CS2104

Dictionary Attack Program Documentation

Juan Jovel
Sami Marzougui

Program Structure

The program is composed of several files that work together to provide a guided user interface that performs a dictionary attack on a given password, providing information with regards to the time taken by the algorithm to figure out the password and the number of guesses it took before figuring it out.
The program consists of 2 main files:

DictionaryAttacker.py
This Python module uses various Python libraries including: hashlib, itertools, binascii, and pathlib. The module includes a class DictionaryAttacker which has various functions that allow it to perform a dictionary attack on a given password. More documentation regarding the specific functions that each method performs can be found in the file. This class has the job of reading the dictionary from a text file, which has been standardized for this project, optimizing the dictionary based on the length of the password, and applying a brute force dictionary attack on the password using the length of it as a constraint.

GUI.py
This Python module uses the tkinter, matplotlib, and time libraries in addition to the DictionaryAttacker class to provide a visual interface for the user to input a password and apply a dictionary attack to it and get the results and statistics of the attack after it is performed.

The file **consoleDemo.py** was used for debugging and testing purposes.

Assumptions and Limitations

This program assumes the following:

- The password given is composed of one or more words contained in the dictionary, if it isn't the program will run for a very long time until it has tried every possible combination of the size of the password and will then return an empty string.
- The user selected a checkbox specifying the hash type, if they didn't the program defaults to SHA256.

Due to time constraints set by the project, the following parameters were selected for the hashing algorithm:

- 50 iterations were used
- A static salt 'staticSalt' was used in order to make the dictionary attack possible.

Because the time to compute a password grows exponentially, the biggest password we could test out was a password composed of 4 words from the dictionary which was cracked in around 125 seconds. All results presented were produced using Windows 10 Home with the following hardware:

- Intel Core i7-9750H (2.60 GHz)
- 32 GB DDR4 RAM

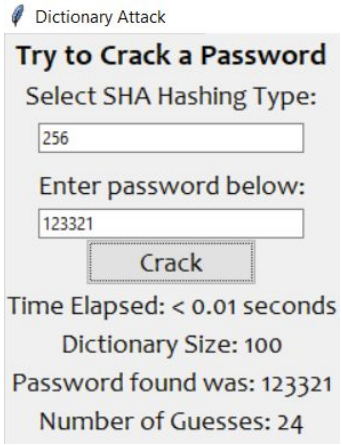
How the Program Works

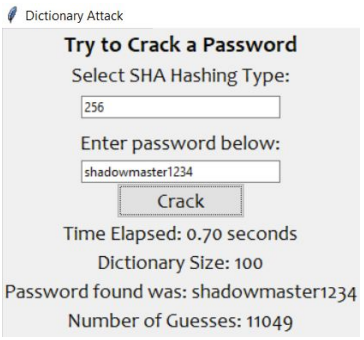
The user inputs a hash type using the checkboxes, the supported choices are 512 and 256, inputs a password into the textbox provided, and clicks the 'Crack' button. The program reads the password from the textbox and takes note of the selected hash type, it instantiates a DictionaryAttacker with the given password and the selected hash type as the parameters. The DictionaryAttacker computes the length of the password and saves it for future reference, the password in string form is never saved, the password is then hashed based on the given parameters and stored. The program runs the readDictionary function which reads the dictionary and saves it into a list form. A timer is started before the attacker begins the process of the dictionary attack. First, the optimizeDictionary method is run which eliminates any impossible words on the list based on the password length saved before. After this, the attack method is run which hashes every word and compares it to the saved hashed version of the password, if the password doesn't match any word then the attackCombination method takes over which tries every combination that matches the length of the password. After the password is figured out the timer is stopped, and the stats of the attack are displayed in the window. After running at least two passwords with the same hashing algorithm, the 'Plot 256' and 'Plot 512' buttons are available and can plot results such as length of password vs. time, length of password vs. number of guesses.

