Image and Video Steganography Tool and Web Application

COMP 8037 – Major Project Proposal

Yashar Nesvaderani – A00984009
12-1-2023

Table of Contents

1. Student Background 2
   1.1. Education 2
   1.2. Work Experience 2
2. Project Description 2
3. Problem Statement and Background 3
4. Scope and Depth 3
   4.1 Required Scope 4
   4.2 Out of Scope 4
5. Test Plan 4
   5.1 Regression Testing 4
   5.2 Acceptance Testing 5
6. Methodology 6
   6.1 Methodology and Approach 6
   6.2 Algorithms 7
   6.3 Technologies Involved 14
7. System/Software Architecture Diagram 14
   7.1 Steganography Tool State Machine 14
   7.2 Basic System Overview 15
8. Innovation 16
9. Complexity 16
10. Technical Challenges 16
11. Development Schedule and Milestones 17
12. Deliverables 18
13. Conclusion and Expertise Development 18
14. Works Cited 19
15. Change Log 20

1. Student Background

Yashar Nesvaderani is a student currently in the second year of the Computer Systems - Network Security Applications Development Option Bachelor of Technology program.

1.1. Education

British Columbia Institute of Technology

- Computer Systems Technology Diploma - Information Systems Option
  - Graduated May 2018

1.2. Work Experience

Software Engineer, Gluwa (Jun 2018 — Jul 2020)

- Developed cross-platform mobile application for clients to manage, deposit and transfer their Gluwacoin cryptocurrency funds using Xamarin Forms.
- Maintained and updated RESTful Gluwacoin internal API in .NET Core.
- Designed and implemented automated C# tests for custom Creditcoin blockchain network.
- Implemented ERC-20 compliant Creditcoin ethereum smart contract.

Development Team Lead, POCU Academy (Jul 2020 — Mar 2021)

- Planned and managed Agile sprints for a small team of developers.
- Designed and implemented automated testing for student coding assignments in C, C++, Java, and C#.
- Planned, developed, tested, deployed, and maintained web application back and front-end for students to enroll and interact with courses and exams using .NET Core, CSHTML, and JavaScript.

2. Project Description

The project will be a software application that will implement multiple steganographic techniques, such as Least Significant Bit (LSB) and Pixel Value Differencing (PVD) and combine these techniques with image Edge Detection to hide and encrypt various types of data in images and the frames of videos. The OpenCV library will be used to manipulate the pixels of images and videos. The images and videos containing said data can then be read and decrypted by the program to retrieve the data hidden inside. Users will also be able to combine certain methods for more effective hiding of data. The application will be developed with C++ and allow the user to hide and retrieve data in images and videos by selecting the algorithms of their choice through a simple GUI.

In addition to the local steganography tool, a web application for users to manage and view their steganographic images and videos will be developed. The website will be developed using ASP.NET Core, will store the images and videos in a SQL Server database, and provide account management through built-in ASP.NET Core Identity and Security features. This will allow users to safely keep their steganographic images and videos in a centralized location for easy access. Using a RESTful API provided by the web application, the images/videos will also be able to be uploaded after being encoded with data, as well as downloaded for decoding of data.

3. Problem Statement and Background

Steganography is a general term used for the hiding of information or messages inside another medium or message. In digital mediums, there are multiple methods and algorithms for hiding information in different types of files while avoiding detection or compromising the integrity of the original file.

One of the simplest types of steganography in computer science is hiding data by changing the least significant bits (LSB) of each pixel in an image to the bits of your data. For instance, if a pixel consists of three bytes, each representing red, green, and blue colour values, changing the last bit of each byte to the values of your data will not change the colour of the pixel significantly. While this is easy to implement and hard to detect with the naked eye, if the number of bits changed per pixel is low, it is highly vulnerable to steganographic detection techniques.

Pixel Value Differencing (PVD) is a more complicated method that compares values between different pixels in an image to decide on the amount of data to hide in those pixels. Pixels are arranged into blocks, the difference between the colour values of the pixels in each block is calculated, and more or less data can be hidden in these pixels depending on the degree of difference (Wu and Tsai). PVD is harder to detect than LSB but can lead to a greater degradation of image quality.

Finally, edge detection is used to determine where the edges of an image are, as these are areas of an image where changes to the pixels will be less noticeable since they tend to be visually noisier (Vishnu et al.).

By allowing users to choose the steganographic algorithm to use, as well as whether to apply those algorithms to the edges of the images or not, the hidden data will be more difficult to detect, both by the naked eye and by existing steganographic analysis techniques. Additionally, by supporting video and inserting data into each frame, more data can be hidden than just in a single image.

The GUI of the program will allow users to choose the image/video to hide data in, the file to hide, the steganographic algorithm (LSB or PVD), the type of edge detection (None, Canny, or Sobel), and set a password for encryption of the data.

The website will let users create accounts to access their images from anywhere they are logged in. This will allow the users to keep track of the steganographic images/video they have created and provide easy access when they want to extract data from the images/videos at a later time. Additionally, as the website will appear as a normal image/video hosting site, the true purpose of storing hidden data is obfuscated.

4. Scope and Depth

The scope of this project is to create an application that will allow the user to select multiple steganographic algorithms, combined with edge detection, to hide data in images and the frames of videos. In concert with the local tool, a website that allows users to create accounts to store/retrieve these images and videos will be created.

4.1 Required Scope

As such, the following functionality is required:

- The application will support both the Least Significant Bit (LSB) and Pixel Value Differencing (PVD) algorithms for the hiding of data.
- The application will support the Canny and Sobel edge detection algorithms.
- The application will encrypt the data using AES 256 encryption.
- The application will at least support 24-bit PNG images.
- The application will support at least .mp4 video files.
- The application will support the hiding of any type of data (images, documents, videos, etc.)
- The application will have a GUI to allow the selection of algorithms, images/videos to hide data in, and the data to be hidden itself.
- The website will allow users to create/manage their accounts.
- The website will allow users to download, upload, view, or delete images/videos.
- The website will allow users to sort and filter images/videos based on name and size.
- The application will allow users to directly upload images/videos or download images/videos to/from the website after providing login credentials to the application and saving an auth token.

