ECE 2420 Programming Exercise #2

(AES Library Use)

# Overview

This programming exercise is a follow on to the initial cryptography interface programming exercise. This assignment builds upon the last exercise to create an encryption tool featuring a robust transform called the Advanced Encryption Standard (AES).

# First Cryptographic Algorithm

The AES standard is a NIST standard. It is a symmetric cypher, (i.e. the same key is used for encryption and decryption, see pg. 140 of the Cormen text). The NSA has approved the algorithms use to protect U.S. government data classified at levels up to and including TOP SECRET. When properly implemented, this is a strong cypher capable of protecting your sensitive data. More information can be found about the AES algorithm at: <https://en.wikipedia.org/wiki/Advanced_Encryption_Standard>.

At this point, you need not understand the details of this cryptographic transform. Rather, you are tasked with integrating a well-known implementation of the AES algorithm into the interface developed in previous assignment. For this exercise you will use a crypto library called mcrypt. There are many examples of how this code can be used found on-line. The man pages for this library are also quite useful and can be displayed with the command: man libmcrypt.

# Programming Concepts

To correctly complete this programming task, the student must possess an understanding of basic Linux system operation, IO streams, inheritance, polymorphism, interfaces, abstract classes, dynamic linking, functional objects/lambdas, and basic memory management.

The challenging aspects of this integration are not cryptographic (We’ll save that for the RSA PEX). Rather, most of the complexity revolves around the streaming of data into and out of the algorithm, supporting data that must be in complete block sizes, and linking against the mcrypt dynamic library.

# System Requirements

This design is intended to be quite flexible not only in the transforms that are supported, but in the ways that the transforms are used. It would be equally easy to use this code to transfer data over a socket as it would be to read/write files on a disk. The system is designed to support encryption, decryption, and key management in a single simple library. Note that these requirements are identical to the last PEX; this is by design.

A typical use of this library could be as follows (secure data file storage):

1. The user instantiates a Crypto object. During this instantiation, the user is required to provide callbacks. The purpose of these callbacks is to provide instruction to the library on what do with data that has been successfully transformed, (encrypted or decrypted).
2. The static factory function parameters dictate what cryptographic transform shall be used.
3. The library will be used to generate keys for this action
4. The keys may be stored for use on a later execution, (e.g. to decrypt the file)
5. The file to be encrypted/secured is opened by the user and read into a memory buffer.
6. The contents of the memory buffer are then written to the encryption engine.
7. The encryption engine will transform the data appropriately. It may then do any of the following:
	1. Call the callback in which the user program can write the obfuscated data to disk.
	2. Call the associated callback with only a portion of the encrypted data, (some encryption algorithms need to handle data in specific block sizes.)
	3. Make no callbacks and buffer the data until a certain amount has been collected
8. Once all of the data has been written, one additional write shall be made to the engine where the length of the data is specified as zero. This will instruct the encryption engine to flush all buffers completely. Note that some algorithms may need padding to fill a complete block.
9. The entire process is executed in reverse to decrypt the file when needed.

Other requirements:

1. The system shall utilize the interface header file provided (Crypto.hpp) verbatim. This will allow the instructor to execute test/grading code using your implementation. This header file is located at: <http://classes.ece.usu.edu/2420/>. It is very well documented in-line and students are encouraged to study it closely.
2. The student may construct the implementation of the provided header in any way they choose.
3. Additional classes/files may be used as the student sees fit.
4. Students should consider building a new class derived from the provided interface for each new transform supported.
5. The system shall support transforms in which the size of the plain text and cipher text streams are not the same.
6. The system shall accept all data provided to an encrypt/decrypt function call.
7. The system may buffer and return encrypted/decrypted data via any number of callback executions.
8. The system may execute callbacks from the encrypt/decrypt function, (i.e. it is possible to see a callback before the encrypt/decrypt call completes)
9. The user callback shall consume all data provided to it before returning to the caller.
10. The system shall support shifting binary data, not just alpha-numeric strings

# Turn-in Procedures

Turn in all source code via canvas by 11:59p.m. on <DUE DATE>

Grading Rubric

(ECE 2420 PEX2)

|  |  |  |
| --- | --- | --- |
| Requirement / Criteria | Available Points | Student’s Score |
| Compiles and links correctly; uses provided interface verbatim.Links against the libmcrypt shared library.  | 10 |  |
| Static factory function creates correct object | 10 |  |
| Key management operates correctly | 10 |  |
| Writes smaller than block size are correctly encrypted | 20 |  |
| Writes larger than block size are correctly encrypted | 20 |  |
| Byte streams not evenly divisible by block size are correct | 20 |  |
| Encrypted/decrypted complex file checked with MD5 sum | 10 |  |
| Total | **100** |  |