Encryption Analysis Engine  
By: James Lockhart

Table of Contents

[1. Overview 7](#_Toc475822840)

[i. Phase I 7](#_Toc475822841)

[ii. Phase II 8](#_Toc475822842)

[iii. Phase III 10](#_Toc475822843)

[iv. Phase IV 12](#_Toc475822844)

[2. Introduction 13](#_Toc475822845)

[a. Purpose 13](#_Toc475822846)

[b. Scope 13](#_Toc475822847)

[c. Background 14](#_Toc475822849)

[d. Project Plan 15](#_Toc475822850)

[3. Requirements & Specifications 16](#_Toc475822851)

[a. Problem Statement 16](#_Toc475822852)

[b. Requirements 16](#_Toc475822853)

[c. Possible Scenarios for Use 16](#_Toc475822854)

[d. Definitions 16](#_Toc475822855)

[e. System Specifications 16](#_Toc475822856)

[f. Functional Specifications 17](#_Toc475822857)

[4. User Guide 18](#_Toc475822858)

[a. Welcome to the Encryption Analysis Engine 18](#_Toc475822859)

[b. Minimum System Requirements 18](#_Toc475822860)

[c. Instructions 18](#_Toc475822861)

[i. Step 1: Launching the Application 18](#_Toc475822862)

[ii. Step 2: Accepting Terms & Conditions 18](#_Toc475822863)

[iii. Step 3: Reviewing the Main Application Interface 19](#_Toc475822864)

[iv. Step 4: Brute Force Feature 20](#_Toc475822865)

[v. Step 4: Hash Collision / MD5 20](#_Toc475822865)

[vi. Step 4: Strong Password Generator 20](#_Toc475822865)

[vii. Step 5: Terminating the Application 24](#_Toc475822866)

[d. Frequently Asked Questions 24](#_Toc475822867)

[i. Can I enter more than 6 characters for a password? 24](#_Toc475822868)

[ii. Will I be able to enter special characters such as @ or !? 24](#_Toc475822869)

[iii. Why does it take so long to find my password? 24](#_Toc475822870)

[iv. Is my password stored in a database after I run the program? 24](#_Toc475822871)

[e. Troubleshoot 24](#_Toc475822872)

[5. Test Plan and Results 27](#_Toc475822883)

[a. Introduction 27](#_Toc475822884)

[i. Project Overview 27](#_Toc475822885)

[ii. Testing Process Overview Diagram 27](#_Toc475822886)

[b. Testing Schedule 27](#_Toc475822887)

[i. Phase I (Complete) 27](#_Toc475822888)

[ii. Phase II (Complete) 28](#_Toc475822889)

[iii. Phase III (Complete) 28](#_Toc475822890)

[iv. Phase IV (Complete) 28](#_Toc475822891)

[c. Use Case Design 28](#_Toc475822892)

[i. Purpose 28](#_Toc475822893)

[ii. Use Case Design Diagram 29](#_Toc475822894)

[d. Test Execution Strategy 29](#_Toc475822895)

[i. Purpose 29](#_Toc475822896)

[ii. Objectives 29](#_Toc475822897)

[iii. Assumptions 29](#_Toc475822898)

[iv. Test Execution Process Diagram 29](#_Toc475822899)

[e. Defect Tracking & Reporting 29](#_Toc475822900)

[i. Defect Categorization 29](#_Toc475822901)

[ii. Defect Tracking Diagram 30](#_Toc475822902)

[f. Use Cases 31](#_Toc475822903)

[i. Case ID#: 001 31](#_Toc475822904)

[ii. Case ID#: 002 32](#_Toc475822905)

[iii. Case ID#: 003 32](#_Toc475822906)

[iv. Case ID#: 004 32](#_Toc475822907)

[v. Case ID#: 005 32](#_Toc475822908)

[vi. Case ID#: 006 32](#_Toc475822909)

[vii. Case ID#: 007 32](#_Toc475822910)

[viii. Case ID#: 008 32](#_Toc475822911)

[ix. Case ID#: 009 32](#_Toc475822912)

[x. Case ID#: 010 32](#_Toc475822913)

[xi. Case ID#: 011 32](#_Toc475822914)

[xii. Case ID#: 012 32](#_Toc475822915)

[xiii. Case ID#: 013 32](#_Toc475822916)

[xiv. Case ID#: 014 32](#_Toc475822917)

[xv. Case ID#: 015 32](#_Toc475822918)

[xvi. Case ID#: 016 32](#_Toc475822919)

[xvii. Case ID#: 017 32](#_Toc475822920)

[g. Standardize I/O 32](#_Toc475822921)

[h. Approved Tests 32](#_Toc475822922)

[6. Design Approach 32](#_Toc475822923)

[a. UML Diagram 32](#_Toc475822924)

[b. PassBreak – Class 32](#_Toc475822925)

[i. Methods 32](#_Toc475822926)

[c. BruteCrack – Class 32](#_Toc475822927)

[i. Methods 32](#_Toc475822928)

[d. InterfaceConsole – Class 32](#_Toc475822929)

[i. Methods 32](#_Toc475822930)

[e. InterfaceOption – Class 32](#_Toc475822931)

[i. Methods 32](#_Toc475822932)

[f. InterfacePanel – Class 32](#_Toc475822933)

[i. Methods 32](#_Toc475822934)

[g. InterfaceReader – Class 32](#_Toc475822935)

[i. Methods 32](#_Toc475822936)

[h. MD5 – Class 32](#_Toc475822937)

[i. Methods 32](#_Toc475822938)

[i. MD5Database – Class 32](#_Toc475822939)

[i. Methods 32](#_Toc475822940)

[j. Timing – Class 32](#_Toc475822941)

[i. Methods 32](#_Toc475822942)

[i. AnalogClock – Class 32](#_Toc475822943)

[i. Methods 32](#_Toc475822944)

[7. Timing Report 32](#_Toc475822945)

[a. General Analysis 32](#_Toc475822946)

[b. Timing Algorithm Tests 32](#_Toc475822947)

[c. Timing Conclusion 32](#_Toc475822948)

[8. Development History 32](#_Toc475822949)

[a. Update 0.20 32](#_Toc475822950)

[i. PassBreak – Class 32](#_Toc475822951)

[ii. PassBreak – Pseudo Code Structure 32](#_Toc475822952)

[iii. BruteCrack – Class 32](#_Toc475822953)

[iv. BruteCrack – Pseudo Code Structure 32](#_Toc475822954)

[v. Developer Comments 32](#_Toc475822955)

[b. Update 0.30 32](#_Toc475822956)

[i. BruteCrack – Class 32](#_Toc475822957)

[ii. BruteCrack – Pseudo Code Structure 32](#_Toc475822958)

[iii. Developer Comments 32](#_Toc475822959)

[c. Update 0.40 32](#_Toc475822960)

[i. Developer Comments 32](#_Toc475822961)

[d. MD5 Update 0.30 32](#_Toc475822962)

[i. MD5 – Class 32](#_Toc475822963)

[ii. MD5Database – Class 32](#_Toc475822964)

[iii. Developer Comments 32](#_Toc475822965)

[e. Update 0.50 32](#_Toc475822966)

[i. Timing – Class 32](#_Toc475822967)

[ii. Timing – Pseudo Code 32](#_Toc475822968)

[iii. Developer Comments 32](#_Toc475822969)

[g. Updates 1.0-1.6 32](#_Toc475822970)

[i. StrengthChart – Class 7](#_Toc475822967)9

[ii. InterfaceConsole - Class 8](#_Toc475822968)1

[iii. InterfaceReader - Class](#_Toc475822969) 83

[iv. InterfaceOption – Class 8](#_Toc475822967)3

[v. InterfacePanel - Class 8](#_Toc475822968)4

[vi. MD5Database - Class](#_Toc475822969) 85

vi[i. HashGenerator – Class 8](#_Toc475822967)6

v[iii. Developer Comments](#_Toc475822969) 87

[h. Update 1.7 32](#_Toc475822971)  
[i. PasswordGenerator – Class 87](#_Toc475822967)

[ii. PassDialong - Class 89](#_Toc475822968)

[iii. Developer Comments](#_Toc475822969) 90

[9. Conclusions 32](#_Toc475822972)

[10. References 32](#_Toc475822973)

Figures

Figure 1. Types of Data Breaches 15

Figure 2. Project Plan Gantt Diagram 15

[Figure 3. Terms & Conditions 19](#_Toc475188630)9

[Figure 4. Schematic of EAE Main Application Interface 20](#_Toc475188631)

[Figure 5. Drop Down Menu](#_Toc475188630) 20

[Figure 6. Time Analysis 2](#_Toc475188630)1

[Figure 7. Invalid Input 2](#_Toc475188630)1

Figure 8. Successful Execution …………………………………………………………………………………………………..22

[Figure 9. Strong Password Generator Selections 2](#_Toc475188630)3

[Figure 10. Example of Strong Password Generator 2](#_Toc475188630)3

[Figure 11. Terminating Application 2](#_Toc475188630)4

[Figure 12. SSL Handshake Application 2](#_Toc475188630)5

[Figure 13. Properties, Change Settings 2](#_Toc475188630)5

[Figure 14. System Properties 2](#_Toc475188630)6

[Figure 15. Environment Variables 2](#_Toc475188630)7

[Figure 16. Testing Process Overview Diagram 27](#_Toc475188632)9

[Figure 17. Use Case Design Process Diagram 29](#_Toc475188633)9

[Figure 18. Test Execution Process Diagram](#_Toc475188634) 30

[Figure 19. Defect Categorization Table 30](#_Toc475188635)

[Figure 20. Defect Tracking Diagram 48](#_Toc475188636)

[Figure 21. Standard I/O 49](#_Toc475188637)

[Figure 22. Approved Tests Table 52](#_Toc475188638)

[Figure 23. Initial UML Diagram 62](#_Toc475188639)

[Figure 24. Representation of Operation Time as a Function of Elements 63](#_Toc475188640)

[Figure 25. Six Highest Frequency Characters](#_Toc475188640) 64

[Figure 26Six Lowest Frequency Characters](#_Toc475188640) 65

[Figure 27. Crack and PassBreak UML Diagram 66](#_Toc475188640)

[Figure 28. Update UML Diagram for BruteCrack and PassBreak](#_Toc475188641) 69

Figure 29. UML Diagram for Phase II Implementation ……………………………………………………………….75

Figure 30. UML Diagram for Phase III Implementation ………………………………………………………..…….79

Figure 31. UML Diagram for Phase IV Implementation …………………………………………………..…..…….88

# Overview

### Phase I

#### **Activities Per Member**

* Travis Parrot – Additions to test cases to reflect our intent to incorporate MD5.
* Leigh Daniels – Additions to EAE final documentation (Overviews and Development History revisions), UML diagrams for Phase 1 coding documentation.
* James Lockhart – Coding for initial brute force algorithm and source code documentation.
* Matthew Seefeldt – Updating Design Approach documentation to reflect changes in our code development. Additions and review of EAE final documentation.
* Diana Van Hook – Additions to EAE final documentation (Purpose Draft and Decisions Reevaluation).
* Kendrick Walters – Coding for mandatory legal disclaimer GUI, documented test cases on version 1 code.
* Nathan Miranda – Researching timing functionality and developing performance related testing scenarios.

#### Original Milestones & Schedule

Our original milestone for this week were to write our primary brute force algorithm, and then successfully analyze a password that has up to 2 alphanumeric characters. We originally decided to work with just 2 alphanumeric characters in order to keep things simple and to know that the algorithm at least works as intended.

At this time, we are ahead of schedule. Our initial milestones have been met, and we have begun further analysis of our code and testing in order to assess the performance when executing our program with more than 2 alphanumeric characters. As far as documentation goes, we are also ahead of schedule as we have already begun writing as if this were our final report. Documenting our development in this way allows us to stay flexible, and keeps revisions much easier to manage.

#### Problems Encountered

None yet.

#### Reevaluation of Decisions

* We are continuing to evaluate the use of symbols and password length that will be allowed for the end user.
* Considerations about the length of time needed to crack a password as well as the amount of computer resources needed caused some changes in our source code.
* Changes are being made to the GUI to make it more interesting for the user.

#### Changes to Documentation

* Overviews have been revised. Overviews are now broken down by phase, and a template has been built to address all questions/requirements from the professor.
* A draft of the Purpose has been added.
* Development History has been revised. Development History will now be the location for the source code/documentation per each phase.

### Phase II

#### Activities Per Member

* Travis Parrott – Wrote Pseudocode for AnalogClock class. Revised User Guide to reflect changes that have been implemented throughout development. Completed problems encountered section of phase II documentation. Completed the scope (Sec 1c.) of the final report document. Updated testing documents to reflect current progress in development. Participated in decision making about various goals and milestones during 2 team meetings.
* Leigh Daniels – Built Phase II UML Diagram. Wrote Phase II Overview, and Background for final report. Updated Figures for User Guide and Devleopment History. Added in Timing Section.
* James Lockhart – Wrote and troubleshot multithreading for the EAE application. Revisions to the code for the random password generator.
* Matthew Seefeldt – Wrote Developer Comments for Phase II. Wrote pseudocode for AnalogClock.
* Diana Van Hook – Created new introduction for user guide. Wrote pseudocode for Timing class.
* Kendrick Walters – Designed and coded the AnalogClock class, graphically illustrating time-to-crack as a clock face with dynamically changing color scheme to reflect increasing strength over time. Provided feedback concerning Timing multithreading class. Shared test data concerning Command-Line Interface (CLI) version performance.
* Nathan Miranda – Phase II Lead. Wrote code for the Timing class, documentation for Phase II development history, approach, and timing/performance data.

#### Original Milestones & Schedule

The original milestones for Phase II were to conclude the coding for Phase I - Algorithms, successfully analyze a password consisting of up to 2 alphanumeric characters, conclude coding for Phase II - Timing, optimize EAE to handle up to 6 alphanumeric characters, and to develop the security level baselines based on execution times.

Since we have reevaluated our original decision to conduct baseline testing this week, we are right on schedule. The brute force algorithm is functioning well, and EAE is fully capable of analyzing password consisting of up to 4 alphanumeric characters. We have completed coding for Timing, however we are still working on optimizing EAE to handle up to 6 alphanumeric characters. Therefore, we will not conduct baselines until next week once the brute force feature has been optimized.

#### Problems Encountered

1. GUI failing and collapsing/disappearing halfway through (~10+ minutes)
   1. During BruteCrack run of 6 character passwords the GUI collapse and disappears from the screen
   2. The CLI version continues to function properly
   3. Ongoing issue, unresolved
2. We were attempting to utilize multi-threading to enable the timing feature of the GUI to continuously count throughout program execution.
   1. No viable solution was discovered as every successful implementation of a multithread caused BruteCrack to slow significantly
   2. Seems that JDK has a threshold on how many resources it can consume from the machine
3. Clock GUI color implementation was malfunctioning to cause an undesired change in color once past max time limit of 43 minutes
   1. Reprogramed AnalogClock class to set 255-240 as the max color decrement, therefore the analog clock will print green forever once it surpasses 43 minutes of attempted cracking

#### Reevaluation of Decisions

One decision we adjusted was to save timing baselines and further testing for next week, Phase III. We are still working on multithreading problems, and feel that it would be best to conduct testing next week after we have worked out bugs.

Multithreading has become too problematic for our application. The incorporation of this feature adds a substantial amount of overhead; with this implementation, execution times are unacceptable. In light of this, we have decided to incorporate a password generator and other features to make up for this fault.

