**Cryptography Assignment Report**

Vince Verdadero - 19009246

University of the West of England – UFCFT4-15-3

Text

Description automatically generated with medium confidence

Table of Contents

[Table of Figures 2](#_Toc92971347)

[Table of Tables 2](#_Toc92971348)

[Introduction 2](#_Toc92971349)

[Task 1: Verifying Credit Card Numbers 3](#_Toc92971350)

[Task 2: Generating and correcting BCH (10,6) code 4](#_Toc92971351)

[Task 3: Using Brute force to break passwords 5](#_Toc92971352)

[Task 4: Implement rainbow tables to break password 6](#_Toc92971353)

[Conclusion: 9](#_Toc92971354)

[References: 10](#_Toc92971355)

# Table of Figures

[Figure 1 – Determine a valid ISBN (Yang, 2021c) 3](#_Toc92971379)

[Figure 2 - Formulas for generating 4 extra digits (Yang, 2021a). 4](#_Toc92971380)

[Figure 3 - Formula to calculate one error from lecture slides (Yang, 2021b) 4](#_Toc92971381)

[Figure 4 - Equation to find two positions and two error magnitudes (Yang, 2021b) 5](#_Toc92971382)

[Figure 5 – shows how to solve chain collision (Yang,2021d) 7](#_Toc92971383)

# Table of Tables

[Table 1 - Set A passwords 5](#_Toc92971392)

[Table 2 - Set B passwords 6](#_Toc92971393)

[Table 3 - Test data for a-z Rainbow table 8](#_Toc92971394)

[Table 4 – Test data for 0-9 Rainbow Table 8](#_Toc92971395)

[Table 5 - More test data for 0-9 Rainbow table 9](#_Toc92971396)

**Terms of Reference**

A report was presented to the University of the West of England in completion of the criteria for module UFCFT4-15-3.

# Introduction

For this project, I was given four tasks to do. For my first task, I had to verify whether it is a credit card number or not. Generating and correcting BCH (10,6) is the second task. Using brute force to break passwords was the third challenge, and the last task was to implement rainbow tables to break the password. Additionally, I'll describe what I have learnt from those programming tasks, keeping in mind the productivity of the algorithms and the approaches I've used.

# Task 1: Verifying Credit Card Numbers

For task 1, we have specified a task to receive a 16—digit string given by a set of test data to test and establish if it is a credit card number or not.  As well I had to verify ISBN (International Standard Book Number) too. This task was mainly on error detection.

To detect errors, the 10th digit of an ISBN is declared as a check digit. All numbers are increased from their index before being added to the outcome. If the total of all nine digits is divisible by 11, the code will be valid, otherwise, there will be at least one error because modulus by 11 is employed in ISBN. 10 is a possible value that must be represented by a symbol, in this instance an 'X.'

The symbol 'X' is essential and signified once the modulus operation returns 10.  Figure 1 demonstrates whether the given number can be used to determine whether it is a valid ISBN.

Chart, scatter chart, box and whisker chart

Description automatically generated

Figure 1 – Determine a valid ISBN (Yang, 2021c)

“Int [] digits = new int [16]; “was also given from the practical worksheet and used to determine and declare as an integer array to show the maximum length of 16 when programming for this task. This programming task was used in Java.

Credit card numbers work similarly to ISBNs in that they identify errors by utilising the last digit as a check digit. However, instead of using modulo 11, the last number is created to meet a set of criteria known as the Luhn Algorithm or Modulo-10 check. The weighted sum is determined whether the index is odd or even. All even indexes of the digits will be considered for testing, ensuring that every part of evenly totalled digits is expanded from doubling while the odd totalled digits remain unchanged (Hussein, 2013).

Although a single check digit cannot recognise all faults, the Luhn method can determine several single errors. If the result >=10, subtract 9.  The credit card number passes the Luhn test and is valid once all digits are combined, and the total sum is divisible by 10. If it's not correct, the number has been tampered with or a transposition issue has happened. Only because even numbers are doubled, the final checksum's weight is determined from adjacent positioning. Therefore, will correct simple transposition errors in which two integers are swapped, one in the even’s position and the other in the odds position. If Luhn's approach included index directing in the weighted checksum, all transposition problems would be detected. This approach is designed to detect single errors rather than harmful attempts like multiple nonadjacent transposition errors (Hussein, 2013).

# Task 2: Generating and correcting BCH (10,6) code

For 2nd task, requires employing BCH codes for multi-error correction to discover and rectify errors. BCH is a combination of the first letters of the three inventors, which are Bose, Ray-Chaudhuri and Hocquenghem (Yang, 2021b). BCH codes are a type of multiple random error-correcting code that falls into a large category. It is also widely used in applications such as Voyager 2 Spacecraft and CD players.

