CS 3843 Computer Organization I

Project Report

December 5, 2018 6:00 p.m.

Team\_18

Group Members

Caleb Evans – umo850

Bernardo Flores – vic398

By signing this report I affirm that I know and agree with the contents.

Signatures:

Bernardo Flores

Name #1: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Caleb Evans

Name #2: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Objectives:**

1. **Learn to read/write x86 assembly code**
2. **Learn about cryptography**
3. **Learn to follow directions and implement a specific algorithm**
4. **Learn to work in a small group**

**Algorithm Milestone #1**

Milestone 1’s encryption algorithm loops through an input file that contains data. For every byte in this file, the data is operated on by a logical xor with an index of a key file; the index being a starting index that takes the 0th index of a gPasswordHash, multiplies it by 256, and then adds it to the 1st index of gPasswordHash. For guaranteed success in this milestone, the password was to be set as “SECRET”. The decryption algorithm is the inverse of this encryption process.

**Algorithm Milestone #2**

For Milestone 2, the algorithm from Milestone 1 is kept, although while looping through each byte of data of an input file; our teams Crypto Order, ‘CDEBA’, is applied to that byte in order. The byte was to be manipulated in order by C: swapping the lower and upper nibbles of the byte (8-bits), D: swap the byte with a byte located in a CodeTable array at the index of the given byte, E: reverse the order of bits in the byte, B: invert the middle 4 bits of the byte, and A: swap even bits with the odd bits. For the decryption process, the process is inversed as well as the crypto order.

**Algorithm Milestone #3:**

In Milestone 3, 5 steps of our crypto order ‘CDEBA’ are applied for the iteration of each byte in the file. This is nested in a for-loop until the end of the file length based on the starting index given to the initial start of this loop. Inside this loop, the index increments based on a hop count with an index of the rounds entered at input. The process mentioned previously is nested in a for-loop based on the rounds given at input. Most importantly, the variables local to this loop, our starting index, is our gPasswordHash (array containing input password) with an index of [1+the number of rounds\*4] \* 256 + our gPasswordHash with an index of 3+the number of rounds\*4]. Local to this for-loop, the hop count is given the index of our hop count which equals gPasswordHash with an index of [2+the number of rounds\*4] \* 256 + our gPasswordHash with an index of 3+the number of rounds\*4].

**Technical Issues:**

Had a slight issue on decryption for mile 3. This was due to a slight misunderstanding with the index increment step. Was resolved after a good night’s sleep.

**Group Member Contributions:**

Bernardo contributed by creating a Github.com for file sharing, writing source code for steps: D, E of the crypto order, and submitting group zip files to Blackboard.

Caleb contributed by creating source code for both decryption and encryption algorithms on Milestones 1, 2, and 3, as well as debugging any program errors.

**Group Issues and Resolutions:**

None.

**Conclusion:**

A demanding project to enforce assembly programming skills is great!

An encryption decryption program is a great representation of the instruction set for x86.