#### Changes to Documentation

* Overview for Phase II – Timing
* Scope added in preparation for final report
* Background added in preparation for final report
* Development History for Phase II - Timing
* New additions to our User Guide Section
* New additions to testing plan
* Added a section for discussing Performance testing aside from test case scenarios.
* Additions to Design Approach to include Timing and AnalogClock

### Phase III

#### Activities Per Member

* Travis Parrot – Updated Test Plans to incorporate new feature, MD5 and possible future password generator. Conducted testing per each test case, providing findings to development lead. Retested all fix actions taken. Provided testing feedback to enable development of additional application functionality. Provided documentation of all tests.
* Leigh Daniels – Documented execution levels. Built UML Diagram for Update 1.6. Updated Gantt Diagram to reflect the group’s actual schedule thus far per Project Plan Requirement. Approved Test Cases. Wrote Overview, and combined all Phase III artifacts for the final report. Added a Troubleshoot section to the User Guide.
* James Lockhart – Coding preparation for Phase IV – random password generator. Troubleshot issues encountered from testing.
* Matthew Seefeldt – Coded time estimator, and wrote pseudocode and developer comments.
* Diana Van Hook – Wrote pseudocode for GUI. Added a Reference and a link to the external MD5 API site. Added new images from updated EAE GUI to the User Guide.
* Kendrick Walters – Phase III Lead. Wrote code to incorporate the additional features from this week’s updates to the GUI. Troubleshot issues encountered from testing.
* Nathan Miranda – Coded the StrengthChart GUI class. Updated User Guide to incorporate new features.

#### Original Milestones & Schedule

The original milestones for Phase III consisted of having a functional and easy to use GUI that provides statistical analysis information to the user after execution, provides safe input validation, and proper error handling. All original goals have been met.

At this point, the EAE tells the user whether their input was “Very Weak”, “Weak”, “Fair”, “Good”, “Strong”, or “Excellent” based on the total execution time, rather, the time taken to determine the input via brute force. These ratings are evenly spaced in five minute increments, spanning over 30 minutes. Brute Force is a difficult, hardware intensive task to accomplish, and time consuming. Much more complex passwords can most certainly take longer than 30 minutes to be broken. However, within the limits of our application, we feel that any user input that is acceptable to EAE that takes 30 minutes to resolve is considered to be Excellent.

EAE does not accept special characters, and does not accept any input strings that are longer than six characters. Also, our engine has been optimized and tested to not crash mid-execution with the given limitations to input, and informs the user that the random password generator is not available in this version. Therefore, the EAE provides safe input validation, and proper error handling.

#### Problems Encountered

In the testing phase, errors were encountered regarding SSL Handshake problems, simple input returning strong results, and timeouts. All of these problems have been addressed. SSL Handshake problems were as a result of incorrect environment configuration. Therefore, a new error prompt was established regarding this matter, and a new section has been added to our User Guide in order to teach our users how to rectify this issue if it arises in their use of our application.

#### Reevaluation of Decisions

In the latest update, the Timing class has been deprecated. The timing class provided no enhancement to the overall function of the program, and did not significantly impact application performance.

Per the professor’s suggestion, the group brainstormed the idea of incorporating a ‘kill button’. In the latest version of EAE, if the user clicks the Reset button, all fields will clear and analysis will cease. The group plans to reevaluate this function in the final week to see how it can be improved.

After Peer Review 1, we adjusted our Class Titles on our UML Diagrams to include extensions and implementations. Since our application code has grown so much, we have excluded these extensions from this point forward for the sake of keeping our UML diagram legible within this report.

#### Changes to Documentation

* Reordered the Introduction and Overview Sections
* Overview for Phase III
* Section now available for Phase IV Overview
* Updated User Guide to incorporate new features
* New images have been added from the latest version of our GUI
* Phase III Coding Documentation and Development History
* Updated Test Cases to incorporate new features
* Completed Testing Information from EAE
* Updated UML Diagram to reflect new class additions
* Updated Gantt Diagram / Project Plan depicting our actual timeline
* Troubleshoot section added to the User Guide
* New Reference added for MD5 API

### Phase IV

#### Activities Per Member

* Travis Parrot – Completed Test Cases and related documentation for Phase IV.
* Leigh Daniels – Wrote Phase IV Overview. Built updated UML Diagram. Updated the User Guide to include Random Password Generator functions. Approved final test cases and final documentation.
* James Lockhart – Coded Random Password Generator classes, and wrote Phase IV coding development history and developer comments.
* Matthew Seefeldt – Wrote pseudocode for Update 1.7 classes.
* Diana Van Hook – Wrote conclusions for Final Report
* Kendrick Walters – Incorporate Random Password Generator into GUI. Provided developer comments for GUI incorporation.
* Nathan Miranda – Validated Table of Contents. Final read-through of documentation.

#### Original Milestones & Schedule

The original milestones for the final week of our project consisted of concluding our Final Analysis. By March 5th, we anticipated making any final necessary changes to the Encryption Analysis Engine. We are on schedule, and have successfully completed this milestone. We also took this project a step further, and was able to incorporate an additional feature into our application prior to our deadline; the random password generator feature.

#### Problems Encountered

No problems were encountered while incorporating the random password generator.

#### Reevaluation of Decisions

Not applicable.

#### Changes to Documentation

* Phase IV Overview
* Phase IV Development History and Developer Comments
* Updated UML Diagram
* Updated User Guide to incorporate Random Password Generator feature
* Updated Design Approach to include new classes
* Test Results from password gen test case
* Conclusions for EAE Project

# Introduction

## Purpose

The purpose of the Encryption Analysis Engine (EAE) is to test the strength of various strings designed to be used as passwords. It will tell the user how long it took their desired string to be cracked. The engine will also tell the user how strong or weak the entered password is based on the amount of time it took to be cracked. The engine has real world application because passwords are used as a security measure for almost everything. In theory, stronger passwords take longer to crack. The more characters and changes a user makes to the string, the more secure it will be.

## Scope

1. Development will begin on January 22, 2017
2. Development will be completed by February 28, 2017
3. Application Includes
   1. The EAE will have a GUI that allows a user to experiment with password strength
   2. GUI will include:

* A drop down menu with three options: Brute Crack, MD5 hash, and password generator
* A single input field that will accept input between three and six alphanumeric character
* A running time clock to show the time-to-crack of a given input
* A strength categorization field to inform the user of the strength of their inputThree action buttons: Analyze, Rest, and Exit
  1. The EAE will contain three modules: Brute Crack, MD5 Hash, and password generator
  2. The brute crack module will:
  + Allow the user to input a password between 3 and 6 character
  + Upon entry, the EAE will crack the password by attempting every character combination possible until there is a match
  + Upon completion, the EAE will return the time-to-crack and the strength categorization of the user’s password
  1. The MD5 hashing module will:
  + Hash a user’s input password using MD5 algorithm
  + Provide the user with the MD5 hash
  + Indicate the strength of the password
  1. This module will utilize an external database to obtain the MD5 hash
  + https://md5db.net/api/
  1. The password generator module will:
  + Provide the user with a very strong password
  + Allow the user to select how many numbers, characters and special characters they would like to include in their password
  + Using the input, the generator will generate and display a password for the user to use
  + This password will meet “Strong” standards at a minimum

1. Application Does Not Include
   * Cannot accept more than six alpha numeric characters

## Cannot accept the special character input

## Background

Today, the world has a higher dependence on computing systems and the internet than ever before. Almost any transaction that you can imagine can be done from a computer. However, what the general public was not cognizant of was the lack of proper protection of their personal data during these transactions. When these processes were new, companies fell victim to all sorts of attacks: email viruses, loss or theft of removable media, or even accidental loss due to lack of cyber awareness. Many of these attacks, especially those concerning big names in business, have been reported to the public. But, for every one breach reported, dozens go unreported.

So, information security professionals started developing standards and procedures for combatting these attacks and for providing better protection for personal information. One of the most valuable contributions to this cause, which in fact has been around since the days of Julius Caesar, was cryptography. There are several different methodologies for using cryptography, but the most well-known method is encryption. Encryption is basically comprised of a mathematical algorithm for manipulating data, and a specific key that personalizes this algorithm to a distinct user or group. Generally, the mathematical algorithms are public knowledge. It’s the key that is kept secret.

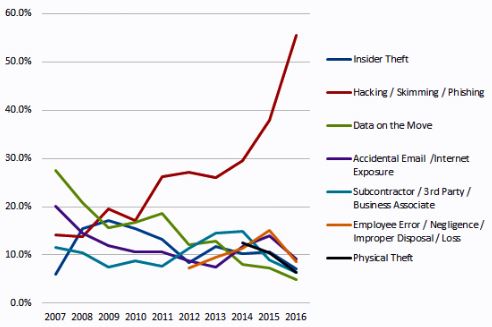
As you can see from the graph, as all other methods involved in data breaches have declined, Hacking/Skimming/Phishing has grown exponentially (Identity Theft Resource Center; Cyber Scout, 2017). Since the threat of hacking and phishing is continuing to grow, there’s a desperate need for analyzing encryption standards and password policy. There’s also a need for providing more education to users in regards to this matter. This is where the Encryption Analysis Engine comes into play.

Figure 1. Types of Data Breaches (Identity Theft Resource Center; Cyber Scout, 2017)

The Encryption Analysis Engine (EAE) shows its user how easy or how hard it is to perform cryptanalysis on user-input. The EAE gives the user different methods to choose from in order to learn more about the strength of their current password implementation, and how they can improve by providing newer, stronger password generation.

## Project Plan

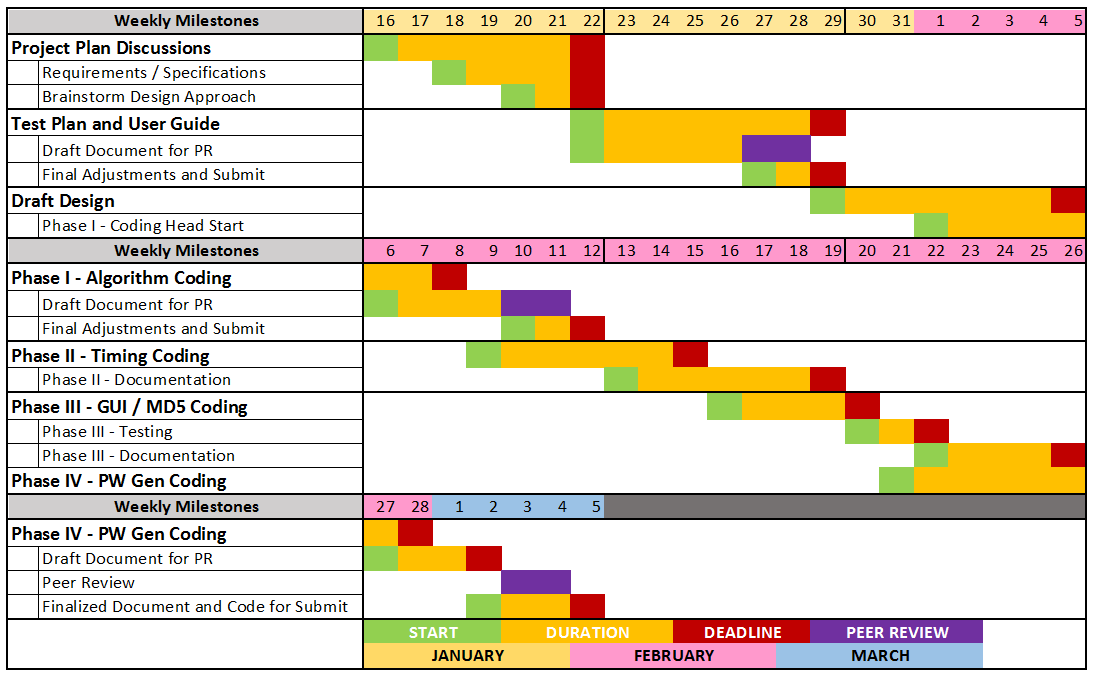


Figure 2. Project Plan Gantt Diagram

# Requirements & Specifications

## Problem Statement

Test the strength of passwords, and inform users of discovered strength and weakness.

## Requirements

* + 1. Must be user friendly.
    2. Must be able to take in user input that is no longer than a defined range and give output back to the user in a way that is understood by the end user.
    3. Will require the use of a graphical user interface (GUI).
    4. Will tell the user whether their password is weak or strong.
    5. Will use a timer to determine how long it takes for the password to be guessed.
    6. Time-to-crack will be used to determine the strength of the provided password.
    7. Should detect if a password is less than 6 characters.
    8. Should detect if a password is more than 6 characters.
    9. Should warn the user if an entered password is greater than 6 characters.
    10. Should be secure.

## Possible Scenarios for Use

* + 1. This application would be useful for a user when they create login accounts for e-commerce sites or banking applications to test the strength of a password before creating the account.
    2. This application would be useful for a business that needs to use secure passwords such as a bank, online store, or even a medical provider that must protect patient information.

## Definitions

1. User: The end user of the engine
2. Developer: The developer of the engine
3. EAE: Encryption Analysis Engine
4. GUI: Graphical User Interface
5. Milestone: A project synchronization point (Microsoft, 2008)
6. Timer: The time that it takes the EAE to guess the user’s password
7. Password: Alphanumeric String entered by the user
8. Brute Force: A method of breaking security measures that attempts all possible password combinations (PCMag, 2016)

## System Specifications

1. Hardware: PCs running Microsoft Windows or Mac operating systems will be used to develop and run code. Systems running the EAE should have at least 4GB of RAM and a Dual Core Processor in order to analyze at an acceptable speed.
2. Software: The software that will be used for the EAE will be NetBeans IDE Versions 8.1 or 8.2 and the Java Development Kit. The required version of Java to successfully run the program will be Java 8u192 or higher.

## Functional Specifications

1. EAE will be secure against malicious user input strings.
2. EAE will properly handle time-out and other programmatic errors that could otherwise cause deadlock for users’ computers.
3. EAE source code visibility will be limited in order to minimize the risk that the analysis algorithms can be used for malicious purposes.

# User Guide

## Welcome to the Encryption Analysis Engine

The Encryption Analysis Engine (EAE) is a tool designed to determine the strength of a password as entered by a user. The tool determines the strength of the password through the use of a brute force algorithm. EAE uses a timer to determine how long the password took to be guessed, then reports back to the user. The information gained from the use of this engine is useful in many different settings. This user guide was created for three reasons:

1. Provide information about the EAE
2. Give instructions on how to use the EAE, and
3. Provide answers to some frequently asked questions (FAQs) about the EAE that may come about as you use the engine.

## Minimum System Requirements

OS: Microsoft™ Windows™ 7 or Higher

Processor: Intel Core 2 Duo 2GHz+ or Better

Memory: 4 GB RAM

Java Run-Time Environment (JRE) Version 8 Update 1 Required

## Instructions

### Step 1: Launching the Application

In order to open the EAE’s Graphical User Interface (GUI), navigate to the application’s parent directory and run the executable JAR file.

To do this:

* Open Command Prompt (Type cmd in windows search box)
* Change your directory to wherever the EAE is stored (i.e. “cd C:\path\where\app\is\located”
* Type ‘java EncryptionAnalysisEngine’ and press enter to run the application.

### Step 2: Accepting Terms & Conditions

Each time you run the application, you will be shown a legal disclaimer indicating the application name, version number, and guidelines for acceptable use. You will be given options to either accept or decline the terms of use.

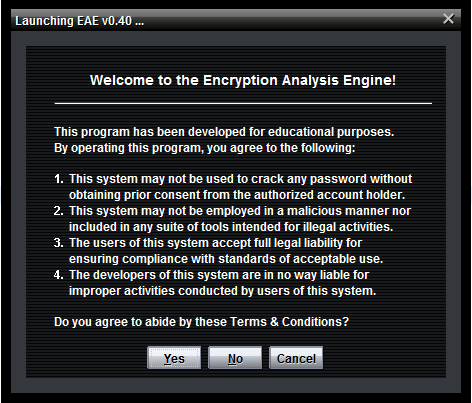


Figure 3. Terms & Conditions

If you agree to these terms, click the “OK” button to proceed to the main application interface. If, however, you choose not to accept these conditions by clicking “Cancel,” the application will immediately and automatically terminate. In order for the application to continue, you must accept the terms and conditions of use in order to access the main application interface.

### Step 3: Reviewing the Main Application Interface

Upon accepting the terms and conditions of use, the splash screen will disappear and immediately be replaced by main application interface.

