(200 pts) CS 3843 Computer Organization I – Project

Final Due Date: Wed Dec 5, 2018 6:00pm

NO LATE ASSIGNMENTS ACCEPTED

Write an assembly language program to implement the specified encryption/decryption algorithm. Your program reads in a key file, a message file to encrypt/decrypt, a user-entered password, and as an option, the number of rounds as specified by the following command line: (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 keyfile 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 that implements the mundane tasks such as parsing the command line input and 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.

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 a 4-byte register/variable to track the current index into the gKey[] array.

Implement the following C code functionality using assembly language:

```
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;

      for ( x = 0; x < fileLength(input_file); x++) { // Note: length passed in
            file[x] = file[x] ^ gKey[index];
            index += hop_count[round];
            if(index ≥ 65537) index –= 65537;

            // for each file[x]:
                  // (#A) swap even/odd bits          0xA9 → 0x56
                  // (#B) invert middle 4 bits        0x56 → 0x6A
                  // (#C) swap nibbles                0x6A → 0xA6
                  // (#D) code table swap             0xA6 → CodeTable[0xA6]
                  // (#E) reverse bit order           0xCA → 0x53
      } // end for loop through file
} // 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. The source code files I provide you will work right out of the box.

Notes:

1. The algorithm is slightly different than that of previous semesters.
2. 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"`

   I will assign your team crypto code
3. 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 at each milestone
4. Each team will have 2 or 3 members
5. The point value of the project is 200 points (that's 20% of your grade for the mathematically challenged)
6. There are 3 milestones prior to final turn-in.

7. ***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 points.
      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. So help each other out!
         2. Share copies of the code.
         3. Make frequent backups.
         4. Make sure to double-check when it is due.
         5. Whoever is responsible for turning it in, BE RESPONSIBLE!
         6. Verify it's been turned in.
   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 are welcome to submit a partly completed submission and later follow up with a more recent one.
   e. FAILURE to strictly follow any of the turn-in directions will result in 20% loss of points.
   f. Remember: A LATE milestone is worth ZERO points

NOTE: 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: For each milestone 1 to 3, the .zip file shall include the appropriately named executable file and the C/C++ files you modified.

(30 pts) Milestone #1, Due: 11:59 p.m. Fri Nov 2, 2018: 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 keyfile [starting index]. Then, on a separate run of the program, the decryption program must loop through the encrypted file and xor every byte with keyfile [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 awarded.

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.

Note4: `starting_index = gPasswordHash[0] * 256 + gPasswordHash[1];`

Name your zip file: "Team_00_CS3843_Project_01.zip" where 00 is your team's number.

(60 pts) Milestone #2, Due: 11:59 p.m. Fri Nov 16, 2018: 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_00_CS3843_Project_02.zip" where 00 is your team's number.

(60 pts) Milestone #3, Due 11:59 p.m. Fri Nov 30, 2018: 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 remove that first encryption step in M#01 and M#02. You will be encrypting using the full keyfile at this point making that step unnecessary.

Name your zip file: "Team_00_CS3843_Project_03.zip" where 00 is your team's number.

(50 pts) Final Project, Due: 6:00 p.m. Wed Dec 5, 2018: It's done.

Goal: Working encryption/decryption program.

At this point, the complete program should have already been working so you can grab 20 points for correct turn-in. If it wasn't working before and now it is you can reclaim 20 points from M#03. The other 30 points are independent of whether the program works or not.

Name your zip file: "Team_00_CS3843_Project_Final.zip" where 00 is your team's number.

NOTE: For the milestones, each .zip file should contain the executable file and the source code files that you modified. **Name the .exe file the same as the .zip except for the .exe extension**.

FAILURE to name your files correctly costs 10 points each time.

NOTE: You may email me programs for test using .xz data compression (7 zip does this) since email scanners do not decompress that or upload on blackboard.

Final Deliverables: Due Wed Dec 5, 2018, 6:00 pm

Hard Copies are due at class time, 6:00 p.m. Printing it right before class is NOT recommended as it can easily be LATE – which will cost you 30 points.

1. (10 pts + 20 Point Reclamation) Softcopy: A working project! You can submit softcopy material on USB (which will be returned) or CD/DVD or on blackboard. This should be in a zip file named as such:

   Team_00_CS3843_Project_Final.zip

   00 – Replace with assigned team number

2. The zip file must contain the following:
   a. Executable file, named "**Team_00_CS3843_Project_Final.exe**" that accomplishes the above encryption/decryption function.
   b. Project Report
   c. Commented source code – this should include all of the source code that your team wrote – I do not want the source code for SHA-256.

3. (10 pts) Softcopy: Commented source code files ONLY INCLUDING the code your team wrote and/or modified. Extraneous code will result is loss of points! Include a cover page with class, date, and everyone's name. The comments should explain what you're doing and why and not simply repeat the assembly instruction. You do not need to comment every single line – block comments are preferred.

4. (30 pts) Hardcopy & Softcopy: A report following the template provided.

Notes:
The printouts should be in a font such that they are readable, i.e. don't have such a large font that the majority of the lines carry over.

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 onto 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 can do it faster and better by themselves) and you have made an effort to work with them, 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 let them do some testing, report writing, double-checking the turn-in, etc.