Contents

1. Overview of the Assignment

2. Programming Task Requirements

3. Report Requirements

4. Marking Criteria

5. Deliverables

6. Plagiarism

1. Overview of the Assignment

This assignment requires you to complete 4 programming tasks and to write a brief report on your findings. The programming tasks carry 80% of the marks whilst the report carries the remaining 20 marks.

Module learning outcomes are described in the module specification and the ones that are relevant to this assignment are:

* Understand and manipulate the mathematical and theoretical methods on which designs are based.
* Implement algorithms and protocols for particular coding schemes, recognising the need for efficiency in terms of delay, throughput, jitter, computing resources and quality of service.
* Use cryptographic and coding classes available in modern programming language environments, such as Java Security, to implement secure applications
* Evaluate the performance of various coding schemes under application load and change configuration parameters to optimise them
* Explain the strategies that need to be employed whilst attempting to break a cipher.

|  |
| --- |
|  |

The assignment is worth *75%* of the overall mark for the module.

The assignment is described in more detail in section 2.

This is an individual assignment.

Working on this assignment will help you to consolidate your understanding of the material that is presented in the lecture. You will also improve your general software development and problem solving skills. If you have questions about this assignment, please post them to the discussion board on Blackboard or contact to the module leader.

1. Programming Task Requirements
   1. Task 1. Verifying credit card numbers. 5 marks

Your code should accept a 16-digit string and determine whether or not it is a valid credit card number.

* 1. Task 2. BCH Generating and Correcting. 25 marks

For the BCH(10,6) generator, unusable numbers must be identified. For the BCH(10,6) error corrector, single and double errors must be corrected, and more-than-two-error cases must be detected

* 1. Task 3. Brute Force Password Cracking. 10 marks

The brute force program should be able to break any password which contains a maximum of six lower-case letters and/or numbers.

* 1. Task 4. Using Rainbow Tables to Break Passwords. 30 marks

You need to build a rainbow table, and then use it to break passwords. The program should make it easy to change the configuration of the alphabet set, the length of string, the size of the rainbow table, and the length of the chains. Due to time and memory limits, you are not required to try very large tables, but in order to get a full mark for this task your code should be able to handle about 100 million passwords. This task consists of two steps (i.e. two programs):

* Step 1 - Building a rainbow table (15 marks);
* Step 2 - Using the rainbow table to break passwords (15 marks).
  1. General Requirements (applies to all tasks) 10 marks

Invalid inputs to the program must be identified. For example, if the input for the credit card number is not a 16 digit long number, a warning message must be displayed.

There is no restriction on what language to use. The chosen language should be able to handle big integers and have SHA1 function in its library

Your code should exhibit the characteristics of professional code. For example, it should well-structured and concise. It should be appropriately commented, variables names should be meaningful. The interface should be user friendly.

1. Report Requirements 20 marks

Write a report on what you have learnt from the above programming tasks. This is an open topic, but as a minimum requirement you should consider the efficiency of the algorithms used, and explain the strategies applied. For tasks 2.3 and 2.4 (i.e. password cracking), you should conduct timing tests then include your results and your analysis of them in the report. The literatures related tasks should be referenced. There is no word limit on the report, however as a rough guide, 700-800 words should be sufficient.

1. Marking Criteria

See marking criteria table in Appendix on the next page.

1. Deliverables and Demonstration

Students are required to submit the following through blackboard:

1. A listing of the source code (please do not use zip to avoid unexpected issues with unzip)
2. A report (in Microsoft Word)

Students can optionally demonstrate programming tasks in scheduled lab sessions to tutors. The provisional marks and feedbacks will be given. It is allowed to redo demonstration to improve marks before the final submission deadline.

1. Plagiarism

While it is acceptable to discuss your assignment with your peers, this piece of work is intended as an individual assignment. Works that are substantially similar will be subject to investigation according to University regulations. Directly copying entire lines of code from the internet is strictly forbidden. You may use partial lines of code from the internet however you must clearly reference the source.

Appendix - Marking Criteria

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0-20% | 21-40% | 41-60% | 61-80% | 81-100% |
| Task 1  Verifying credit card numbers  (5 marks) | Little or no attempt at the task | The code can’t compile, but it shows some correct understanding of Luhn algorithm | Partially working.  The code has some minor logic errors | Mostly working but failing to pass a test example | Fully working and passing all random tests |
| Task 2  BCH(10,6) code generating, and error correcting  (25 marks) | Little or no attempt at the task | Only produced partial code for some minor functions. For example, generating BCH(10,6) code | Can detect and correct both single error and double errors for some simple cases. But code still contains logic errors | Can detect and correct not only single error and double errors but also identity more than two errors. Code is almost perfect but it fails to pass a few more complex cases | Fully working and passing all random tests cases provided by module leader.  The code is robust and efficient |
| Task 3  Brute Force Password Cracking  (10 marks) | Little or no attempt at the task | The code can’t compile, but it shows some correct understanding of brute force algorithm | The code is correctly working for smaller set of passwords, but can’t scale. | The code is almost perfect but fails to pass a few tests. | Passing all test data provided by module leader, and collecting timing results |
| Task 4  Using Rainbow Tables to break passwords  (30 marks) | Little or no attempt at the task | Some pseudo code which shows correct logic of rainbow tables | The code is partially working, e.g. only generate rainbow table. Or it only works for short passwords and can’t scale up | The code can generate correct rainbow tables with required configuration and it can crack some pre-set passwords. | Fully working and passing all random tests provided by module leader |
| General Requirement  For Programming Tasks  (10 marks) | Little comments in code.  Totally Unstructured code | Limited comments  some good program structure  but program tasks only achieved less than 40% | Some comments  good program structure  but program tasks only achieved less than 60% | Detailed comments  Excellent use of program structures  but program tasks only achieved less than 80% | Code quality has reached professional level, and all tasks are completed |
| Report  (20 marks) | No discussion; no results and/or analysis. Little evidence of insight into the implications of the work completed | Some discussion but misses the main points. Few results and/or results are unrepresentative  Little analysis. | Adequate discussion of what was learnt. Some results are presented and analysed. | Good discussion of what was learnt. Results for functions are well-presented and all are analysed | An excellent discussion of what was learnt contextualised with reference to academic writing on the subject. Results are well-presented and analysed in depth. All programming tasks are completed. |