ENGG1410: Introductory Programming for Engineers

"Mini Project #2": Encryption/Decryption Utility

Mohamad Abou El Nasr
School of Engineering, University of Guelph
Fall 2024
Special thanks to Prof. Shawki Areibi

Start Date: Week #10 2024
Due Date: Week #12 2024, Demo in class and report upload in Drop Box

1 Objectives:

In the previous Labs you learned several skills that can assist you in developing complex applications and projects, including effective debugging and revision control. In this lab, you will be asked to develop an encryption/decryption utility that can be used as a helpful stand-alone application.

The main objectives of this lab are to:

- Introduce new C programming techniques – File I/O using ASCII files and Command-Line Processing (using main function parameters argc and argv[])
- Deal with strings and Introduce <string.h> for handling strings (for example: using fgets() and file name change using combination of functions from <string.h>)
- Practice creating a small software project.
- Practice working in a programming team, creating a small software project.

2 Introduction

Encryption is a way of scrambling data so that only authorized parties can understand the information. In technical terms, it is the process of converting human-readable plain text to incomprehensible text, also known as cipher text. In simpler terms, encryption takes readable data and alters it to appear random. Encryption requires the use of a cryptographic key: a set of mathematical values that both the sender and the recipient of an encrypted message agree on.

A brute force attack is when an attacker who does not know the decryption key attempts to determine the key by making millions or billions of guesses. Brute force attacks are much faster with modern computers, which is why encryption has to be extremely strong and complex. Most modern encryption methods, coupled with high-quality passwords, are resistant to brute force attacks, although they may become vulnerable to such attacks in the future as computers become more and more powerful. Weak passwords are still susceptible to brute force attacks.

2.1 Types of Encryption

The two main kinds of encryption are symmetric encryption and asymmetric encryption. Asymmetric encryption is also known as public key encryption. In symmetric encryption, there is only one key, and all communicating parties use the same (secret) key for both encryption and decryption. In asymmetric, or public key, encryption, there are two keys: one key is used for encryption, and a different key is used for decryption. The decryption key is kept private (hence the "private key" name), while the encryption key is shared publicly, for anyone to use (hence the "public key" name). Asymmetric encryption is a foundational technology for TLS (often called SSL). Commonly used symmetric encryption algorithms include (a) AES, (b) 3-DES, (c) SNOW. Commonly used asymmetric encryption algorithms include (a) RSA, (b) Elliptic Curve Cryptography.

3 Encryption/Decryption

An encryption algorithm is the method used to transform data into cipher-text. An algorithm will use the encryption key in order to alter the data in a predictable way, so that even though the encrypted data will appear random, it can be turned back into plain-text by using the decryption key.

3.1 Encryption/Decryption: Useful Resources

Here are useful Encryption/Decryption resources:

- Encryption 101 (YouTube)
  Encryption 101
- Cryptography for Beginners)
  Crytpography for Beginners
- Introduction to Encryption
  Introduction to Encryption
- Data Encryption: An Introduction
  Data Encryption: An Introduction

4 Requirements

This mini project has you writing an encrypting / decrypting utility. This utility will take any ASCII file and encrypt it in such a way that its contents are not readable until they are decrypted by the utility.

1. The utility needs to be called cryptoMagic and needs to be written in C
   (a) The utility has 2 command-line switches – they are "-E" for Encrypt and "-D" for Decrypt.
   (b) If none of these switches is specified, then encrypt is assumed
   (c) The utility also takes the name of an ASCII input file to encrypt/decrypt as an argument.
   (d) For example:
      - cryptoMagic -E myFile.txt ← will encrypt the contents of the myFile.txt file
      - cryptoMagic myFile.txt ← will encrypt the contents of the myFile.txt file
      - cryptoMagic -D myFile.crp ← will decrypt the contents of the myFile.crp file
2. When the utility is asked to -E encrypt an ASCII file, it will take the input filename and produce the encrypted file with the same base filename and an .crp file extension
   (a) For example:
   cryptoMagic -E myFile.txt ← will produce an encrypted file called myFile.crp
3. When the utility is asked to -D i.e. decrypt an encrypted file, it will take the input filename and produce the decrypted file with the same base filename and an .txt file extension
   (a) For example:
   cryptoMagic -D myFile.crp ← will produce a decrypted file called myFile.txt
4. It should be noted that the input file can have any file extension. When asked to encrypt, you need to replace the existing file extension (if any) with .crp. Similarly when asked to decrypt, you need to replace the existing file extension (if any) with .txt
   (a) Encrypting always produces a file with a .crp extension, decrypting always produces a file with a .txt extension
   (b) Also, note that you may be asked to encrypt or decrypt a file that has no extension! In this case, you have no existing extension to replace – you only need to append the .txt (if decrypting) and .crp (if encrypting).
5. Each line (up to and including the carriage return (noted as <CR> below)) in the unencrypted ASCII file is guaranteed of being 120 characters of less. While processing the input ASCII file you need to process one line at a time. Continue to process the input file until you reach the end of the file.
   (a) Please note that it is not guaranteed that each line actually ends in a carriage return .... how will you handle that?
6. The encryption scheme is applied to each character in the line:
   (a) If the character is a <tab> (ASCII value 9) then simply transform it into the output character sequence TT.

