**Software Requirements and Design Document**

**For**

***Group 13***

Version 1.0

**Authors**:

*Brandon Goldsmith*

*Marshall Copeland*

*Eduardo Neira*

*Katterin Soto*

*Joshua Flashman*

# Overview (5 points)

*Give a general overview of the system in 1-2 paragraphs (similar to the one in the project proposal).*

*NoleWare is a multi-faceted web application focusing on cybersecurity features to help cyber security enthusiasts and students with basic operations needed to thrive in the industry. It will feature encryption and security algorithms, conversion tools, and logical operations, etc. This app is similar to CyberChef, where the user enters specific input and chooses what they’d like to do with that input. The features offered are there to further understand security and encryption algorithms and methodology. It is very user-friendly, as it is easy to use to be able to get accurate results that come from python programs, each specifically focused on each feature. We also want to be able to provide an additional tool for students or anyone interested in the cyber security industry.*

*Github Link: https://github.com/jflashman/N0l3Ware*

# Functional Requirements (10 points)

*List the* ***functional requirements*** *in sentences identified by numbers and for each requirement state if it is of high, medium, or low priority. Each functional requirement is something that the system shall do. Include all the details required such that there can be no misinterpretations of the requirements when read. Be very specific about what the system needs to do (not how, just what). You may provide a brief design rationale for any requirement which you feel requires explanation for how and/or why the requirement was derived.*

1. *The Hashing section consists of different types of hashing functions that vary in methodology and the amount of security they provide. The more bytes, the more secure the function is, and that usually means it is a more recent version*
2. *Blake2b*

*i. The input will be converted to a Blake2b hash*

*input: “hello”*

*output: “e4cfa39a3d37be31c59609e807970799caa68a19bfaa15135f165085e01d41a65ba1e1b146aeb6bd0092b49eac214c103ccfa3a365954bbbe52f74a2b3620c94”*

1. *Blake2s*

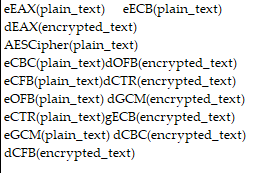
*i. The input will be converted to a Blake2s hash*

1. *MD2*

*i. Same as above, all hashing functions are used the same, with medium priority*

1. *MD4*
2. *MD5*
3. *SHA1*
4. *SHA224*
5. *SHA256*
6. *SHA384*
7. *SHA3\_224*
8. *SHA3\_256*
9. *SHA3\_384*
10. *SHA3\_512*
11. *SHA512*
12. *There is a feature that generates QR codes from strings*
13. *QR Code Generator*

*i. If a user types in a string such as “*[*https://www.cs.fsu.edu/*](https://www.cs.fsu.edu/)*”, a QR code will be generated so that when scanned by a QR reader, the user can be sent to that website. The string does not need to be a website, a QR code will be created for any words that are entered.*

1. *Encryption*
2. **
3. *Conversions (Binary<->Decimal)*
   1. *Binary (binary->decimal)*
      1. *Once input is given, Baking will start working and there will be the accurate decimal value of the binary. An error will be printed if the input is not in binary form. i.e.*

*input: 110101*

*output: 53*

*input: 1203560*

*output: error.*

* 1. *Decimal*
     1. *Once input is given, Baking will start working and there will be a binary output form of the decimal given. An error will be printed if the input is not in decimal form*

*input:1235*

*output: 10011010011*

*input: 1dc!34*

*output: error.*

1. *Frequency Analysis*
   1. *Frequency Analysis module takes in a text string, analyzes its frequency, and generates a bar graph image. This image is then serialized into base64 to be sent over the flask API*

