CS 3843 Computer Organization I – Project

Final Due Date: Thu Nov 30, 2023 11:59 pm

NO LATE ASSIGNMENTS ACCEPTED

Objective: Write an assembly language program to implement the specified encryption/decryption algorithm. Your program reads in a message file to encrypt/decrypt, a key file, a user-entered password, and optionally, the number of rounds and/or an output file name: (order is irrelevant).

To encrypt:

```
cryptor.exe -e <input_file> -k <keyfile> -p <password> [-r <#rounds> -o <out_file>]
```

To decrypt:

```
cryptor.exe -d <input_file> -k <keyfile> -p <password> [-r <#rounds> -o <out_file>]
```

When no password is entered, it defaults to "password" for that run of the program. The number of rounds "-r" is an optional parameter and defaults to one. Maximum is three. I provide a key file called "Key.dat" which should be used when executing your program. The default output filename is the input filename with a ".enc" extension if encrypted and a ".dec" extension if decrypted.

I provide a C program for each group. The program implements the mundane tasks such as parsing the command line input, combining the crypto order and then hashing the password. Make sure to modify the following line in "Main.h" for your team: `#define CRYPTO_ORDER "ABCDE\0"` with your team's Crypto_Order which is provided on blackboard in a file named "2023_08_CryptoOrderTeamValues.txt."

The key file is 65537 bytes in length (indexed 0 to 65536) and that length is an integral part of this simple algorithm – do not alter it. The program reads the key file into a global byte array exactly 65537 bytes in length.

The SHA-256 hash function coverts the password into a 32 byte (256 bit) "randomized" array. This data is used to get both the starting index and the hop count for the gKey[] array used by the encryption/decryption algorithm. The first two bytes of the hash are used as the starting index (0 – 65535) and the next two bytes set the hop count (1 to 65536, use a zero to mean 65536). **NOTE**: You will need to use a full 32-bit register or variable to track the current index into the gKey[] array.

Implement all of the following C code functionality using assembly language:

```
// MILESTONE #1    round = 0
// MILESTONE #2    includes MileStone #1, round = 0
// MILESTONE #3    includes Milestone #2, round = 0, 1, or 2

for( round = 0; round < #rounds; round++) {

    Starting_index[round] = gPasswordHash[0+round*4] * 256 + gPasswordHash[1+round*4];

    hop_count [round] = gPasswordHash[2+round*4] * 256 + gPasswordHash[3+round*4];
    if(hop_count == 0) hop_count = 0xFFFF;

    index = Starting_index[round];

    for ( x = 0; x < datalength; x++) {        // Note: datalength is passed in
        data[x] = data[x] ^ gKey[index];

        index = index + hop_count[round];
        if(index ≥ 65537) index = index - 65537;

        // for each data[x]:                 Note: 0x43 == 'C'
            // (#A) code table swap          0x43 → CodeTable[0x43] == 0xC4
            // (#B) nibble rotate out        0xC4 → 0x92  abcd efgh → bcda hefg
            // (#C) reverse bit order        0x92 → 0x49  abcd efgh → hgfe dcba
            // (#D) invert bits 0,2,4,7      0x49 → 0xDC  abcd efgh → XbcX dXbX
            // (#E) rotate 3 bits left       0xDC → 0xE6  abcd efgh → defg habc

    } // end for loop through data
} // end for loop through number of rounds
// save the file, report success
```

The C program provided handles all of the input so you can focus on the encrypt/decrypt. The SHA-256 hashing code is freely available and provided with the code given. Set up the visual studio project as you have for the labs.

NOTES:

1. Each group gets a unique permutation of steps such that one group's encryption will not work with another group's encryption. This is the Crypto Order Code

   Main.h: `#define CRYPTO_ORDER "ABCDE\0"` Each team is assigned a crypto code
2. I have a program that will automatically run your program to grade it.
   a. 4 Levels of execution:
      i. Fails to run (0%)
      ii. Crashes (0% to 50% - I'll grade source code)
      iii. Runs, but incorrect (0% to 80% of points)
      iv. Runs correctly (100% of points)
   b. This will be done for each milestone
3. Each team will have 2 or 3 members
4. The point value of the project is 200 points (that's 20% of your grade for the mathematically challenged)
5. There are 3 coding milestones and then the final report.

6. Definition of LATE:

a. FAILURE to complete a milestone on time will result in ZERO points for that milestone

b. An example of late is a project turned in past the due date / time.

   i. If you finish at 11:50pm and your Internet connection fails due to a thunderstorm resulting in a FAILURE to submit your code by the 12:00am deadline, you get ZERO

   ii. If your submission goes through at 12:01 a.m. when it's due at 12:00 a.m. that is called "LATE" and you will receive ZERO points.

   iii. If your group member fails to turn in the project because they "forgot", "the dog ate it", "hard drive crashed" or any other reason … ZERO points for the group.