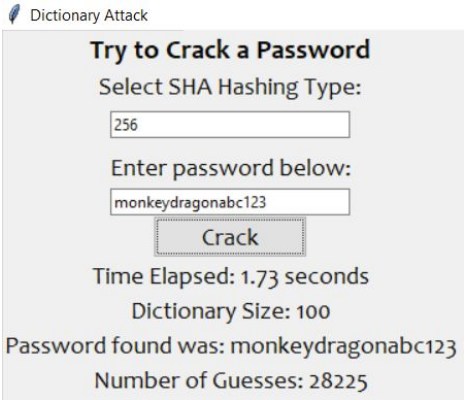
Demo

As required by the assignment, the following is a demonstration of our program using 10 passwords of increasing difficulty using both SHA256 and SHA512.
This is a list of the passwords that were used:

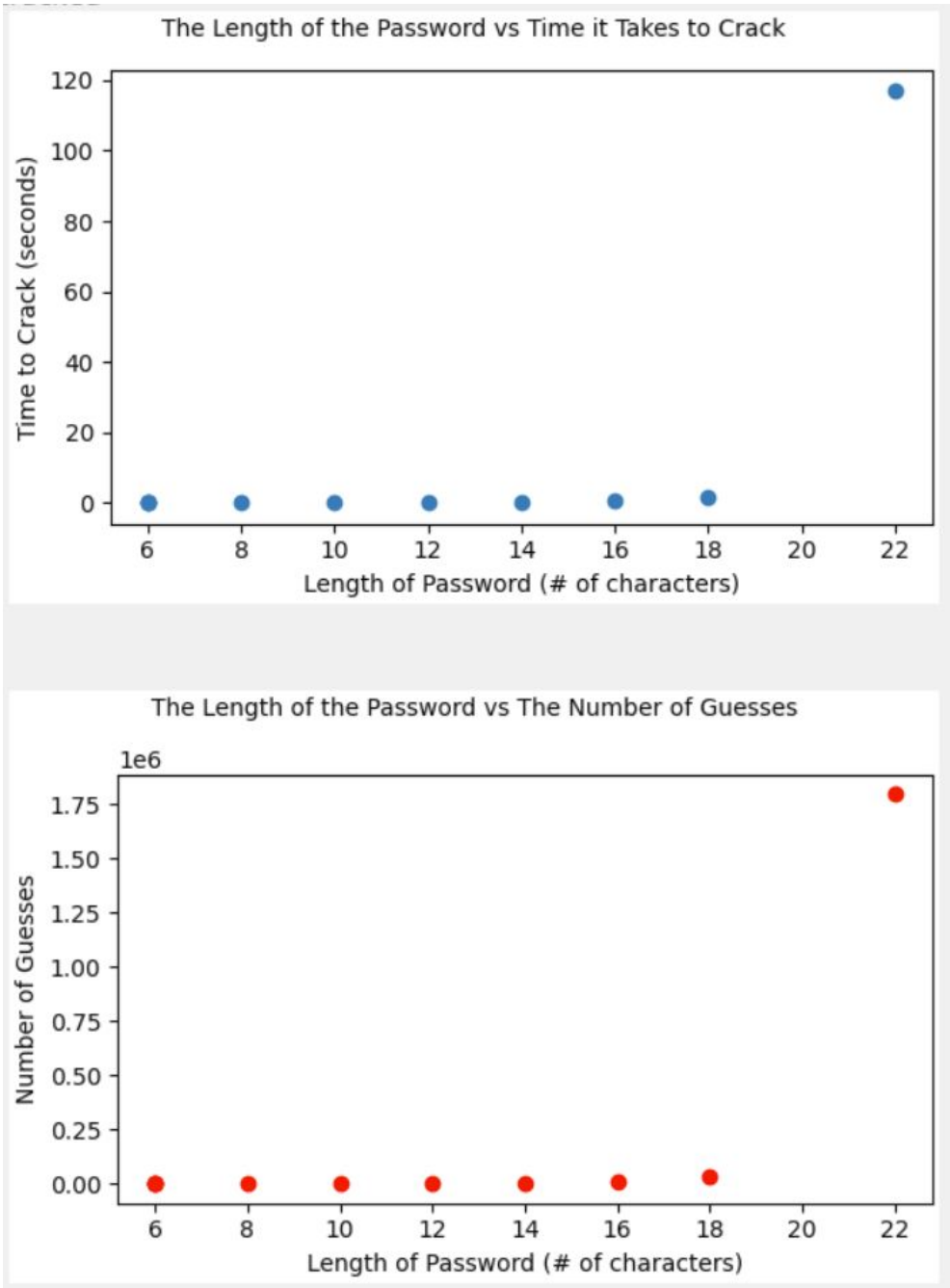
1. ‘123123’
2. ‘123321’
3. ‘abc123’
4. ‘password’
5. ‘qwertyuiop’
6. ‘password1234’
7. ‘hunterbaseball’
8. ‘shadowmaster1234’
9. ‘monkeydragonabc123’
10. ‘jordanfootballpass1234’