4.2 Out of Scope

The following functionality will not be included:

- The application will not be cross-platform.
- The application will not support all image and video types for embedding data in.

5. Test Plan

Testing for this project will consist of regression and acceptance testing and will be carried out through both manual and automated testing.

5.1 Regression Testing

Regression tests will be carried out with automated unit testing and will include:

- Testing each algorithm to ensure it is implemented properly.
- Testing encoding/decoding various file formats.
- Testing encoding data in images.
- Testing encoding data in videos.
- Testing all UI functionality works as expected.

5.2 Acceptance Testing

Acceptance testing will consist of ensuring the user is able to complete all expected use cases of the application. Some examples are:

Case	Select an image or video for encoding/decoding.
Verification	The image or video file can be seen loaded into the application.
Pass Condition(s)	Image or video is successfully selected and loaded into application.
Fail Condition(s)	Image or video fails to be selected and loaded into application.

Case	Least Significant Bit (LSB) algorithm can be selected and properly applied.
Verification	1. Apply LSB algorithm to the image or video. 2. Confirm that the image or video is visually the same. 3. Confirm that the embedded file can be retrieved from the encoded image.
Pass Condition(s)	The LSB is applied to the encoding when selected as an option.
Fail Condition(s)	The LSB is not applied to the encoding or fails to work when selected as an option.

Case	Pixel Value Differencing (PVD) algorithm can be selected and properly applied.
Verification	1. Apply PVD algorithm to the image or video. 2. Confirm that the image or video is visually the same. 3. Confirm that the embedded file can be retrieved from the encoded image or video.
Pass Condition(s)	The PVD is applied to the encoding when selected as an option.
Fail Condition(s)	The PVD is not applied to the encoding or fails to work when selected as an option.

Case	Steganographic algorithms can be combined with Canny edge detection.
Verification	1. Apply stenographic and Canny algorithm to the image or video. 2. Confirm that the image or video is visually the same. 3. Confirm that the embedded file can be retrieved from the encoded image or video.
Pass Condition(s)	The steganographic algorithm is applied only to provided edges from Canny algorithm.
Fail Condition(s)	The steganographic algorithm is not applied only to provided edges from Canny algorithm.

Case	Steganographic algorithms can be combined with Sobel edge detection.
Verification	1. Apply stenographic and Sobel algorithm to the image or video. 2. Confirm that the image or video is visually the same.

	3. Confirm that the embedded file can be retrieved from the encoded image or video.
Pass Condition(s)	The steganographic algorithm is applied only to provided edges from the Sobel algorithm.
Fail Condition(s)	The steganographic algorithm is not applied only to provided edges from the Sobel algorithm.

Case	Any type of data can be embedded in an image or video.
Verification	The program will be run with multiple file types (text, images, audio, video, etc.) to see if the files can be successfully hidden and then retrieved without affecting the files hidden.
Pass Condition(s)	Multiple file types can be stored and retrieved from the given images and videos.
Fail Condition(s)	Multiple file types cannot be stored and retrieved from the given images and videos.

Case	Upload an image or video to the website
Verification	The image or video file can be seen on the website.
Pass Condition(s)	Image or video is successfully uploaded to the website.
Fail Condition(s)	Image or video fails to be uploaded to the website.

Case	Download an image or video from the website
Verification	The image or video file is downloaded to the local machine.
Pass Condition(s)	Image or video is successfully downloaded to the local machine.
Fail Condition(s)	Image or video fails to be downloaded from the website.

Case	Delete an image or video to the website
Verification	The image or video file can be no longer seen on the website.
Pass Condition(s)	Image or video is successfully deleted to the website.
Fail Condition(s)	Image or video fails to be deleted to the website.

6. Methodology

6.1 Methodology and Approach

The student will be using an Agile approach to development, working on one main feature of the application at a time. Tasks will be broken down into achievable chunks in order for them to be completed in a single sprint (two weeks) and managed in a backlog on a Trello board.

Each feature will go through the following stages of development:

- Requirements Gathering
- Planning

- Design
- Implementation
- Testing
- Documentation

6.2 Algorithms

The steganographic algorithms that will be implemented are the Least Significant Bit (LSB) and Pixel Value Differentiation (PVD) algorithms. LSB simply replaces the X least significant bits (the bits that would have the least affect on the colour of the pixel if they were changed) of each pixel in an image or frame of a video with the bits of the data to be hidden.

PVD calculates the differences between pixels in blocks of the image to determine exactly how many bits can be changed without affecting the image noticeably (Wu and Tsai). To account for the multiple colour values found in RGB, as well as the case where inserting data would lead to values outside of the acceptable RGB range (0-255), the PVD algorithm proposed is as follows (Mandal and Debashis):

PVD Insertion Design:

1. Split each colour element of the image/frame into separate matrices (1 Red, 1 Green, 1 Blue).
2. Determine the difference between each consecutive block of two elements in each colour matrix. ($d_i=|p_i - p_{i+1}|$)
3. Using the calculated difference, get the number of bits of data, *t*, that can be inserted into the colour value from $\log_2(u_i – l_i + 1)$, where u_i and l_i are the upper bound and lower bounds of the range d_i falls into. (The range table for this will use the ranges 0-7, 8-15, 16-31, 32-63, 64-127, and 128-255)
4. Convert *t* bits of the data to hide into a decimal value and add it to the lower bound of the range, l_i to get the new difference value, d_i'. ($d_i' = l_i + d_i$)
5. Calculate the new pixels after inserting data into the block of elements using the following formulae (where $m = |d_i'-d_i|$):
    a. $(p_i', p_{i+1}') = (p_i + \lceil m/2 \rceil , p_{i+1} - \lfloor m/2 \rfloor)$ if $p_i \geq p_{i+1}$ and $d_i' > d_i$
    b. $(p_i', p_{i+1}') = (p_i - \lfloor m/2 \rfloor , p_{i+1} + \lceil m/2 \rceil)$ if $p_i < p_{i+1}$ and $d_i' > d_i$
    c. $(p_i', p_{i+1}') = (p_i - \lceil m/2 \rceil , p_{i+1} + \lfloor m/2 \rfloor)$ if $p_i \geq p_{i+1}$ and $d_i' \leq d_i$
    d. $(p_i', p_{i+1}') = (p_i + \lceil m/2 \rceil , p_{i+1} - \lfloor m/2 \rfloor)$ if $p_i < p_{i+1}$ and $d_i' \leq d_i$
6. Check the new pixel values to see if they exceed the limit of 255 for 8 bits.
    a. If it does not, go to step 7.
    b. If it does, look at the most significant bit (MSB) of the data to be inserted.

