Programming Assignment 1

Kollin Labowski

800296755

January 30, 2022

Status Sheet:

As of the program's completion on January 29th, it works as specified in the program handout document. There are not any known issues that cause runtime errors or invalid data to be produced as of its completion.

Description of Programming Environment:

The program is intended to be run using the LOUD OVA, specifically with Python version 3.8.10 installed. While it was specifically made to run in this environment, it should work as intended in any Python programming environment with the appropriate version installed. No additional libraries need to be installed for the program to run correctly.

Description of Program:

The program will take an input message and a key (16 characters long) and will encrypt the given message using the given key. The input message must be placed in the input.txt file by the user prior to program execution, and the key should be placed in the key.txt file as well.

The first stage of the encryption algorithm was preprocessing, which included removing all whitespace and punctuation from the input message. Because it was given that all characters in the input message will be uppercase, this part was accomplished by simply omitting any characters from the input message that are not uppercase characters.

The next part of the encryption algorithm required that the newly processed message be encrypted using a Vigenere cipher. This is typically done by lining up each character in the input message with its corresponding character in the key, then looking up the character these two correspond to in a table. In a program, this can be accomplished without the table by first converting each of the characters to their corresponding indices in the alphabet (for example, ‘A’ as index 0, ‘B’ as index 1, and so on), then adding the two characters together, and taking their remainder when dividing by 26 (because this is the length of the alphabet). This number can then be converted back to the correct ASCII value for its character by adding the ASCII value of the lowest index uppercase character, ‘A’.

The next step in the encryption process included partitioning the newly generated ciphertext into 4x4 array structures, filling in any additional remaining space in the final array structure with ‘A’. In Python, this was accomplished by using a list to contain all the 4x4 arrays, then 2 additional levels of lists for each element in the outer list to represent these 2D arrays. The

ciphertext was added into this structure as each level was created, and if the ciphertext characters were exhausted while an array was still being filled, this array was filled with ‘A’.

The next step in the encryption process was to shift the rows in each of these 2D arrays such that the second row in each was shifted left by 1, the third row was shifted left by 2, and the fourth row was shifted left by 3. These could be accomplished using a loop on each row which would perform a left shift operation on that row a set number of times depending on which of the four rows it was on. The actual shift was performed using a temporary variable to track the first element in the row, then have each element an index ahead of another overwrite the index preceding it, and finally overwriting the last element in the row with the first character which was saved in the temporary variable.

The next step was to replace each of the characters with their corresponding ASCII value (represented in hexadecminal for the output), and if the parity of the ASCII value for a character was odd, the MSB of that value would be set to one to make the parity even. This was accomplished using a helper function which determined the parity of an input value, and if the parity was odd, it would return the input integer with the MSB set to one. This was used on every integer ASCII value in the structure, replacing the values with their MSB set when necessary.

The final step in the encryption process was to mix the columns in the same way as in AES encryption. This involved multiplying each column with the circulant MDS matrix using multiplication in the Rijndael Galois field. As suggested in the project document, a helper function was created to perform multiplication in this form, and it was created to follow the specifications in the project document. Similarly, the project document instructions were followed to perform the appropriate calculations on each column of every 4x4 array in the structure. Upon the completion of this step, the results were generated.