# Non-functional Requirements (10 points)

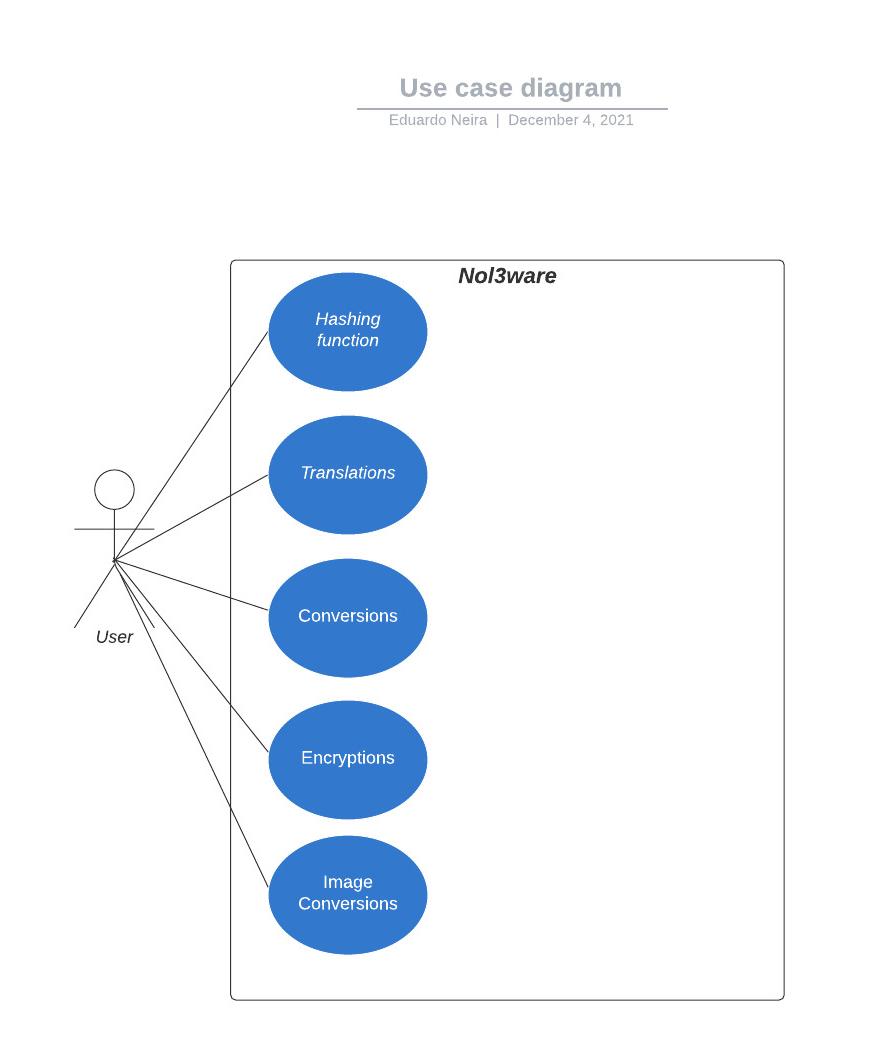
*List the* ***non-functional requirements*** *of the system (any requirement referring to a property of the system, such as security, safety, software quality, performance, reliability, etc.) You may provide a brief rationale for any requirement which you feel requires explanation as to how and/or why the requirement was derived.*

1. *User must have access to internet and associated libraries listed in dependencies.*
2. *Device must have reasonable performance to host the simultaneously host the webapp and backend react server*

# Use Case Diagram (10 points)

*This section presents the* ***use case diagram*** *and the* ***textual descriptions*** *of the use cases for the system under development. The use case diagram should contain all the use cases and relationships between them needed to describe the functionality to be developed. If you discover new use cases between two increments, update the diagram for your future increments.*

***Textual descriptions of use cases****: For the first increment, the textual descriptions for the use cases are not required. However, the textual descriptions for all use cases discovered for your system are required for the second and third iterations.*

**

*The hashing functions are sets of functions that allow the user to input a string and have it output a hash of whichever of the fourteen hashing functions there are in the hashing tab..*

*The conversions are functions that allow the user to either convert from a decimal number (base 10) to a binary number (base 2), or the reverse, which would be a binary number to a decimal number.*

*The encryption functions are similar to the hashing ones, however they are able to securely encrypt the string based on a private key. The ciphertext can then be decrypted with that key to reveal the original message.*

