# CS 451/551 Project 5: WASM Steganography (version 2020.01)

May 3, 2020

***DUE DATE: May 13 at 11:59PM***

## 1 Introduction

This document first provides the aims of this project, followed by a discussion of its background. It then lists the requirements as explicitly as possible.

### 1.1 Aims

The aims of this project are as follows:

- To write a WebAssembly version of our steganography program.

## 2 Functionality

In this project, you will reproduce the *encoding* portion of Project 01, but targeting WebAssembly.

The user should be able to choose a `.ppm` file locally, type some text into a text box, and then click a "hide" button. Clicking this button should download a ppm (with proper extension) that has the supplied message encoded in it.

All error conditions that had to be handled in Project 01 must be handled here is as well. For example, if the supplied message is too long to fit into the supplied ppm, then an `alert()` should be shown.

**NB: All PPM data should live in WebAssembly and be accessed via WebAssembly memory.**

# 3 Setup and Allowed Crates

**Setup:** There are several prerequisites that you need to have set up for things to work.

1. `wasm-pack` is the tooling we will use for this project. Visit `https://rustwasm.github.io/wasm-pack/installer/` to get it installed.
2. `cargo-generate` allows you to use a pre-existing git repository as a template for a rust project. You can install it with `cargo install cargo-generate`
3. `npm` is a package manger for JavaScript and will be used to actually run our program. You can install `npm` by following the instructions at `https://www.npmjs.com/get-npm`. Make sure that you have the latest version of npm by running `npm install npm@latest -g` as well.

Once you have everything installed, you can create a new project *OR* you can use the `wasm-ppm` project that we were working on as a starting point.

If you want to start a project from scratch, you can type `cargo generate --git https://github.com/rustwa` and when prompted give it the name `wams-steg`.

While `cargo check` et al. will work, you need to use `wasm-pack build` to produce the full WebAssembly binary and bindings.

To create the web server portion of the project, you can type (from within the project directory) `npm init wasm-app www`.

Once your `www` portion of the project is initialized, you can (from within the `www` directory) type `npm install` to get all the requisite packages needed.

To start the server you can run (from within the `www` directory created via the npm init command) `npm run start`.

**Crates:** You are allowed to use and modify the previously distributed `libsteg` that was sent to the class. You are allowed to use the `regex` crate. **No other crates besides those used by `wasm-pack` are allowed.**

# 4 Grading

All students:

- An earnest effort was made to complete the project.
  - **20 points**
- Program compiles.
  - **30 points**
- Your program can properly encode a message.
  - **30 points**
- Your program handles errors correctly.
  - **20 points**