6.1 If the MSB is 1, discard it and calculate the new decimal value from the remaining bits and calculate the new pixel values as in steps 4 and 5. If the new pixel values are in the limit, go to step 7.

6.2 If the MSB was 0 or the pixel value was still above the limit after removing the MSB of 1, store the data with the following formulae:

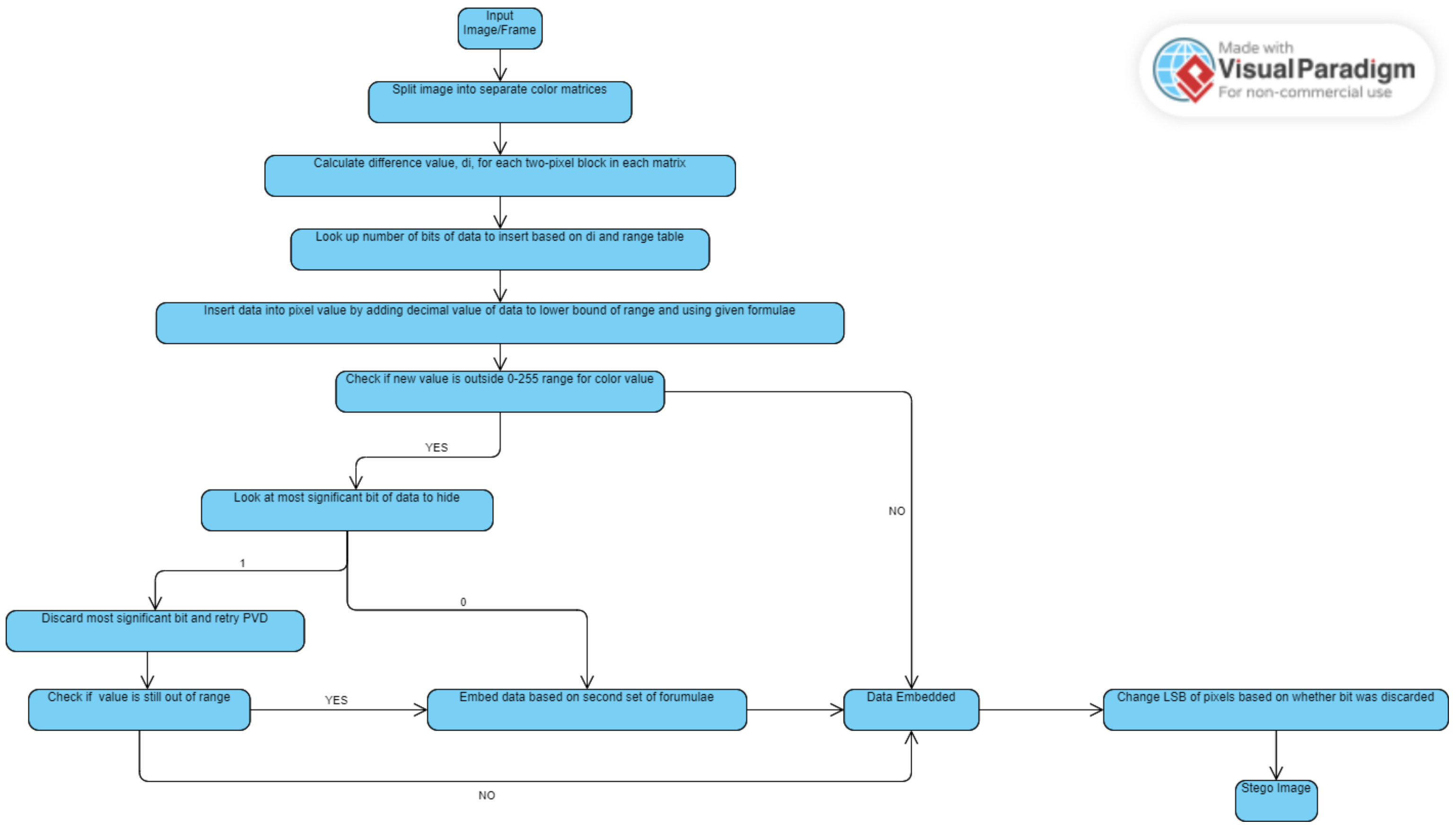
   a. $(p_i', p_{i+1}') = (p_i - m, p_{i+1})$ if $p_{i+1} \geq p_i$ and $p_{i+1} > 255$ (after insertion)

   b. $(p_i', p_{i+1}') = (p_i, p_{i+1} - m)$ if $p_{i+1} < p_i$ and $p_i > 255$ (after insertion)

   c. $(p_i', p_{i+1}') = (p_i, p_{i+1} + m)$ if $p_{i+1} \geq p_i$ and $p_i < 0$ (after insertion)

   d. $(p_i', p_{i+1}') = (p_i + m, p_{i+1})$ if $p_{i+1} < p_i$ and $p_{i+1} < 0$ (after insertion)

