Assignment 2
CS-452
Spring 2018

Due Date: 4/13/2018 at 11:59 pm (No extensions)
You may work in groups of 5

Goals:

1. **Design** an elegant solution for encrypting multiple blocks of text using modern block ciphers.
2. **Utilize** DES and AES block ciphers in order to implement file encryption utility.
3. **Experiment** with `openssl`: a powerful cryptographic library widely used in the real-world.
4. **Appreciate** the challenges of developing cryptographic solutions.

Overview

In this assignment you will gain hands-on experience with DES and AES encryption covered in class. You will do so by utilizing openssl library, a powerful cryptographic library widely used in the real-world in applications (e.g., HTTPs, VPN, etc.), to implement a file encryption/decrypting utility.

Note: You will not be implementing your own DES and AES algorithms. Instead, you will be using the DES and AES encryption functions provided by the `openssl` library. **you may work in groups of 6**

The sections that follow give the specifics.

Block Cipher File Encryptors

File `assig2skeleton.zip` provides the skeleton codes for DES and AES classes. Specifically, it includes the following files:

- `CipherInterface.h` and `CipherInterface.cpp`: contains functions defining the DES and AES encryption classes described below. **You must not modify this file**.
- `DES.h` and `DES.cpp` files: contain functions implementing DES encryption. It is your job to fill the bodies of these functions. Guidelines are provided in the comments found in these files as well as in the later sections.
- `AES.h` and `AES.cpp` files: contain functions implementing AES encryption. It is your job to fill the bodies of these functions. Guidelines are provided in the comments found in these files as well as in the later sections.
- `cipher.cpp`: the driver program.

- `mydes.cpp:` a program illustrating basic encryption and decryption of a single text block using DES.
- `myaes.cpp:` a program illustrating basic encryption and decryption of a single text block using AES.
- `Makefile:` the file used to build the encryptor program as well as the `mydes.cpp` and `myaes.cpp` demos.

Your finished program shall be called `cipher`, shall integrate both DES and AES encryption functionalities and, shall be executed as:

```
./cipher <CIPHER NAME> <KEY> <ENC/DEC> <INPUT FILE> <OUTPUT FILE>
```

Each parameter is defined as follows:

- `CIPHER NAME:` The name of the cipher:
  - DES: indicates the DES cipher
  - AES: indicates the AES cipher
- `KEY:` the encryption key to use (must be 16 characters representing a 64-bit hexadecimal number for DES and 128-bit number for AES)
- `ENC/DEC:` whether to encrypt or decrypt, respectively
- `INPUT FILE:` the file from which to read the input
- `OUTPUT FILE:` the file to which to write the output

When invoked, the program shall read the input file block by block, use the functions in DES and AES classes in order to encrypt/decrypt each block using the specified key and finally write the decrypted blocks to the specified output file. The size of the block for DES is 64 bits and for AES 128 bits (in this assignment).

Your program shall be tested on very large as well as small files.

For example: `./cipher DES "0123456789abcdef" ENC in.txt out.txt` will read the contents of file `in.txt`, encrypt them using the DES cipher and key `0123456789abcdef`, and shall write the encrypted contents to `out.txt`.

You must use C++, Java, or Python. The skeleton was provided for C++. Your implementation must conform to the CipherInterface class.

The sections that follow describe the specifics of DES and AES classes and your tasks.

DES File Encryptor

The functions in the `DES` class are as follows:

- `bool setKey(const unsigned char*& key)`: takes a 16-character string, `key`, where each character corresponds to a hexadecimal digit in the DES key, and converts each pair of characters into a hexadecimal byte: for example pair `"ab"` is converted into a byte with value `0xab`. Each byte is then stored in the array `des_key` (see `DES.h`). This is necessary, because the `openssl` library requires the key to be an 8-byte array, but the key specified, at e.g. the command line, is usually a string. Finally, `des_key` array is used to initialize the `des_key_schedule key` structure containing the actual key used for DES encryption and decryption (declared in `DES.h`). This is quite possibly the most complex function in this part of the assignment. Good news: it has already been done for you (just trying to help...). Hence, no need to worry about implementing this part. You are, however, highly encouraged to understand how it works.

