the vessel image into informative and noise-like regions by using a threshold value (). A typical value is .
3. Group the bytes of the secret file into a series of secret blocks.
4. If a secret block (B) is less complex than the threshold (), then conjugate it to make it a more complex block (B\*). The conjugated block must be more complex than as shown previously.
5. Embed each secret block into the noise-like regions of the bit-planes. If the block is conjugated, then record this fact in the conjugation map.
6. Also embed the conjugation map as was done with the secret blocks.
7. Convert the embedded dummy image from CGC back to PBC.

The decoding algorithm is the reverse procedure of the embedding steps.

In later phases of the project, I altered step 4 because it caused a pattern analysis vulnerability. The reason and specifics for this alteration are detailed in the problems with interesting solutions chapter of the paper.

### BPCS Token Structure

As mentioned previously, each token generated when using steganography algorithms has an extra parameter:

The minimum complexity coefficient; . This parameter is a floating-point number and thus represented by 8 bytes in memory. We can use the same number of bytes to save it into a token.

## LSB vs BPCS

The LSB and BPCS steganography methods differ from each other in many ways, the most notable ones are speed, maximum embedding capacity, and the ability to hide data effectively.

### Speed

Because BPCS’s core principle is to consider certain blocks of data as singular units and iterate through the whole image and test each bit plane block for with the threshold before even changing any data, it’s almost always much slower than LSB. This is because LSB doesn’t need to calculate anything, it just starts to change the data.

Speed-wise, LSB wins.

### Maximum Capacity

LSB’s maximum capacity in each image is easy to calculate, you need the number of pixels in each image, the number of channels in each pixel, and the number of bits you are willing to use for data storage in each channel. The maximum capacity of LSB without changing the image too much for the changes to be noticed is about one fourth of the total uncompressed image size.

Estimating the maximum capacity for the BPCS principle is trickier. Because BPCS uses a criterion to filter out the blocks that shouldn’t be changed in an image, it has a per-image capacity. If the image is computer generated, the image will likely have very low complexity blocks, because normally CGI doesn’t generate noise where real photos do have them. We can see this in the following pictures. The first image is a fully CGI image of a tree floating in some space. The second image is the bit plane of the least significant bit in the green channel. When computers generate imagery, they don’t add noise unless they are programmed to – and most people choose the A black and white image of a wave

AI-generated content may be incorrect.simple and better-looking solution of having a gradient or even the same color.

In real images the first few (1-4) bit planes are mostly comprised of pure noise. This is because in real life there are impurities and causes that can affect the color of a pixel when taking the image even in the most controlled environments.

Capacity-wise (for non-CGI), BPCS wins.

### Data Hiding Quality

As previously presented, LSB’s hiding method is simple, changing the bits with the least significance to the image. LSB makes very small changes, but it changes the whole image disregarding the adjacent bits in the image. Meaning, if there is a portion of an image that is expected to have one solid color (for example the above computer-generated image), after LSB’s changes we could use steganalysis algorithms to see that it isn’t one solid color – even if our eyes can’t.

Because BPCS’s method can filter out the block who are ‘informative’ in the image, with a proper it won’t change how the human eye sees the image. And because it only changes noise only, we won’t be able to detect that BPCS was used on the image without the original copy. Of course, this is when not pushing the method to its limits (hiding the maximum capacity possible and using a low that will allow for informative blocks to be accepted for changing data).

Where LSB might fail, BPCS won’t – because it won’t accept the bit plane blocks unless they pass the threshold.

Quality-wise, BPCS wins.

### Overall

Considering all the brought-up points, I would choose a BPCS based method over an LSB based one, this is because when doing steganography, the things that matter most are hiding quality and hiding capacity – not speed.

## Steganalysis Algorithms

### Bit Plane Slicing Algorithm

As explained previously, each image can be separated into 24 separate binary images (or bit planes) that represent the composition of bits in the image. Using this algorithm, we can observe the layers of bits in each channel.

### Image Difference Calculation Algorithm

As the name suggests, this algorithm can calculate the exact difference between two given images of same size. Additionally, I implemented this algorithm to accept a Boolean parameter, if it’s set to true, the algorithm calculates the exact difference between two images, if it’s set to false, the algorithm highlights every pixel that is different in the channel it differs in.

Meaning, for the pixel combination (20, 50, 100) and (40, 50, 180), the resulting highlighted pixel will be (255, 0, 255). Because the pixels differ from each other in the red and green channels.

### The Combination of Both Algorithms

Using both algorithms, we can calculate the difference between two images and analyze how every bit changed. By doing this, we might uncover underlying patterns or steganography weaknesses.