Assignment 2
CS-433
Spring 2014

Due Date: 4/28/2014 at 11:59 pm (No extensions)

You may work in groups of 5

Goals:

1. To gain hands on experience using DES library.
2. To help acquire skills useful for the final project.
3. To prepare for the final exam.

Overview

In this assignment you shall extend your assignment 1 program to encrypt/decrypt files using DES and RSA. **You shall not be implementing your own DES and RSA algorithms**. Instead, you shall be using the DES and RSA encryption functions provided by the openssl library. **you may work in groups of 5**

DES File Encrypter

File `assig2skeleton.zip` provides the skeleton codes. The skeleton codes comprise the `CipherInterface` class as given in Assignment 1, Makefile, and, most importantly, the `DES` class. The `DES` class, similar to the classical ciphers you implemented in Assignment 1, also implements the `CipherInterface` interface. The job of `DES` class is to encrypt/decrypt data using the DES algorithm. The class contains 3 functions `setKey()`, `encrypt()`, and `decrypt()`. The functions perform the following tasks:

- `bool setKey(const string& 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 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.

- `string encrypt(const string& plaintext)`: this function encrypts an 8-character (i.e. 64-bit) string, `plaintext`, using DES and key `key` (in `DES.h`), and returns the resulting cipher text string. The body of this function contains comments explaining what you need to do.
- `string decrypt(const string& ciphertext)`: decrypts an 8-character (i.e. 64-bit) string, `ciphertext`, using DES and key `key`, and returns 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 C++ string (i.e. `plaintext` or `ciphertext`) into a cstring. E.g. `const char* cstrText = plaintext.c_str()`.
   (c) Convert the first 4 bytes of `cstrText` 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]`.
   (d) 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.
   (e) 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`.
   (f) Convert txtText into a C++ string, and return the resulting string.
2. Extend your 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.**

Your driver program shall be called `cipher` and shall be executed as:

```
./cipher DES <KEY> <ENC/DEC> <INPUTFILE> <OUTPUT FILE>
```

where

- `KEY`: the encryption key to use (must be 16 characters representing a 64-bit hexadecimal number).
- `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.

The program shall then read the input file, encrypt/decrypt the contents using the specified key and, finally, write the output to the specified output file. **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 Sample files have been provided for C++. Each cipher shall be implemented as a separate class. All classes shall implement a common interface defined in `CipherInterface.h` (see sample codes and description below). This class **shall not** be modified. You shall then write a driver program, called `cipher`, that shall use your classes. `CipherInterface` class defines the following abstract methods:

RSA FILE ENCRYPTER

Implement the methods of class `RSA_433`. Like all classes in this assignment, `RSA_433` also inherits from the `CipherInterface` class. Use the RSA algorithm implemented by the openssl library, for encryption and decryption (openssl, if you are using C++). Please note the following:

- The algorithm name to be specified at the command line is: `RSA`.
- Openssl library expects the RSA public keys to be stored in separate, specially formatted `.pem` files. The sample codes include `pubKey.pem` and `privKey.pem` files which contain sample public and private key, respectively. If you want to, you can also use the following sequence of commands to generate your own keys:

  ```
  openssl genrsa -out privKey.pem 2048
  openssl rsa -in privKey.pem -outform PEM -pubout -out pubKey.pem.
  ```

  At the command line, when specifying the RSA algorithm, for a key, you will need to specify the name of the file containing the key to be used for encryption/decryption, instead of the actual key.
- File `RSA_specific.cpp` in the skeleton codes contains sample code on how to encrypt using RSA in openssl (this includes how to read the keys stored in `pem` files).

COMPILING YOUR PROGRAM

Your programs must be implemented using C++, Java, or Python, and must compile and run on the Lisp server. Please follow the following steps in order to connect to the Lisp server or Topaz1 server. To access the Lisp server:

1. Connect to the Titan server: `ssh <your user name>@ecs.fullerton.edu`.
   E.g. `ssh mgofman@ecs.fullerton.edu`.

2. From the Titan server console: ssh `<your user name>@lisp`.
   E.g. `ssh mgofman@lisp`

3. Enter your Titan server credentials (both Lisp and Titan servers have the same home directory and require the same login credentials).

If you would like access to the Topaz1 server, please ask the instructor.
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 encryption 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 RSA 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.