The programme begins by getting 6-digit inputs and uses the mod 11 methods to produce four additional parity digits. Since we are using mod 11, 10 is a feasible outcome from the calculations. By using the 4 checking digits created can fix double errors in 10 digital decimal numbers.

Text

Description automatically generated

Figure 2 - Formulas for generating 4 extra digits (Yang, 2021a).

Rather than using a symbol to represent 10, such as 'X' in ISBN, we would simply stop using numbers that require 10 in their checking digits, which are referred to as "unusable” numbers.

The first phase of the error correction (decoding) part of my task 2, is to initialise 4 -digit syndromes employing a Vandermonde matrix e.g., taking the 10 digits input: *d1, d2, …d10.* After these syndromes, we can determine that no faults have occurred in the process if all values are equal to 0 (S1, S2, S3, S4) == (0,0,0,0), Otherwise, the difficulty of the single error/double error must be determined and worked out by correcting them.

Three more values are calculated if there is at least one error: P, Q, and R, all of which are zero, indicating that there is only one error. To figure it out, use the following formula:

Text, letter

Description automatically generated

Figure 3 - Formula to calculate one error from lecture slides (Yang, 2021b)

Text, letter

Description automatically generated

Figure 4 - Equation to find two positions and two error magnitudes (Yang, 2021b)

There are more than two errors if either place of error given is 0 or there is no square root available for the quadratic equation.

# Task 3: Using Brute force to break passwords

This task is requiring merely the generation of plaintext and ciphertext, as well as a comparison of one ciphertext to another and will use a 6-character long password. In cryptography, the plaintext is an unencrypted message inputted into cryptographic algorithms that are frequently transmitted or stored ‘in clear’ or readable format.   SHA-1 is a hash function example and is used to generate a hash value (message digest).

To achieve this task, the software/application must crack two sorts of passwords from two sets.  One of them is Set A, which is any password with no more than six lower case characters and/or numbers in it. Then, there is Set B, which is any valid BCH (10,6) code as a password.

The most used method of decryption is brute force. It takes all available characters in the alphabet and tries all conceivable combinations and permutations of a given length before comparing the produced hash to the original hash.  The decrypted data can be found if a match occurs.

Iteration and Recursion are two approaches that helped me understand that I can use either one of them, which can be implemented for the brute force technique. Recursion is beneficial since it reduces the problem by repeating the situation and concluding to its simplest solution. Alternatively, Iteration can also be used where loops can be used to go through a range, for instance for this task the maximum length of the password is six and it will end the loop and terminate.

Here are the SHA-1 Hashes that were cracked from Set A and B (all the passwords were given from the practical worksheet)

Table 1 - Set A passwords

|  |  |  |
| --- | --- | --- |
| **Passwords from Set A** | **Plain Text** | **Milliseconds** |
| c2543fff3bfa6f144c2f06a7de6cd10c0b650cae | this | 881 |
| b47f363e2b430c0647f14deea3eced9b0ef300ce | is | 1 |
| e74295bfc2ed0b52d40073e8ebad555100df1380 | very | 872 |
| 0f7d0d088b6ea936fb25b477722d734706fe8b40 | simple | 181463 |
| 77cfc481d3e76b543daf39e7f9bf86be2e664959 | fail7 | 38473 |
| 5cc48a1da13ad8cef1f5fad70ead8362aabc68a1 | 5you5 | 35188 |
| 4bcc3a95bdd9a11b28883290b03086e82af90212 | 3crack | 411058 |
| 7302ba343c5ef19004df7489794a0adaee68d285 | 1you1 | 32845 |
| 21e7133508c40bbdf2be8a7bdc35b7de0b618ae4 | 00if00 | 1161647 |
| 6ef80072f39071d4118a6e7890e209d4dd07e504 | cannot | 802855 |
| 02285af8f969dc5c7b12be72fbce858997afe80a | 4this4 | 1268573 |
| 57864da96344366865dd7cade69467d811a7961b | 6will | 13427 |

Table 2 - Set B passwords

|  |  |  |
| --- | --- | --- |
| **Passwords from Set B** | **Plain Text** | **milliseconds** |
| 902608824fae2a1918d54d569d20819a4288a4e4 | 0000118435 | 2762 |
| 88d0b34055b79644196fce25f876bc1a5ef654d3 | 1111110565 | 1395 |
| 5b8f495b7f02b62eb228c5dbece7c2f81b60b9a3 | 8888880747 | 201 |

When trying to crack these hashes, I have noticed that when its length is short, the processing time will be quick. Although, if the length is longer, the more processing time increases.