  If the `key` is a 16-character string containing only hexadecimal digits (numbers 0-9 and letters a-f), the function initializes `this->key` in `DES.h` and returns true. Otherwise, the key is considered invalid, and the function returns false.

- `unsigned char* encrypt(const unsigned char*& plaintext)`: this function encrypts an 8-character (i.e. 64-bit) string, `plaintext`, using DES and key `key` (in `DES.h`), and returns the pointer to the resulting ciphertext string. The body of this function contains comments explaining what you need to do.

- `unsigned char* decrypt(const unsigned char*& ciphertext)`: decrypts an 8-character (i.e. 64-bit) string, `ciphertext`, using DES and key `key`, and returns the pointer to the resulting plaintext. The body of this function contains comments explaining what you need to do.

- `DES_LONG ctol(unsigned char *c)`: converts a 4-character array, `c`, into a long integer. `DES_LONG` type is basically equivalent to C/C++ `long`.

- `void ltoc(DES_LONG l, unsigned char *c)`: Takes a long integer and converts it into an array of 4 characters (i.e. reverses the effects of `ctol()`).

Your tasks are as follows:

1. Fill in the bodies of `encrypt()` and `decrypt()` functions in `DES.cpp`. The bodies of these functions (`in DES.cpp`) contain comments explaining the logic of what you must do. The logic is similar for both functions. The steps can be summarized as follows:

   (a) Declare an array of 2 long integers e.g. long textArr[2];

   (b) Convert the first 4 bytes of `plaintext` into a long integer, using `ctol()`. Store the result in `textArr[0]`. Next, convert the last 4 bytes of `cstrText` into a long integer. Store the result in `textArr[1]`.

   (c) Call DES library function `des_encrypt1(textArr,key,ENC/DEC)`. This function will encrypt/decrypt the contents of `textArr` (based on the value of the last flag). The result will replace the original contents of `textArr`. Please see the sample codes demoed in class for more details about how to use this function.

   (d) Declare an array of 8 characters e.g. `unsigned char txtText[8]`. Use `ltoc()` to convert `textArr[0]` into 4 characters, and store them in the first 4 characters of `txtText`. Next, use `ltoc()` to convert `textArr[1]` to 4 characters, and store them in the last 4 characters of `txtText`.

(e) Return the resulting string.

2. Write a driver program (i.e. `cipher.cpp`) to use your DES class in order to encrypt/decrypt files. Please recall: DES processes text in units of 64-bit blocks (that is, 8 characters at a time). Because the last block of the file may be less than 8 characters, you will need to pad the unused bytes within a block with NULLs (i.e. 0's). **Your program will be tested on both small and large files.**

AES File Encryptor

The functions in the `AES` class are described below:

- `setKey(unsigned char* key)`: this function sets the key to be used for encryption/decryption. Please be aware that the AES implementation of openssl uses different functions to set the key for encryption and decryption. That is, for encryption is uses function `AES_set_encrypt_key(aes_key, 128, &enc_key)` and for decryption it uses function `AES_set_decrypt_key(aes_key, 128, &dec_key)`, respectively where:
  - aes_key: a 16-byte array representing the encryption/decryption key.
  - `128`: the key size (recall, AES supports 128-, 192-, and 256-bit key sizes; we will be using a 128-bit key).
  - `enc_key and dec_key`: an object of type `AES_KEY` representing the encryption/decryption key to use. The functions initialize this object using the user-supplied `aes_key` array.
- `encrypt(unsigned char* plainText)`: uses the `AES_ecb_encrypt(enc_out, dec_out, &enc_key, AES_ENCRYPT` function to encrypt a 128-bit (i.e., 16-byte) block text and return a pointer to the encrypted text. The parameters are as follows:
  - `enc_out`: the buffer that will store the ciphertext
  - `enc_key`: the key of type `AES_KEY` to use for encryption
  - `AES_ENCRYPT`: a directive telling the function to perform encryption
- `encrypt(unsigned char* cipherText`: uses the `AES_ecb_encrypt(dec_out, enc_out, &dec_key, AES_DECRYPT` function to decrypt a 128-bit (i.e., 16-byte) block text and return a pointer to the decrypted text. The parameters are as follows:
  - `dec_out`: the buffer that will store the decrypted plaintext
  - `enc_out`: the buffer that stores the original ciphertext
  - `dec_key`: the key of type `AES_KEY` to use for decryption

Your tasks are as follows:

- Follow the guidelines in the skeleton in order to fill in all three functions.