From the main interface, you may use the drop-down menu to select an action to perform. In addition to the drop down menu, there is an input field and three action buttons. The input field will accept 3-6 alpha numeric characters that you are interested in analyzing. The analyze button causes the application to execute. The reset button clears the input field and any previous data. Finally, the exit button, closes the application. You may also choose to minimize, maximize, or close the application using the standard buttons provided at the top right-hand corner of the window:

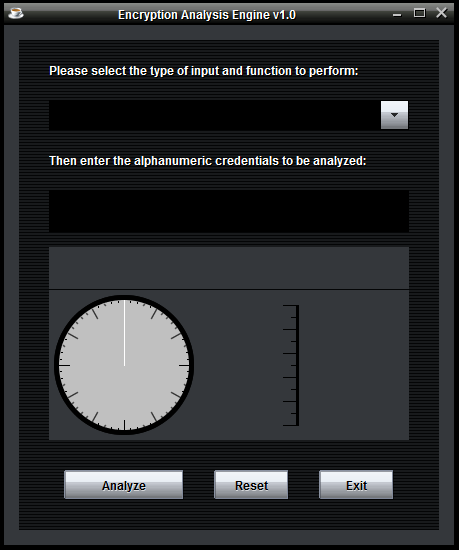


Figure 4. Schematic of EAE Main Application Interface

### Step 4: Brute Force Feature

* In the drop down menu, select Brute Force Crack

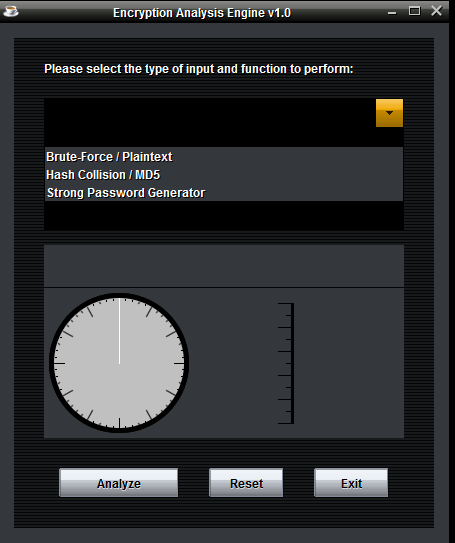


Figure 5. Drop Down Menu

* In the input field, input 3-6 alphanumeric characters you are interested in analyzing
* Select analyze
* The application will now begin attempting a brute crack

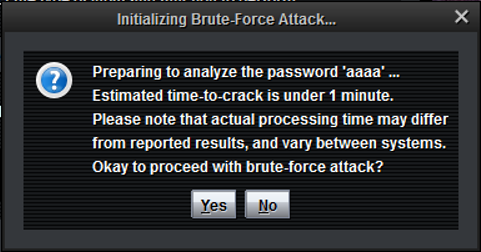


Figure 6. Time Analysis

* Upon the discovery of a match, the application will identify how strong your password was based upon the difficulty of cracking it and the time it took to crack the password
* Providing invalid input formats, such as special characters, whitespaces, less than three alphanumeric characters or longer more than 6 alphanumeric characters will result in a descriptive error message reminding you of the correct input parameters.

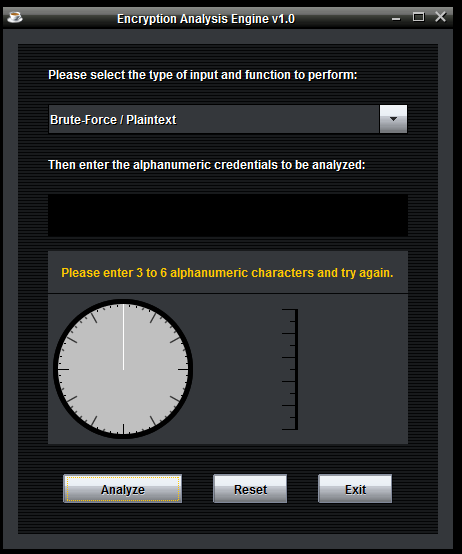


Figure 7. Invalid Input

* Successful Run Example:

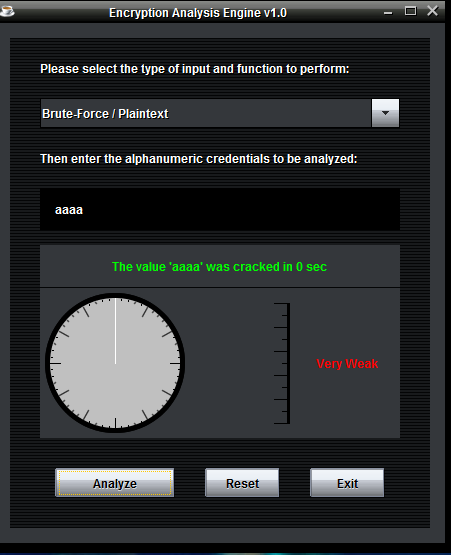


Figure 8. Successful Execution

### Step 4: Hash Collision / MD5

In the drop down menu, select Hash Collision / MD5. Then, input up to 6 alphanumeric characters, and click Analyze. Your input will be compared with an online rainbow table to determine how weak or strong your input it is.

### Random Password Generator

In the drop down menu, select Strong Password Generator. This feature will give you an example of a much stronger password you can use instead of a weak one. Enter up to 6 alphanumeric characters, and then click Analyze.

A new window will pop up. From here, you can select what new characters you would like to have added to your new, Strong password. Select as many, or as little additions as you’d like, and then click Ok.

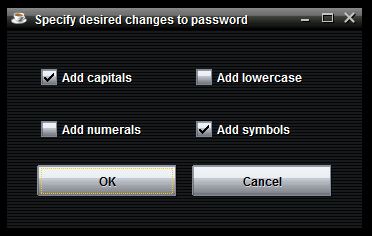


Figure 9. Strong Password Generator Selections

On the other window, you will receive a new, stronger password, based on your original input and the new additions you have selected. A stronger password such as this will allow you to better protect yourself from cyber criminals!

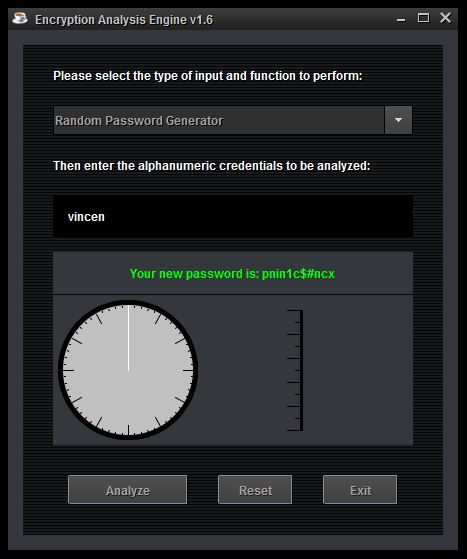


Figure 10. Example of Strong Password Generator

### Step 5: Terminating the Application

Upon program termination, the application will display a confirmation dialogue asking you to acknowledge that you wish to exit the application. Clicking “OK” will close this dialogue and terminate the application.



Figure 11. Terminating Application

## Frequently Asked Questions

### Can I enter more than 6 characters for a password?

The program will only accept between 2 and 6 alphanumeric characters. Due to the varying processing times of an individual’s computer, the crack may take longer than others. Adding a 7th character will increase the time it takes to crack the password exponentially.

### Will I be able to enter special characters such as @ or !?

As of right now, no. This program will only accept alphanumeric characters from A – Z and 0 – 1. Adding special character to your password will increase its strength though.

### Why does it take so long to find my password?

The algorithm searches for the most common letters used in the English language first. Using the search pattern of ETAONRISHDLFCMUGYPWBVKJXQZ we can quickly find if your password is of a standard English word. If your password is a mix of letters and numbers, the algorithm takes longer to solve because it must iterate through each alphanumeric character. Every letter/number that is added to a password will exponentially add to the time it takes to crack. The longer it takes to find the password, the stronger the password is.

### Is my password stored in a database after I run the program?

Your password isn’t stored in any type of database. The program will store the password temporarily to be tested and then immediately deleted once it has completed its task.

## Troubleshoot

## While trying to use the Hash Collision/MD5 feature, you may run into the problem below.

## 

Figure 12. SSL Handshake Error Message

## There are two things that you will need to check in order to resolve this problem. First, you must make sure that you have at least Java Version 8 Update 1, as described in the minimum system requirements. To get the latest updates, you can go to <http://www.oracle.com/technetwork/java/javase/downloads>.

## Next, you must ensure that your Environment Variable has been updated. To do this, click on the Start button, Right Click on Computer, and click on Properties.

## 

Figure 13. Properties, Change Settings

## From here, you will need to click the link to Change Settings. Then, you will Right Click on the Advanced Tab, and then click Environment Variables.

## 

Figure 14. System Properties

## Under the User Variables, you need to make sure that JAVA\_HOME and PATH are pointed to the correct folders. Please see an example below.

## 

Figure 15. Environment Variables

## Once these changes have been made, try to run the Encryption Analysis Engine’s MD5 option again. You shouldn’t see the error come up anymore.

# Test Plan and Results

## Introduction

### Project Overview

The Encryption Analysis Engine (EAE) will analyze plaintext passwords, and give the user statistical information about the level of security of the given password or hash that the user has input. The purpose of utilizing the EAE is to understand, in real-world terms, the exponential increase in security when longer credential strings are used.

### Testing Process Overview Diagram

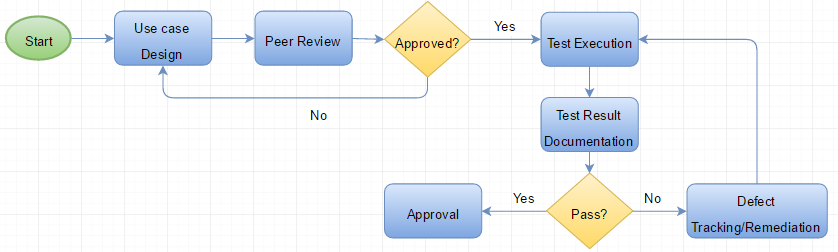


Figure 16. Testing Process Overview Diagram

## Testing Schedule

### Phase I (Complete)

Phase I will focus on the design of the core element of the application, designing an efficient password cracking algorithm capable of cracking 6 character alphanumeric passwords.

1. Design password cracking algorithm
2. Utilizing the functional algorithm, test 2 character, numeric only, passwords to ensure functionality
3. Test 2-character alphanumeric password
4. Test 4-character alphanumeric password
5. Test 6-character alphanumeric password

Once the algorithm is capable of cracking 6 character alphanumeric passwords the testing will move on to phase II.

### Phase II (Complete)

Phase II of the application testing will focus on ensuring the proper input validation restrictions are put in place to protect the program from crashing.

1. Test for empty input (Fail, but not crash)
2. Test for >6-character input (Fail, but not crash)
3. Test for 1-6-character input
4. Test for all numeric input
5. Test for all letter input
6. Test for special character input (Fail, but not crash)
7. Test for upper case input
8. Test for lower case input

Once all input validation is handled as expected by the System requirements and specifications (SRS) document, testing will move onto phase III.

### Phase III (Complete)

Phase III of the testing schedule will incorporate the time-to-crack aspect of the application. This phase will aim to achieve the most efficient algorithm possible for cracking 6 character alphanumeric passwords.

1. Add timer
2. Run Test cases and discover averages of each character combination

The test results will be provided to the design lead and further time-to-crack enhancements could be pursued, then retested.

### Phase IV (Complete)

During phase IV the graphical user interface (GUI) will be added to the application. Using testing and feedback, this phase of the testing process will be focused on achieving the most user-friendly application possible.

1. Utilize GUI to test all previously defined password combinations from Phases I-III
2. Document possible quality improvements
3. Provide feedback to design team
4. Retest enhanced GUI

After each cycle of phase IV testing, feedback will be provided to the design lead. Once adjustments are completed, the GUI will be retested.

## Use Case Design

### Purpose

Use cases will provide the exact steps that are required from the tester to test a specific requirement or function included in the application. Each use case will have a specific goal or requirement that is being tested. Exact input and the expected output of the test will be included prior to the test. Once the test is complete, the tester will document the pass or fail criteria.

### Use Case Design Diagram

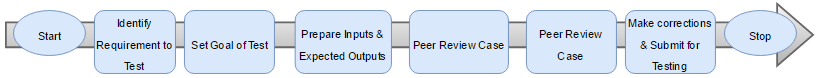


Figure 17. Use Case Design Process Diagram

## Test Execution Strategy

### Purpose

The purpose of this section is to describe the exact execution strategy that will occur throughout the development of the application. Ensuring all testers follow the same execution strategy will help to ensure the accuracy of each test performed.

### Objectives

Discover and eliminate application defects. Ensure that final product is suitable for production environment.

### Assumptions

Personal computer with a minimum of 4 GB of RAM.

### Test Execution Process Diagram

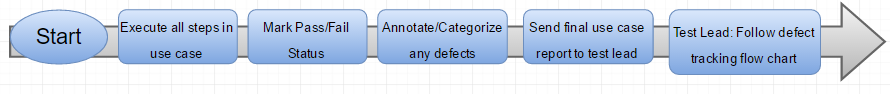


Figure 18. Test Execution Process Diagram

## Defect Tracking & Reporting

If a use case fails, the use case will be returned to the test lead, who will validate and categorize the defect and then explain the defect to the development team lead. If there are no defects and the use cases passes testing, the case will be submitted to the project manager (PM) for approval.

### Defect Categorization

|  |  |
| --- | --- |
| **Severity** | **Impact** |
| Category 1 | * Critical enough to crash system * Causes program to hang infinitely * <Additional criteria> |
| Category 2 | * Causes program to run for extended duration * Causes user confusion * Overlooks a portion of the results (i.e. no time-to-crack provided to the user) * <Additional criteria> |
| Category 3 | * Unclear prompts, causing negative impact on application * Cosmetic issues * <Additional Criteria> |

Figure 19. Defect Categorization Table

### Defect Tracking Diagram

Tester: ID Defects

Test Lead: Submit Defects to Dev Lead

Test Lead: Validates Defects

Development Lead: Assign Defects

No

Resolved?

Developer: Fix Defects

Test Lead: Retest Defect

Developer: Submit new code to test lead

No

Yes

Approve?