# Task 4: Implement rainbow tables to break password

To do this assignment, two different files are required, one of them is to generate and one of them is to crack the password. This task is devised to achieve a balance between memory space and processing time. This task is related to task 3, however, it relies on data compression.

Time management can be an issue when trying to predict the password and compare the hash. Utilising a wordlist, for instance, the most used passwords can be a method for breaking a password.  Depending on the size of the wordlist, this can also be time-consuming if it is near the complete search field for the password.   Rainbow tables solve both issues; they don't take up as much space as a standard brute force method and they don't take as long to get the password. A rainbow table works that produces chains, which take a limited amount of storage space.

From the beginning, there are sequences of chains, and at the end of the plaintext, there will be a mixture of these with reduction strings too. Firstly, the beginning will produce a random string, then at the end, a chain length is produced after a loop of reduction/hashing of this randomly generated string. A reduction function converts a hash into plaintext.  Having a plaintext can be used to match with the chain's end.

When attempting to crack and reduce the hash, it will be kept in one of the hash chains if there's a match. Since it's saved, the plaintext of the original hash is utilised within that chain, and it still is verified (Geek for Geeks, 2021).

Effectively it conserves space while also saving time. The brute force task, for example, took a long time because it required comparing the output hash to the goal hash and then regenerating the password. This method is much faster since it won't have to switch back and forth through checking and generation, instead of concentrating on producing chains and cracking basic loop through and contract.

Although, this strategy will encounter collisions in the hash table throughout implementation. If a non-unique hash is generated from a hash function, a collision may occur.  When a chain is formed with a unique key or end reduction, collisions in the rainbow table are dramatically reduced. Therefore, if it takes an endless amount of input length and produces a big outcome, a collision could happen through the process.

For instance, Figure 5 shows how to solve the chain collision. H stands for the hash function, R1....R3 is different reduce functions. Therefore, it is called a rainbow table since different reduced functions are like the difference of colours.

Table

Description automatically generated

Figure 5 – shows how to solve chain collision (Yang,2021d)

Furthermore, it needs to be satisfied when completing the table which can occur if bad values are implemented for the chain length/number. Also, it can result in not being able to establish a new chain. For it to work, the user must be aware before using this application otherwise, the application will never finish and must be terminated (Privacy Canada, 2021).

For table 3, I have generated 150 million passwords with the character set a – z. These are lower case hashes. These sets of hashes were given by the module teacher. To create this rainbow table, I did 150000 as the number of chains, the length of the chain was 1000 and the length of the password was 6. The table shows 7/10 passwords that were cracked.

The results on my rainbow table task for a-z:

Table 3 - Test data for a-z Rainbow table

|  |  |  |
| --- | --- | --- |
| **SHA-1 Hash** | **Passwords cracked** | **Milliseconds** |
| ad9966bd4b4a82e086b3f96fb4132cbf284efdb9 | zzaapf | 191 |
| dc551ddda247d9307a340a57ce2679f9fbf70b71 | oidhje | 146 |
| cadf11f0ed2fbdba016fa2935a466fe424e30565 | Not found | N/A |
| d9fba47a4be2b9c68349ecef481fc90fb10cca73 | wichen | 35 |
| 4eb3078d5c65f923173b5dc0a2be5af361362a61 | okrmrw | 38 |
| 907916580d665b4fe4323f316cb3acd2d814e6f4 | uuuxxx | 41 |
| a2b7caddbc353bd7d7ace2067b8c4e34db2097a3 | zzzzz | 37 |
| aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d | Not found | N/A |
| fbc8fae6b1390136c802d43f16890134bfe73df7 | Not found | N/A |
| 37e7e58c067169fb08c97978fff80a00dc57fdfd | cieoe | 38 |

The results on my rainbow table task for 0-9:

Table 4 – Test data for 0-9 Rainbow Table

|  |  |  |
| --- | --- | --- |
| **SHA-1 Hash** | **Passwords cracked** | **Milliseconds** |
| fe635ae88967693bc7e7eead87906e62e472c52f | 187494 | 186 |
| 3ac2d907663deccd843f9bbcf0c63bd3ad885a0e | 940376 | 82 |
| 3557c095ed6c16a90febda48d6b3a4490107b0d9 | Not Found | N/A |
| 85e04129ed328d4a2b3eedabca74d08b3e6badc1 | 0987593 | 65 |
| 70352f41061eda4ff3c322094af068ba70c3b38b | Not found | N/A |
| 052bd5b02559d1270866c5626538e720cec0c135 | Not found | N/A |
| 3e71f65d56cb29521ac16ff1f92ecace156b1db5 | Not found | N/A |
| bfc52d4e36cb45cb667749982755e63630f3bc93 | 09680243 | 258 |
| 8cb2237d0679ca88db6464eac60da96345513964 | 12345 | 46 |
| 38bbc0a1ca7e9b3e9f6ab33782e0f780f009db1f | 99887766 | 38 |