*The image conversions were scrapped.*

*The translations were scrapped*

# Class Diagram and/or Sequence Diagrams (15 points)

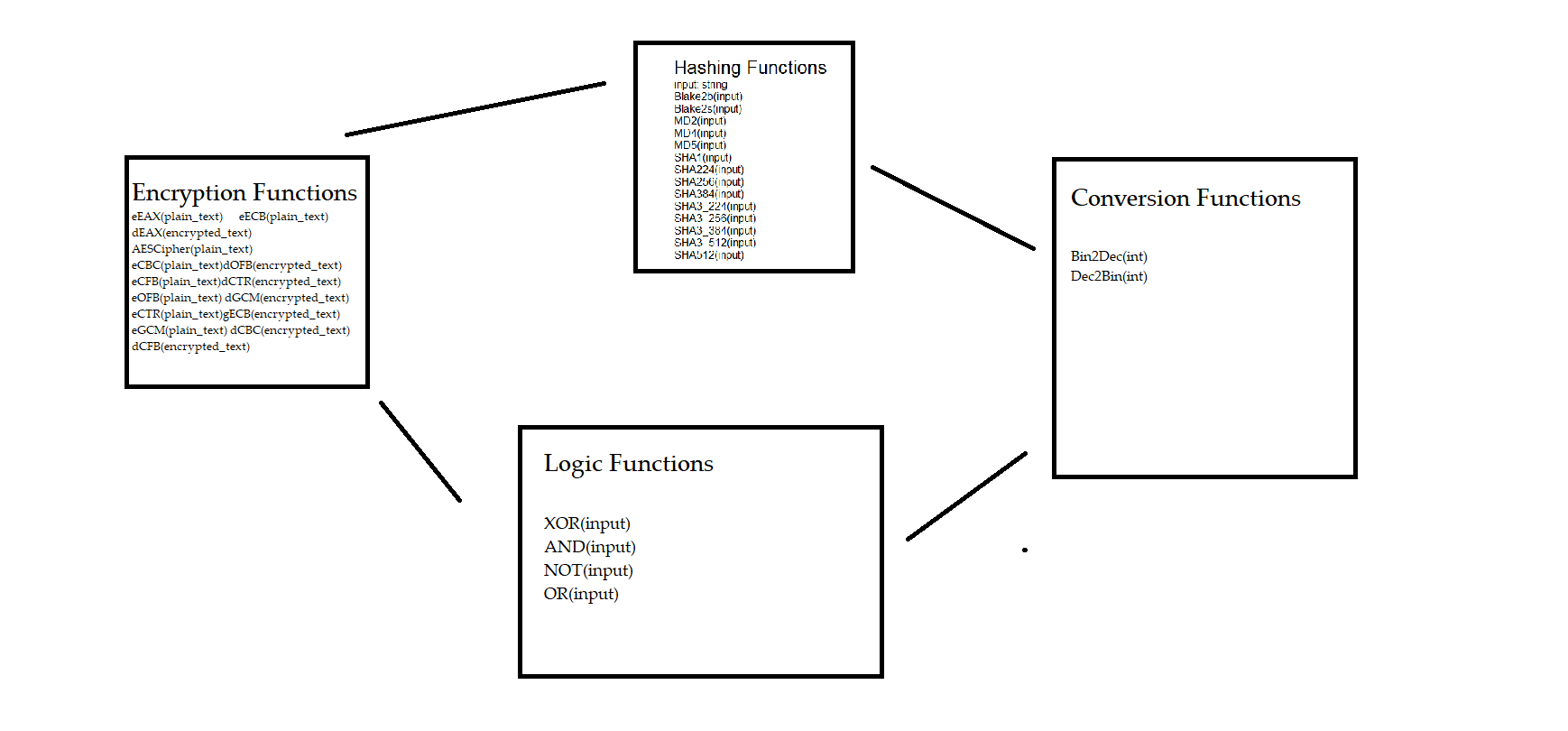
*This section presents a high-level overview of the anticipated system architecture using a* ***class******diagram*** *and/or* ***sequence diagrams****.*

*If the main* ***paradigm*** *used in your project is* ***Object Oriented*** *(i.e., you have classes or something that acts similar to classes in your system), then draw the* ***Class Diagram******of the entire system and Sequence Diagrams for the three (3) most important use cases in your system.***

*If the main* ***paradigm*** *in your system is* ***not Object Oriented*** *(i.e., you* ***do not*** *have classes**or anything similar to classes in your system) then only draw* ***Sequence Diagrams****,* ***but for all the use cases of your system.*** *In this case, we will use a modified version of Sequence Diagrams, where instead of objects, the lifelines will represent the functions in the system involved in the action sequence.*

***Class Diagrams*** *show the* ***fundamental objects/classes*** *that must be modeled with the system to satisfy its requirements and* ***the relationships*** *between them. Each class rectangle on the diagram* ***must also include the attributes and the methods of the class*** *(they can be refined between increments). All the* ***relationships between classes and their multiplicity*** *must be shown on the class diagram.*

*A* ***Sequence Diagram*** *simply depicts* ***interaction******between objects*** *(or* ***functions -*** *in our case - for non-OOP systems) in a sequential order, i.e. the order in which these interactions take place. Sequence diagrams describe how and in what order the objects in a system function.*

**

# Operating Environment (5 points)

*Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.*

*Nol3Ware runs on any operating system as long as the machine has access to a web browser, as we used JavaScript and Python to build it. Nol3Ware utilizes a backend api written using Flask. This api is able to receive and send data to/from the frontend via url requests. The user must have python, flask, flask-cors, and virtualenv installed in order to be able to run the server api locally on their machine. As for the frontend, it is built in JavaScript using the React framework. The user can start the frontend if node.js is installed on their local machine. Once both the backend and frontend are online in different ports, the http requests can be made, and they can communicate with each other.*

# Assumptions and Dependencies (5 points)

*List any assumed factors (as opposed to known facts) that could affect the requirements stated in this document. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project.*

* *The user will not try to overflow the programs, nor will it try to break the website with the input.*
* *The user will not use Nol3Ware to gain illegal information through any of the functionalities.*
* *For the purpose of this project, the user will not put invalid inputs in each function.*
* *The user will not attempt to change any of the functionality of the programs, nor attempt to access any files that they are not authorized to access.*
* *The user should only try to use this program in environments where it has been tested and proven to work, any other platforms will likely not be able to allow the program to be run.*
* *Requirements/dependencies: Pycryptodome, Flask, Node, Python, flask-cors, internet access.*