      1. Help each other out!
      2. Share copies of the code – back it up! Submit early and modify later.
         a. I will grade latest copy that is not late
         b. Submit one per group – you MUST coordinate
      3. Make sure to double-check when it is due.
      4. Whoever is responsible for turning it in, BE RESPONSIBLE!
      5. Other team members, verify – don't fire and forget!!!

c. I will do one thing with LATE code: I will let you know if it runs and if it produces the correct answer. It will still be ZERO points.

d. I DO ALLOW multiple submissions, so you can submit a partly completed assignment and later follow up with a more recent one. I will grade the most recent that is not **LATE**.

e. FAILURE to strictly follow any of the turn-in directions will result in loss of up to 20 points.

f. Remember: A LATE milestone is worth ZERO points

NOTE #A: MAKE SURE to compile with the Mutli-threaded Debug option so that your program runs on my version of Windows. Check the slides for details. I STRONGLY suggest you submit a shell program to me early that I will run just to ensure you have the settings correct.

NOTE #B: For **milestones #1, #2, and #3**, each **.zip** file should contain **ONLY** the two source code files with your encryption/decryption code and the executable file.

NOTE #C: Failure to name your files correctly costs 10 points each time. Do NOT include extra files. Do NOT zip up the entire project folder – you will lose 10 points!

(30 pts) Milestone #1, Due: 11:59 pm, Sun, Nov 5th, 2023: It runs!

Goal: Working program that accesses the input file data and applies a simple encryption to that data.

The encryption program must loop through the input file and xor every byte with gkey [starting index]. Then, on a separate run of the program, the decryption program must loop through the encrypted file and xor every byte with gkey [starting index]. The resulting output file must match, byte for byte, the original input file.

Note 1: If the encryption works but not the decryption or vice versa, then 20 points earned.

Note 2: I will have my own test input files of different lengths.

Note 3: Use the word "SECRET" for your password for testing. It WILL matter on milestone #1.

Note 4: `starting_index = gPasswordHash[0] * 256 + gPasswordHash[1];`

Name your zip file: "Team_##_CS3843_Project_01.zip" where '##' is your team's number.
Name the exe file: "Team_##_CS3843_Project_01.exe"

(70 pts) Milestone #2, Due: 11:59 p.m. Sun Nov 19th, 2023: Bit manipulation

Goal: Working program that accesses the input file data and correct applies the 5 steps of the encryption/decryption process.

Each team must implement their 5 assigned steps in the correct order (as specified by the Crypto_Order) for both encryption and decryption. KEEP the milestone #1 step.

Name your zip file: "Team_##_CS3843_Project_02.zip" where ## is your team's number.

Name the exe file: "Team_##_CS3843_Project_02.exe"

(70 pts) Milestone #3, Due 11:59 p.m. Thu Nov 30th, 2023: Hopping Around

Goal: Working program that accesses the input file data, correctly applies the 5 steps, and implements the random hop portion of the encryption including the 1 to 3 rounds.

Note: IF milestone #2 is not correct, this portion will not be successful either. So even if you failed to get #2 on time, you still must get it to work correctly in order to get full credit for #3.

Make SURE to replace that first encryption step used in M#01 and M#02.

Name your zip file: "Team_##_CS3843_Project_03.zip" where ## is your team's number.

Name the exe file: "Team_##_CS3843_Project_03.exe"

(30 pts) Final Report, Due: 11:59 p.m. Thu Nov 30th, 2023:: It's done.

Goal: Short write-up on the experience – no more code changes.

Turn in the final report using the template provided. It is a fairly short report, but still worth 20 points.

Name your file: "Team_00_CS3843_Project_FinalReport.docx" where 00 is your team's number.

Notes:

To receive full credit, everyone in a group MUST participate in some definable way on the project. It doesn't have to be an even split nor does everyone need to code. HOWEVER, code similar to the project tends to find its way onto exams, particularly the final exam. If your group completes the project and you are not at least familiar with the code and how it works, you will likely not do well on the final exam.

If you have a group member that is not making an effort to contribute, and you have attempted to get participation, come talk to me and I will talk with them. ALSO, if you have a group member that just wants to do everything (because he/she thinks they can do it faster and better by themselves) and you have made an effort to work with him/her, then come talk to me and we'll get that issue resolved.

This is a GROUP project and part of your education is to learn to work with people with diverse backgrounds and skills. You will need to do it on the job, so learning how to deal with it now is important. If someone is weaker in coding, then ask them do some testing, report writing, double-checking the turn-in, etc.