7. Replace the original pixel values with the new pixel values.
8. To keep track of whether any bits were discarded:
   a. If no bits were discarded:
      i. If both the LSB of p_i' and p_{i+1}' are 0, add 1 to p_{i+1}'
      ii. If the LSB of p_i' is 0 and p_{i+1}' is 1:
         1. If $p_{i+1}' < 255$ and $p_i' \geq 0$, add 1 to p_{i+1}'
         2. If $p_i' > 0$ and $p_{i+1}' = 255$, $p_i' – 2$ and $p_{i+1}' – 1$
         3. If $p_i' = 0$ and $p_{i+1}' = 255$, add 1 to p_i'
      iii. If the LSB of p_i' is 1 and p_{i+1}' is 0, subtract 1 from p_i'
      iv. If both the LSB of p_i' and p_{i+1}' are 1, subtract 1 from p_i'
   b. If a bit has been discarded:
      i. If both the LSB of p_i' and p_{i+1}' are 0, add 1 to p_i'
      ii. If the LSB of p_i' is 0 and p_{i+1}' is 1, add 1 to p_i'
      iii. If the LSB of p_i' is 1 and p_{i+1}' is 0:
         1. If $p_{i+1}' > 0$ and $p_i' \leq 255$, subtract 1 from p_{i+1}'
         2. If $p_i' < 255$ and $p_{i+1}' = 0$, $p_i' + 2$ and $p_{i+1}' + 1$
      iv. If both the LSB of p_i' and p_{i+1}' are 1, subtract 1 from p_{i+1}'
9. All given blocks are now encoded with secret data.



Input Image/Frame
Split image into separate color matrices
Calculate difference value, di, for each two-pixel block in each matrix
Look up number of bits of data to insert based on di and range table
Insert data into pixel value by adding decimal value of data to lower bound of range and using given formulae
Check if new value is outside 0-255 range for color value
YES
NO
Look at most significant bit of data to hide
1
0
Discard most significant bit and retry PVD
Check if value is still out of range
YES
NO
Embed data based on second set of forumulae
Data Embedded
Change LSB of pixels based on whether bit was discarded
Stego Image
Made with Visual Paradigm For non-commercial use

Diagram of the steps involved in the PVD insertion algorithm.

PVD Extraction Design:

1. Split each colour element of the image/frame into separate matrices (1 Red, 1 Green, 1 Blue).
2. For each block, when calculating the difference value, if the LSB of p_i is 0, add one to it. If the LSB is 1, subtract one.
3. Determine the difference between each consecutive block of two elements in each colour matrix. ($d_i=|p_i - p_{i+1}|$)
4. Find the range used for the block based on the difference value to get the lower bound of the range, l_i.
5. Get the hidden data by subtracting l_i from d_i and converting the result to binary.
6. Look at the LSB of p_i. If it is 1, replace the MSB of the extracted bits with 1.
7. Once all blocks are processed, the data has been retrieved.

