CS5750 Programming Assignment #3: `sget` **and** `sgetd`

Due Date: *Friday, April 19, 2019 @ 5pm*

In this project, you will implementing the functionality of `get` and `getd` using encrypted communication. Rather than sending a message in plain text, the two processes will use encrypted communication.

Requirements

There are two subjects: USER and OWNER. OWNER is a subject that owns a set of files protected by access-control lists. USER is a subject that wishes to access OWNER's files. Here is a scenario illustrating how this system works.

1. The daemon `sgetd` listens for a connection from `sget`. The purpose of `sgetd` is to read a file, encrypt the file, and send the encrypted contents of that file to `sget` via a socket.
2. USER runs the binary `sget` just as in Program 2. However, there is some initial communication that occurs to allow the two programs to encrypt communication. This is outlined below.

The binary `sget` communicates with `sgetd` through sockets using encryption. The daemon `sgetd` communicates with `sget` and reads files owned by `owner`. `sgetd` relies on `sget` to write files owned by USER. `sget` has no direct access to OWNER's files.

Protocol.

USER attempts to get a file by executing the command

```
sget <source> <destination>
```

Below is a description of how communication between SGET and SGETD is established.

1. `sget` sends a request to establish a session to `sgetd` with a **Type 0** message as in Program 2. However, the message now contains USER's public key e.
2. `sgetd` stores USER's public key.
3. `sgetd` sends a **Type 1** message to `sgetd` that contains a 128-byte session id and 128-bit session key for the Blowfish algorithm, k. The **Type1** message is encrypted with USER's' public key e.
4. Follow the rest of the protocol from Program 2 except all messages between `sget` and `sgetd` are encrypted with the blowfish algorithm using k.

In this assignment, `sget` will follow the protocol faithfully. You do not need to validate the input.

New Type 0 Message Format. A **Type 0** message is the same as in program 2 except with the addition of a 2048-bit public RSA key. The C definition for a **Type 0** message is

```
typedef struct _type0 {
   Header header;
   unsigned int dnLength;
   char distinguishedName[33];
   char publicKey[256];
} MessageType0;
```

New Type 1 Message Format. A **Type 1** message is the same as in program 2 except with the addition of a 16-byte symmetric blowfish key. This message is encrypted as described above. The C definition for a **Type 1** message is

```
typedef struct _type1 {
  Header header;
  unsigned int sidLength;
  char sessionId[129];
  char symmetricKey[16];
} MessageType1;
```

Openssl Encryption must be done using `openssl`. To install `openssl` on Ubuntu 18.04 execute the command

```
sudo apt install libssl-device
```

To compile and link a file that uses `openssl`'s cryptographic library, you will need to link against the `openssl` library. For example, if I were to compile and link `symEncDec.c` after I installed `openssl`, I would use the command

```
 gcc -o sym symEncDec.c -lcrypto
```

Miscellaneous. The project must be coded in C and will be tested under Ubuntu 18.04.

Generating RSA Keys For this assignment you will generate the RSA keys using the command `openssl`. To generate the keys, use the following:

```
  openssl genrsa -out private.pem 2048
```

To extract the public key from `private.pem`, use the following:

```
  openssl rsa -in private.pem -outform PEM -pubout -out public.pem
```

The file `private.pem` will contain the RSA private key. The file `public.pem` will contain the RSA public key. You can store these keys wherever you like within your project. They will need to be submitted with your project for grading.

Cryptographic Routines The documentation for using cryptographic functions from `openssl` can be found at

`https://www.openssl.org/docs/manmaster/man3/`.

For an example of how to encrypt and decrypt using RSA, please see the webpage

`http://hayageek.com/rsa-encryption-decryption-openssl-c/`.

For an example of how to using blowfish symmetric encryption and decryption, see

`https://stackoverflow.com/questions/993780/`
`assistance-with-openssl-blowfish-simple-example-inserting-garbage-characters#999151`.

Make sure you look at the code in the checked answer, not the original. Also, read bytes from `/dev/urandom` rather that `/dev/random`. The second device may fail.

Provided Code I have supplied my code for `get` and `getd`. You may use this as the basis for your project.

Collaboration Rules

This project must be performed individually. Each person must work independently. An individual may neither show any other its code nor look at the code of another person. (This policy extends to any external resource, including code found on the web or individuals who are not enrolled in the course.)

Submissions

You must prepare a `makefile` and all necessary source files so that I can simply do a `make` and build `sgetd`. Submissions will be made through eLearning.