COMP 4140 (Fall 2022) Assignment 3

Due date: Thursday November 17, by 11:59 PM (CST). Late assignments not accepted.

This assignment consists of a programming question and some written questions.

The programming part will be handed in using UMLearn while the written part will be handed in using crowdmark.

Porgramming Question (12 Marks)

You will write a complete program to implement the encryption and decryption algorithms as described in class and the FIPS 197 document.

Your program must be written in either C/C++ or Java.

Your implementation only need to handle a single 128-bit (16 byte) plaintext message block and 128-bit key size.

The goal of this assignment is to make sure you understand the how AES works and can implement it.

Please use good programming practices, comment your code appropriately, and ensure that your **name and student number** is provided at the start of each of your source code file(s). Your program should be modular, easy to read, well-documented. Before each routine, provide some comments on what the routine does, its input(s) and output(s). You should also comment parts of code that is non-trivial.

Please get yourself a copy of the FIPS 197 document.

Grading Environment

The grading will be done on the Unix system (aviary.cs.umanitoba.ca). The building and execution of your program will be done from the command-line terminal. You'll need to ensure you have the University VPN set up beforehand if you want access to aviary off-campus.

Make sure you test your program in a Unix environment (best to test it on aviary.cs.umanitoba.ca before submission) and can build and execute from the command-line terminal. If you have a Mac computer, then it already comes with Unix. If you have a PC running Windows OS, you can install the Windows Subsystem for Linux.

Input Files

Your program will take two command-line arguments, the first specifying the file containing the plaintext message to be encrypted, and the second containing the key.

For example

(for c/c++) or

(for Java)

The plaintext and the key files are specified in hexadecimal format (see the provided sample data files).

The key files test1key.txt, test2key.txt, test3key.txt along with the plaintext files test1plaintext.txt, test2plaintext.txt, test3plaintext.txt are provided for testing purposes.

Note: The input plaintext file test1plaintext.txt, key file test1key.txt is the input used in Appendix A.1,B of the FIPS 197 document.

Modularity

As a mininum, you should implement separates routines to perform the following operations (I'm using the names from the FIPS document):

- SubBytes(), InvSubBytes() that performs the s-box substitutions.
- KeyExpansion() that expands the input key into the 11 round keys.
- ShiftRows(), InvShiftRows() that shifts the rows.
- MixColumns(), InvMixColumns() that does the matrix multiplication in $GF(2^8)$.

You should also have a routines for the encryption and decryption algorithms (say encrypt(), decrypt()).

Please use the names provided above to make your code easier to read and grade.

You will probably have other routines in addition to the ones stated above. This is perfectly acceptable.

Mainline

The main() driver function will read in the plaintext message and the key from the files specified on the command line. Then it will do the following:

1. Encrypt the plaintext using the key and output intermediate results (see below) and the final ciphertext.
2. Take the ciphertext generated by the encyption process and decrypt it, outputting intermediate results and the resulting plaintext.

Summary of outputs

As your program runs, it should output (to the console):

- The key and the plaintext message.

- The round keys, one on each line. Each round key should be display as 4 32-bit values (in hexadecimal format).
- During the encryption process:
  - output the plaintext message
  - output the state immediately after returning from each call to MixColumns().
  - At the end of the encryption process, print the ciphertext.
- During the decryption process:
  - output the ciphertext message
  - output the state immediately after returning from each call to InvMixColumns().
  - At the end of the decryption process, print the plaintext message computed.

For specific details regarding what the output should look like, see out1.txt, which is generated from the message given by test1plaintext.txt using the key given by test1key.txt.

Note : The outputs of the keys, plaintexts, ciphertexts and states are given in hexadecimal format.

What to Hand In

- All source code files needed to build your program.
  - If you are using C/C++, a Makefile should also be provided to compile and build your executable.
- A README file explaining how to build and run your program.
- Submit everything in one .zip, .tar, or .tar.gz file.

Other things to consider

If you are writing your program in C/C++, ensure that your code is using portable data types. For example, instead of using an unsigned int variable to hold a 4-byte value, use a uint32_t variable instead. This is because the number of bytes used to store an unisgned int variable may differ across systems. See this link for more information.

As mentioned earlier, your program should be able to be built and executed without an IDE. Instead, it should be built and executed from a Unix command-line terminal on aviary.cs.umanitoba.ca.

Files Provided to Support this Assignment

1. sbox.txt, inv_sbox.txt - these two files contain the sbox and the inverse sbox. Alternatively, I've provided sbox.h which contains both as arrays. You can use either version.
2. Test files:
   - key files test1key.txt, test2key.txt, test3key.txt.
   - plaintext message files test1plaintext.txt, test2plaintext.txt, test3plaintext.txt.
3. Sample Output file: out1.txt which is the output when key file test1key.txt and plaintext message file test1plaintext.txt is used as input to the program.

Academic Integrity

Code that you submit for this assignment must be entirely written by you (except for the code provided to support this assignment). Copying code from the Internet or from other students will be considered academic dishonesty.

Questions and Clarifications

Please use the UMLearn forums for asking questions and seeking clarifications.

Written Questions

Written questions must be handed in using Crowdmark.

1. (3 Marks) Suppose n is a positive integer (the security parameter). Consider the following MAC for messages of length n using a psuedorandom function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ defined as follows: On input k (the key) and message $m = m_1 m_2 ... m_n$, the algorithm $\mathsf{Mac}_k(\cdot)$ is defined by

$$\mathsf{Mac}_k(m) = F_k(1||m_1 m_2 ... m_{n-1}) \oplus F_k(0||m_2 m_3 ... m_n).$$

The algorithm Vrfy can be defined using canonical verification. Is this a secure MAC? Prove your answer.

2. (3 Marks) In the basic CBC-MAC construction, some simple changes can make the construction not secure (that is, is not a secure MAC).

Show that the following changes to the construction makes it not secure: Outputting all the t_i's instead of just the final one.

Hint: Consider messages of length $2n$, where n is the security parameter.

Optional Questions (Do not hand in)

Question:

For security parameter n, consider the following MAC for messages of length $2n-2$ using a psuedorandom function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ defined as follows: On input a message $m_0||m_1$ (with $|m_0| = |m_1| = n-1$) and a key $k \in \{0,1\}^n$, the algorithm $\mathsf{Mac}_k(\cdot)$ is defined by $\mathsf{Mac}_k(m_0||m_1) = F_k(1||m_0)||F_k(0||m_1)$. The algorithm Vrfy is defined in the natural way.

Is this a secure MAC? Prove your answer.

Question:

Let n be the security parameter. Consider using a block cipher (that is, an pseudorandom permutation that is length-preserving) $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ in CBC-mode. Show that this is not CCA-secure.

Question:

Provide formal definitions for second-preimage resistance and preimage resistance. Then:

- Prove that any hash function that is collision resistant is second-preimage resistant.
- Prove that if a compression function mapping a $2n$-bit inputs to n-bit outputs is second-preimage resistant, then it is preimage resistant.

Question

Suppose (Gen, H) is a collision-resistant hash function. Is (Gen, H_1) defined by $H_1^s(x) = H^s(H^s(x))$ necessarily collision resistant?