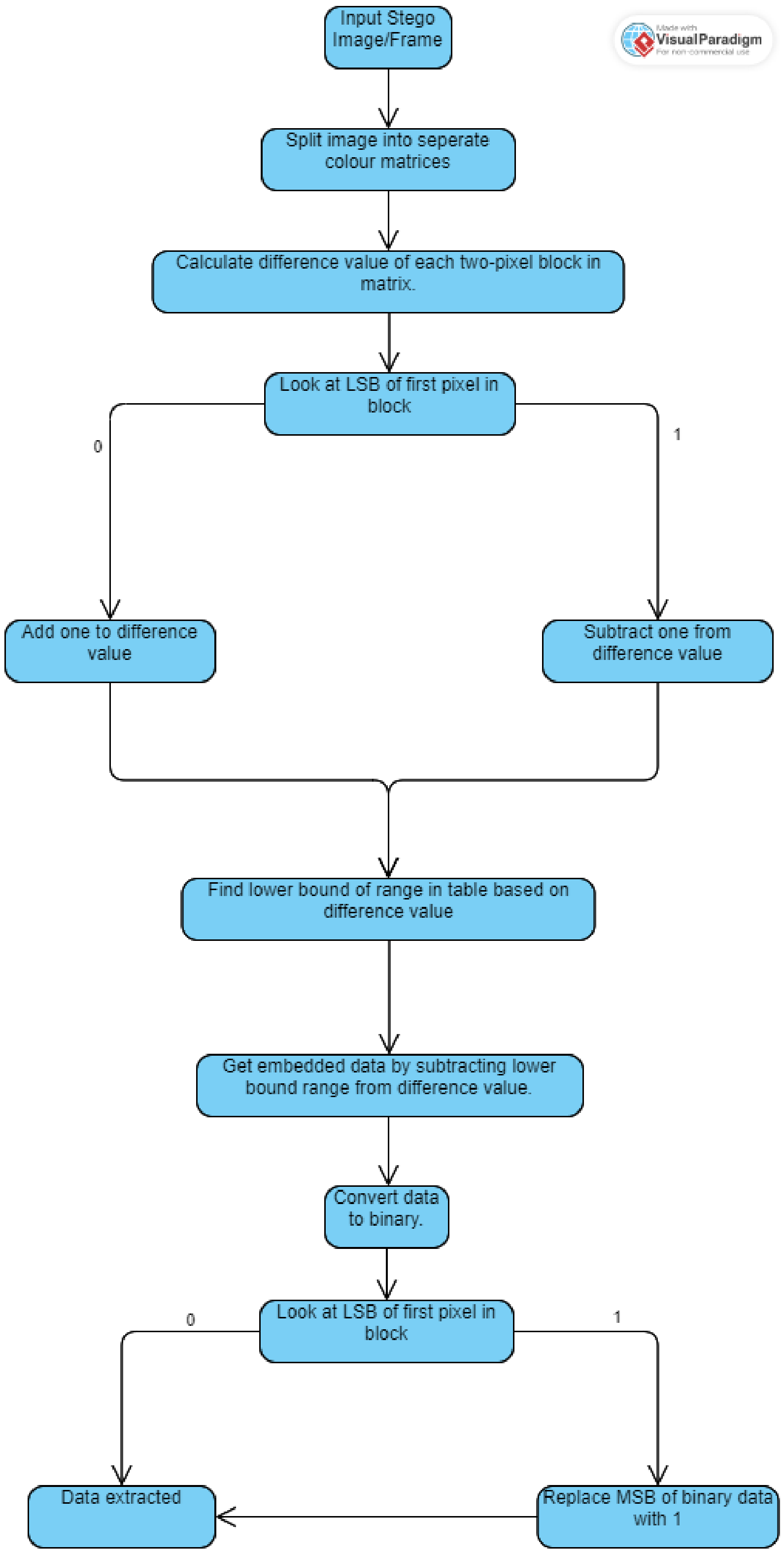

Input Stego Image/Frame
Made with Visual Paradigm For non-commercial use
Split image into seperate colour matrices
Calculate difference value of each two-pixel block in matrix.
Look at LSB of first pixel in block
0
1
Add one to difference value
Subtract one from difference value
Find lower bound of range in table based on difference value
Get embedded data by subtracting lower bound range from difference value.
Convert data to binary.
Look at LSB of first pixel in block
0
1
Data extracted
Replace MSB of binary data with 1

Diagram of the steps involved in the PVD extraction algorithm.

In addition to the steganographic algorithms above, the Sobel and Canny edge detection algorithms will also be implemented to find noisy edges in the images most suitable for hiding data. Sobel edge detection works by convolving an image with a vertical and horizontal kernel to obtain gradient. The magnitude of the gradients is then calculated, as well as their directions to find the final edges. (Vincent and Folorunso). Canny edge detection works by first applying a Gaussian filter to remove image noise, finding the intensity gradients in the image using another detector such as Sobel, removing edge pixels that are less intense than their neighbours, validating the edges as strong, weak, or invalid against given thresholds, and removing any weak edges not connected to strong edges. (Canny).

Canny Edge Detection Design:

1. Apply Gaussian Filter to image to smooth image out.
    a. Convolve the image/frame with precalculated 5x5 Gaussian Kernel (see below for kernel).
        i. Overlay the kernel on each pixel in the image.
        ii. Calculate weighted average for the pixel based on surrounding pixel values (for each RGB value) multiplied by the kernel values in the same position. (For pixels where the overlay of the kernel is outside the bounds of the image, mirror the pixels on the inside to corresponding outside positions for calculating average.)
        iii. Set the pixel value to the calculated value.
2. Locate the Intensity Gradients.
    a. Convolve the image/frame with a pair of Sobel Kernels, G_x and G_y (see below for kernels).
        i. Overlay G_x and G_y kernels on each pixel.
        ii. Calculate the x and y weighted value based on surrounding pixel values (for each colour value of the pixels) multiplied by the kernel values in the corresponding positions.
        iii. Calculate the overall magnitude for each colour of the pixel G = $\sqrt{G_x^2 + G_y^2}$ and store the largest value between the colour magnitudes to get the intensity gradient for that pixel.
        iv. Calculate and store the direction of the pixel gradient with $\theta = \text{atan2}(G_y, G_x)$, rounded to the nearest direction (0, 45, 90, 135 degrees for vertical, horizontal, and both diagonals, respectively).
3. Gradient Magnitude Thresholding.
    a. Compare the pixel magnitude of each pixel to the pixels in the positive and negative directions of the angle.
    b. If the pixel magnitude is lower than either adjacent pixel, suppress the pixel by setting it 0 (black).
4. Apply high and low thresholds to categorize strong, weak, and further unwanted edges.
    a. Compare each pixel to previously set high and low threshold values.
    b. If pixel value is higher than high threshold, mark it as a strong edge pixel.
    c. If pixel value is between high and low thresholds, set it as a weak edge pixel.
    d. If pixel value is lower than the low threshold, suppress the pixel by setting it to 0 (black).
5. Apply hysteresis to remove weak edges that are not connected to strong edges.

    a. Look at each remaining weak edge pixel.
    b. If the weak edge is connected to a strong edge, it is marked as a strong edge.
    c. If not, suppress the weak edge pixel by setting it to 0 (black).
6. The result is an image only containing the strong edges in which the positions of those edges can be used for embedding data in the original image.

5x5 Gaussian Kernel to be used in step 1 of the Edge Detection algorithm:

