(150 pts) CS 3843 Computer Organization I – Project – Rev #4
Due Thu. Apr 27, 2017

Write an assembly language program to implement the following encryption/decryption algorithm. Your program will read in a key file, a message file to encrypt/decrypt, a user-entered password, and as an option, the number of rounds as in the following command line: (order is irrelevant)

To encrypt:
```
cryptor.exe –e –i <input_file> –k <keyfile> –p <password> [–r <#rounds> –o <out_file>]
```

To decrypt:
```
cryptor.exe –d –i <input_file> –k <keyfile> –p <password> [–r <#rounds> –o <out_file>]
```

If no password is specified, the password will be "password" for that run of the program. The number of rounds "-r" is optional and will default to one with a maximum of three. I will provide a keyfile called "key.dat" and that should be used for testing your program, however you can rename it. The default output filename is the input filename with a ".enc" extension if encrypted and a ".dec" extension if decrypted.

I provide a C shell program for every group to handle some of the mundane tasks such as parsing the command line input and hashing the password.

The keyfile is 65537 bytes in length (indexed 0 to 65536) and that length is an integral part for allowing this simple algorithm to work – do not alter the length. Your program should read that keyfile into a global byte array that is exactly 65537 bytes in length.

The SHA-256 hash will input the password and output 32 bytes of hash data into an array. This data will be used to get both the starting points and the hop counts for the key file. The first two bytes of the hash will be the first starting point (0 – 65535) and the next two bytes will get the first hop count (1 to 65536, use a zero to mean 65536). NOTE: You will need to a 4-byte variable to hold the current index into the key file array.

```
for( round = 0; round < #rounds; round++) {
      Starting_point1[round] = gPasswordHash[0+round*4] * 256 + gPasswordHash[1+round*4];
      hop_count1[round] = gPasswordHash[2+round*4] * 256 + gPasswordHash[3+round*4];
      if(hop_count1[round]  == 0) hop_count1[round]  = 0xFFFF;
      // repeat for starting_point2[round], except 16+round*4, 17+round*4, etc.
      index1 = starting_point1[round]; index2 = starting_point2[round];

      for ( x = 0; x < fileLength(input_file); x++) {
            file[x] = file[x] ^ gKey[index1];
            index1 += hop_count1[round];
            if(index1 ≥ 65537) index1 –= 65537;

            // for each file[x]:
                  // rotate 1 bit to right      0xA5 → 0xD2
                  // swap nibbles               0xD2 → 0x2D
                  // reverse bit order          0x2D → 0xB4
                  // swap half nibbles          0xB4 → 0xE1
                  // rotate 1 bit to left 0xE1 → 0xC3

            file[x] = file[x] ^ gKey[index2];
            index2 += hop_count2[round];
            if(index2 ≥ 65537) index2 –= 65537;
      }
}
// save the file, report success
```

You are NOT required to use the shell program, but your project program must work the same way and produce the same results. The SHA-256 hashing code is freely downloadable and you may incorporate that any way you determine to be best. The source code files I provide you should work in both Windows and Linux, so you may use either one.

I STRONGLY suggest you get pieces of the program to work, one at a time, rather than write all the code and then attempt to debug it. A program that successfully encrypts/decrypts an arbitrary file, even if not all of the steps are done, is better than a non-working program with all the steps attempted.

Deliverables: April 27, 2017

1. A working project! You can submit softcopy material on USB (which will be returned) or CD/DVD.

2. (25 pts) Softcopy: Executable file that accomplishes the above encryption/decryption function.

3. (25 pts) Softcopy: All source code files ready to be compiled. For Windows, provide a project directory. For Linux, an appropriate makefile and directory structure. In either case, I should be able to plug-n-play to compile your code and evaluate your project.
    a. For Linux users: I would strongly suggest that you submit a preliminary set of project files to me long before the due date to make sure I can easily compile and link. This program could simply read the message file and print it out. If it fails we will then have time to determine the issue.
    b. For Windows users: I currently have Visual Studio 2013 installed but can install 2015 if needed. If you have a more recent version, let me know.

4. (25 pts) Softcopy & Hardcopy: A brief report consisting of a cover page, a description of the overall flow of your program and what each function accomplishes, any major issues you had in development, each person's contribution, a brief conclusion, and links to any reference material. Each person in the group should sign the cover page.

5. (75 pts) Hardcopy: Commented source code files NOT including SHA-256 files with a cover page including 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.