# **Steganography Application**

# **Project Report**

**Zachary Blouin - 100298392**

*12/12/2018*

# **Table of Contents**

Contents

[**Steganography Application** 1](#_Toc532392711)

[**Project Report** 1](#_Toc532392712)

[**Table of Contents** 2](#_Toc532392713)

[**Introduction** 3](#_Toc532392714)

[**Project Scope** 3](#_Toc532392715)

[**How to Use** 4](#_Toc532392716)

[**Coding Design** 6](#_Toc532392717)

# **Introduction**

Steganography is the practice of concealing information inside of another file. Steganography is advantageous over [cryptography](https://en.wikipedia.org/wiki/Cryptography) as the intended secret message does not attract attention to itself. Encrypted messages even if they are harder to break draw attention to themselves as they as they are not inherently hidden. Cryptography deals with the practice of protecting the contents of a message compared to steganography which is concerned with concealing the fact that a secret message is being sent or even exists.

For this project I will be building and implementing a steganography tool for hiding information includes any type of information file and image files and the path where the user wants to save image and extruded file.

# **Project Scope**

The scope of this project will be to create a windows form application using C# that can take and image file and secretly encode plain text so that it is embedded into the image. The embedded plain text should not be easily readable by humans without access to the tool and should not change the created image’s likeness significantly. The application will output a second Image file when a message is saved to not overwrite the original non embedded file.

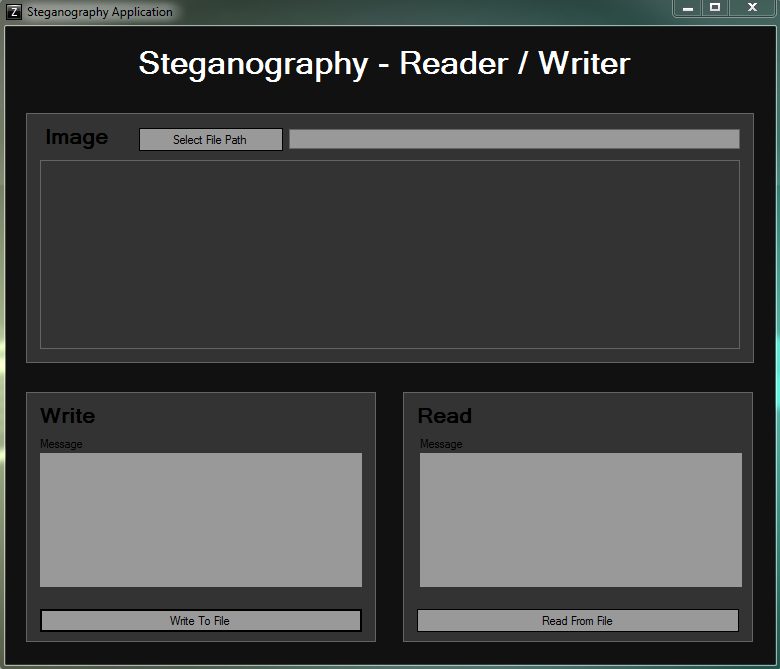
# **How to Use the Application**

The application has a fairly straight forward use case and a wide range error checking covers many situations applicable to the program. The application is available as an .exe format for easy use across windows systems and should not require administrator privileges although it may open flags in some antivirus applications.

Opening up the .exe file will start the user in the default menu.