$$\frac{1}{256} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

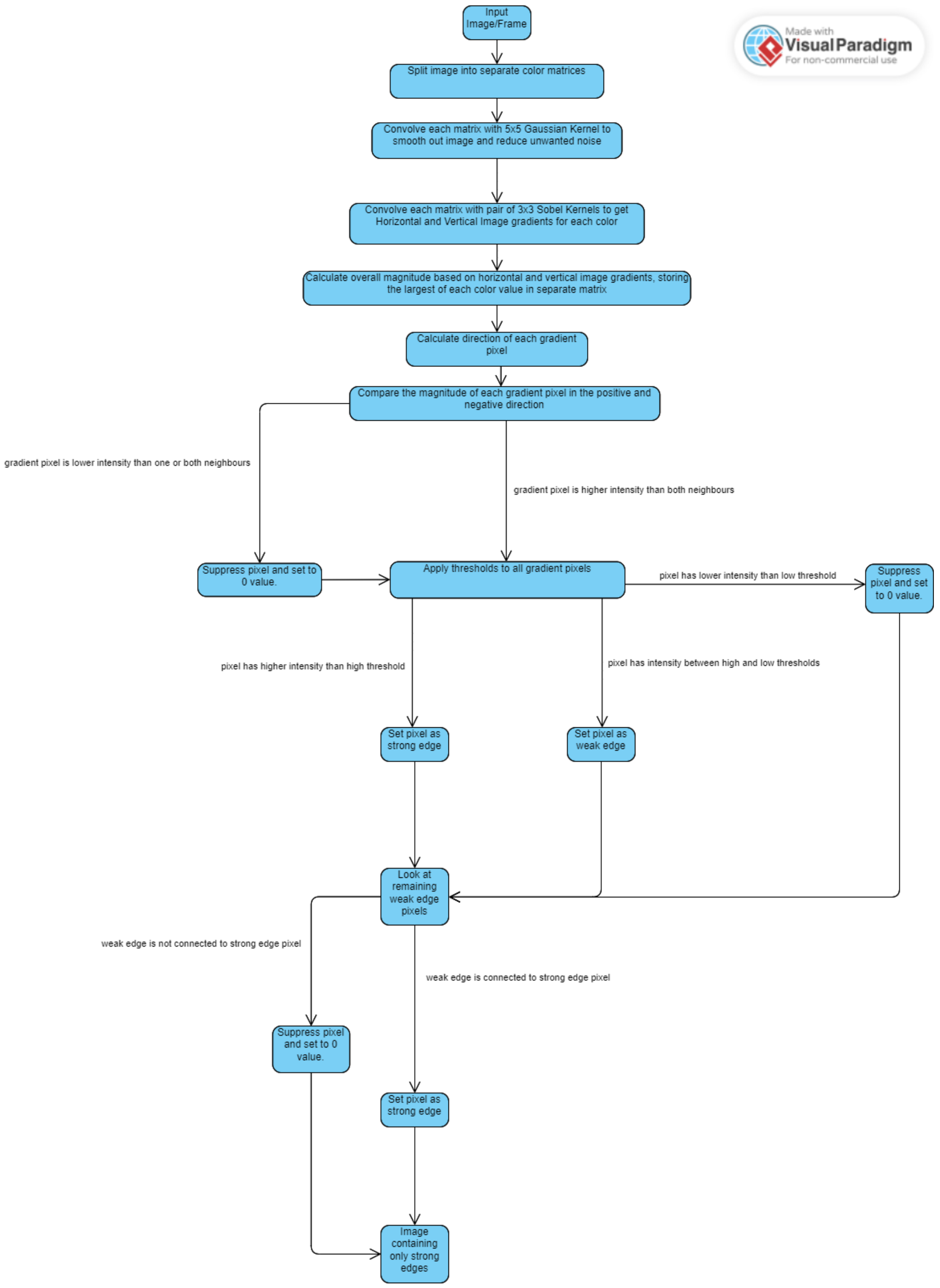
Sobel Kernels to be used in step 2 of the Edge Detection algorithm:

G_x =

-1	0	1
-2	0	2
-1	0	1

G_y =

1	2	1
0	0	0
-1	-2	-1


Made with Visual Paradigm For non-commercial use
Input Image/Frame
Split image into separate color matrices
Convolve each matrix with 5x5 Gaussian Kernel to smooth out image and reduce unwanted noise
Convolve each matrix with pair of 3x3 Sobel Kernels to get Horizontal and Vertical Image gradients for each color
Calculate overall magnitude based on horizontal and vertical image gradients, storing the largest of each color value in separate matrix
Calculate direction of each gradient pixel
Compare the magnitude of each gradient pixel in the positive and negative direction
gradient pixel is lower intensity than one or both neighbours
gradient pixel is higher intensity than both neighbours
Suppress pixel and set to 0 value.
Apply thresholds to all gradient pixels
pixel has lower intensity than low threshold
Suppress pixel and set to 0 value.
pixel has higher intensity than high threshold
pixel has intensity between high and low thresholds
Set pixel as strong edge
Set pixel as weak edge
Look at remaining weak edge pixels
weak edge is not connected to strong edge pixel
weak edge is connected to strong edge pixel
Suppress pixel and set to 0 value.
Set pixel as strong edge
Image containing only strong edges
Diagram of the steps involved in the PVD extraction algorithm.

