ECE 2420 Programming Exercise #1 (Cryptography Interface)

Overview

The purpose of this exercise is to introduce several fundamental programming concepts through the implementation/use of various encryption algorithms/transforms. Students will first design a front end interface which provides flexibility to support a variety of different algorithms. These algorithms will range from a contrived shift cipher to an AES-256 block cipher (a standard currently approved to secure TOP SECRET government data). Finally, the student will be tasked with implementing an RSA asymmetric encryption system and integrating that into the common project/interface.

First Cryptographic Algorithm

The first cryptographic algorithm to be implemented is a trivial shift cipher, (also known as a Caesar Cipher due to its use by the famous Roman emperor Julius Caesar). The details of this algorithm are available here: https://en.wikipedia.org/wiki/Caesar_cipher. While this transform provides virtually no confidentiality to the user, it will serve as an easily debugged transform to ensure that the interface functions as expected and can serve as a robust home for much more complicated transforms. Unlike the traditional cipher, the produced implementation must work on all possible data streams, not just alpha numeric strings.

Programming Concepts

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

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.

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:
    a. Call the callback in which the user program can write the obfuscated data to disk.
    b. Call the associated callback with only a portion of the encrypted data, (some encryption algorithms need to handle data in specific block sizes.)
    c. 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 PEX1)

Requirement / Criteria	**Available Points**	**Student's Score**
Compiles and links correctly; uses provided interface verbatim.	10	
Correctly implements constructor which accepts functional callbacks.	10	
Static factory method builds correct derived type; no underlying derived class implementation are exposed.	10	
Correctly implements ability to construct, expose, set, and destroy keys for chosen algorithm (eNONE and eCAESAR in this portion). Considers corner cases and handles them correctly.	20	
Correctly implement the eNONE and eCAESAR cipher correctly. All input data is returned via a callback. The result of an encryption/decryption cycle is identical to original input. All possible byte streams are supported. Handles corner cases correctly.	50	
Total	**100**	