HW 2: Communication with Block Cipher

1. Introduction

In the last assignment, we are familiar with the stream cipher, and how it should be implemented to secure the channel. In this assignment, we will learn how to use and implement another symmetric key cipher, the block cipher.

A block cipher is a deterministic algorithm operating on fixed-length groups of bits, called blocks. They are specified as elementary components in the design of many cryptographic protocols and are widely used to encrypt large amounts of data, including in data exchange protocols [1]. It uses blocks as an unvarying transformation.

2. Tasks

Specifically, we'll implement one encryption function and one decryption function using the block cipher. Implementing the block cipher from scratch is not an easy task, in this task, we will use the same public API we used in HW1, OpenSSL [2]. You can use your own code from HW0 or the answer code to build the socket channel.
Note: We recommend this assignment be done in C/C++ language.

2.1 Implement Encryption/Decryption Function

The general encryption/decryption function contains three steps:

1. Initialize the encryption/decryption function by choosing the encryption/decryption mode, key, and iv. There are many modes to choose from as taught in the lectures, such as AES-256-CBC, you can choose any mode you want. It is worth mentioning that for most modes, IV size is the same as block size, so be sure to set it right. **Be sure to choose a block cipher!**
2. Use the initialized cipher to encrypt/decrypt and obtain the encrypted/decrypted output.
3. Clean up and free the cipher.

Be sure to include your name in the message you send (e.g., "My name is …. And my password is xxxxx") 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. 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>
```

```c
#include <arpa/inet.h>
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>

int block_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):

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

```c
#include <openssl/evp.h>
#include <openssl/err.h>

// Define the decryption function
int block_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();

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

   /* Clean up */
   EVP_CIPHER_CTX_free();

   return;
}
```

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:
sudo apt-get install libssl-dev

Then Compile the program through:
gcc crypto.c -L/usr/lib -lssl -lcrypto -o server

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.

4. References

[1] Cusick, Thomas W., and Pantelimon Stanica. *Cryptographic Boolean functions and applications*. Academic Press, 2017.
[2] OpenSSL. https://www.openssl.org/.

5. 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 encrypted version 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);*

For more details and usage, please refer to https://www.openssl.org/docs/.