**Software Implementation and Testing Document**

**For**

**Group 13**

Version 1.0

**Authors**:

Brandon Goldsmith

Marshall Copeland

Eduardo Neira

Katterin Soto

Joshua Flashman

# Programming Languages (5 points)

*List the programming languages used in your project, where you use them (what components of your project) and your reason for choosing them (whatever that may be).*

* *Python - we are using Python to implement all of the features used in our web app. None of us have had much experience with Python, so we chose it as a way to learn.*
* *React - Brandon is utilizing React to aid with the front-end design.*
* *HTML / CSS - Also utilized for front-end.*
* *JavaScript - used along with the React framework*

# Platforms, APIs, Databases, and other technologies used (5 points)

*List all the platforms, APIs, Databases, and any other technologies you use in your project and where you use them (in what components of your project).*

* *Front end - Webapp built using ReactJS. Uses functions file to speak with Flask backend.*
* *Custom Flask API- Used to send and retrieve information for each function via URL http requests. Executed using virtual environment.*
* *Python backend - Individual functions were defined in server.py to communicate with the Flask API*

# Execution-based Functional Testing (10 points)

*Describe how/if you performed functional testing for your project (i.e., tested for the* ***functional requirements*** *listed in your RD).*

* *For each function that is provided to the users, there is a test file that was previously used to avoid discrepancies with user input prior to loading the function into the web application.*
* *Hashing: For testing the 14 Hashing functions, all of the functions were placed in a HashTesting class. In the class, each hashing function was included, as well as an initializing function that allows the input to be passed to the functions. After the class was constructed, a separate python file was created that imported the class to be able to test the functions. A list was then initialized that was full of different types of strings like numbers, words, words with numbers, and sentences. Then for each element in the list, the hashed version of the element was added to a specific file depending on which hash is going to be tested. For example, if the MD2 hash is being tested, a MD2 test file was created that contained the correct hashes of each element of the list on each line of the file. These correct hashes were collected from the website CyberChef. and served as the key. Then when it was time to test the MD2 function, the function takes in the elements as input and compares the output to the key file with the correct hashes. If there is an issue, on the terminal the element that had the issue would be printed out, as well as the two different hashes for that element. This was done for all 14 of the hashing functions, and by the end of the testing there were no issues with any of the functions. That means that each function’s output matched up with the output of a secondary function that was published online. Now this could mean that both just happened to produce the wrong output based on the elements, but that is highly, highly unlikely. There could also be other issues because only certain words and character combinations were tested. The program might only work for those words, but not others, however like the previous concern, it is very unlikely that the output would be correct for all of the elements in the list and not others outside of the list. A larger sample size would fix this issue, but with a list of 50 different words for each of the 14 functions, and with each function giving the correct output, it is safe to say that each of these functions has been adequately tested and works as intended.*

# Execution-based Non-Functional Testing (10 points)

*Describe how/if you performed non-functional testing for your project (i.e., tested for the* ***non-functional requirements*** *listed in your RD).*

* *Testing of the frontend:*
  + *Average user course of action. This use case assumes the user enters correct input and format, along with user events such as clicking. In order to guard against errors, Nol3Ware implements error checking whenever the user interacts with the application. Ex: when a user clicks the ‘Bake’ button, there is a check that ensures all input forms are correct and cohesive with one another. Otherwise, the request fails and the user gets an error response.*
  + *Rapid-event test to attempt to break the frontend.*

# Non-Execution-based Testing (10 points)

*Describe how/if you performed non-execution-based testing (such as code reviews/inspections/walkthroughs).*

*Throughout the semester, we as a group had meetings to discuss the project and what each group member was working on. Since mostly everyone had separate components that they were working on, like encryption, conversion, and hashing, it was difficult to perform code analysis at the beginning of the semester. But as each member continued to post their code to the group’s GitHub, and as the project took shape, it was easier to open somebody's file and see how it worked. Each member is able to not only see their own backend code, but how it interacts with the frontend, and to see from start to finish which code is being executed when the website is running. This familiarity with each other’s code was important because it holds each person accountable to their own work, and encourages them to make progress on their assigned segments. Ideally this would also mean that each member can test another member’s code to second-handedly verify that everything works as intended, and perform reviews for the author to consider feedback from that analysis. However, as a group it was decided that it would be easier to test their own code, as it is easier to produce testing materials and results when the tester has an understanding of the inticieces and niche features of that code. This speeds up the process of testing because the tester does not have to spend time trying to understand what the code does, how it is supposed to work, and how to actually test the code, without having to sacrifice any accuracy of tests. A final overview of the code was also performed by all of the group members to ensure that the program is including everything that needs to for it to run, as well as making sure that there are no errors that were made from any of the different sections of the project, whether it was the frontend, backend, or any accessory features that communicated the information between the two.*