(b) The carriage return characters are not to be encrypted – they are left as is in the resultant output file (the <CR> in the example below is meant for illustration only – do not output <CR>) NOTE: The term "carriage return" in this document does not apply to any specific ASCII code. It applies to the typical end-of-line character that exists in TEXT files within the Linux OS. You may want to investigate what constitutes a carriage return (i.e. end-of-line character in a TEXT file) in the Linux OS

(c) If the character is not a tab or a carriage return character, then apply the encryption scheme in steps d through f below

(d) Take the ASCII code for the input character and subtract a value of 16 from it. Let's call this resultant value "outChar"

(e) If the resulting outChar value is less than 32, then another step must be taken: outChar = (outChar - 32) + 144

(f) You need to write the ASCII value of the new encrypted character (i.e. outChar) to the destination file as a 2 digit hexadecimal value. Note that this will effectively double the length of the input line in terms of size ... For example – if the input (unencrypted) file is:
Hello There how are you?<CR>
My name is Sean Clarke.<tab>I like software!<CR>
Then the encrypted file is:
38555C5C5F80445855625580585F678051625580695F652F<CR>
3D69805E515D55805963804355515E80335C51625B558ETT39805C595B5580635F56646751625581<CR>

7. Each line (up to an including the carriage return (if it exists)) in the encrypted ASCII file is guaranteed of being 255 characters or less. Remember - while processing the input ASCII file you need to process one line at a time.

8. The decryption scheme is applied to each pair of characters in the input line:

   (a) You need to see if the pair of characters you are processing is the sequence TT – if so, then simply transform this pair of characters into a <tab> character (ASCII value 9) in the output file.

   (b) If the pair of characters is not the sequence TT, then translate the first character of the pair by multiplying its face value by 16. Remember that hex values of A through F take on the face values of 10 through 15. Then add the face value of the second character in the pair. Let's call the resulting value "outChar". For example:

   - Reading the pair of characters "38" from the encrypted file will translate into an outChar value of 56 decimal.
   - Reading the pair of characters "5C" from the encrypted file will translate into an outChar value of 92 decimal.

   (c) Now you need to add 16 to outChar.

   (d) If the resulting outChar value is greater than 127, then another step must be taken: outChar = (outChar - 144) + 32

   (e) The outChar value now contains the decrypted ASCII code for the character that you have just decoded. So take this decrypted character value (i.e. outChar) and write it to the destination file as a character.

(f) The carriage return characters are not to be decrypted – they are left as is in the resultant file. For example – if the input (encrypted) file is:
4458596380555E5362696064595F5E80635358555D55805963806062556464698067555962548E<CR>
39635E87648059642F812F<CR>
Then the decrypted file is:
This encryption scheme is pretty weird. <CR>
Isn't it?!? <CR>

9. It is expected that your cryptoMagic utility has been tested (perhaps by using the above examples) as well as with any other encrypted / decrypted examples you can think of. Don't forget about the extreme / boundary test cases!

10. It is expected that your cryptoMagic utility has been designed using **modular techniques (i.e. the program has been written using functions that you've created).**

5 Deliverables

1. **Lab Preparation File**:
   - Name your file as follows: ENG1410_F24_MiniProject2_Group#_Preparation.pdf
   - The preparation file can be hand written or typed. But has to be neat.
   - The preparation file should consist of the flow chart and pseudo code.
   - Only one submission is allowed.
2. **Lab Final Report File**:
   - Name your file as follows: ENG1410_F22_MiniProject2_Group#_FinalReport.pdf
   - Check the format of the final report below in the next section.
   - Keep the first page only for the title page.
   - Only one submission is allowed.
3. **Zipped Project File**:
   - Name your file as follows: ENG1410_F21_MiniProject2_Group#_Project.zip
   - The project file should have a README file that explains the input and output of your program.
   - The project file should have your source file(s)
4. **DEMO**:
   - You will need to demonstrate to the Teaching Assistant that your program fulfills the requirements.
   - The DEMO will take place in the last week (week 12) during the LAB hours and not outside these hours. Unless the TAs inform you otherwise.

6 Report

Below is the general format of the report required:

1. Title Page:
    - Course #, and Date
    - Lab # and miniproject#
    - Your Group #, and Names
2. **Start a new page** and explain how you implemented your design by providing the following:
    (a) **Problem Statement**,
        i. Briefly describe the problem solved in the lab.
    (b) **Assumptions and Constraints**.
        i. Constraints could be for example no optimization with compilation.
    (c) **How you solved the Problem**,
        i. Flowchart.
        ii. Pseudo Code.
        iii. Block Diagram.
    (d) **System Overview & Justification of Design**
        i. Give an overview of the system to be designed.
        ii. Briefly explain how the system works and reasons behind the design.
3. **Error Analysis**
    (a) Confirm no syntax errors are present.
    (b) Confirm no semantic and logical errors are present.
    (c) Describe any problems with your program.
    (d) If no problems in the final system, describe problems/errors encountered during the development and how they were resolved.