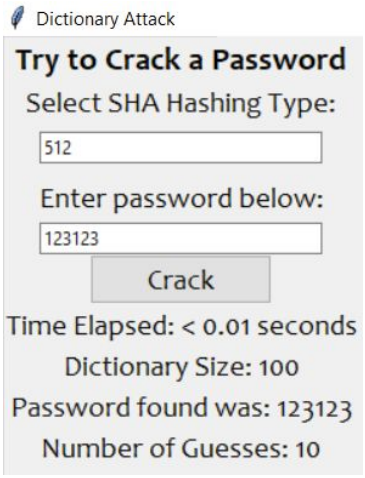


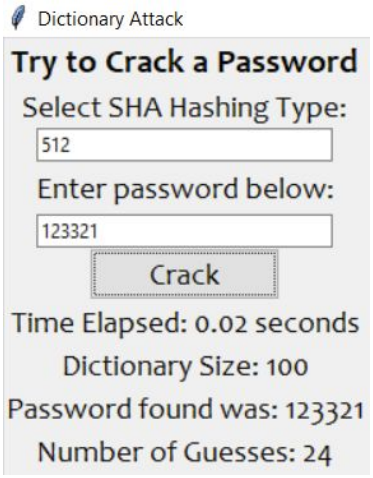


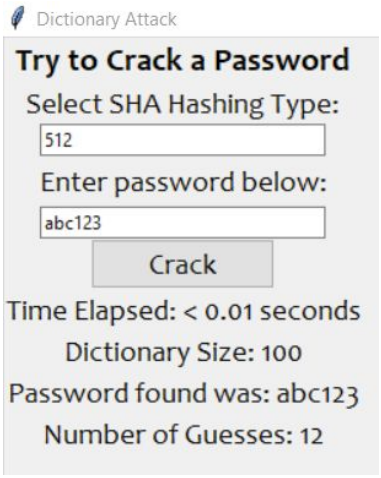
The results can be plotted using the ‘Plot 256’ button:



It is evident that both plots have an exponential nature to them. The longer the password is, the more guesses, and consequently the more time, it takes to crack it.


Dictionary Attack
Try to Crack a Password
Select SHA Hashing Type:
512
Enter password below:
123123
Crack
Time Elapsed: < 0.01 seconds
Dictionary Size: 100
Password found was: 123123
Number of Guesses: 10


Dictionary Attack
Try to Crack a Password
Select SHA Hashing Type:
512
Enter password below:
123321
Crack
Time Elapsed: 0.02 seconds
Dictionary Size: 100
Password found was: 123321
Number of Guesses: 24


Dictionary Attack
Try to Crack a Password
Select SHA Hashing Type:
512
Enter password below:
abc123
Crack
Time Elapsed: < 0.01 seconds
Dictionary Size: 100
Password found was: abc123
Number of Guesses: 12

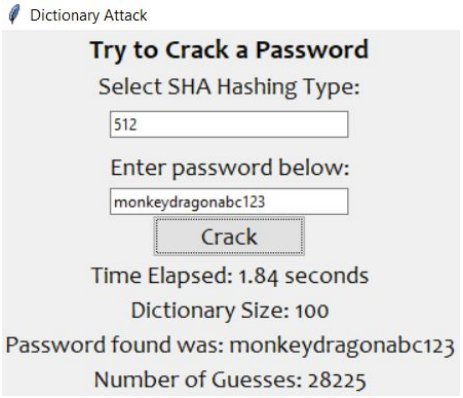

Dictionary Attack
Try to Crack a Password
Select SHA Hashing Type:
512
Enter password below:
password
Crack
Time Elapsed: < 0.01 seconds
Dictionary Size: 100
Password found was: password
Number of Guesses: 2


