COMP2300/6300: Applied Cryptography

Assignment 1 Description

Macquarie University
Session 1, 2024

1 Assignment Deadline

Assignment 1 (15%): Applying symmetric encryption/decryption and cryptographic hashing on data files or messages using the Python programming language.

- **Submission due**: Sunday 07 April 2024 11:55PM (Sydney time, AEDT)
- **Type**: Individual assignment

2 Late Submissions, illness, or special consideration

- (From the unit guide) Unless a Special Consideration request has been submitted and approved, a 5% penalty (of the total possible mark) will be applied each day a written assessment is not submitted, up until the 7th day (including weekends). After the 7th day, a grade of '0' will be awarded even if the assessment is submitted. Submission time for all written assessments is set at 11:55 pm. A 1- hour grace period is provided to students who experience a technical concern.

3 Learning Outcomes Linked to This Assessment

By completing this assignment, you should demonstrate your ability to:

- ULO1: Explain the concepts and principles on which modern cryptography relies upon.
- ULO2: Employ adapted cryptographic tools and techniques to encrypt, decrypt and sign messages.
- ULO3: Decipher simple encrypted messages using a range of cryptanalysis methods.

4 Submission

- **Submission method**: In the iLearn Assignment 1 submission box, there are six files to submit for marking. DO NOT zip or compress the files, and submit the files as is. No folders, no zip, no rar, and no compressed formats.
  - A txt file (with the filename “assignment_answers_<YourStudentID>.txt”) containing your answers to the tasks described below. Note: Replace <YourStudentID> with your own student ID.
  - Three Python files with the completed tasks for Task A, Task B, and Task C, respectively. Completed programs will be executed to validate the submitted answers.

- Two files from Task B: <YourStudentID>_Task-B-2_plain-text.txt from Subtask B.2, and <YourStudentID>_Task-B-3_cipher-text.txt from Subtask B.3. They will be generated from the completed Python code.

- **Number of submissions**: You can submit or update as many times as you like between now and the deadline. You are encouraged to submit early drafts in case of any last-minute technical issues (even if only partially completed).

5 Marking Rubric

General marking criteria:

- All the required programming tasks have been reasonably completed and all the questions have been answered.
- The correctness and readability of the answers in the submitted report.
- The right decrypted and encrypted files.
- Working code and the quality of source code.

Task A (5 marks)

- Subtask A.1 (2 marks): 1 mark for the right answers to (1) and (2), and 1 mark for the source code producing the right answers.
- Subtask A.2 (3 marks): 0.5 mark for the right answer to (1), 0.5 mark for the justification to the answers to (1), 0.5 mark for the right answer to (2), 0.5 mark for the right answer to (3), and 1 mark for the source code producing the right answers.

Task B (4 marks)

- Subtask B.1 (2 marks): 0.5 mark for the right answer to (1), 0.5 mark for the right answer to the modified plain text and its cipher text in (2), 0.5 mark for the right answer to the percentage in (2), and 0.5 mark for the source code producing the right answers.
- Subtask B.2 (1 mark): 0.5 mark for the successfully decrypted file (the decrypted text should be readable and meaningful in English), and 0.5 mark for the source code producing the output file.
- Subtask B.3 (1 mark): 0.5 mark for the successfully encrypted file (markers will check if it is able to be decrypted with the given key), and 0.5 mark for the source code producing the output file.

Task C (6 marks)

- Subtask C.1 (2 marks): 0.5 mark for the reported answer to (1) and (2), 0.5 mark for the right answer to (3) and (4), and 1 mark for the source code producing the right answers.
- Subtask C.2 (2 marks): 0.5 mark for the right answer to (1), 1 mark for the right answer to (2), and 0.5 mark for the source code producing the forged message and its hash value.

- Subtask C.3 (2 marks): 0.5 mark for the right answer to (1), 0.5 mark for the right answer to (2), 0.5 mark for the right answer to (3), and 0.5 mark for the source code producing the right answers.