- Special note about `the setKey()`: Again, this function uses `AES_set_encrypt_key()` and `AES_set_decrypt_key()` to set keys for encryption and decryption, respectively. Hence, you need to tell the function whether to encrypt or decrypt ***WITHOUT MODIFYING THE FUNCTION INTERFACE***. Since the AES key is a 16-byte array, you can pass in a 17-byte array where if the first byte is `0x00`, the function will use `AES_set_encrypt_key()` and the remaining 16 bytes of the array to set the key. Otherwise, it will use `AES_set_decrypt_key()` and the remaining bytes of the array in order to set the key.

COMPILING YOUR PROGRAM

Your programs must be implemented using C++, Java, or Python, and must compile and run on the Topaz1 server. Please follow the following steps in order to connect to the Topaz1 server. To access the Topaz1 server, please ask the instructor for credentials. Once the credentials are obtained, you can use the following process to connect:

1. Connect to the Topaz1 server: `ssh <your provided user name>@topaz1.ecs.fullerton.edu`.

   E.g. `ssh mgofman@topaz1.ecs.fullerton.edu`.

2. Enter your Topaz server credentials.

You must also include a `Makefile` which compiles all of your code when the user types `make` at the command line. Simply type `make` at the terminal in order to compile the program.
If you are not sure how to write or modify the `makefile` please check the following resources.

- C++ make files: http://www.delorie.com/djgpp/doc/ug/larger/makefiles.html
- Java make files: http://www.cs.swarthmore.edu/ newhall/unixhelp/javamakefiles.html
- Google "writing Makefiles"
- Ask the instructor (don't be afraid!)

EXTRA CREDIT:

Extend your program to perform DES and AES encryption in CBC and CFB modes. You cannot use the CBC and CFB library functions provided by openssl.

SUBMISSION GUIDELINES:

- This assignment may be completed using C++, Java, or Python.
- Please hand in your source code electronically (do not submit .o or executable code) through **TITANIUM**. You must make sure that this code compiles and runs correctly.
- **Only one person within each group should submit.**
- Write a README file (text file, do not submit a .doc file) which contains
  - Names and email addresses of all partners.

  - The programming language you use (e.g. C++, Java, or Python)
  - How to execute your program.
  - Whether you implemented the extra credit.
  - Anything special about your submission that we should take note of.
- Place all your files under one directory with a unique name (such as `p1-[userid]` for assignment 1, e.g. `p1-mgofman1`).
- Tar the contents of this directory using the following command. `tar cvf [directory_name].tar [directory_name]` E.g. `tar -cvf p1-mgofman1.tar p1-mgofman1/`
- Use TITANIUM to upload the tared file you created above.

Grading guideline:

- Program compiles: 5'
- Correct DES cipher: 45'
- Correct AES 45'
- README file included: 5'
- BONUS: 20 points
- Late submissions shall be penalized 10%. No assignments shall be accepted after 24 hours.

Academic Honesty:

Academic Honesty: All forms of cheating shall be treated with utmost seriousness. You may discuss the problems with other students, however, you must write your **OWN codes and solutions**. Discussing solutions to the problem is **NOT** acceptable (unless specified otherwise). Copying an assignment from another student or allowing another student to copy your work **may lead to an automatic F for this course**. Moss shall be used to detect plagiarism in programming assignments. If you have any questions about whether an act of collaboration may be treated as academic dishonesty, please consult the instructor before you collaborate. Details posted at http://www.fullerton.edu/senate/documents/PDF/300/UPS300-021.pdf.