Dictionary Attack
Try to Crack a Password
Select SHA Hashing Type:
512
Enter password below:
qwertyuiop
Crack
Time Elapsed: < 0.01 seconds
Dictionary Size: 100
Password found was: qwertyuiop
Number of Guesses: 2


Dictionary Attack
Try to Crack a Password
Select SHA Hashing Type:
512
Enter password below:
password1234
Crack
Time Elapsed: 0.02 seconds
Dictionary Size: 100
Password found was: password1234
Number of Guesses: 57

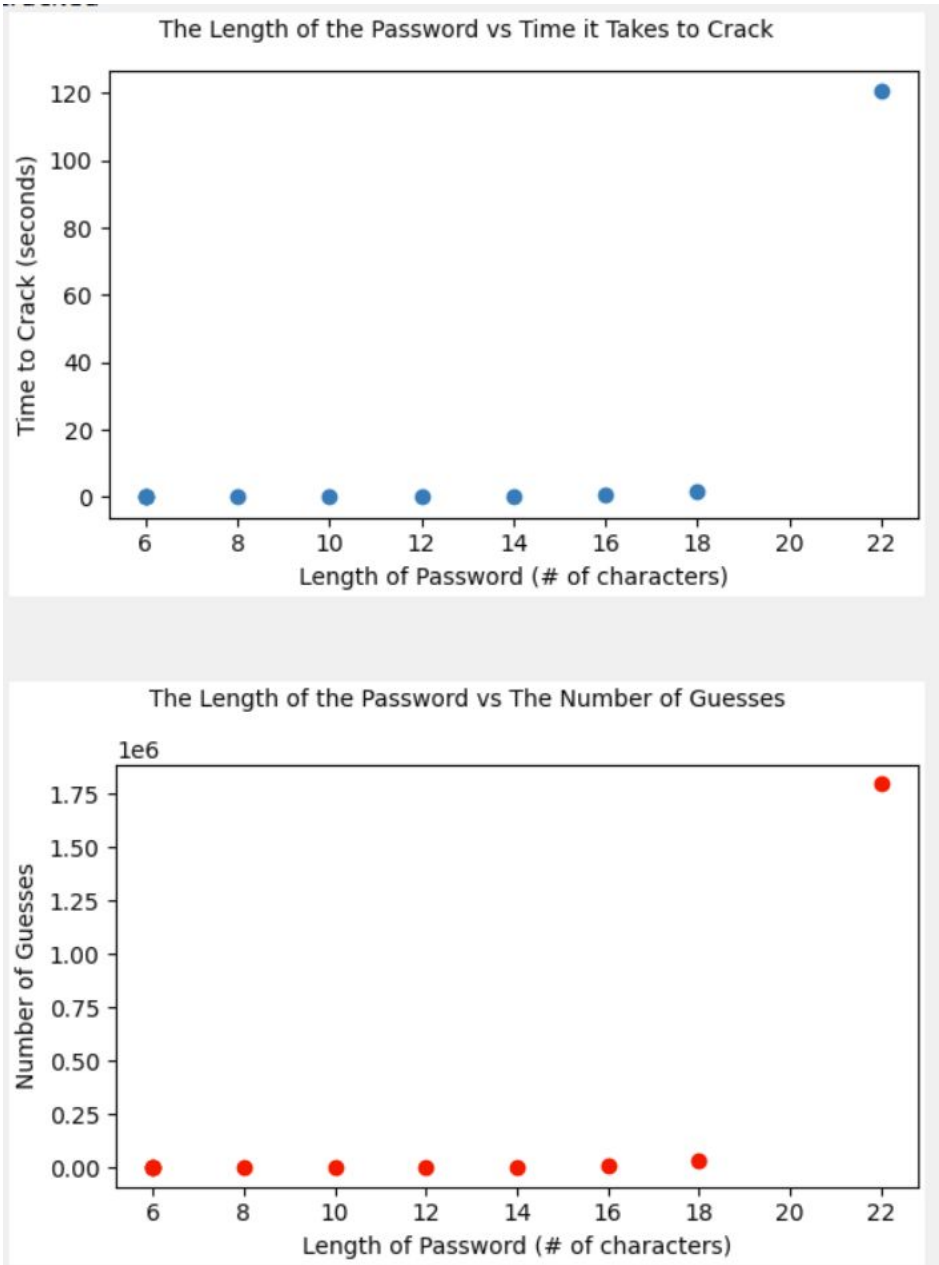

Dictionary Attack
Try to Crack a Password
Select SHA Hashing Type:
512
Enter password below:
hunterbaseball
Crack
Time Elapsed: 0.07 seconds
Dictionary Size: 100
Password found was: hunterbaseball
Number of Guesses: 885


