OpenSSL Blowfish

1 Introduction

Blowfish is a symmetric-key block cipher that provides good encryption rate with no disclosed effective cryptanalysis. Blowfish uses 64-bit blocks and typically 128-bit keys; you can find a description of the Blowfish algorithm at `https://www.schneier.com/academic/blowfish/` and `http://www.counterpane.com/blowfish.html`. OpenSSL is a community-maintained general-purpose cryptography and secure communication library, arguably the most popular in the world. In this assignment, you will use OpenSSL's libcrypto to implement the Blow-fish block cipher. You can find a current description of all the OpenSSL func-tions at `https://www.openssl.org/docs/man1.1.1/man3/`, but we will specifi-cally be using the `BF cbc encrypt`, `BF ecb encrypt`, and `BF set key` functions, which are described at `https://www.openssl.org/docs/man1.1.1/man3/BF_encrypt.html`.

2 Directions

2.1 CBC-mode Encryption/Decryption

Using OpenSSL version 1.1.1k, implement the Blowfish block cipher for encryption and decryption using CBC mode of encryption in **fscrypt.c**. The two functions you will implement have the following signatures:

```
void *fs_encrypt(void *plaintext, int bufsize, char *
    ↪ keystr, int *resultlen);
void *fs_decrypt(void *ciphertext, int bufsize, char
    ↪ *keystr, int *resultlen);
```

For an additional **20 bonus points**, implement the following functions in **fscrypt2.c** with the same functionality as above. However this time, only use `BF_cbc_encrypt` and `BF_set_key`. Note that you will have to modify the provided files (see below).

```
void *fs_encrypt2(void *plaintext, int bufsize, char
    ↪ *keystr, int *resultlen);
void *fs_decrypt2(void *ciphertext, int bufsize, char
    ↪  *keystr, int *resultlen);
```

You will be provided with three files:

- **fscrypt.h**, which contains a block size definition, and the two function prototypes.
- **main.c**, which contains a driver program.
- **Makefile** to compile the driver executable(s).

2.2 Assumptions

We will make the following design decisions and assumptions:

- Use CBC mode of encryption and decryption.
- Pad the buffer with `null` bytes so that the input is always a multiple of the block size.
- The initialization vector must be comprised of all `null` characters.
- Both functions must allocate the result buffer of at least the required size (using `malloc()` or `new`).
- Both functions also return the number of valid bytes in the result buffer pointed to by `resultlen`.
- The application "caller" code is responsible for subsequently freeing the buffer pointed to by `plaintext` or `ciphertext`, but not to free any of the "callee" locally-allocated buffers. We will check your code for memory leaks using Valgrind.

3 Examples

```
matthew@remote:~/cs458/project2$ make
clang -g -Wall -O2  fscrypt.c main.c -o exec  -lcrypto
clang -g -Wall -O2  fscrypt2.c main2.c -o exec2  -lcrypto

matthew@remote:~/cs458/project2$ ./exec "hello world" "top
    ↪ secret"
length after encryption =  16
ciphertext = 3de8c0996489214b412cb350ecdd6fbc
plaintext = hello world

matthew@remote:~/cs458/project2$ ./exec2 "hello world" "top
    ↪ secret"
length after encryption =  16
ciphertext = 3de8c0996489214b412cb350ecdd6fbc
plaintext = hello world
```

4 Optional: Installing OpenSSL

You may desire to have a local installation of OpenSSL in your home directory or on a personally owned machine. In that case, perform the following steps on Remote in your home directory:

```
$ wget https://www.openssl.org/source/old/1.1.1/openssl-1.1.1
    ↪ k.tar.gz
$ tar -xvzf ./openssl-1.1.1k.tar.gz
$ cd openssl-1.1.1k/
$ mkdir install
$ ./config --prefix=$(pwd)/install
$ make && make test TESTS=test_bf && make install
```

Installation is similar for MacOS, Windows, and other Linux distributions, see the **INSTALL** file included with **openssl-1.1.1k.tar.gz**. This, however, is left as an exercise to the reader.

5 Submitting Work

In addition to your file implementation, use a **README.md** file to document how to build the executable(s) **exec** and **exec2**, and how to execute the programs from the command line interface.

Include at minimum:

- Instructions to build your executable for compiled languages using the Makefile.
- Command line interface usage instructions for your implementation of **exec** and **exec2**.
- Acknowledgements of assistance you received and citations for works to which you referred.

Note that we will be building and testing your program on Remote, linking against the system-wide installation of libcrypto and including its headers. If you modified the Makefile for a local installation (see below), ensure you restore the Makefile as appropriate.

Place all your files under one directory with the unique name `p2-<userid>`, where "userid" is your PODS username. Archive the contents of this directory – do not include executable binaries or intermediate files – using the following command:

```
tar -cvf <directory_name>.tar <directory_name>
```

For example,

```
tar -cvf p2-ghyan.tar p2-ghyan
```