PM: Approve or Deny Use Case

Test Lead: Submit Use case to PM for Approval

Figure 20. Defect Tracking Diagram

## Use Cases

### Case ID#: 001

|  |  |
| --- | --- |
| **Use Case ID#: 001** | **Testing for > 6 Character Input** |
| Goal | Ensure that the end-user inputs 6 characters or less. If not, provide errors prompt |
| Requirements being tested | (Taken from SRS v3)  9. Should detect if a password is more than 6 characters.  10. Should warn the user if an entered password is greater than 6 characters. |
| Application event | A user inputs more than 6 characters into the input field of the GUI and hits “Analyze” button |
| Standard I/O | Input: 1234567  Expected Output: Error Prompt |
| Pre-Conditions | Application must be running. User must input 7+ characters. User must select “Analyze” button |
| Expected Post-Conditions | User is prompted with an error message box stating: “You have entered more than 6 characters. Please enter 6 or fewer characters and try again.”  Input field is cleared in the GUI  Timer resets for “Time-to-Crack field” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| n/a |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Sucess Outcomes | Input | Expected Output |
| Upon entry of 7 characters, the proper error was displayed | 1234567 | Error |

### Case ID#: 002

|  |  |
| --- | --- |
| **Use Case ID#: 002** | **Testing for All Letters Input** |
| Goal | Ensure that the application can still accept, and analyze, input that is composed of only letters. |
| Requirements being tested | 3.2 Must be able to take in user input that is no longer than a defined range and give output back to the user in a way that is understood by the end user. |
| Application event | A user enters only letters into the input field of the GUI and hits “Analyze” button |
| Standard I/O | Input: aaaaaa  Expected Output: Time-to-Crack prompt |
| Pre-Conditions | Application must be running. User must input only letters. User must select “Analyze” button |
| Expected Post-Conditions | The EAE analyzes this input very quickly.  The EAE will let the user know that the password is very weak.  Input field is cleared in the GUI.  Timer resets for “Time-to-Crack field”. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| GUI displayed crack time as “0 sec” when time to crack was greater than 0 seconds | aaaaa | Crack time & Password strength | Processing overhead | 2 |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| Added disclaimer to the alert pop-up indicating that processing times may differ from reported time-to-crack. This is likely the result of processing overhead in the application itself. | aaaaa | Crack time & Password strength |

### Case ID#: 003

|  |  |
| --- | --- |
| **Use Case ID#: 003** | **Testing for All Numeric Input** |
| Goal | Ensure that the application can still accept, and analyze, input that is composed of only numbers. |
| Requirements being tested | 3.2 Must be able to take in user input that is no longer than a defined range and give output back to the user in a way that is understood by the end user. |
| Application event | A user enters only numbers into the input field of the GUI and hits “Analyze” button. |
| Standard I/O | Input: 111111  Expected Output: Time-to-Crack prompt |
| Pre-Conditions | Application must be running. User must input only numbers. User must select “Analyze” button. |
| Expected Post-Conditions | The EAE analyzes this input very quickly.  The EAE will let the user know that the password is very weak.  Input field is cleared in the GUI.  Timer resets for “Time-to-Crack field”. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| n/a |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| Upon entry of 11111, the result: 7 second & very weak was displayed | 11111 | Crack time & Password strength |

### Case ID#: 004

|  |  |
| --- | --- |
| **Use Case ID#: 004** | **Testing for A Mixture of Alphanumeric Characters** |
| Goal | Ensure that the application can accept, and analyze, input that is composed of alphanumeric characters. |
| Requirements being tested | 3.2 Must be able to take in user input that is no longer than a defined range and give output back to the user in a way that is understood by the end user. |
| Application event | A user enters alphanumeric characters into the input field of the GUI and hits “Analyze” button |
| Standard I/O | Input: aaa123  Expected Output: Time-to-Crack prompt |
| Pre-Conditions | Application must be running. User must input both letters and numbers. User must select “Analyze” button |
| Expected Post-Conditions | The EAE analyzes this input.  The EAE will let the user know that the password is strong.  Input field is cleared in the GUI.  Timer resets for “Time-to-Crack field”. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| n/a |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| Upon entry of aaa12 , the result: 16 second & very weak was displayed | aaa12 | Crack time & Password strength |

### Case ID#: 005

|  |  |
| --- | --- |
| **Use Case ID#: 005** | **Testing for > 6 Character Input** |
| Goal | Ensure that the user inputs 6 or fewer characters; if this condition is not satisfied, provide a descriptive error message noting proper input parameters |
| Requirements being tested | (Taken from SRS v4)  3.2 Must be able to take user input that is no longer than a defined range.  3.8 Should detect if a password is more than 6 characters.  3.9 Should warn the user if an entered password is an incorrect length.  3.10 Should be secure (sanitize any potentially-malicious user input) |
| Application event | A user inputs more than 6 characters into the input field of the GUI and hits “Analyze” button |
| Standard I/O | Input: aaaaaaa  Expected Output: Error prompt |
| Pre-Conditions | Application must be running.  User must input 7 or more characters.  User must select “Analyze” button. |
| Expected Post-Conditions | User is prompted with an error message box stating: “Your input exceeds the maximum allowed input length. Please enter 6 or fewer characters and try again.”  Input field is cleared in the GUI  Timer resets for “Time-to-Crack field” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| n/a |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| Upon entry of >2 characters, the proper result was displayed | aa | Error prompt |

### Case ID#: 006

|  |  |
| --- | --- |
| **Use Case ID#: 006** | **Testing for < 3 Character Input** |
| Goal | Ensure that the user inputs 3 or more characters; if this condition is not satisfied, provide a descriptive error message noting proper input parameters |
| Requirements being tested | (Taken from SRS v4)  3.7 Should detect if a password is less than 6 characters.  3.9 Should warn the user if an entered password is an incorrect length. |
| Application event | A user inputs less than 3 characters into the input field of the GUI and hits “Analyze” button |
| Standard I/O | Input: aa  Expected Output: Error prompt |
| Pre-Conditions | Application must be running.  User must input 2 or fewer characters.  User must select “Analyze” button. |
| Expected Post-Conditions | User is prompted with an error message box stating: “Your input does not meet the minimum allowed input length. Please enter 3 or more characters and try again.”  Input field is cleared in the GUI  Timer resets for “Time-to-Crack field” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| There were no errors handling special input | aaa(+ every special character) | Error prompt: Special characters not allowed | n/a | n/a |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| Error prompt: Special characters not allowed | aaa(+ every special character) | Error prompt: Special characters not allowed |

### Case ID#: 007

|  |  |
| --- | --- |
| **Use Case ID#: 007** | **Testing for Invalid Character Input** |
| Goal | Ensure that the user inputs only alphanumeric characters (A-Z, 0-9); if this condition is not satisfied, provide a descriptive error message noting proper input parameters |
| Requirements being tested | (Taken from SRS v4)  3.10 Should be secure (sanitize any potentially-malicious user input) |
| Application event | A user inputs non-alphanumeric characters into the input field of the GUI and hits “Analyze” button |
| Standard I/O | Input: aaa#  Expected Output: Error prompt |
| Pre-Conditions | Application must be running.  User must input non-alphanumeric characters.  User must select “Analyze” button. |
| Expected Post-Conditions | User is prompted with an error message box stating: “Your input includes invalid characters. Please enter 3 to 6 alphanumeric characters and try again.”  Input field is cleared in the GUI  Timer resets for “Time-to-Crack field” |

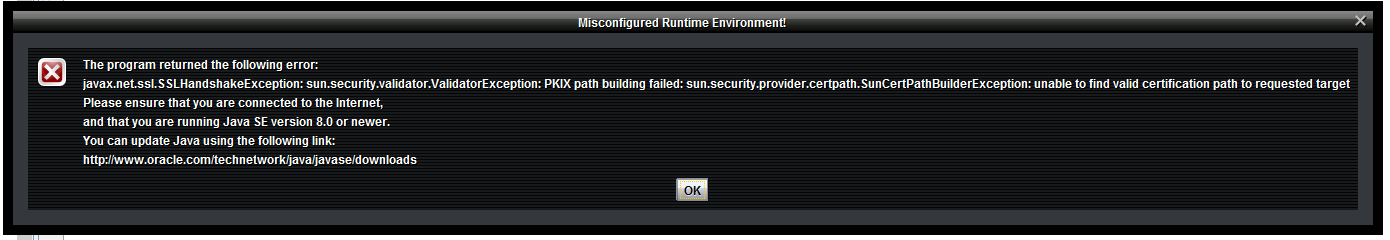
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| n/a |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| Error message has been changed to read "Please enter 3 to 6 alphanumeric characters and try again." | ‘Blank’ | Error prompt |

### Case ID#: 008

|  |  |
| --- | --- |
| **Use Case ID#: 008** | **Testing for Timing Assigned to Strong** |
| Goal | Accurately count the number of computational steps. |
| Requirements being tested | Verify that the time complexity has an asymptotic-growth rate that is quadratic. |
| Application event | A user inputs 6 characters into the input field of the GUI and hits “Analyze” button. |
| Standard I/O | Input: Pwd999  Expected Output: Time-to-Crack prompt & Strong categorization assigned |
| Pre-Conditions | Application must be running. User must input >6 characters. User must select “Analyze” button |
| Expected Post-Conditions | User is prompted with the output:  Time-to-Crack should be near the established ‘strong average |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| Caused memory warning | Pwd999 | Time-to-Crack prompt & Strong categorization assigned | Testing Java environment was outdated causing the program to hang Team added error prompt to update java if environment is incorrect (See below) | 1 |



|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| PWd999 was cracked in 1080 seconds | Pwd999 | Time-to-crack |

### Case ID#: 009

|  |  |
| --- | --- |
| **Use Case ID#: 009** | **Testing for Timing Assigned to Strong** |
| Goal | Accurately count the number of computational steps. |
| Requirements being tested | Verify that the time complexity has an asymptotic-growth rate that is quadratic. |
| Application event | A user inputs more than 6 characters into the input field of the GUI and hits “Analyze” button. |
| Standard I/O | Input: Pwd999  Expected Output: Time-to-Crack prompt & Strong categorization assigned |
| Pre-Conditions | Application must be running. User must input >6 characters. User must select “Analyze” button |
| Expected Post-Conditions | User is prompted with the output:  Time-to-Crack should be near the established ‘strong average |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| n/a |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| Time-to-Crack: 0 sec  Password categorization: very weak | abc | Time-to-Crack should be near the established ‘weak’ average |

### Case ID#: 010

|  |  |
| --- | --- |
| **Use Case ID#: 010** | **Testing for Response – Application Re-Use** |
| Goal | The goal of this test case is to tell if the user enters a duplicate string or not. |
| Requirements being tested | 3.1 Program must be user friendly.  3.10 Program should be secure. |
| Application event | A user puts in the password of their choosing. User click analyze button. Upon results, clicks reset |
| Standard I/O | Input: abc123  Expected Output: Time-to-Crack prompt (User selects reset) All fields are cleared for re-use |
| Pre-Conditions | Application must be running. User must input characters of their password. User must click analyze button. |
| Expected Post-Conditions | Input field is cleared in the GUI  Timer resets for “Time-to-Crack field” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| n/a |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| Time-to-Crack prompt. upon reset all fields are cleared | abc | Time-to-Crack prompt. upon reset all fields are cleared |

### Case ID#: 011

|  |  |
| --- | --- |
| **Use Case ID#: 011** | **Testing for Response – MD5 numeric input** |
| Goal | Ensure all numeric input can be hashed with MD5 and proper results displayed |
| Application event | A user puts in the password of numeric input choosing MD5 option. User click analyze button. |
| Standard I/O | Input: 1234  Expected Output: Hash value and strength categorization |
| Pre-Conditions | Application must be running. User must input characters of their password. User must click analyze button. |
| Expected Post-Conditions | User is prompted with a message box telling them if their password MD5 hash and strength |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| The results are not explained to the user. Highly suggest a prompt that explains the results | 1234 | Hash results | Testing Java environment was outdated causing. Team added error prompt to update java if environment is incorrect (See figure above) | 2 |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| '81dc9bdb52d04dc20036dbd8313ed055' matched in 1 sec  Strength: very weak | 1234 | Hash results |

### Case ID#: 012

|  |  |
| --- | --- |
| **Use Case ID#: 012** | **Testing for Response – MD5 letter input** |
| Goal | Ensure all letter input can be hashed with MD5 and proper results displayed |
| Application event | A user puts in the password of letter input choosing MD5 option. User click analyze button. |
| Standard I/O | Input: aaaa  Expected Output: Hash value and strength categorization |
| Pre-Conditions | Application must be running. User must input characters of their password. User must click analyze button. |
| Expected Post-Conditions | User is prompted with a message box telling them if their password MD5 hash and strength |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| The results are not explained to the user. Highly suggest a prompt that explains the results | aaaa | Hash results | Incorrect java environment. Error has been added (See figure above) | 2 |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| '74b87337454200d4d33f80c4663dc5e5' matched in 0 sec  Very weak | aaaa | Hash results |

### Case ID#: 013

|  |  |
| --- | --- |
| **Use Case ID#: 013** | **Testing for Response – MD5 alphanumeric input** |
| Goal | Ensure alphanumeric input can be hashed with MD5 and proper results displayed |
| Application event | A user puts in the password of letter input choosing MD5 option. User click analyze button. |
| Standard I/O | Input: aaa12  Expected Output: Hash value and strength categorization |
| Pre-Conditions | Application must be running. User must input characters of their password. User must click analyze button. |
| Expected Post-Conditions | User is prompted with a message box telling them if their password MD5 hash and strength |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| The results are not explained to the user. Highly suggest a prompt that explains the results | aaa12 | Hash results | Incorrect java environment. Error has been added (See figure above) | 2 |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| No match found for '6cfbbefaedf841d8537b39bd9a582333'  Strength: Excellent | aaa12 | Hash results |

### Case ID#: 014

|  |  |
| --- | --- |
| **Use Case ID#: 014** | **Testing for timing** |
| Goal | Ensure proper crack time is displayed.  During this test the tester started a stop watch upon the analyze click. If time to crack vs stopwatch time was >30 second difference, the test resulted in a failure |
| Application event | A user puts in the password. User click analyze button. |
| Standard I/O | Input: aaa1, aaa12, aaaaaa, Pwd999  Expected Output: Hash value and strength categorization |
| Pre-Conditions | Application must be running. User must input characters of their password. User must click analyze button. |
| Expected Post-Conditions | User is prompted with a message box telling them the time to crack and strength |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| Time-to –crack: 18 sec  Stopwatch: 1m 52sec | aaa12 | Time-to-crack close to stopwatch time | Likely the result of processing overhead. | 2 |
| Application hung | aaaaaa | Time-to-crack close to stopwatch time | Likely the result of processing overhead. | 1 |

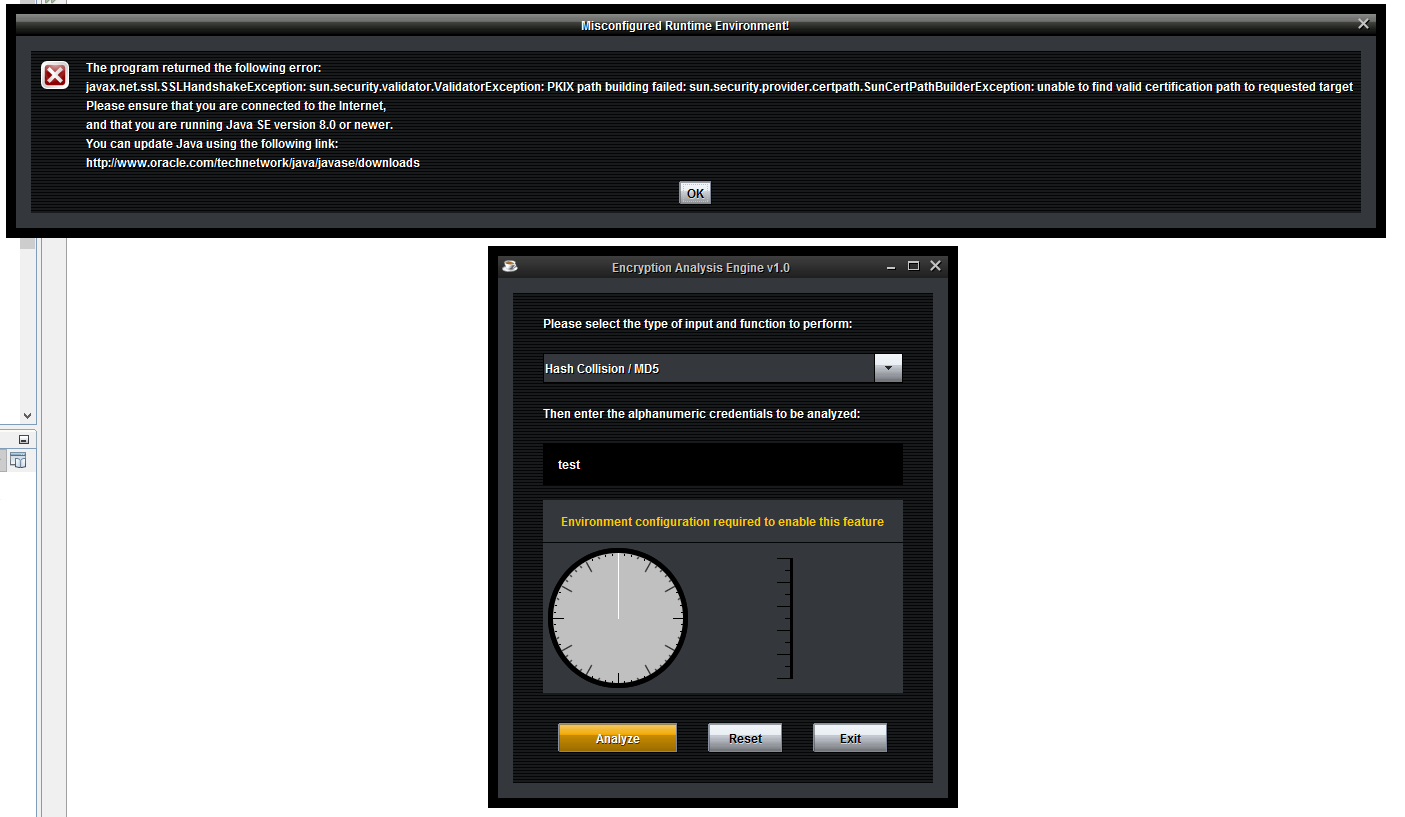
|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| Time-to –crack: 0 sec  Stopwatch: 2 sec | aaa1 | Time-to-crack close to stopwatch time |
| Time-to –crack: 1080 sec  Stopwatch: 1096 sec | aaa12 | Time-to-crack close to stopwatch time |
| Time-to –crack: 0 sec  Stopwatch: 0 sec | aaaaaa | Time-to-crack close to stopwatch time |
| Time-to –crack: 2400 sec  Stopwatch: 2405 sec | Pwd999 | Time-to-crack close to stopwatch time |