*Digital Forensics Steganography Application .exe*

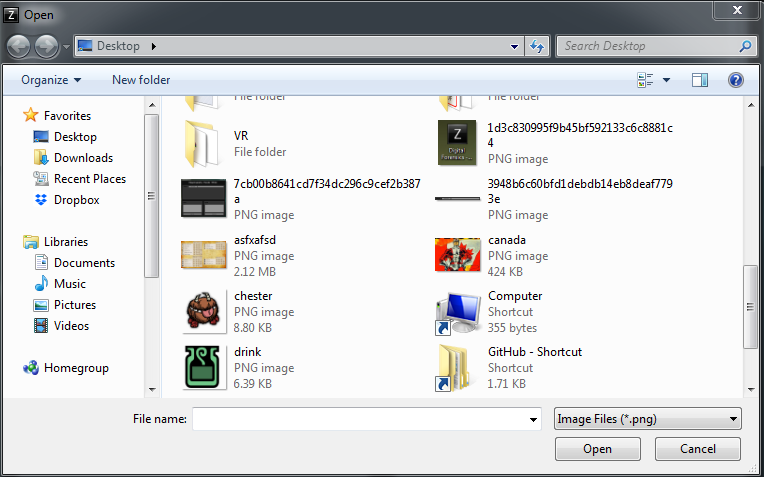


*Default Menu*

Selecting a file path will open up the dialog box to select the file that you want to either read or write to.

D:\Desktop\3948b6c60bfd1debdb14eb8deaf7793e.png

*File Selection Button*

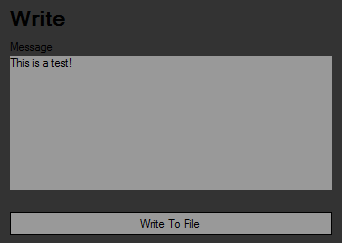


*File Selection Dialog*

It is also possible to write directly to inside of the file path text box it is not recommended as the file type is required to be a .png to function correctly and produce errors. After selecting a .png file users will now be able to read an embedded message or write inside of the image using the bottom two buttons.

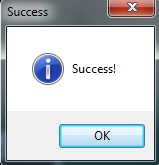
**Writing to File**

To write a message to the file input a message less than 255 characters to hide into the write textbox while and image has been selected. Then select the “Write to File” button.



*Write Textbox and Button with sample text*

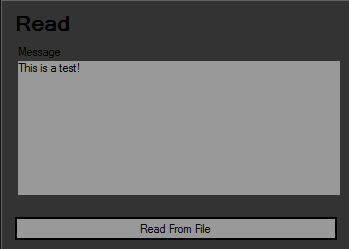
The user will now be prompt for a name on what to store the new encoded file. Save the file with any name. A completion message will popup once it’s done.

**

*Success Message*

**Reading from File**

Selecting the read button when an image is loaded will load the embedded text into the read textbox, this text is not editable in this box but is over writable from the write box. It is possible to read random data from images that have not hidden any text yet.



*Read Textbox and Button with hidden text*

# **Coding Design**

The version of the steganography that I am using for this application hides the information on the inside of the pixels blue value and on the last pixel the size of the message is stored instead of a value. The highest possible RGB value to be stored in the last pixel is 256 resulting as a maximum of 255 characters that can be stored for the message. For this reason files with transparency borders or low pixels counts are much easier to identify as having hidden messages.

**File Setting**

* The application opens a dialog box for the user to choose a file
* The file types are limited to .png format.
* The program error checks for the validity of the file then set the textbox value.

**Writing**

* The write button takes the file location from the textbox value of the path to load the image. It then takes the message inside of the writing text box and transforms each character into a numerical value.
* The characters are taken from the numerical values of the message. Each pixel along the left size of the image has its blue value changed to represent a character.
* The blue value of the last pixel is then set to the size of the message.
* A dialog box prompts saving the new file.
* An error or confirmation message is then generated and displayed to the user.
* The new file is set as the default path, removing the old file from the textbox.

**Reading**

* The read button takes the file location from the textbox value of the path to load the image.
* The last pixel is read so determine the size of the message.
* Using the size of the message each pixel along the left is loaded and the blue numerical value is taken and converted into a character
* All of the characters are put together and displayed as a string in the read textbox.
* An error or confirmation message is then generated and displayed to the user.

# **Test Values**

Included with the project I supplied 3 .png files with hidden messages inside of them to demo the use of the application, feel free to use your own images as well or re-write them!

**File:** drink.png

**Message:** Spacing And Special Characters Work Too!?

!@#$%^&\*()\_+

1234567890-=

**File:** canada.png

**Message:** This image has no transparency making it hard to detect!

**File:** chester.png

**Message:** This is a test message!