6 Assignment Details

Task Overview: In Assignment 1, you are required to complete three tasks, Task A, Task B, and Task C, covering the topics in Affine cipher, symmetric ciphers, and hash functions, respectively. In each task, you are provided with a Python file where some base source code for necessary function definitions, code examples of running the functions, and the task descriptions are detailed. Each task consists of several subtasks to assess your understanding and skills from different perspectives. First of all, you need to comprehend the existing source code that defines some basic functions and demonstrates several examples of using the functions. Based on the understanding, you need to further write your own code to complete the subtasks, and the results from the execution of the code should answer the questions in each task. The answers should be reported in the file "assignment_answers_<YourStudentID>.txt", where <YourStudentID> should be replaced by your own student ID. The document with the answers and the 3 completed Python files will be submitted for marking. Task B also produces files as outputs, which should be submitted for marking as well. Note that there some dependencies and data sharing among the three tasks, e.g., to complete Task B you need the information from the results of Task A. Each task is further detailed subsequently.

Task A (5 marks):

The corresponding Python source code file is "Task-A_affine-cipher.py". To run this program, you can use the command if you use a command line interface: *python Task-A_affine-cipher.py*. Please do not alter the code with the comment line "Please do NOT modify the following code". You can start writing your code after the comment line "TODO: Your code …". **Please read the comment lines carefully because the task details are described in these comment lines**.

In this task, we aim to let students develop a solid understanding on historical ciphers, or more specifically, the Affine cipher. In the Python file, we have implemented an extended version of Affine cipher where the extended alphabet set is all the printable characters rather than the 26 letters. The encryption and decryption functions have been implemented and are ready for use. In the examples, we set the alphabet set with all the printable characters (from ' ' to '\x7f'), and demonstrate how to encrypt plain text and decrypt cipher text with a pair of keys.

In **Subtask A.1 (2 marks)**, you are required to choose a pair of keys and encrypt a given plain text "Hello World, COMP2300/6300!". The challenge here is to choose an appropriate pair of keys with respect to the size of the extended alphabet set.

In **Subtask A.2 (3 marks)**, you are required to decrypt a given piece of cipher text without the full information of the key pair. The cipher text is "Au*-`*^-z(*1-nK-Hz?!F(`-_+v-n(-!u*-(!z1*K!-.Q-a}}}}}}}9-FK1-!u*-`*^-z(*1-nK-Hz?!F(`-_+o-n(-^Dz/-DlK-(!z1*K!-.Q+". The second key in the key

pair “key_b” is already known as 13, but the first key “key_a” is missing. There are two possible options to achieve this subtask. One option is to use the exhaustive method to try all the possible keys (this option is more recommended for simplicity). If you follow this option, you are required **to minimise the number of possible keys to attempt**. The challenge is how to ensure the minimisation. The other option is to use frequency analysis, which would be more advanced and complicated. Note that you are just required to take one of these two options to complete this subtask, but you are welcome to take both if you wish. This decrypted information from this subtask will be used in Task B.

Note: In the implementation of this extended affine cipher, specifically for the decryption function, we make use of the Python module *egcd* which implements the extended Euclidean Algorithm to obtain the multiplicate inverse of an element in the group $\mathbb{Z}_n$. The module might not be installed in your Pyhton environment, and you need to install it with the following command: *pip install egcd*. Please refer to https://pypi.org/project/egcd/ for more information on how install and use this module.

Task B (4 marks):

The corresponding Python source code file is “Task-B_symmetric-cipher.py”. To run this program, you can use the command if you use a command line interface: *python Task-B_symmetric-cipher.py*. Please do not alter the code with the comment line “Please do NOT modify the following code”. You can start writing your code after the comment line “TODO: Your code …”. **Please read the comment lines carefully because the task details are described in these comment lines**.

In this task, we aim to let students experience in using an implementation of symmetric cipher with Fernet and develop further understanding of some features of modern symmetric cipher like the avalanche effect. Refer to https://cryptography.io/en/latest/fernet/ for detailed documentation for Fernet. In the Python file, we have implemented a function to produce a key from a given string. You can use this function to create your key to use Fernet functions. We have demonstrated the examples of using Fernet to encrypt and decrypt some texts. Please pay attention to the code achieving conversion between a string and a binary string which is the input or output of a Fernet function.