### Case ID#: 015

|  |  |
| --- | --- |
| **Use Case ID#: 015** | **Testing for Java Environment error** |
| Goal | Ensure application throws an error if the user attempts MD5 module with java < 8u121 |
| Application event | A user puts in the password of choosing and selects MD5 option. User click analyze button. |
| Standard I/O | Input: test (With outdated Java)  Expected Output: Error |
| Pre-Conditions | Application must be running. User must input characters of their password. User must click analyze button. |
| Expected Post-Conditions | User is provided java error prompt |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| n/a |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| See Figure below | test | java error prompt |



### Case ID#: 016

|  |  |
| --- | --- |
| **Use Case ID#: 016** | **Testing for GUI Running Clock** |
| Goal | Ensure GUI has running clock through brute force execution |
| Application event | A user puts in the password of choosing and selects brute-force option. User click analyze button. |
| Standard I/O | Input: aaa12  Expected Output: GUI clock ticks every second and changes color as time progresses |
| Pre-Conditions | Application must be running. User must input characters of their password. User must click analyze button. |
| Expected Post-Conditions | User is provided the time-to-crack. GUI ticks once every second throughout processing |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| n/a |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| GUI ticked properly 1080 times adding a minute tracker each 60 ticks. | aaa12 | Time-to-crack; Password strength; Proper GUI function |

### Case ID#: 017 (Pending functional Development)

|  |  |
| --- | --- |
| **Use Case ID#: 016** | **Testing for password generator** |
| Goal | Ensure password generator can generate various passwords |
| Application event | A user puts clicks the password generator option. User click analyze button. |
| Standard I/O | Input: Need  Expected Output: Strong password |
| Pre-Conditions | Application must be running. User must input characters of their password. User must click analyze button. |
| Expected Post-Conditions | User is provided the strong password including the selections they chose |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Failure Outcomes | Input | Expected Output | Cause of failure | Defect Severity |
| n/a |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Success Outcomes | Input | Expected Output |
| Several random words, number combinations were tried, and all resulted in a successful, randomly generated password based on the selections from the user. | Abc123 | A strong, random password as expected: 13b\*132pcbg/-8L |

## Standardize I/O

The table below is used to document the exact strings that will be used for the test cases. The purpose of providing standard I/O is to ensure that tests are accurate when performed by various testers. This table will be a reference tool for testers.

|  |  |  |
| --- | --- | --- |
| **Use case ID** | **Input** | **Expected Output** |
| 1 | 1234567 | Error: “Too many characters” |
| 2 | aaaaaa | Time-to-crack results |
| 3 | 111111 | Time-to-crack results |
| 4 | aaa123 | Time-to-crack results |
| 5 | aaaaaaa | Error prompt: “too many characters” |
| 6 | aa | Error prompt: “too few characters” |
| 7 | aaa# | Error prompt: “Special character entry” |
| 8 | Blank | Error prompt: “Blank entry” |
| 9 | Pwd999 | Time-to-Crack prompt & Strong categorization assigned |
| 10 | abc | Time-to-crack results are close to equal to established average |
| 11 | abc123 | Time-to-Crack prompt (User selects ok) All fields are cleared for re-use |
| 12 | qaz123 | Time-to-Crack prompt & Strong categorization assigned |
| 13 | abc | Time-to-Crack prompt & Weak categorization assigned |
| 17 | Abc123 | Generates a Strong, Random Password |

Figure 21. Standard I/O

## Approved Tests

All approved tests will be tracked on the table below for record keeping.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID** | **Tester** | **Approving Official** | **Date Approved** | **Notes** |
| 001 | Travis | Leigh | 2/24/2017 | Concur |
| 002 | Travis | Leigh | 2/24/2017 | Team made changes to disclaimer addresses this issue. |
| 003 | Travis | Leigh | 2/24/2017 | Concur |
| 004 | Travis | Leigh | 2/24/2017 | Concur |
| 005 | Travis | Leigh | 2/24/2017 | Concur |
| 006 | Travis | Leigh | 2/24/2017 | Concur |
| 007 | Travis | Leigh | 2/24/2017 | Team made change to error message. |
| 008 | Travis | Leigh | 2/25/2017 | No errors once environment is properly configured. |
| 009 | Travis | Leigh | 2/24/2017 | Concur |
| 010 | Travis | Leigh | 2/24/2017 | Concur |
| 011 | Travis | Leigh | 2/25/2017 | No errors once environment is properly configured. |
| 012 | Travis | Leigh | 2/25/2017 | No errors once environment is properly configured. |
| 013 | Travis | Leigh | 2/25/2017 | No errors once environment is properly configured. |
| 014 | Travis | Leigh | 2/24/2017 | Team added disclaimer to address this problem. |
| 015 | Travis | Leigh | 2/25/2017 | Concur |
| 016 | Travis | Leigh | 2/25/2017 | Concur |
| 017 | Travis | Leigh | 3/2/2017 | Concur |

Figure 22. Approved Tests Table

# Design Approach

This design contains the building blocks for coding phases. It details the classes, methods, interfaces, fields, and parameters that will be needed at each stage of the program. The source code should be in a package named “passbreak”. Any images, dictionaries, test files, libraries, and any other files needed for the program to work should also be in this folder. Each major function will be in its own class, along with any associated methods needed for that class to work. The GUI will require four classes, one for each of the following: text field panel, menu panel, button panel, and the main GUI builder. The GUI will be implemented in phases. To start, the legal notice will be implemented. This will be followed by a simple GUI with simple functions. Finally, advanced themes and features will be implemented towards the end of the project in order to create a great user experience along with a reliable algorithm. We are taking this approach because if the algorithms don’t work, there is no point in having a GUI.

The PassBreak and BruteCrack classes will be the first algorithms to be designed for the EAE program. In the first couple of updates, the program will implement command line interface (CLI) for initial testing. The PassBreak class will push the necessary variables to the BruteCrack class and pass the results to the CLI. We will measure the efficiency of the BruteCrack class through the timing of methods and reevaluate if the algorithm needs to be adjusted. Once a class is approved through the testing phases, it will be modified for easy implementation into a GUI. We are taking this approach because if the algorithms don’t work, there is no point in having a GUI. The design of the MD5 class will be built as a separate program for initial testing. This class will also take the same approach as PassBreak and BruteCrack classes and be separated for easy implementation into the GUI phases of coding. All changes and designs will be listed as updates in the development history section.

## UML Diagram

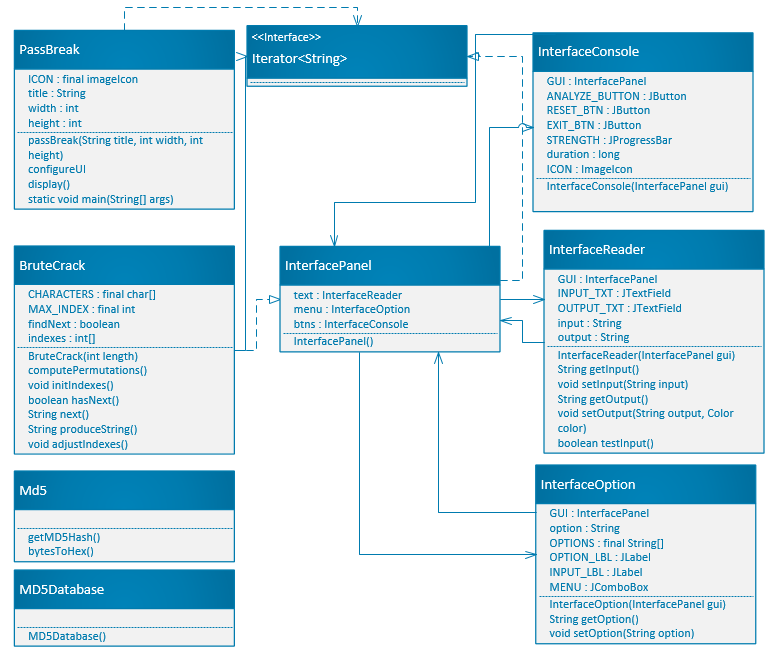


Figure 23. UML Diagram

## PassBreak – Class

PassBreak is the main class in the EAE program. In later phases, it will be used to configure the theme of the GUI windows, start the necessary GUIs, and end the program on GUI window close. Additionally, it will handle the legal notice at the beginning of the program and determine which action to take based off of user selection.

### Methods

**passBreak**

The PassBreak method is used as the primary constructor for this class. Its job will be to create the GUI window for the main application. It will set the title, size, and close operation.

1. Create GUI window.
2. Set window title.
3. Set window size.
4. Set the window’s default closing operation.

**configureUI**

The configureUI method is used to configure the global user interface settings and applies them to all windows. A for loop will iterate through all existing interface elements and apply the configured theme.

1. Handle exceptions resulting from missing or invalid directory path to extension libraries.
2. Iterate through each user interface instance to retrieve default configuration.
3. Modify default configuration to use a custom theme based on included extensions.
4. Add a line border and padding around each user interface instance.

**display**

The display method’s only purpose is to set the visible property of the primary interface to be true. This will make the GUI pop up on the user’s desktop.

* + - 1. Set window visibility to true

**main**

The purpose of the main method is to execute the program. It will call the configureUI method, create and display the legal notice window, and (in later phases) create and display the program GUI; note that initial implementation (Phase I-III) will use a Command Line Interface (CLI).

1. Start the configureUI method.
2. Create the legal notice window.
3. Display the legal notice window.
4. Create program GUI.
5. Display program GUI.

## BruteCrack – Class

The BruteCrack class implements the brute force password cracker capability of the program. It defines a character array that includes all characters to be tried during a brute force attempt.

### Methods

**BruteCrack**

The BruteCrack method is the primary constructor for this class. Its job is to create a BruteCrack object with a specified length as its only argument. This length is used to set the length of the password to be cracked. It then calls the computerPermutations method and returns the result as the password guess.

1. Receive Int as parameter
2. Define value of variable
3. Call initIndexes method

**computePermutations**

The computePermutations method will attempt to recursively brute force guess a password. It will use an incrementing index and available characters will be stored in a switch statement.

* + - 1. Define string variable
      2. Switch statement that has all of the available letters
      3. Append next indexed letter to string
      4. Return string as password guess

## InterfaceConsole – Class

The InterfaceConsole class will define and create the appearance and layout of buttons within the GUI.

### Methods

**InterfaceConsole**

The InterfaceConsole method is the primary constructor for the InterfaceConsole class. It will create and configure all of the buttons that will be used in the GUI. Additionally, it will add action listeners for the buttons that will call the appropriate methods and conduct some basic timing of how long the methods run for.

1. Set the GUI parameter equivalent to a corresponding private constant value.
2. Define grid layout for the drop-down menu as well as an empty border for padding.
3. Add a visual representation to graphically plot password strength as a function of time.
4. Add buttons to analyze user-provided input, reset parameters, and exit the application.
5. Add an actionlistener to the “analyze” button which retrieves the selected drop-down menu option, reads user text input, and performs input validation; if the input passes validation conditions, perform the selected function and display the results as output.
6. Add an actionlistener to the “reset” button which restores system default settings.
7. Add an actionlistener to the “exit” button which hides the main application interface, displays a goodbye message, and terminates the application.

## InterfaceOption – Class

The InterfaceOption class will define and create the appearance of the drop-down menu within the GUI.

### Methods

**InterfaceOption**

The InterfaceOption method is the primary constructor for the InterfaceOption class. Its job will be to add the drop-down menu to the InterfacePanel.

1. Set the GUI parameter equivalent to a corresponding private constant value.
2. Define grid layout for the drop-down menu as well as an empty border for padding.
3. Add a list of predefined options to the drop-down menu and set default to none.
4. Add the drop-down menu as well as a corresponding label to the GUI panel.
5. Add an actionlistener to the drop-down menu changing options based on user selection.

**getOption**

The getOption method is the getter method for the drop-down menu. It will return the selection option.

1. Return option

**setOption**

The setOption method is the setter method for the drop-down menu. It will set the selected option.

1. Receives String as parameter
2. Set menu’s selected item to be parameter received

## InterfacePanel – Class

The InterfacePanel class will define the placement of individual elements within the GUI. It will create objects for the text, menu, and buttons, and add them to a panel.

### Methods

**InterfacePanel**

The InterfacePanel method is the primary constructor for the InterfacePanel class. It will set the configuration and placement of each of the text, menu, and button panels.

1. Define grid layout for text field elements.
2. Define an empty border for padding.
3. Add a menu element constructed from InterfaceOption.
4. Add a text field element constructed from InterfaceReader.
5. Add a button console element constructed from InterfaceConsole.

## InterfaceReader – Class

The InterfaceReader class will define the appearance of the text input and output fields within the GUI.

### Methods

**InterfaceReader**

The InterfaceReader method is the primary constructor for the InterfaceReader class. It will be used to add the text input and output fields into the GUI.

1. Set the GUI parameter equivalent to a corresponding private constant value.
2. Define grid layout for text field elements as well as an empty border for padding.
3. Define the appearance of the text fields, being sure to set output as not editable.
4. Add the input and output text fields to the GUI panel.

**getInput**

The getInput method will be the getter method for the input text field.

1. Return input

**setInput**

The setInput method will be the setter method for the input text field.

1. Receive String as parameter
2. Set input text as String received

**getOutput**

The getOutput method will be the getter method for the text output field.

1. Return output

**setOutput**

The setOutput method will be the setter method for the text output field.

1. Receive String and Color as parameters
2. Set output text as String received
3. Set foreground color of textbox to color received

**testInput**

The testInput method will be used to conduct input validation of text. It will check for compliance with formatting requirements and will output errors if necessary.

1. Retrieve user-defined input from the text field element.
2. Define a boolean variable to store the validity or invalidity of the input.
3. Construct an “if - else if - else” statement containing the following conditions:
   1. Test for input length less than 3 characters.
   2. Test for input length greater than 6 characters.
   3. Test for input containing non-alphanumeric characters.
4. In the case of any of the three above-listed conditions, print a descriptive error message to the output field; otherwise set the validity variable to true and clear the output field.

## MD5 – Class

The MD5 class will convert the password chosen by the user into a hash string variable. To achieve this, the program will import two classes: MessageDigest and DatatypeConverter. The MessageDigest class provides our application the functionality of the message digest algorithm, such as MD5. Message digests are a secure one-way hash function that takes arbitrary-sized data and outputs a fixed-length hash value. Next, the program will use the DatatypeConverter class to convert the byte data to a readable/printable hash string.

### Methods

**getMD5Hash**

The getMD5Hash method will return the MD5 hash for a given string.

1. Create digest object for MD5 conversion
2. Create byte array value of password(UTF-8)

**bytesToHex**

The bytesToHex method will convert the byte array values to hex.

* + - 1. Convert the byte value to a readable/printable format.
      2. Convert the results to lowercase so it can be passed to the API database

## MD5Database – Class

This application will utilize an API that grants access to a online rainbow table of MD5 Hash values. Using this approach will save the user from having to download the entire database on their computer. Calling the API will produce a web page that has the result of the hash search. The MD5Database class will then translate the page source material and print the results to the screen.

### Methods

**MD5Database**

The MD5Database method is the primary constructor for the MD5Database class. Its job is to query the web-accessible rainbow table in order to find the password for a given hash. If the password is found, it will be returned in plaintext to the user. MD5db.net has provided the hosting of the rainbow table as well as an API to query the rainbow table being hosted.

