Cryptography Practical Coursework Report

# Introduction

Flask is a web framework for Python used to build REST APIs. It is minimal and flexible so that building an app can be as simple as possible (Herman, 2021). Vue is an open-source JavaScript framework used to build user interfaces that is beginner-friendly. Vuetify is a UI library with Material Design based components that can be used within a Vue application. For the purposes of this coursework, the aforementioned stack was used to build a small application to demonstrate the tasks as easily as possible.

# Task 1 - Verifying credit card numbers

For this task, a simple if condition in python, checking the length of the input string and whether it contains digits were used to fulfill the requirements. Appropriate responses were given if the check fails. While this task was entirely possible to complete within the frontend of the application, it was included in the back-end for the sake of consistency.

# Task 2 - BCH Generating and Correcting

# Task 3 - Brute Force Password Cracking

The password requirements for this task are any password which contains a maximum of six lower-case letters and/or digits. A list of lowercase alphabets and digits are used to check against the input password. Random combinations of alphabets and digits of the length of input were checked against the input password in a while loop for a match. Number of iterations and the time taken for match is recorded and provided upon matching.

The time taken to match the following passwords along with the number of iterations is shown below:

|  |  |  |
| --- | --- | --- |
| Password | Time Taken (in seconds) | Iterations |
| a | 0 | 36 |
| a1 | 0 | 15 |
| a1b | 0.23 | 92493 |
| a1b2 | 4.46 | 1532242 |
| a1b2c | 726.54s | 240463263 |
| a1b2c3 | 369.64 | 126056865 |

Since the matching is random there are slight irregularities in the trend. In general, as the input password gets longer, the time taken increases exponentially. For the 6-length password: a1b2c3, the time taken is less than for the 5-length password: a1b2c, this could be because matching is random, and the algorithm got lucky.

## Drawbacks

Random matching would sometimes try to match an already attempted input, thus making the algorithm much less efficient. Common brute-force algorithms use a big list of commonly used passwords to try beforehand. This method skips checking predefined password files and goes straight to the brute forcing part.

# Task 4 - Using Rainbow Tables to Break Passwords

A rainbow table is used to hack cryptographic hash functions to match passwords using a predefined table (Mezquita, 2020). While modern password protection techniques have somewhat invalidated the use of rainbow tables, unprotected passwords can be matched easily using this method.

A rainbow table stores all the one-way hashes of passwords in a certain length and can get large. A table generated in this task got up to 2.2GB in size.

To generate a rainbow table, the algorithm requires two inputs:

1. *possible\_inputs*: Possible characters in the password
2. *input\_hash*: Hash of the password to match

Python library, Permutations is then used to generate possible combinations of *possible\_inputs* which are then hashed using a hashing algorithm which will be explained later. The resulting data set is stored a text file which is the rainbow table itself. The *input\_hash* is then run through a loop for possible matches against the rainbow table.

## Hashing

Hashing transforms a given string into another value which is usually short and of fixed length making it easier to match to the original string (Zola, 2021).

The hashing algorithm used in task takes two arguments:

1. *perm*: Single permutation performed in input generation
2. *possible\_inputs*: Possible characters in the password

The variable *hashed* is the result of this function. The following steps are taken to reach *hashed*:

1. Trailing character is multiplied by 2
2. *perm* is added
3. result is reversed

## Testing

|  |  |  |
| --- | --- | --- |
| *possible\_input* | *Input\_hash* (hashed password) | Time taken (in seconds) |
| a | a | 0.01 |
| a1 | a1 | 0.01 |
| a1b | a1b | 0.01 |
| a1b2 | a1b2 | 0.01 |
| a1b2c | a1b2c | 0.01 |
| a1b2c3 | a1b2c3 | 0.03 |
| a1b2c3d | a1b2c3d | 0.28 |
| a1b2c3d4 | a1b2c3d4 | 1.57 |
| a1b2c3d4e | a1b2c3d4e | 13.62 |

The trend seems to be exponential. The generation of rainbow tables is the major contributor to the time taken as it takes a fraction of the time for the matching process.