Introduction to Secure Systems
University of Crete
Spring 2024
Prof. Evangelos Markatos

Assignment 1

Cryptography Algorithms & Key Store

Konstantina Papafragkaki (csdp1339)

Assigned: 05/03/2024
Due: 19/03/2024

Office Hours will be held on *Mondays* and *Wednesdays* from 16:00 to 18:00 in rooms *B208*, *B210*, and *B212* or remotely at `https://uoc-gr.zoom.us/j/84560032286`.

The assignment is submitted via turnin in the school machines using this command: **turnin assignment_1@hy457**.

Part A: Cryptography Algorithms

For this section of the assignment, you will implement encryption algorithms and attempt to decrypt them without knowing the key. You are required to create two files: `cs457_crypto.h`, which should contain function declarations, and `cs457_crypto.c`, where you will implement these functions. Only the C programming language should be used, without relying on any external libraries. If you want to examine the corresponding examples of cryptanalysis, you may see the associated Tutorial that is held on the site `https://www.csd.uoc.gr/~hy457/assignments.html`.

You should implement a demo (test file) for the functions described in part A of the assignment, demonstrating successful encryption and decryption processes.

1. The one-time pad is an encryption technique that utilizes a randomly generated key, often referred to as a one-time pad, with a length at least equal to that of the plaintext. During encryption, each bit or character of the plaintext is XOR-ed with the corresponding bit or character of the key.

   - Implement the functions **one_time_pad_encr** and **one_time_pad_decr**. These functions should accept the plaintext or ciphertext, its length, and the randomly generated key as arguments, and return the result accordingly.
   - Use a pseudorandom number generator to generate the key, such as `/dev/urandom`. Since `/dev/urandom` provides a new random value upon each read, it is essential to generate a random secret key and store it in memory to successfully decrypt the encrypted message.

- It is assumed that the plaintext consists only of letters or numbers.

2. The affine cipher is an encryption method where each letter (either in upper or in lower case) is mapped to its numeric equivalent (`'x'` in plaintext and `'y'` in ciphertext), encrypted using a simple mathematical function, especially **(5x + 8) mod 26**, and then converted back to a letter using the function **21(y - 8) mod 26** for decryption.

- Implement the functions **affine_encr** and **affine_decr**. These functions should accept the plaintext or ciphertext and return the result accordingly.
- It is assumed that the plaintext consists only of letters and/or spaces, and the program should handle letters in both upper and lower cases.

3. Write a decryptor for the simple substitution algorithm that decrypts a ciphertext without knowing the key (cipher alphabet).

- Decrypt the following: 'Pfim im k pwbp pfkp fkm nwwx wxqjedpwt smixc pfw kzzixw krcajipfu kxt civwx km kx kmmicuwxp ix pfw Qaudspwj Mqiwxqw Twdkjpuwxp az pfw Sxivwjmipe az Qjwpw.'

The decryptor will execute the following functionalities in each iteration:

(a) Read the ciphertext and prompt the user to enter a mapping (alphabet to cipher alphabet). The tool will display the plaintext that can be decoded so far.

(b) Ask the user to enter a partially decrypted word and the program will print the common words that match the pattern in the ciphertext. To do that, you will follow the below steps:

   i. Take as argument a text file and the program will print the frequency of each character.

   ii. Calculate the frequency of the English Dictionary (`https://github.com/dwyl/english-words`) and the ciphertext using the functionality at (i).

For example:

Given the first cipher text as input and the mapping i -> c the following intermediate plaintext should be printed:

**** ** * **** **** *** **** ********* ****i *** ****** **i****** *** i**** ** ** ****i**** ** *** ******** ******* ********* ** *** ********** ** *****.
Enter partially decrypted word: th**
this, them

Next mapping: c -> e

**** ** * **** **** *** **** *****c*** ****i *** ****** **i****** *** i**** ** ** ****i**** ** *** ******** ******* ********* ** *** *********c ** *****.
Enter partially decrypted word: c*t
cat

Repeat this process until you are able to decode the whole plaintext, you should include the plaintext in the Readme file.

4. The Trithemius cipher, also known as the Trithemian cipher or Tabula Recta cipher is a polyalphabetic substitution cipher that uses an encryption table, often referred as tabula recta for both the encryption and decryption processes. The tabula recta consists of 26 letters arranged along the top of each column, representing the letters A to Z (or a to z, depending on the case of the letters). Additionally, the alphabet table repeats along the left side at the beginning of each row. In each row, the letters are shifted one position to the right in a cyclic manner as the rows progress downwards. For instance, when 'B' moves to the front, 'A' shifts down to the end of the row. This cyclic shifting continues throughout the entire table.

- Implement the functions **trithemius_encr** and **trithemius_decr**. These functions should accept the plaintext or ciphertext and return the result accordingly.
- In the encryption process, each letter of the plaintext is substituted with the corresponding letter from the appropriate row of the alphabet table. The row used for substitution is determined by the position of the letter in the plain text and the corresponding shift value in the shift sequence.
- Decryption reverses this process by applying the inverse shift to each letter.
- It is assumed that the plaintext consists of letters, punctuation, and/or spaces, and the program should handle letters in both upper and lower cases. During encryption, spaces and punctuation should be omitted, and they should be added back to the generated plaintext after decryption.