1. Constructor to pull the hash value and check it against the API.
2. BufferedReader and InputStreamReader will gather the source data from the URL and print to the GUI application.
3. If no result is found, the application will inform the user

**j. Timing – Class**

The Timing class extends the Thread class and is used for benchmarking the time efficiency of the BruteCrack algorithm and will not be seen by the end-user. When implementing the Timing class, the modifications to the PassBreak class are to be uncommented, and a Timing class object is used to measure the time spent running the EAE application code while running the BruteCrack algorithm, the time the OS spent executing processes on behalf of the EAE application, and the sum of the time spent running the EAE application and OS processes used by the EAE.

### Methods

**setThreadCpuTimeEnabled**

Enables or disables thread CPU timeDifference measurement.

1. Inherits the timeDifference method from the Thread class.

**Timing**

Constructor method that generates a polling thread to test at defined timer interval. The time interval is set in the PassBreak method when defining input for the threadTiming object. Convert the byte value to a readable/printable format.

* + - 1. Makes a “super()” call to Thread class.
      2. Instantiates pollingTimeInterval.

**run**

Loops thread until interrupt flag received.

**hashTableUpdate**

Updates hash table values that represent the thread times. byte value to a readable/printable format.

* + - 1. Utilizes if-else statement to update thread id, CPU and user start and stop times.

**getFinalUserTime**

Represents the timeDifference spent running the EAE application code

1. Returns the CPU timeDifference that the current thread has executed in user mode in double precision nanoseconds.

**getFinalCpuTime**

Represents the timeDifference the OS spent executing processes on behalf of the EAE application.

1. Returns the CPU timeDifference that the current thread has executed in user mode in double precision nanoseconds.

**getFinalSystemTime**

Represents the sum of the timeDifference the OS spent executing processes on behalf of the EAE application and the timeDifference spent running the EAE application code.

1. Returns time sum in double precision.

## AnalogClock – Class

This application will utilize an analog clock display to provide the user a visual aid in the analysis of their password. The clock is drawn using GUI programming and is blank prior to and throughout the execution of the program. Upon completion the clock updates with the time taken and is colored with s password strength indicator. (I.e. red = weak)

### Methods

**AnalogClock**

Default constructor

1. Calls updateClock method

**setClcok**

Sets clock variables

1. Set this clock’s seconds variable = 0
2. Set this clock’s minutes variable = 0

**getSecond**

Getter method used to return number of seconds

1. Return seconds

**startClock**

This method starts the timer

1. Call start method from Timer class
2. Set “stopped” variable to false to deactivate display

**updateClock**

This method is used to update the display of the clock upon program completion

1. Call repaint method
2. If seconds is >= 60 increment minutes by 1 & reset seconds to 0

**stopClock**

This method stops the timer and assigns the timing variables

1. Call stop methos from Timer Class
2. If seconds >= 60 round down to last minute
3. Set stopped variable = true to erase clock hands
4. Call repain method to update clock display using current timing variables

**paintComponent**

This method refreshed the clock hand positions upon program completion

1. Use super class paint component
2. Set variables for new clock position
3. If clockImage is null create clock image
4. Draw clock hands using updated clockImage variable
5. Call drawClockHands method to complete drawing

**drawClockHands**

This is the method used to draw the updated clock with the timing varibales set post execution

1. Set varibales
2. Enter for loop based on number of minutes
3. If minutes <= 30 draw ‘Weak’ color
4. Else draw ‘strong’ color
5. Call drawArc method to draw clock hands
6. Call setColor draw on clock face
7. For loop to iterate through 60 seconds
8. Add tick at each second
9. If statement to add a large tick mark at 5 sec intervals
10. If statement to draw second hand if stopped

**drawClockFace**

This method draws the clock

1. Set color to black
2. Draw oval
3. Fill oval

**drawRadius**

This method is used to draw the clock hands

1. Use math sine to calculate angle
2. Use cosine to determine second angle
3. Determine minimum and maximum radius variable

**drawArc**

This method is used to highlight the clocks color

Call fillArc method

# Timing Report

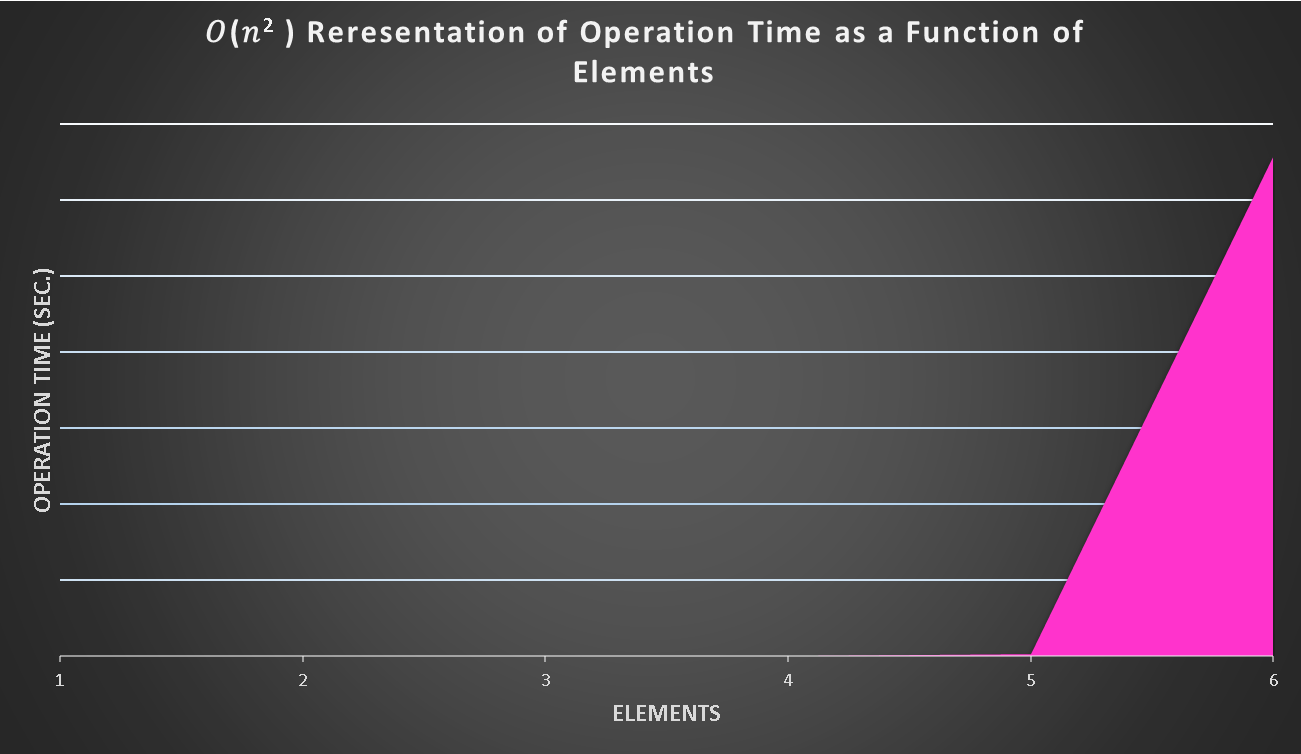
## General Analysis

The *BruteCrack* class serves as the primary mechanism around which the Encryption Analysis Engine (EAE) functions. Analyzing the computational complexity of an EAE instance, reveals how the resource requirements of the *BruteCrack* class scale, under worst-case complexity, as the size of the integer passed to it increases.

*BruteCrack* consists of five methods, *BruteCrack()*, *hasNext()*, next(), *produceString(),* and *adjustIndexes()*, that contribute to the overall operational complexity of the algorithm for any given problem size. Consider passing a password of length to the *BruteCrack()* algorithm. Given the modular design of the BruteCrack algorithm, the complexity of each method for some problem size, , can be expressed as either a constant-time method of order , , a linear-time method of order , , or a quadratic-time method of order n squared, . The following summarizes the operational complexity of the five *BruteCrack()* methods.

* The *BruteCrack()* constructor allocates the initial array by setting the value of length. This operation runs in constant time of
* The *hasNext()* method consists of an *if*-*statement* and a single *for-loop* conducting a linear search on a sorted array. This operation runs in linear time and can be expressed as .
* The *produceString()* method consist of a single for-loop with a nested call to the *append()* method of the S*tringBuilder* class and a call the *toString()* method of the *Object* class. Since the for-loop statement contains a method call, the complexity of the append() method is also taken into consideration. This operation runs in quadratic time and can be expressed as .
* The adjustIndexes() method consists of a linear search on a sorted array and a single for-loop. This operation runs in linear time, and can be expressed as .
* The *next()* method consists of a call to the produceString() method and a call to the adjustIndexes() method. Given that there are two calls to separate methods, the operational complexity of the called methods is inherited. This operation runs in quadratic time, and can be expressed as .

Therefore, the total operational complexity of the BruteCrack algorithm can be expressed as . Figure 10 illustrates how the operation time is a function dependent upon the number of elements.



*Figure 24. Representation of Operation Time as a Function of Elements.*

## Timing Algorithm Tests

* + 1. Six Characters with the Highest Frequency

The six characters with the highest frequency are entered into the EAE, begging with a single character with an additional character added with each test run.

|  |  |  |  |
| --- | --- | --- | --- |
| **Six Highest Frequency Characters** | | | |
| **Input** | **EAE Time (sec.)** | **OS Time (sec.)** | **EAE+OS Time (sec.)** |
| a | 0.46875 | 0.078125 | 0.546875 |
| ae | 0.453125 | 0.0625 | 0.515625 |
| aeo | 0.453125 | 0.09375 | 0.546875 |
| aeor | 0.46875 | 0.140625 | 0.609375 |
| aeori | 3.109375 | 0.140625 | 3.25 |
| aeoris | 221.859375 | 0.390625 | 222.25 |

*Figure 25. Six Highest Frequency Characters*

* + 1. Six Characters with the Lowest Frequency

The six characters with the highest frequency are entered into the EAE, begging with a single character with an additional character added with each test run

|  |  |  |  |
| --- | --- | --- | --- |
| **Six Lowest Frequency Characters** | | | |
| **Input** | **EAE Time (sec.)** | **OS Time (sec.)** | **EAE+OS Time (sec.)** |
| X | 0.375 | 0.171875 | 0.546875 |
| XQ | 0.46875 | 0.078125 | 0.546875 |
| XQZ | 0.484375 | 0.15625 | 0.640625 |
| XQZV | 1.125 | 0.140625 | 1.265625 |
| XQZVY | 34.609375 | 0.171875 | 34.78125 |
| XQZVYW | 2379.15625 | 0.90625 | 2380.0625 |

*Figure 26. Six Lowest Frequency Characters*

## Timing Conclusion

Both the highest frequency and lowest frequency tests show that as the number of elements increase, the operation time grows at a quadratic rate, or, rather, the *BruteCrack* algorithm needs an amount of time that is quadratic proportional to the number of elements passed to it. In terms of password strength, the results indicate that a password that a password consisting of seven low frequency characters will take approximately 26 hours to break, a password of eight low frequency characters will take approximately 28 days to break, and a password of nine low frequency characters will take approximately 821 days to break.

# Development History

## Update 0.20

* Created the PassBreak class (Main Class) and BruteCrack class (Brute force algorithm).
* Initial design will use CLI and will cycle through the password chosen by the user using switch/case statements in the BruteCrack class.
* Structure of classes will be designed after the following UML diagram:

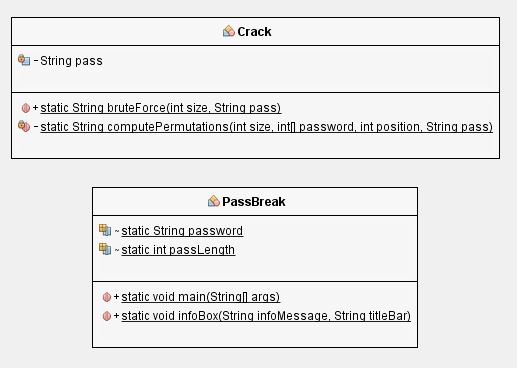


Figure 27. Crack and PassBreak UML Diagram

### PassBreak – Class

PassBreak is the main class in the EAE program. The PassBreak class main function during phase 1 is to pass data to BruteCrack class object. A temporary timing feature will surround the statement passing the password variables to the BruteCrack class. After the value is returned, the program will inform the user of the time it took to crack the password and terminate.

### PassBreak – Pseudo Code Structure

Import Scanner utilities

Import Swing Components

Main Class PassBreak

Static string password = null

Static int passLength

Main method

InfoBox creation (Welcome message)

pass = new Scanner

Print line (Enter password)

Password = input

passLength = password.length

BruteCrack object creation

Time starts

BruteCrack (passlength , password)

Timer ends

Time duration = start – end

Duration < 0

System output (password cracking in duration/1000000 (milliseconds)

Duration >0

System output (password cracking in duration/1000000000 (seconds)

Static void infoBox

JOptionPane ( string, string )

end

### BruteCrack – Class

The BruteCrack class implements the brute force password cracker capability of the program. It defines a character array that includes all characters to be tried during a brute force attempt. Case/Switch statements will be used to quickly assign and compare each character to the original password. To increase efficiency of the algorithm, the search pattern of the case/switch statements will use the overall letter frequency most commonly used in English language. Below you will find the percentages of use:

Overall Character Frequency Analysis (letter/probability):

a       7.52766

e       7.0925

o       5.17

r       4.96032

i       4.69732

s      4.61079

n     4.56899

1      4.35053

t       3.87388

l       3.77728

2       3.12312

m      2.99913

d       2.76401

0       2.74381

c       2.57276

p       2.45578

3       2.43339

h       2.41319

b       2.29145

u       2.10191

k       1.96828

4       1.94265

5       1.88577

g       1.85331

9       1.79558

6       1.75647

8       1.66225

7       1.621

y       1.52483

f       1.2476

w     1.24492

j       0.836677

v       0.833626

z       0.632558

x       0.573305

q       0.346119

A       0.130466

S       0.108132

E       0.0970865

R       0.08476

B       0.0806715

T       0.0801223

M       0.0782306

L       0.0775594

N       0.0748134

P       0.073715

O       0.0729217

I       0.070908

D       0.0698096

C       0.0660872

H       0.0544319

G       0.0497332

K       0.0460719

F       0.0417393

J       0.0363083

U       0.0350268

W       0.0320367

Y       0.0255073

V       0.0235546

Z       0.0170252

Q       0.0147064

X       0.0142182

### BruteCrack – Pseudo Code Structure

class BruteCrack

static string bruteForce (int size , string pass)

system output (wait for pass to be cracked)

int [] password = int

return computePermutations(size, password, 0, pass)

static string computePermutations(int, [], int, string)

string testString = “”

string assemble = “”

for (int i = 0; i<36; i++)

password [position] = i

if pass does not equal string pass to else

else

array position = size

switch (password [position] + 1)

case 1-36

assemble = assemble + “A – Z” and “0 – 9”

return assemble if array matches password

end

### Developer Comments

After initial testing, the average time to find the password “qqqqqq” was 3:40. This would be the longest it would take to crack any variation of a password using the last character in the overall character frequency analysis. Finding the password “aaaaaa” would on average be found in less than a second. The max time is not in line with our initial analysis of how fast it needs to crack a password. The algorithm will need to be redesigned with a more efficient search method.

## Update 0.30

* The BruteCrack class received a massive redesign.
* Implemented the Iterator interface. the algorithm will now iterate through each element assigned to a list.
* The UML diagram has been updated to the following:

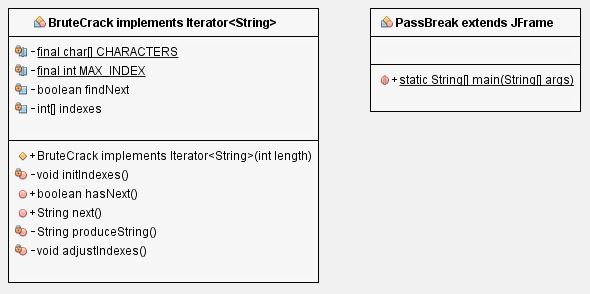


Figure 28. Update UML Diagram for BruteCrack and PassBreak

### BruteCrack – Class

By implementing the iterator, the program will now be able to cycle through a collection and obtain or remove elements. Iterator interface will allow access to each element in the collection, one element at a time. To start the iteration cycle, we will first have to obtain iterator to start the collection. Then start a loop and override the method hasNext() and continue as long as hasNext() returns true. Lastly obtain each element in the loop by calling next(). The new class will define an array of character. Have an array of indexes copy it, and then iterate through it to match the password entered by the user.