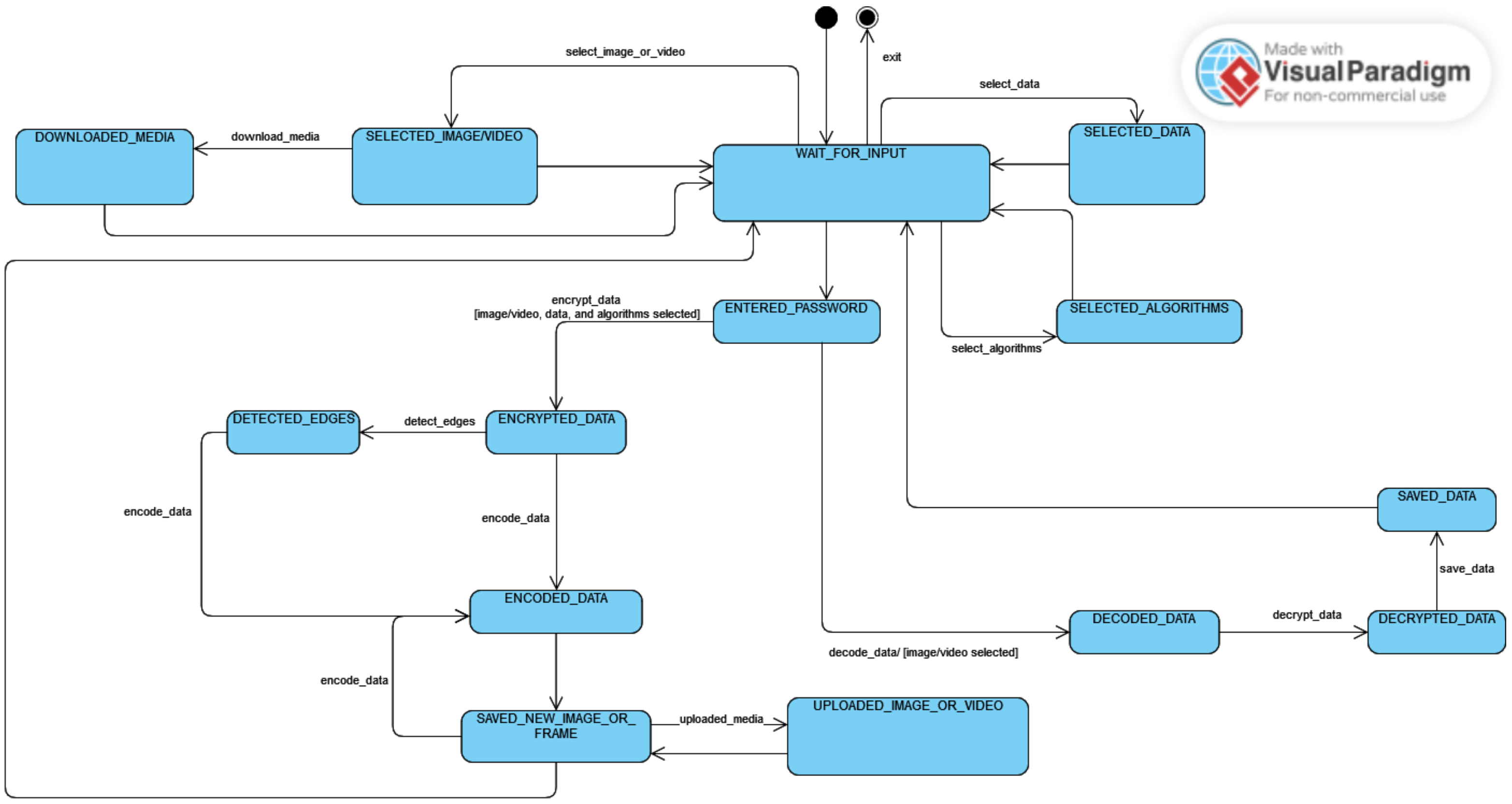
6.3 Technologies Involved

The technologies that will be used in the development of the application are:

- C++20 for the main development of the application.
- Qt for the UI portion of the application.
- OpenCV for manipulating PNG images and video frames.
- The Crypto++ library for encrypting data.
- GitHub for version control.
- Visual Paradigm Online for diagrams.
- Google Drive for documentation and recording test results.
- Trello for keeping track of tasks and their statuses.

7. System/Software Architecture Diagram

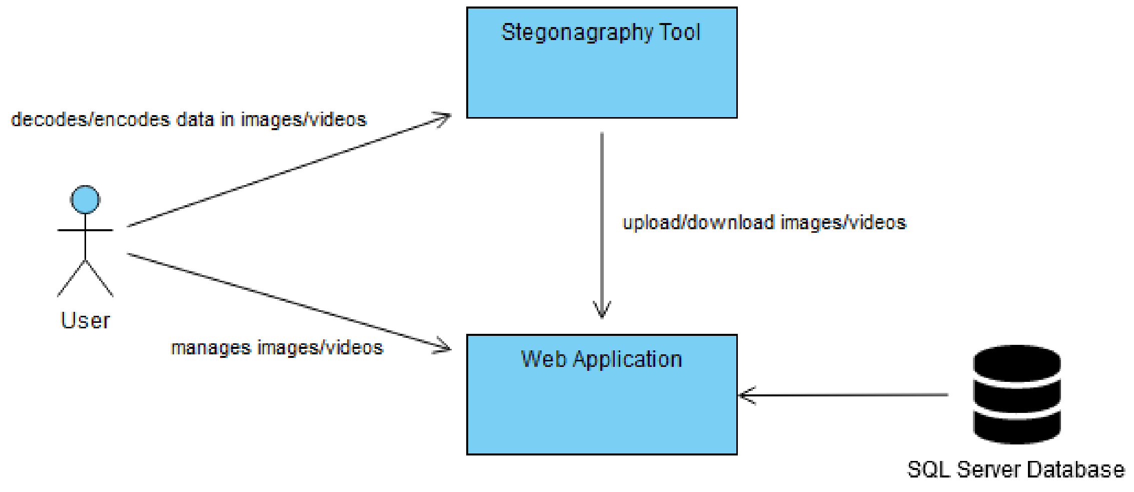
7.1 Steganography Tool State Machine



A diagram of the various states involved in the program.

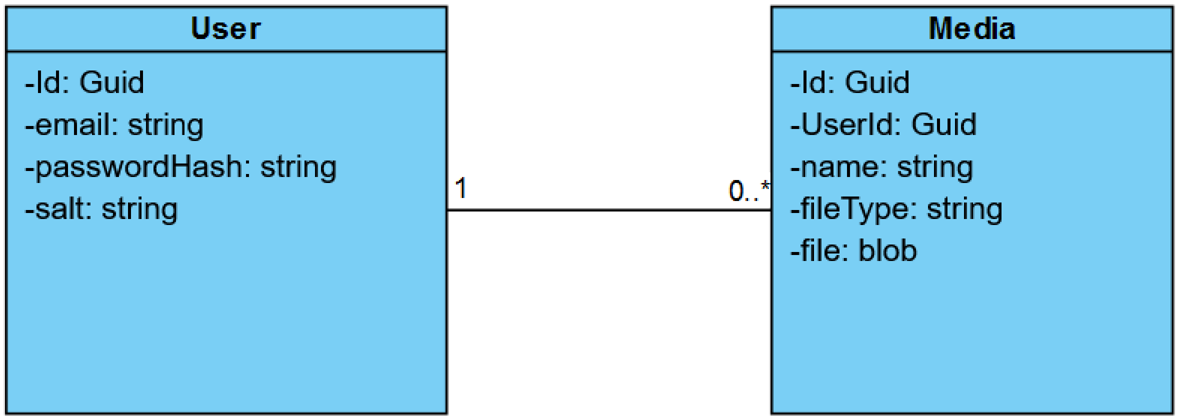
The program will start up and wait for user input. The user will then have to select the image/video to either hide data in or extract data from, either from their local file system or from their images/videos stored on the web application. If hiding data, the user will additionally need to select the data to hide and the algorithms to use in the hiding process. The user will have to input a password for encryption or decryption. The program will then either go through the encoding/decoding process, optionally upload the image/video the web application, and then go back to waiting for more user input.

7.2 Basic System Overview



Basic overview of systems involved in the project.