In **Subtask B.1 (2 marks)**, you are required to observe the padding techniques used in modern block ciphers and the avalanche effect from the diffusion and confusion operations in modern block ciphers. To observe the avalanche effect, you are required to write code for calculating the percentage of overlap between the cipher texts from two plain texts of slight difference.

In **Subtask B.2 (1 mark)**, you are requested to call Fernet APIs to decrypt the cipher text from a given file. There are two txt files containing the cipher text and you just need to choose one of them as the input for the decryption function. For Linux/Mac users, please use the file “mac_linux_data_encrypted.txt”, and for the Windows users, please use the file “windows_data_encrypted.txt”. The key information is from Subtask A.2. The decrypted text

should be saved in a file with the file name “<YourStudentID>_Task-B-2_plain-text.txt” for submission.

In **Subtask B.3 (1 mark)**, you are requested to call Fernet APIs to decrypt the plain text from the Subtask B.2 output. The key information is from Subtask A.2. The encrypted text should be saved in a file with the filename “<YourStudentID>_Task-B-3_cipher-text.txt” for submission.

Task C (6 marks):

The corresponding Python source code file is “Task-C_hash-function-and-MAC.py”. To run this program, you can use the command if you use a command line interface: *python Task-C_hash-function-and-MAC.py*. Please do not alter the code with the comment line “Please do NOT modify the following code”. You can start writing your code after the comment line “TODO: Your code …”. **Please read the comment lines carefully because the task details are described in these comment lines**.

In this task, we aim to let students develop a solid understanding on the construction and the properties of cryptographic hash functions, as well as experience the use of modern cryptographic hash functions such as SHA256. Refer to https://cryptography.io/en/latest/hazmat/primitives/cryptographic-hashes/ for detailed documentation for the SHA family. In the Python file, we have implemented a simple hash function and the birthday attack targeting this hash function. The simple hash function is implemented by taking the product (modulo 365) of the ordinal numbers of characters at the positions indexed by prime numbers. The birthday attack is implemented by uniformly randomly generating a pair of messages from a message space without replacement, and checking if their hash values collide. The number of examined hash values before the first collision is recorded and returned, together with the two collided messages. We use the output file from Task B.2 named “<YourStudentID>_Task-B-2_plain-text.txt” as the message set and each line of in the file is regarded as a message. We have demonstrated examples of using the hash function and performing the birthday attack once.

In **Subtask C.1 (2 marks)**, you are required to perform to the birthday attack multiple times (>=100), and calculate the average number of examined hash values before the first collision. Compare this value with the theoretical one ($\sqrt{0.5 * \pi * n}$, where n is size of the hash value space) to check the compliance, and explain the difference if the difference is significant (say, +/-5).

In **Subtask C.2 (2 marks)**, through understanding the construction of the hash function, you are requested to forge a message for a given message, i.e., to identify a second preimage. The given message is Line 5 in the file "<YourStudentID>_Task-B-2_plain-text.txt" from Task B.2: "Creative Commons Corporation ("Creative Commons") is not a law firm and" The change should be minor with substitution of a few characters, and the modified version still produces the same hash value as the given message.

In **Subtask C.3 (2 marks)**, you are requested to employ SHA256 to generate the hash values for the Line 5 message and the forged message (the same as in Task C.2), and then compare

the two hash values to check if they are equal to each other. Further, you are requested to calculate the percentage of the overlap between the two hash values, in terms of bits, bytes, or hexadecimal numbers, to observe how much avalanche effect can be obtained in modern cryptographic hash functions.

Notes: In the implementation of the hash function, we use the module *sympy.sieve* to generate prime numbers. Refer to https://docs.sympy.org/latest/modules/ntheory.html for more details. To install the *sympy* package (if you don't have it by default), please refer to https://docs.sympy.org/latest/install.html.