### BruteCrack – Pseudo Code Structure

Import iterator utility

Class BruteCrack implements Iterator <String>

Private final char [] CHARACTERS = “a-z, 0-9”

Private static final int MAX\_INDEX

Boolean findNext = true

Private final int [] indexes

BruteCrack(final length)

Indexes = new [length]

Boolean hasNext()

If there is no value next return false

Else return true

String next()

If a value does not exist return null

String next = produceString()

adjustIndexes()

return next

String produceString()

Stringbuilder sb = new stringbuilder

For the length of the index append character found to stringbuilder object

Return sb.toString

Void adjustIndexes()

Int i

For indexes length match MAX\_INDEX and break from loop

end

### Developer Comments

Testing the new algorithm with the iteration interface shows a large improvement over that last algorithm. Tests showed an average increase 96% when cracking the password. Since capital letters weren’t introduced in the first search pattern, this would often hang the program up if the tester chose to add a capitol letter to the password. The statement .toLowerCase() was added to prevent this from happening.

## Update 0.40

* Refined the GUI welcome message to now display a terms of use message.
* Stylized the GUI component with a JTattoo Jar package. Added
* Added capital letters to the search pattern

### Developer Comments

The new terms of use message will inform the users of the intended use of the program. Instead of using the standard swing components to style a GUI, we decide that a JTattoo package will give a cleaner look to the program and increase readability. With addition of capital letters being added to the search pattern, we noticed a small dip in the search efficiency of the program. It now must iterate through more elements to find the letters needed to complete the cycle. If the user enters the password “XXXXXX”, the program will run through ever version of a password available. Searching through 56 billion passwords averaged out to 43-minute crack time.

## MD5 Update 0.30

* Created separate program to turn user’s password into a MD5 hash equivalent
* Run the password through an online rainbow table api to search for the same hash value
* Inform user that the password exists

### MD5 – Class

The MD5 class will convert the password chosen by the user into a hash string variable. To achieve this, the program will import two classes: MessageDigest and DatatypeConverter. The MessageDigest class provides our application the functionality of the message digest algorithm, such as MD5. Message digests are a secure one-way hash function that takes arbitrary-sized data and outputs a fixed-length hash value. Next, the program will use the DatatypeConverter class to convert the byte data to a readable/printable hash string.

### MD5Database – Class

This application will utilize an API that grants access to a online rainbow table of MD5 Hash values. Using this approach will save the user from having to download the entire database on their computer. Calling the API will produce a web page that has the result of the hash search. The MD5Database class will then translate the page source material and print the results to the screen.

### Developer Comments

We currently have a working model of the MD5 hash program that will search through an online api database. Before we make modification to the EAE application, we wanted a separate working program. This will allow us to test and catch bugs faster without the complication of other classes. Originally, we wanted to create our own rainbow table database but discovered that it would take up large amounts of storage on the user’s computer. Due to the SSL security check that the api uses, the users of the program must update their java platform to a version higher than 8u1. If not, they will receive a handshake issue when the program tries to check the hash against the api. Due to time constraints, integrate the MD5 program in to the EAE application will move to phase 2.

## Update 0.50

* Added Timing class for benchmarking
  + 1. **Timing – Class**

The Timing class is not only used for benchmarking the time efficiency of the BruteCrack algorithm, but also adaptable to measure the timing of the entire Encryption Analysis Engine (EAE) application. For measuring the efficiency of the BruteCrack algorithm, the test PassBreak class has been modified to instantiate Timing class objects that are used to measure the time spent running the EAE application code while running the BruteCrack algorithm, the time the OS spent executing processes on behalf of the EAE application, and the sum of the time spent running the EAE application and OS processes used by the EAE.

* + 1. **Timing – Pseudo Code**

Pseudocode Structure For Timing Class:

Import required classes:

Public class Timing extends Thread {

private class TimerType {

declare variables for staring CPU Time;

declare variables for starting User Time;

declare variables for stopping CPU Time;

declare variables for stopping User TIme;

}

Declare variables for polling Time

Enable/Disable parameters for thread CPU time {

Create a method for the CPU Time

}

public Timing {

create Polling thread to check timer interval

get the thread’s ID

}

Create an exception for the thread being interrupted {

Loop through the thread until interrupted {

While the thread is not interrupted {

Update the hash table {

}

Try interrupting the hash table {

}

Catch the interrupted exception

}

Constructor to update hash tables {

Declare Thread variables {

If the id does not equal the thread id {

Continue to update the table.

Get the CPU Thread Time

Get the user Thread Time

}

If the CPU time equals -1 or the user thread equals -1

Continue;

Use the thread history to get the times for each TimerType;

If the times are not null{

Create a Timer Type object

Set the id equal to id

Set the Start CPU Time equal to CPU

Set the Start User Time equal to user

Set the Stop CPU Time equal to CPU

Set the Stop User Time equal to user

Put the id and the times into the thread history

} else {

Set the CPU Stop Time equal to CPU;

Set the User Stop Time equal to user;

}

Get the User’s Final Time(){

Create a timer history object

Create a time difference variable

Time difference equals the user Stop Time mimus The user Start Time

Return the Time Difference

}

* + 1. **Developer Comments**

The Timing class is only used as a developer tool to better understand how resources are used by the Encryption Analysis Engine (EAE). For the current version of the EAE, the Timing class is only used to benchmark the BruteCrack algorithm. The times returned are in double precision to give a more accurate measurement of the BruteCrack runtime efficiency.

**f.** **Update 0.60**

* Added AnalogClock class to visually depict password strength as time-to-crack

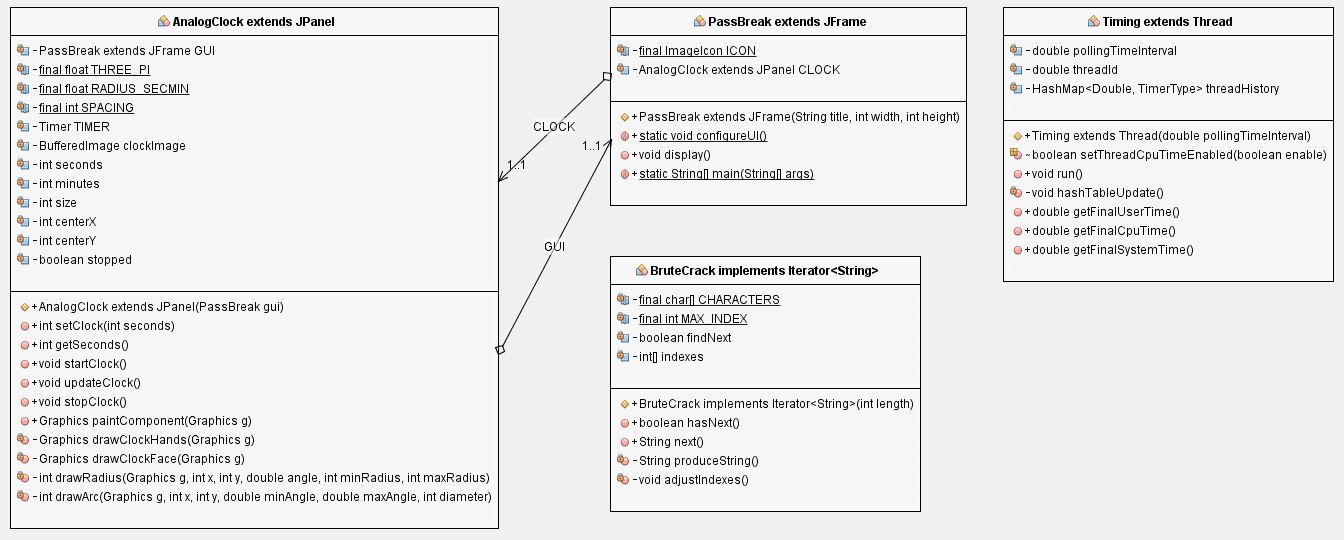


Figure 29. UML Diagram for Phase II Implementation

**i.** **AnalogClock – Class**

The AnalogClock class is used for graphically illustrating the time efficiency of the BruteCrack algorithm, charting duration of the analysis on a clock face as a progressive shading gradient from bright red (weakest) to bright green (strongest). Several methods associated with the BufferedImage and Graphics classes have been modified or overridden to offer dynamic recoloring of the clock face, and to ensure that the final display matches the overall time-to-crack.

**ii.** **AnalogClock – Pseudo Code**

Pseudocode Structure for AnalogClock Class:

Define the AnalogClock class as follows:

public class AnalogClock extends JPanel {

Declare constants to hold PassBreak and Timer objects

Declare static constants for mathematical values

Declare variables for BufferedImage, seconds, minutes, image size, horizontal/vertical axis, and trigger

Define the class constructor:

public AnalogClock(PassBreak gui) {

Initialize PassBreak using the supplied parameter gui

Initialize Timer to increment and call the updateClock() method once per second

}

Define the clock setter method:

public void setClock(int seconds) {

Set class variable seconds equal to supplied parameter

Set class variable minutes equal to seconds divided by 60

}

Define the clock getter method:

public int getSeconds() {

Return the current value of class variable seconds

}

Define the Timer start method:

public void startClock() {

Call the start() method associated with the Timer

Set class variable stopped equal to false

}

Define the Timer update method:

public void updateClock() {

Call the repaint() method

Increment class variable seconds if stopped is set to false

Increment class variable minutes if seconds is divisible by and greater than or equal to 60

}

Define the Timer stop method:

public void stopClock() {

Call the stop() method associated with Timer

Round down seconds to nearest factor of 60 if more than one minute has elapsed

Set class variable stopped to true in order to erase clock hands

Call the repaint() method

}

Define the overridden paint method:

public void paintComponent(){

Call super class paintComponent() method with g as parameter

Set size to 150 – (2 \* spacing)

Set centerX to size / 2 plus spacing

Set centerY to size / 2 plus spacing

Call drawImage() method to draw clock face from the precomputed image

Call drawClockHands() if program is still running

}

Define the drawClockHands method:

public void drawClockHands(){

Set secondRadius to size / 2

Set fseconds to seconds

Set secondAngle to THREE\_PI – (RADIUS\_SECMIN \* fseconds)

Iterate through each minute on the clock face to shade from red to green over 30 min

Add the black tic marks to the clock face on top of the color

}

Define the drawClockFace method:

public void drawClockFace(Graphics g){

Call setColor() method and set to Color.BLACK

Call fillOval() method and fill with Color.BLACK

Call setColor() method and set to Color.LIGHT\_GRAY

Call fillOval() method and fill with Color.LIGHT\_GRAY

}

Define the drawRadius method to draw the clock hands:

public void drawRadius(Graphics g){

Set sine to Math.sin(angle)

Set cosine to Math.cos(angle)

Set dxmin to minradius \* sine

Set dymin to minradius \* cosine

Set dxmax to maxradius \* sine

Set dymax to maxradius \* cosine

Call drawLine() method with defined constraints

}

Define the drawArc method for clock highlighting:

public void drawArc(Graphics g){

Call fillArc() method with passed diameters and anglesx

}

}

**iii.** **Developer Comments**

The AnalogClock class started out as a dynamically updated clock, but we quickly realized that it added significantly more processing time needed to brute force crack the supplied password. Additionally, we had concerns that the length of 1 second was variable due to time drift and the number of clock cycles needed to keep accurate time. To reduce the resources needed, we update the clock once after the password has been successfully brute force cracked.

1. **Updates 1.0-1.6**

* Added StrengthChart class to graphically depict password strength as a color spectrum
* Added InterfaceConsole class to provide GUI buttons and define actions-on-click
* Added InterfaceReader class to provide GUI input and output text fields
* Added InterfaceOption class to provide GUI drop-down menu selection
* Added InterfacePanel class to define appearance and placement of GUI elements
* Added MD5Database and HashGenerator classes to provide MD5 hash collision function
* Modified AnalogClock class constructor to read to InterfaceConsole instead of PassBreak
* Modified PassBreak class to construct GUI using InterfacePanel rather than running CLI
* Modified InterfaceConsole to include processing delay alert prior to running BruteCrack
* Modified InterfaceConsole to run BruteCrack as a background process via SwingWorker

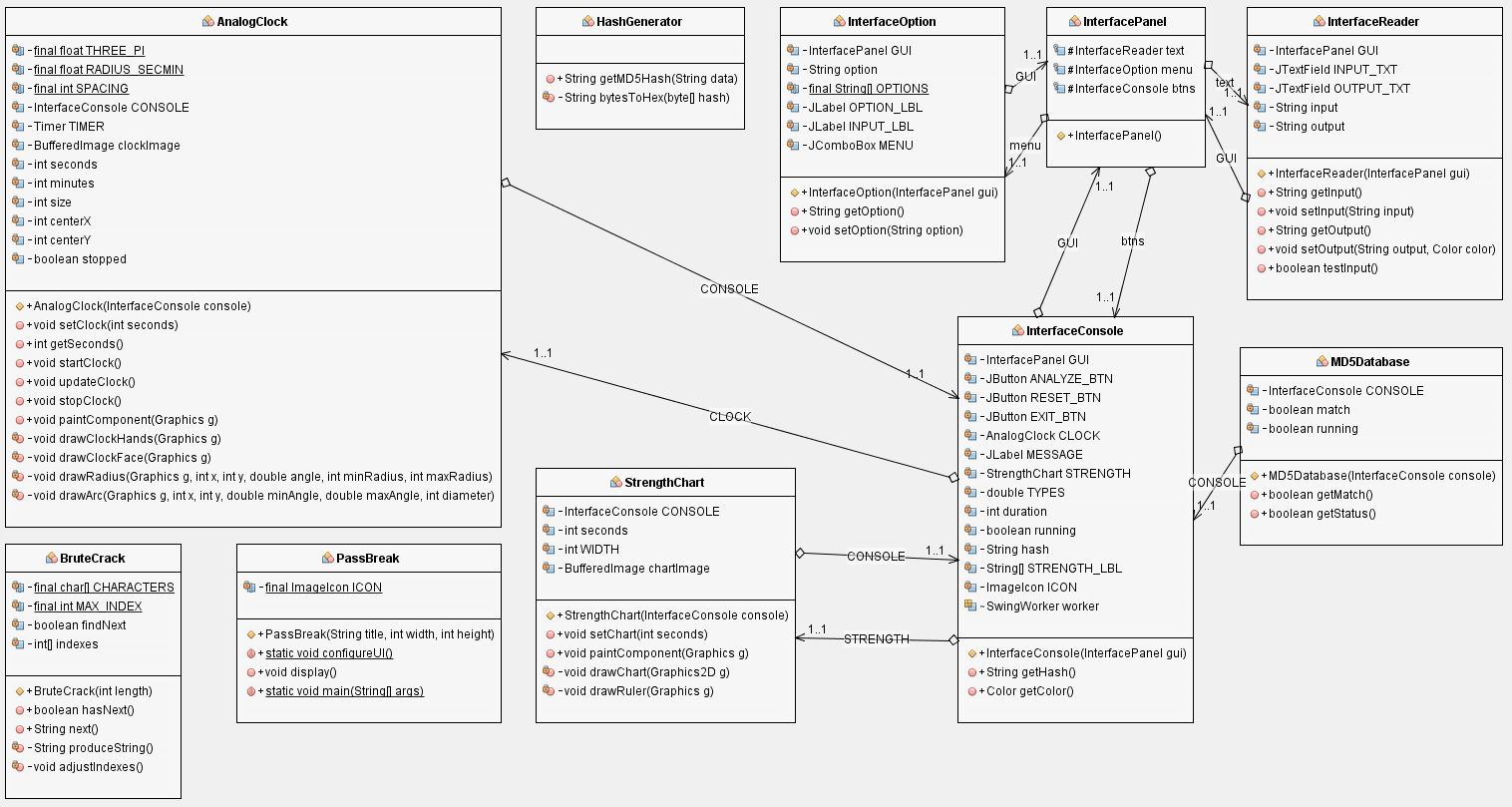
****

Figure 30. UML Diagram for Phase III Implementation

* + 1. **StrengthChart – Class**

The StrengthChart class creates and defines the appearance of a bar graph plotted vertically against a ruler to aid in the visual analysis of password strength. The bar graph is drawn using Java Graphics and is blank prior to and throughout the execution of the program. Upon completion the chart dynamically scales to a height corresponding to the ruler measurement of strength, and is colored with the password strength indicator (i.e. red = weak).