The user directly interacts with both the steganography tool and the web application, while the tool can also upload/download images/videos to and from the website.



Database classes for storing User Images/Videos.

8. Innovation

While steganography programs do exist, many of them only support image and not video. Additionally, they often do not allow the user to select the algorithms used in the hiding of their data and only implement a single algorithm. The innovative component of this project comes from allowing multiple steganographic techniques to be combined, as well as from supporting both images and video.

One such tool is OpenStego, a free, open-source steganography program that allows the hiding of data in images. However, this program does not offer options for the steganographic algorithms used, only supports image files for hiding data in, and is implemented in Java.

Another tool is Steghide, a free, open-source steganography program the supports hiding data in image files, but not video as this project intends to do. As with OpenStego, there is no option to choose the algorithms used and while it is implemented in C/C++, it is only a command line tool and does not offer a GUI.

9. Complexity

The complexity of this project comes from implementing multiple steganographic algorithms and then combining those with each other while working on both image and video files.

A diploma student would be unable to implement such complex algorithms, let alone combine them in an effective manner. They would also not have learned about security topics such as steganography or encryption in the diploma program.

The real problem being solved here is creating an effective way to hide data using steganography while being more difficult to detect than current methods due to the various combinations of algorithms that can be used.

The student will require specialty knowledge of steganography and the various algorithms involved in order to successfully implement the project. Currently, the student only has a surface-level familiarity with most of the algorithms involved and will need to study them in detail in order to implement them.

10. Technical Challenges

The technical challenges for this project will come from implementing multiple, complicated steganographic algorithms. LSB steganography has been previously taught to the student in the third term of the BTech program, but Pixel Value Differencing and Edge detection are more complicated to implement. The student has also never worked with video files or implemented a GUI using C++ so there will be a learning period associated with those, as well.

11. Development Schedule and Milestones

Milestones	Tasks	Estimated Hours
Project Setup	• Create Project. • Setup GitHub. • Import Libraries (Qt).	5
GUI	• Familiarise self with Qt. • Design GUI. • Implement GUI.	40
MILESTONE 1 COMPLETE		Total Hours: 45
Image and Video Handling	• Implement handling of PNG images. • Implement handling of .mp4 videos.	40
Bit Manipulation Algorithms	• Implement Least Significant Bit (LSB) Algorithm. • Implement Pixel Value Differencing (PVD) algorithm.	30
MILESTONE 2 COMPLETE		Total Hours: 70
Edge Detection Algorithms	• Implement Sobel edge detection. • Implement Canny edge detection. • Ensure edge detection works concurrently with Bit Manipulation Algorithms.	40
MILESTONE 3 COMPLETE		Total Hours: 40
Web app	• Design and implement frontend. • Implement backend database and account system. • Implement web API for tool to upload/download data from	60
API handling	• Add handling of API to steganography tool for user auth, and	20

	uploading/downloading images and video	
MILESTONE 3 COMPLETE		Total Hours: 80
Testing	• Designing test cases • Implementing automated Unit Tests • Implementing automated UI Tests • Manual Testing	80
Documentation	• User Documentation • Writing Report	30
MILESTONE 5 COMPLETE		Total Hours: 110
Total Hours		345

12. Deliverables

The deliverables for this project will be:

- A working C++ executable of the final program.
- C++ project and automated test source code.
- Link to image/video storage website
- ASP.NET Core web app source code
- Data used for testing (images, videos, etc.)
- COMP 8037 Final Report.
- Any other design documentation developed over the course of the project.

13. Conclusion and Expertise Development

This project involves multiple aspects that the student does not have much experience with, such as GUIs in C++, implementing complex algorithms, and dealing with video files. By working with and implementing these unfamiliar aspects, the student hopes to increase the breadth of their technical knowledge, as well as be able to better deal with complex algorithms in the future.

As a BTech student in the Network Security option, steganography is directly related to what the student has been learning in the security portions of the program. The student also hopes to expand their C++ expertise through the development of the project as it is a widely used language in the network and security field.

14. Works Cited

Canny, John. “A Computational Approach to Edge Detection.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, 1986, pp. 679–98, https://doi.org/10.1109/TPAMI.1986.4767851.

Dai, Wei. *Crypto++ Library 8.9 | Free C++ Class Library of Cryptographic Schemes*. June 2023, https://www.cryptopp.com/.

Mandal, J. K., and Debashis Das. "Colour image steganography based on pixel value differencing in spatial domain." *International journal of information sciences and techniques* 2.4 (2012): 83-93.

The Qt Company Ltd. *Qt Documentation*. https://doc.qt.io/. Accessed 26 Oct. 2023.

Vaidya, Samir. *OpenStego*. https://www.openstego.com/. Accessed 4 Nov. 2023.

Vincent, Olufunke, and Olusegun Folorunso. *A Descriptive Algorithm for Sobel Image Edge Detection*. 2009, https://doi.org/10.28945/3351.

Vishnu, B., et al. “Enhanced Image Steganography with PVD and Edge Detection.” *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, IEEE, 2020, pp. 827–32, https://doi.org/10.1109/ICCMC48092.2020.ICCMC-000153.

Wu, Da-Chun, and Wen-Hsiang Tsai. “A Steganographic Method for Images by Pixel-Value Differencing.” *Pattern Recognition Letters*, vol. 24, no. 9–10, 2003, pp. 1613–26, https://doi.org/10.1016/S0167-8655(02)00402-6.

15. Change Log

Date	Description
November 4, 2023	1. Initial Proposal Submission
December 1, 2023	1. Added detailed description and diagrams of proposed algorithms. 2. Added which library will be used for encryption of file data. 3. Added information about web application for managing steganographic images/videos to Project Description, Problem Statement, and Scope. 4. Updated test plan with example cases for web application. 5. Updated state machine diagram to include more detail and state for uploading images/videos to website. 6. Added overview of system interactions. 7. Updated deliverables to include web application portion of project. 8. Update Development Schedule and Milestones based on new features added.