Cryptography with OpenSSL: A Tutorial

Introduction

Cryptography is essential for securing communications between a client and a server. In this tutorial, we will explore how to use **OpenSSL** to perform **encryption and decryption** in both **C** and **Python**. This knowledge will help you implement basic encryption for your **Simple Client-Server Communication** assignment.

Using OpenSSL for Encryption and Decryption

C Example: Encrypting and Decrypting with AES

OpenSSL provides a rich set of cryptographic functions, including AES encryption. Below is a simple example demonstrating AES-256 encryption and decryption in C.

C Code: AES Encryption/Decryption

```c
#include <openssl/evp.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

void handleErrors() {
    fprintf(stderr, "An error occurred.\n");
    exit(EXIT_FAILURE);
}

int encrypt(unsigned char *plaintext, int plaintext_len, unsigned char *key,
            unsigned char *iv, unsigned char *ciphertext) {
    EVP_CIPHER_CTX *ctx;
    int len;
    int ciphertext_len;

    if (!(ctx = EVP_CIPHER_CTX_new())) handleErrors();

    if (1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv))
handleErrors();

    if (1 != EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext_len))
handleErrors();
    ciphertext_len = len;

    if (1 != EVP_EncryptFinal_ex(ctx, ciphertext + len, &len)) handleErrors();
    ciphertext_len += len;

    EVP_CIPHER_CTX_free(ctx);
    return ciphertext_len;
}
```

```
int decrypt(unsigned char *ciphertext, int ciphertext_len, unsigned char *key,
            unsigned char *iv, unsigned char *plaintext) {
    EVP_CIPHER_CTX *ctx;
    int len;
    int plaintext_len;

    if (!(ctx = EVP_CIPHER_CTX_new())) handleErrors();

    if (1 != EVP_DecryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv))
handleErrors();

    if (1 != EVP_DecryptUpdate(ctx, plaintext, &len, ciphertext, ciphertext_len))
handleErrors();
    plaintext_len = len;

    if (1 != EVP_DecryptFinal_ex(ctx, plaintext + len, &len)) handleErrors();
    plaintext_len += len;

    EVP_CIPHER_CTX_free(ctx);
    return plaintext_len;
}

int main() {
    unsigned char key[32] = "This is a key123This is a key123";
    unsigned char iv[16] = "1234567890123456";
    unsigned char plaintext[] = "Hello, Secure World!";
    unsigned char ciphertext[128];
    unsigned char decryptedtext[128];

    int ciphertext_len = encrypt(plaintext, strlen((char *)plaintext), key, iv,
ciphertext);
    printf("Ciphertext: ");
    for (int i = 0; i < ciphertext_len; i++) printf("%02x", ciphertext[i]);
    printf("\n");

    int decryptedtext_len = decrypt(ciphertext, ciphertext_len, key, iv,
decryptedtext);
    decryptedtext[decryptedtext_len] = '\0';
    printf("Decrypted text: %s\n", decryptedtext);

    return 0;
}
```

Explanation:

- **encrypt(plaintext, plaintext_len, key, iv, ciphertext)**:
  - `plaintext`: The input text to be encrypted.
  - `plaintext_len`: The length of the plaintext.
  - `key`: The encryption key (32 bytes for AES-256).
  - `iv`: The initialization vector (16 bytes for AES-CBC mode).

  - `ciphertext`: The buffer where the encrypted output will be stored.
  - This function initializes an encryption context, encrypts the plaintext in blocks, finalizes encryption, and stores the result in `ciphertext`.
  - Returns the length of the encrypted text.
  - It uses **AES-256-CBC** mode for encryption.

- **decrypt(ciphertext, ciphertext_len, key, iv, plaintext)**:

  - `ciphertext`: The encrypted input data.
  - `ciphertext_len`: The length of the ciphertext.
  - `key`: The decryption key (same as encryption key).
  - `iv`: The initialization vector (same as used for encryption).
  - `plaintext`: The buffer where the decrypted output will be stored.
  - This function initializes a decryption context, decrypts the ciphertext in blocks, finalizes decryption, and stores the result in `plaintext`.
  - Returns the length of the decrypted text.

- The `main()` function demonstrates calling encryption and decryption, printing the results.

Python Example: Encrypting and Decrypting with PyCrypto

For Python, we can use **PyCryptodome**, a widely used cryptography library.

Python Code: AES Encryption/Decryption

```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import os

def encrypt(plaintext, key):
    iv = os.urandom(16)  # Generate a random IV
    cipher = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(pad(plaintext.encode(), AES.block_size))
    return iv + ciphertext  # Prepend IV for use during decryption

def decrypt(ciphertext, key):
    iv = ciphertext[:16]  # Extract IV from the message
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = unpad(cipher.decrypt(ciphertext[16:]), AES.block_size)
    return plaintext.decode()

# Example Usage
key = b'This is a key123This is a key123'  # 32 bytes for AES-256
plaintext = "Hello, Secure World!"
ciphertext = encrypt(plaintext, key)
decrypted_text = decrypt(ciphertext, key)

print(f"Ciphertext: {ciphertext.hex()}")
print(f"Decrypted: {decrypted_text}")
```

Explanation:

- **encrypt(plaintext, key)**:
  - The function encrypt() uses AES CBC mode and a random IV.
  - `plaintext`: The input string to be encrypted.
  - `key`: The encryption key (must be 32 bytes for AES-256).
  - Generates a random 16-byte IV.
  - Uses AES-CBC mode for encryption.
  - pad() ensures that the plaintext length is a multiple of the block size.
  - The IV is prepended to the ciphertext, which is necessary for proper decryption.
- **decrypt(ciphertext, key)**:
  - The decrypt() function extracts the IV, decrypts the message, and removes padding.
  - `ciphertext`: The encrypted data, with the IV prepended.
  - `key`: The decryption key.
  - Extracts the IV from the ciphertext.
  - Decrypts the remaining part using AES-CBC.
  - Removes padding and returns the original plaintext.

Why Use Encryption in Your Assignment?

1. **Confidentiality:** Ensures that data exchanged between the client and server cannot be read by unauthorized entities.
2. **Integrity:** Guarantees that messages are not altered during transmission.
3. **Authentication:** Confirms that messages originate from trusted sources.

Next Steps:

- Implement **encryption** in your TCP client-server application.
- Use **AES encryption** for messages exchanged between client and server.
- Log encrypted messages to understand how data changes before and after encryption.