HW 1: Communication with Stream Cipher

1. Introduction

In the last assignment, Alice and Bob created a private channel to communicate with each other. However, this communication channel is not completely secure. For example, there may exist one Malory who is able to intercept the messages during the transmission (e.g., a man-in-the-middle attack). In this assignment, we take a further step to secure the channel, by using the stream cipher.

A stream cipher is a symmetric key cipher where plaintext digits are combined with a pseudorandom cipher digit stream (keystream). In a stream cipher, each plaintext digit is encrypted one at a time with the corresponding digit of the keystream, to give a digit of the ciphertext stream [1].

2. Tasks

Specifically, we'll implement two module functions in addition to our code in HW 0. We will learn to use stream ciphers by calling OpenSSL APIs. You can use your own code from HW0 or the answer codes to build the socket channel (answer code will not be provided until after all late homeworks are submitted or assigned a 0).
Note: We recommend this assignment be done in C/C++ language.

2.1 Communication using Stream Cipher

To communicate using a stream cipher, we recommend using OpenSSL, a well-documented and simple-to-use public library. You can also use other public libraries if you want. The general procedures for encryption/decryption consist of the following three steps:

1. Initialize the encryption/decryption function by choosing the encryption/decryption mode, key, and iv. In the stream cipher, we need to set the mode, key, and iv. There are many modes that OpenSSL provides, such as *EVP_rc4*, *EVP_rc4_40*, etc. (***You can choose any mode you want, and you need to specify in the report why you choose the mode***. **Be sure to choose a stream cipher rather than a block cipher**)
   Please refer to https://docs.openssl.org/1.1.1/man3/EVP_EncryptInit/ for more details.
2. Use the initialized cipher to encrypt/decrypt and obtain the encrypted/decrypted output.
3. Clean up and free the cipher.

We need to write a new encryption function and decryption function. Also, be sure to send and receive the same messages (e.g., "My password is xxxxxx") and print out the message to be sent and its encrypted text in the client, and the received message and its decrypted text in the server. **NOTE: You must include your name in the message you send!** The code skeleton is as follows, please fill in the function parameters according to the documentation in Section 5 below.

Encryption function in client.c (Alice):

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>

int stream_encrypt(unsigned char *plaintext, int plaintext_len, unsigned
char *key, unsigned char *iv, unsigned char *ciphertext)
{
  /* Declare cipher context */
   EVP_CIPHER_CTX *ctx;

   int len, ciphertext_len;

   /* Create and initialize the context */
   EVP_CIPHER_CTX_new();

   /* Initialize the encryption operation. */
   EVP_EncryptInit_ex();

   /* Provide the message to be encrypted, and obtain the encrypted
output. EVP_EncryptUpdate can be called multiple times if necessary */
    EVP_EncryptUpdate();

   /* Finalize the encryption. Further ciphertext bytes may be written at
this stage. */
   EVP_EncryptFinal_ex();

   /* Clean up */
   EVP_CIPHER_CTX_free();

   return ciphertext_len;
}
```

Decryption function in server.c (Bob):

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>

// Define the decryption function
int stream_decrypt(unsigned char *ciphertext, int ciphertext_len, unsigned
char *key, unsigned char *iv, unsigned char *plaintext)
{
   /* Declare cipher context */
   EVP_CIPHER_CTX *ctx;

   int len, plaintext_len;

   /* Create and initialize the context */
   EVP_CIPHER_CTX_new();

   /* Initialize the decryption operation. */
   EVP_DecryptInit_ex();

   /* Provide the message to be decrypted, and obtain the plaintext
output. EVP_DecryptUpdate can be called multiple times if necessary. */
   EVP_DecryptUpdate();

   /* Finalize the decryption. Further plaintext bytes may be written at
this stage. */
   EVP_DecryptFinal_ex();

   /* Clean up */
   EVP_CIPHER_CTX_free();

   return plaintext_len;
}
```

2.3 Compile and Troubleshooting

Note we use the OpenSSL header in our code, so we need to include the library during the compilation process.

For Ubuntu users, please first install OpenSSL through (**Note: openssl is likely already installed on the SeedVM, but if you are working on a local or fresh install this might be helpful**):
sudo apt-get install libssl-dev
Then Compile the program through:
gcc crypto.c -L/usr/lib -lssl -lcrypto -o server

(**Note: if you are working on a VM from your Mac, you can ignore below. If you want to build locally on your Mac from outside the VM, this might be useful**)
For Mac users, the compilation process is relatively complex, there is a tutorial to follow: https://unix.stackexchange.com/questions/346864/how-to-link-openssl-library-in-macos-using-gcc.

3. Things to Turn-in

You are expected to turn in 1) all of your source code used to build the server and the client, and 2) your report that explains what you have done in this lab. Be sure to include the screenshot of the results on both the server and the client. To turn in the source code, just copy and paste the code and put it in the appendix of the report (a screenshot is ok).

4. **Extra credit (0.1 points)** In homework 0, you used wireshark to simulate an eavesdropper capturing packets in transit between client and server. The Eavesdropper was able to read the message in plaintext. Run the same experiment and determine if you are able to view the message in plaintext now that it has been encrypted. Show a screenshot of your wireshark output examining the corresponding packet to where the plaintext message was in homework 0. Describe your observations.

5. References

[1] Christof Paar, Jan Pelzl, "Stream Ciphers", Chapter 2 of "Understanding Cryptography, A Textbook for Students and Practitioners". Springer, 2009.

6. Useful Functions

Create and initialize the cipher context, the context will hold state data and intermediate calculations.
*EVP_CIPHER_CTX *EVP_CIPHER_CTX_new(void);*

Initialize the encryption cipher, *EVP_EncryptInit_ex()* sets up cipher context *ctx* for encryption with cipher type from ENGINE *impl.* If impl is NULL then the default implementation is used. *ctx* must be created before calling this function. *type* is normally supplied by a function such as EVP_aes_256_cbc().
```
int EVP_EncryptInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type,
                        ENGINE *impl, const unsigned char *key, const unsigned char *iv);
```

Encrypt the plaintext, *EVP_EncryptUpdate* encrypts *inl* bytes from the buffer *in* and writes the encrypted version to *out*. For stream ciphers, the amount of data written can be anything from zero bytes to *inl* bytes. Thus, *out* should contain sufficient room for the operation being performed. If padding is enabled (the default), then *EVP_EncryptFinal_ex()* encrypts the "final" data.
```
int EVP_EncryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out,
                       int *outl, const unsigned char *in, int inl);
int EVP_EncryptFinal_ex(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl);
```

Initialize the decryption cipher, the usage is the same with initializing the encryption cipher.
```
int EVP_DecryptInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type,
                        ENGINE *impl, const unsigned char *key, const unsigned char *iv);
```

Decrypt the ciphertext, *EVP_DecryptUpdate* decrypts *inl* bytes from the buffer *in* and writes the decrypted string to *out*. *EVP_DecryptFinal_ex* is used to decrypt the final data when the padding is enabled (the default).
```
int EVP_DecryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out,
                        int *outl, const unsigned char *in, int inl);
int EVP_DecryptFinal_ex(EVP_CIPHER_CTX *ctx, unsigned char *outm, int *outl);
```

Clean up the cipher:
*void EVP_CIPHER_CTX_free(EVP_CIPHER_CTX *ctx);*