Dictionary Attack
Try to Crack a Password
Select SHA Hashing Type:
512
Enter password below:
shadowmaster1234
Crack
Time Elapsed: 0.73 seconds
Dictionary Size: 100
Password found was: shadowmaster1234
Number of Guesses: 11049


Dictionary Attack
Try to Crack a Password
Select SHA Hashing Type:
512
Enter password below:
monkeydragonabc123
Crack
Time Elapsed: 1.84 seconds
Dictionary Size: 100
Password found was: monkeydragonabc123
Number of Guesses: 28225


Dictionary Attack
Try to Crack a Password
Select SHA Hashing Type:
512
Enter password below:
jordanfootballpass1234
Crack
Time Elapsed: 122.18 seconds
Dictionary Size: 100
Password found was: jordanfootballpass1234
Number of Guesses: 1796775

The results can be plotted using the ‘Plot 512’ button:

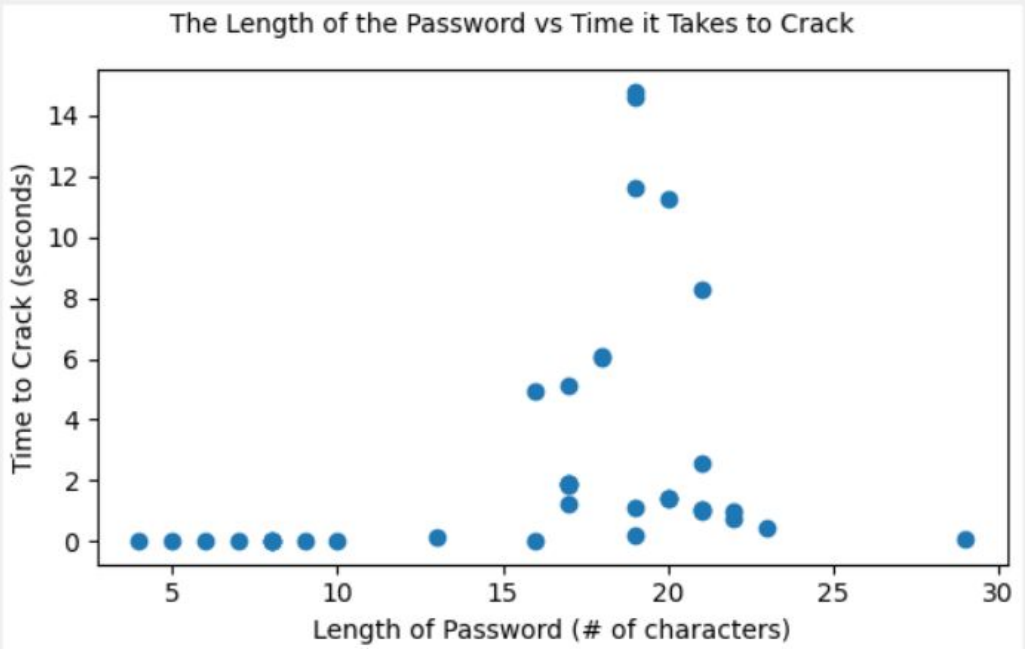


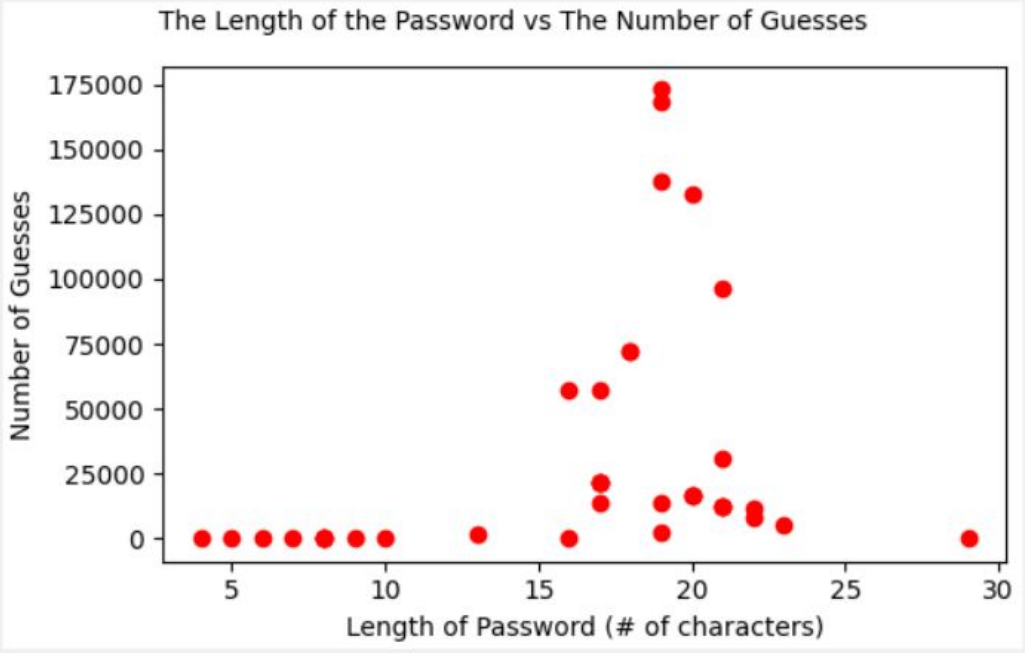
Although the hashing type has changed to SHA512, the behavior of the plot continues to be exponential. Although we do notice that, on average, the passwords take a little longer to be cracked when using the SHA512 as opposed to SHA256.

Plots

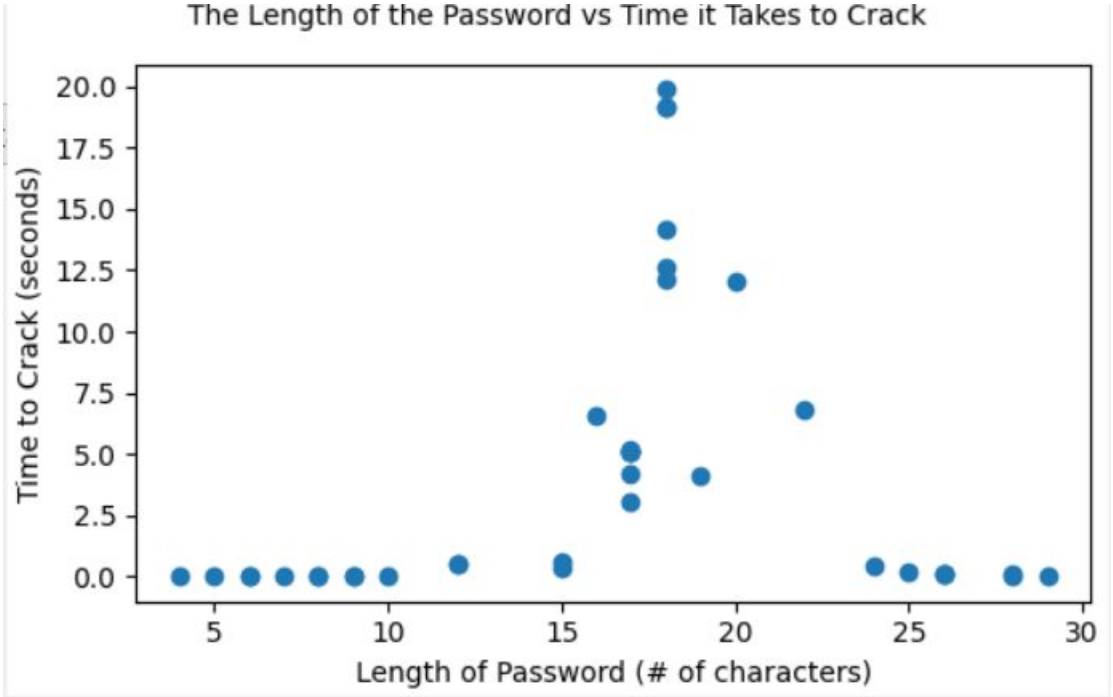
After cracking more passwords, our plot starts to change shape and look like this.