Pseudocode Structure for StrengthChart Class:

Import required classes

Public class StrengthChart extends JPanel{

Create InterfaceConsole;

Set width of chart bar;

Create bar graph image;

Create chart constructor {

}

Create chart setter method {

}

Override paintComponent method {

Create Graphics2D object;

Turn on Anti-Aliasing to render image as quickly as possible;

If bar graph image is null {

Create new bar graph image;

Turn on Anti-Aliasing to render image as quickly as possible;

Call drawChart () method;

}

Draw the chart from the precomputed image;

Call drawChart () method;

}

Create drawChart method {

Set background color;

Fill bar graph rectangle with color;

Create start and ending colors to highlight in the bar graph;

If seconds <= 30 minutes {

Create new color shade based on number of seconds elapsed;

If seconds is between 0-15 {

Set height to 1;

}

Else {

Set height to seconds divided by 15;

}

}

Else (greater than 30 minutes) {

Set color shade to 0, 255, 0 (green);

Set maximum bar graph height;

}

Set bar graph paint color to gradient paint from red to green;

Fill bar graph;

Set background color to black;

Call drawRuler method;

}

Create drawRuler method {

Fill rectangle;

For each i 0 to 12{

If i mod 2 = 0 {

Draw a longer line;

}

Else i mod 2 =/= 0 {

Draw a shorter line;

}

}

}

* + 1. **InterfaceConsole – Class**

The InterfaceConsole class defines and create the appearance and layout of buttons within the GUI. It creates and configure all of the buttons that will be used in the GUI. Additionally, it adds action listeners for the buttons that call the appropriate methods and conduct some basic timing of how long the methods run for. The “analyze” button retrieves the selected drop-down menu option, reads user text input, and performs input validation; if the input passes validation conditions, it prompts the user to acknowledge potential processing delays (in the case of BruteCrack), and then proceeds to perform the selected function (including instantiating a SwingWorker process to run BruteCrack) and display the results as output. The “reset” button restores system default settings and terminates any currently-running BruteCrack process. The “exit” button hides the main application interface, displays a goodbye message, and terminates the application. Finally, InterfaceConsole constructs and displays both the AnalogClock and StrengthChart graphics in response to results obtained from the specified input.

Pseudocode Structure for InterfaceConsole Class:

Import required classes

public class InterfaceConsole extends JPanel {

Create GUI elements

Create InterfacePanel GUI;

Create new JButton named Analyze;

Create new JButton named Reset;

Create new JButton named Exit;

Create new AnalogClock();

Create new JLabel();

Create new int duration;

Create new String;

Create new String array;

Create a new ImageIcon;

Create a hash getter method;

}

Create a color getter method{

}

Create a constructor for the GUI buttons {

Add each button and elements to the GUI

Create ActionListener for Analyze button;

Get the values of the GUI from the console;

If the results are valid {

Clear the previous analysis;

Analyze the results only if the choice is valid;

Create a message box to inform user of possible processing delay;

Create a bar graph to represent user’s password

}

Create action listener for reset button{

Clear existing selections;

)

Create action listener for exit button;

}

* + 1. **InterfaceReader – Class**

The InterfaceReader class defines the appearance of the text input and output fields within the GUI. It also conducts input validation of text by checking for compliance with formatting requirements, and outputs error messages if necessary.

Pseudocode Structure for InterfaceReader Class:

Import required classes

Create GUI elements;

Create GUI constructor (){

Add GUI elements;

}

Create a getter method for text input{

Method;

}

Create a setter method for text input {

Method;

}

Create a setter method for text output{

Method;

}

Check input for formatting isses {

Method;

}

* + 1. **InterfaceOption – Class**

The InterfaceOption class defines and create the appearance of the drop-down menu within the GUI. Its job is be to add the drop-down menu to the InterfacePanel.

Pseudocode Structure for InterfaceOption Class:

Import required classes

public class InterfaceOption extends JPanel {

Create GUI elements;

}

Create a new JLabel

Create a new JLabel

Create a new JComboBox with several options

Create a GUI constructor {

Add dropdown menu elements;

Add an action listener to the combobox;

}

Create a getter method for dropdown menu {

}

Create a setter method for the dropdown menu {

}

* + 1. **InterfacePanel – Class**

The InterfacePanel class will define the placement of individual elements within the GUI. It creates objects for the text, menu, and buttons, and adds them to a panel.

Pseudocode Structure for InterfacePanel Class:

Import required classes

public class InterfacePanel extends JPanel {

Create InterfaceReader text set equal to new InterfaceReader();

Create InterfaceOption menu set equal to new InterfaceOption();

Create InterfaceConsole btns set equal to new InterfaceConsole();Create GUI constructor

InterfacePanel(){

Add GUI parts to the panel

setLayout equal to new BoxLayout();

setBorder equal to new LineBorder;

add the InterfaceOption;

set the InterfaceOption’s preferred size;

add the InterfaceReader;

set the InterfaceReader’s preferred size;

add the InterfaceConsole

}

}

* + 1. **MD5Database – Class**

This application will utilize an API that grants access to a online rainbow table of MD5 Hash values. Using this approach will save the user from having to download the entire database on their computer. Calling the API will produce a web page that has the result of the hash search. The MD5Database class will then translate the page source material and print the results to the screen. Its job is to query the web-accessible rainbow table in order to find the password for a given hash. If the password is found, it will be returned in plaintext to the user. MD5db.net has provided the hosting of the rainbow table as well as an API to query the rainbow table being hosted.

Pseudocode Structure for MD5Database Class:

Import required classes;

Public class MD5Database extends JPanel {

Create InterfaceConsole object;

Create boolean values for match and running and set initial states;

Create getter method for match {

Return match value;

}

Create getter method for running status {

Return running value;

}

Create MD5Database constructor {

Create URL object with md5 database url and hash to check;

Create buffered reader and input stream reader to send hash through;

While loop monitoring for hash return{

Set match = true;

}

}

}

* + 1. **HashGenerator – Class**

The HashGenerator class converts the password chosen by the user into a hash string variable. To achieve this, the program imports two classes: MessageDigest and DatatypeConverter. The MessageDigest class provides our application the functionality of the message digest algorithm, such as MD5. Message digests are a secure one-way hash function that takes arbitrary-sized data and outputs a fixed-length hash value. Next, the program uses the DatatypeConverter class to convert the byte data to a readable/printable hash string.

Pseudocode Structure for HashGenerator Class:

Import required classes;

Public class HashGenerator {

Create getMD5Hash method to build hash {

Try {

Create MessageDigest object of MD5 type;

Create byte array of MD5 hash using MD5 algorithm on supplied password;

Call bytesToHex method and return results;

}

Catch exceptions {

Print the exception;

}

return result;

}

Create bytesToHex method {

Return lowercase hex version of the byte array;

}

}

* + 1. **Developer Comments**

The classes and methods provided this week allow for construction of a responsive, scalable Graphical User Interface that serves as the single point of access to all password analysis functionalities, including brute-force cracking, hash collision, and strong password generation (to be added later). Furthermore, the GUI displays any error messages associated with incorrect environment configuration or failed input validation. The GUI depicts password strength as a function of time-to-crack and plots the results as a bar graph chart with descriptive labelling. Finally, the GUI provides buttons to run, reset/cancel, and terminate the password analysis processes.

1. **Update 1.7**

* Added PasswordGenerator class to transform user input read via “Analyze” button-click action listener into a new password based on including randomly-generated sequences of letters, numbers, and special characters, based on user-defined options made using a series of checkboxes.
* Added PasswordDialog class to construct and display a new window allowing the user to specify modifications by means of the PasswordGenerator methods.
* Modified InterfaceConsole to display the PasswordDialog window and disable all other main application interface controls upon selecting the “Random Password Generator” option from drop-down menu and clicking “Analyze;” added resetButtons() method to re-enable main application interface controls upon exiting the PasswordDialog window.

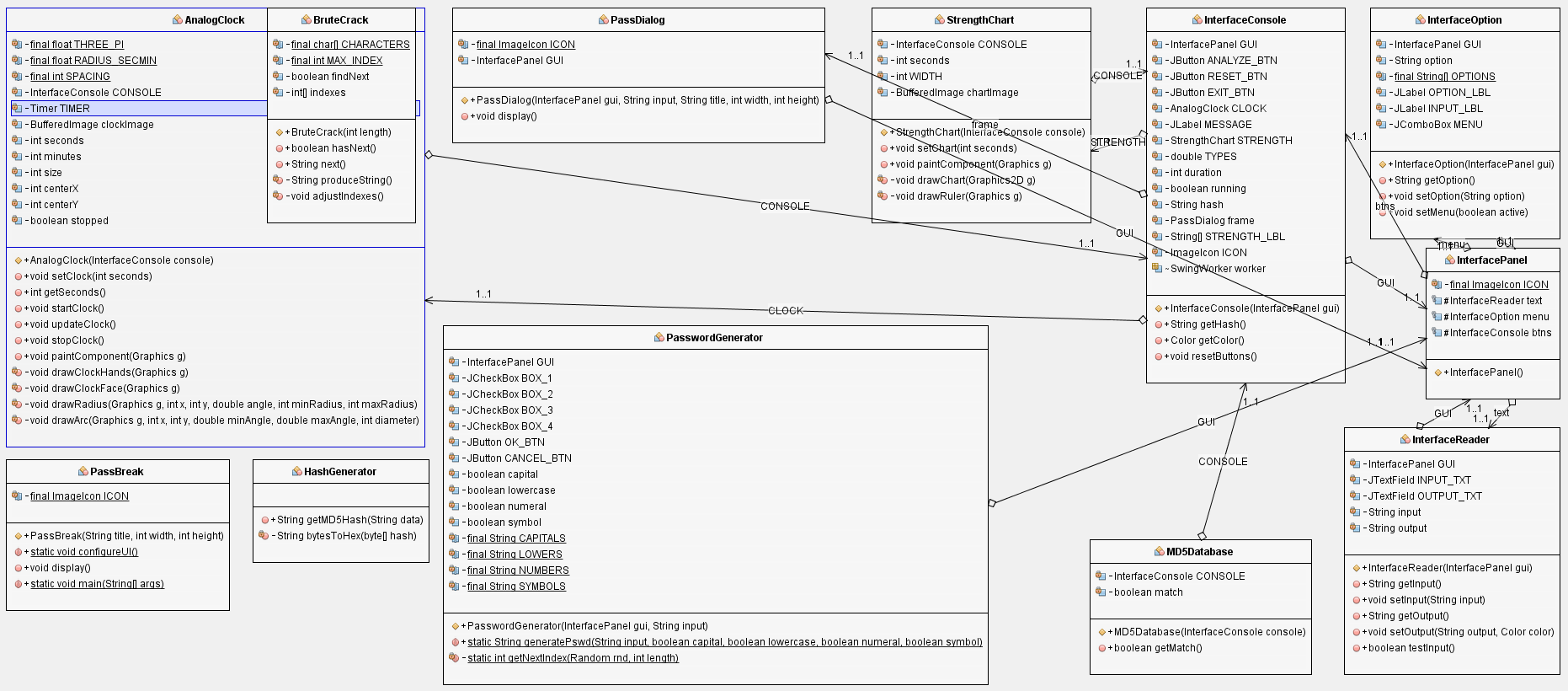


Figure 31. UML Diagram for Phase IV Implementation

* + 1. **PasswordGenerator – Class**

The PasswordGenerator class interfaces directly with the InterfaceReader and InterfaceConsole classes to retrieve user-provided alphanumeric password input and perform modifications based on a series of four checkboxes. These checkboxes allow for the addition of a random number of lowercase, uppercase, numerals, and symbols interspersed with the originally-specified password characters. The resulting is printed to the main application interface output text using the previously-defined setOutput() method. Once the user has finished performing password generation, the “Cancel” button re-enables buttons on the main application interface.

Pseudocode Structure for PasswordGenerator Class:

Import required classes;

Public class PasswordGenerator extends JPanel{

Create GUI InterfacePanel;

Create check boxes for GUI;

Create buttons for GUI;

Create Boolean variables to store selection;

Create PasswordGenerator constructor method {

Define GUI Layout;

Define GUI border;

Add GUI components;

Define action listeners for each check box and button;

}

Create generatePswd method {

Create random object;

For each check box selected, generate a random number between 1 and 3;

Set max to the length of the input plus randomly generated numbers;

Create new character array with max as the size;

For loop for each of the character types put random characters at randomly selected indices;

For loop to fill the remaining length with original characters;

Returns randomly generated password;

}

Create getNextIndex method {

Set index to randomly generated length between 0 and the max length;

Return index;

}

}

* + 1. **PassDialog – Class**

The PassDialog class defines and creates the password generation window, including window close operations and visibility to ensure superimposition on top of the main application interface when active.

Pseudocode Structure for PassDialog Class:

Import required classes;

Public class PassDialog extends JFrame {

Set image icon;

Define GUI InterfacePanel;

Create PassDialog constructor method {

Set size of GUI;

Set Close operation of GUI;

Add PasswordGenerator object to GUI;

Create Window adapter {

Override windowClosing method {

Set visible to false;

Reset GUI buttons;

}

}

Add window adapter as a window listener;

}

Create display method {  
 Set visible to true;

}

}

* + 1. **Developer Comments**

Once the program was completed, the group felt that something was missing. We initially started to formulate ideas about having the program create a password. After initial testing, we felt that the design wasn’t very user intuitive and remembering the password to be quite difficult. To create a better user experience, we implemented a method that allows the user to create a password that would be easier for them to remember. They can select from a predefined selection and the program will output the selected value. We found this route to be easier and offer a richer experience for the user.

The classes and methods provided this week add a random password generation functionality to the Graphical User Interface that also serves as the single point of access to all other analysis functionalities, including brute-force cracking and hash collision. The additional window, which is overlaid on top of the main application interface, provides options to allow the user to create and define a randomly-generated password. It also disables and re-enables main application interface controls upon window open and close operations, respectively.

# Conclusions

The Encryption Analysis Engine works as designed. Users can choose to have their entered password cracked using Brute Force or MD5 methods. Once the user has chosen their desired method and entered their password, it is analyzed after the user clicks the Analyze button. If the user chooses, they can even have a stronger password generated for them using the Password Generator option.

Since the engine works as intended, the developers consider the engine to be a success and may consider implementing additional features such as SHA1 in the future. Some possible future improvements include SHA1, an option for the user to be able to customize the user interface with different color schemes, deployment to Amazon Web Services or other web hosting service. The user interface being able to be customized by the user adds additional value for the user because then EAE would better match the look and feel of the user’s other applications. Deployment to Amazon Web Services was considered by the developers but due to the resource intensive nature of the EAE and needing to pay for most Amazon Web Services, deployment was considered for future development. The program could also feature a SHA1 option for the user. SHA1 is not currently implemented because it is feature that would have to be paid for by the developers.

# References

FRICHET, T. (2017). *MD5DB.net API*. Retrieved from MD5DB.net API: https://md5db.net/api

Identity Theft Resource Center; Cyber Scout. (2017, January 19). *Breaches Increase 40 Percent in 2016*. Retrieved from Identity Theft Resource Center: http://www.idtheftcenter.org/2016databreaches.html

Microsoft. (2008). *Process 2: Write the Function Specification*. Retrieved from https://technet.microsoft.com/en-us/library/cc543349.aspx

PCMag. (2016). *Brute Force Attack*. Retrieved from http://www.pcmag.com/encyclopedia/term/38980/brute-force-attack

W3Schools. (2017). *Java Introduction*. Retrieved from http://www.w3schools.in/java-tutorial/intro