Table 5 - More test data for 0-9 Rainbow table

|  |  |  |
| --- | --- | --- |
| **SHA-1 Hash** | **Passwords cracked** | **Milliseconds** |
| 21e4b025b7b858928ec3ce22e373ff5b28df87ad | 58493072 | 199 |
| d7d5d8b3838452a833f4c6a30a3b9a78cb88e530 | 0139826 | 83 |
| 7236da7fee4ddcb8c389b9732e78c7c4e1fcc1e4 | 60248677 | 95 |
| 564831f5a86d1d0b8f465042c6e17c4e7d1a1e7d | 92763708 | 37 |
| 7f06bee6c1214d732a9e40765bfae0aa3e547c09 | Not found | N/A |
| 6ec7be98bca5d1812fa41dbb719b552d01c253f7 | Not found | N/A |
| cedd8db553409b282deb6ca276203c5edd6f0971 | Not found | N/A |
| 8bc2059150084f16ad065924444d552ce521c090 | 09385847 | 37 |
| 886e177c49dc6dd72176e1be8ba5f4f907f54883 | Not found | N/A |
| 911a45dec90e5ee1d4f5acc8ce17c8068c9512b0 | Not found | N/A |

I have generated 150 million passwords, and these were the results given in table 4 and table 5. These test data was given by the module tutor. To create this rainbow table, I did 150000 as the number of chains, the length of the chain was 1000 and the length of the password was 8. I also saved it as a text file, and this can be opened too when I restart and run it again to save time generating the table. Throughout this task, I only broke 6 of the 10 passwords in table 4, and 5 out of 10 in table 5. The milliseconds happened to be quicker than the brute force as expected.

Furthermore, I believe that throughout these results, the password that was meant to be cracked was not generated to be in the chains that the table was kept. This would mean that collisions have occurred, as previously stated, and this should be avoided in broad search spaces. However, if the tables were larger, the passwords that were not found may be generated and be found, although more collisions may happen too.

One of the issues that I encountered in my program was testing the A-Z character set, sometimes it generates quite slow or doesn’t generate. Also, I had some problems too when generating two-character sets or all character sets that’s been picked because there is a slight bug when trying to generate them. However, if one user selects either one of the 0-9 or a-z, it will work fine.

# Conclusion:

I have done my best to finish all the tasks. A few tasks could be improved to remove the repetition and reduce redundancy.  Additionally, if I had more time, I could've looked at other programming languages, such as Python, to see if I find a more time-efficient approach. Also, since I have a few missing for the test data in some of the rainbow tables, it'd be good to see what the leftover password is and figure out how to avoid collisions for better password hacking.

# References:

Geeks for Geeks (2021) *Understanding Rainbow Table Attack.* Available from:<https://www.geeksforgeeks.org/understanding-rainbow-table-attack/> [Accessed 10 December 2021]

Hussein, K.W., Sani, N.F.M., Mahmod, R. and Abdullah, T. (2013) Enhance Luhn Algorithm for Validation of Credit Cards Numbers. *International Journal of Computer Science and Mobile Computing* [online]. 2 (7), pp. 262-272. [Accessed 01 December 2021].

Privacy Canada (2021). *Hash Collision Attack*. [online] Available at: <https://privacycanada.net/hash-functions/hash-collision-attack/> [Accessed 01 December 2021].

Yang. R. (2021a). *Lecture slides week 2 – Hamming Codes*. PDF and PowerPoint. Cryptography [online]. Available from: Blackboard. [Accessed 03 December 2021]

Yang. R. (2021b). *Lecture slides week 3 – BCH Codes.* PDF and PowerPoint. Cryptography [online]. Available from: Blackboard. [Accessed 03 December 2021]

Yang. R. (2021c). *Starting Block 0 – Cryptography Intro session.* PDF and PowerPoint. Cryptography [online]. Available from: Blackboard. [Accessed 01 December 2021]

Yang. R. (2021d). *Tutorial slides week 5&6 – Password Security (Password Attack).* PowerPoint. Cryptography [online]. Available from: Blackboard. [Accessed 10 December 2021]