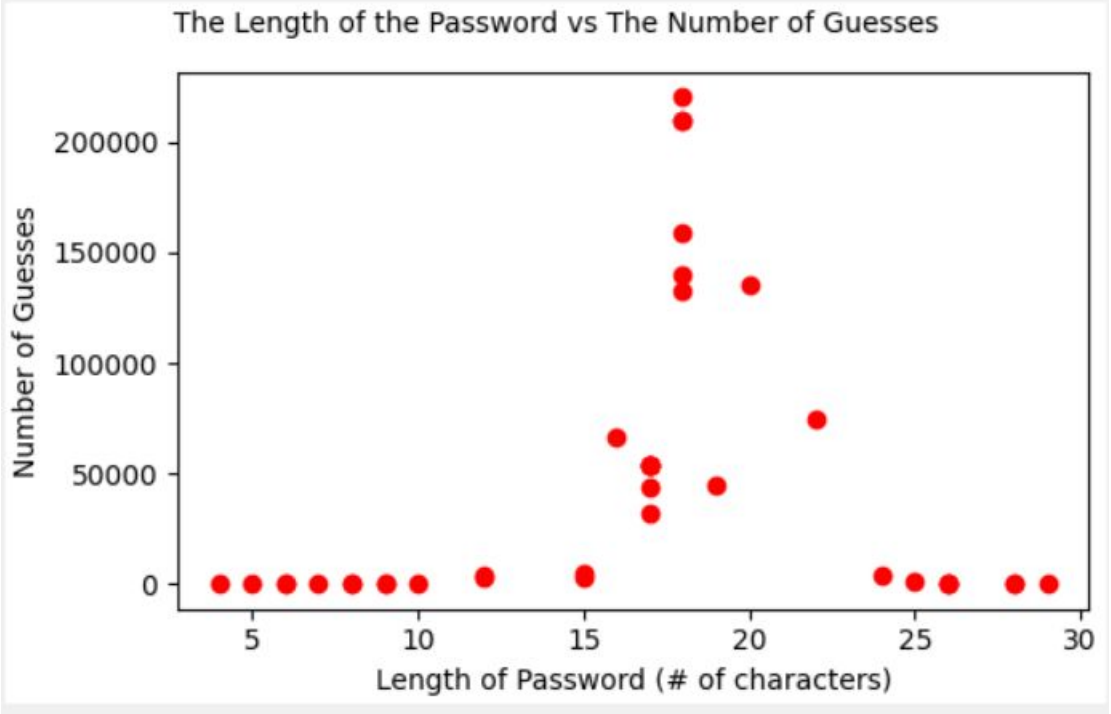
SHA-256





SHA-512


The Length of the Password vs Time it Takes to Crack
Time to Crack (seconds)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
5
10
15
20
25
30
Length of Password (# of characters)


The Length of the Password vs The Number of Guesses
Number of Guesses
200000
150000
100000
50000
0
5
10
15
20
25
30
Length of Password (# of characters)

There is a tendency for each of these graphs to have a sort of spike in the middle. This makes sense, as our program accounts for the length of the password given when it tries to crack it, and longer passwords are less common when combining the common password words, so it is faster in cracking longer passwords. The passwords that take the longest are multiple word combinations of short words, since they get confused with passwords of a less amount of words, taking many more guesses to get. The plots stay relatively consistent when comparing the time taken and number of guesses, indicating that the computer takes almost exactly the same amount of time for each guess. Also, these plots show that the SHA-256 hashing appeared to be cracked slightly faster than SHA-512, and in less guesses.

How to Run the Program on your Machine

To run the program on your own computer unzip the HW13_jjovel_samim19.zip file into your PyDev workspace. Open Eclipse in PyDev mode and select:
File -> New -> PyDev Project
In the dialogue box, uncheck 'Use default' and browse for the directory you unzipped. Once it is selected, name the project. The project will be created and the Python files will be in it. Navigate to the guiInterface package, and open the GUI.py file, run this file and you can start cracking passwords.