5. The Scytale cipher is an encryption method using a simple device known as a Scytale, which consists of a rod and a parchment strip. The rod is typically a cylindrical object, such as a wooden stick, with a specific diameter. The parchment strip is a long, narrow piece of material, such as parchment or leather.

- Implement the functions **scytale_encr** and **scytale_decr**. These functions should accept the plaintext or ciphertext, as well as the diameter of the rod, and return the result accordingly.
- During encryption, the parchment strip is wrapped tightly around the rod, and the message is then written along the length of the strip.
- Decryption involves wrapping the parchment strip around a rod of the same diameter used for encryption. When the strip is wrapped around the rod, the letters align properly, allowing the original message to be read.

- It is assumed that the plaintext consists of letters, punctuation, and/or spaces, and the program should handle letters in both upper and lower cases. During encryption, spaces and punctuation should be omitted, and they should be added back to the generated plaintext after decryption.

6. The Rail Fence cipher rearranges the letters in the plaintext by writing them in a zigzag pattern across a predetermined number of rails and then read them out to form the ciphertext. After each letters of a rail is read, a space is added to the ciphertext. The process is repeated for each rail.

- Implement the functions **rail_fence_encr** and **rail_fence_decr**. These functions should accept the plaintext or ciphertext, as well as the number of rails in the encryption process, and return the result accordingly.
- During encryption, plaintext is written diagonally across the rails. The ciphertext consists of the letters of the fence in each row-rail with a space added after reading each rail.
- In the decryption process, the number of rails are found from the ciphertext and it is written once again diagonally to produce the plaintext.
- It is assumed that the plaintext consists of letters, punctuation, and/or spaces, and the program should handle letters in both upper and lower cases. During encryption, spaces and punctuation should be omitted, and they should be added back to the generated plaintext after decryption.

Part B: Key-Value Store

7. Write a C program that implements a simple Key-Value Store to safely store keys and the corresponding values in a database. The program should support 2 functionalities: **add** a (key,value) pair into the database and retrieve either a value (**read**) or a range of values (**range-read**) from a key or a range of keys accordingly. To ensure security, you have to encrypt the whole database, which means the file and each pair separately using AES. In order to do that, you will use the OpenSSL library (`https://www.openssl.org/`) which provides a wide range of cryptographic algorithms.

More specifically, you are asked to implement the following operations:

- **Add** a (key,value) pair to the database:

  $ kv add -f <filename> key value

  With this command, your program should take the database file, encrypt both key and value using AES with CBC mode and then store them in the database file. If the file exists, you should append the new entry (pair) at the end of the file. If not, you should create the file and then add the new entry. The database file should be in CSV format. Assume that key and value are both positive integers.

  Additionally, the user will have to provide a master password that will be used to generate the key and IV for the AES encryption.

  (Hint: This `https://www.openssl.org/docs/man3.1/man3/EVP_BytesToKey.html` function might help you).

  For example:

  ```
  $ kv add -f db.txt 1 3
  Enter password: pass
  ```

  The user wants to store in the Key-Value Store the key **1** and the value **3**. The program asks him for the master password and the user types 'pass'. The Key-Value Store will use the word 'pass' to generate the key and IV for the AES encryption of the (key,value) pair and the file encryption. After the operation, the encrypted database file should look like this:

  db.txt

  ```
  key,value
  <encrypted key>,<encrypted value>
  ```

- **Read** the value from the (key,value) pair in the store:

  $ kv read -f <filename> key

  With this command, your program should read the database file, find the entry that corresponds to the given key (decrypt the pair to see if the given key matches the decrypted one) and print the decrypted pair to the user.

  For example:

  ```
  $ kv read -f db.txt 1
  Enter password: pass
  Key: 1 has value: 3
  ```

  The user wants to retrieve the value of a specific key. The program asks for his master password and he types the word 'pass' that he used before. This way, the file is successfully decrypted and the decrypted (key,value) pair is printed to the console.

- **Read a range** of values from the store that belongs in (key,value) pairs where key1<=key<=key2:

  $ kv range-read -f <filename> key1 key2

  With this command, your program should read the database file and check that the key1 is smaller or equal to key2. Then for each key in that range find the entry that corresponds to the key (decrypt the pair to see if the key matches the decrypted one) and print the decrypted pair to the user.

  For example:

  ```
  $ kv range-read -f 1 3
  Enter password: pass
  Key: 1 has value: 3
  ```

  The user wants to retrieve the values of a range of keys. The program asks for his master password and he types the word 'pass' that he used before. This way, the file is successfully decrypted and the decrypted range of (key,value) pairs are printed to the console in ascending order.

Notes

- You need to submit all the .c and .h files you created, a Makefile that compiles them, a Readme file explaining your implementation or unimplemented parts and a test file that utilizes the implemented functions.
- If you implement more helper functions or macros, explain their functionality in the Readme file.
- This assignment should be implemented using C on Linux-based machines.
- You can use the course's mailing list or the Office Hours for questions. However, read the previous emails first since your question might have already been answered.
- Do not send private messages with questions to the TAs, since other students might have the same question and everyone deserves the answer.
- Do not send code snippets or pieces of your implementation to the mailing list when asking a question.
- Submitted code will be tested for plagiarism using plagiarism-detection software.