IMPLEMENTING BLOCK CIPHERS USING OPENSSL IN C/C++

SUBJECT NAME: CRYPTOGRAPHY AND NETWORK SECURITY

SUBJECT CODE: CS6008

MODULE: 7

NAME	BHUVANESHWAR S
REG.NO	2019103513
DATE	05/04/2022

AIM:

To implement a block ciphers using openssl.

TOOLS INVOLVED:

- Gcc compilers
- Kali Linux terminal [WSL]
- Visual Studio Code

PROBLEM DESCRIPTION:

A block cipher encrypts data in blocks using a deterministic algorithm and a symmetric key.

As in the case of stream ciphers, most encryption methods encrypt bits one by one (stream ciphers). Block ciphers, on the other hand, encrypt 128 bit blocks with a key of predetermined length: 128, 192, or 256 bits.

A 128-bit block cipher brings 128 bits of plaintext and encrypts it into 128 bits of ciphertext. Where the amount of plaintext is less than 128 bits.

Block ciphers have the advantage of high diffusion and strong tamper resistance without detection. They have the disadvantage of slower encryption speed since the entire block must be captured for encryption/decryption. Block ciphers also breed errors since a mistake in just one symbol could alter the whole block.

Here we demonstrate a classical example of block ciphers popularly known as 256bit Advanced Encryption Standard (AES). AES uses symmetric key encryption, which involves the use of only one secret key to cipher and decipher information.

The Advanced Encryption Standard (AES) is the first and only publicly accessible cipher approved by the US National Security Agency (NSA) for protecting top secret information. AES was first called Rijndael after its two developers

Symmetric, also known as secret key, ciphers use the same key for encrypting and decrypting, so the sender and the receiver must both know -- and use -- the same secret key. AES-256, which has a key length of 256 bits, supports the largest bit size and is practically unbreakable by brute force based on current computing power, making it the strongest encryption standard. There are 14 rounds for 256-bit keys. A round consists of several processing steps that include substitution, transposition and mixing of the input plaintext to transform it into the final output of ciphertext.

Encryption and decryption in the following program are carried out with the help of openssl libraries which gives us the decrypted text as output.

INPUT:

Getting an plain text from the user.

OUTPUT:

Encrypt the plain text using openssl.

SCREENSHOT:

Openssl.c

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

void handleErrors(void)
{
    ERR_print_errors_fp(stderr);
    abort();
}
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;
```

```c
    EVP_CIPHER_CTX_free(ctx);
    return plaintext_len;
}

int main(void)
{
    unsigned char *key = (unsigned char *)" 98765432109876543210987654321109";

    unsigned char *iv = (unsigned char *)"9876543210987644";
    char buffer[1024];
    printf("[ + ]\tEnter plain text :\t");
    scanf("%[^\n]%c", buffer);
    unsigned char *plaintext = (unsigned char *)buffer;
    unsigned char ciphertext[128];
    unsigned char decryptedtext[128];
    int decryptedtext_len, ciphertext_len;
    ciphertext_len = encrypt(plaintext, strlen((char *)plaintext),
                             key, iv,
                             ciphertext);
    printf("\n[ - ]\tCiphertext is:\n");
    BIO_dump_fp(stdout, (const char *)ciphertext, ciphertext_len);
    decryptedtext_len = decrypt(ciphertext, ciphertext_len, key, iv,
                                decryptedtext);
    decryptedtext[decryptedtext_len] = '\0';
    printf("\n[ - ]\tDecrypted text is:\t");
    printf("%s\n", decryptedtext);
    return 0;
}
```

OUTPUT

- Plain Text: College of Engineering, Guindy - Anna University
- Key: 98765432109876543210987654321109
- IV: 987654321098744

```
┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/Crypto&Net Security/assignment/openssl]
└─$ gcc -o openssl openssl.c -lcrypto

┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/Crypto&Net Security/assignment/openssl]
└─$ ./openssl
[ + ]   Enter plain text :      College of Engineering, Guindy - Anna University

[ - ]   Ciphertext is:
0000 - 6c c4 84 3b 00 98 9e 21-d2 d5 ea e7 a0 e0 c5 a4   l..;...!........
0010 - 3e 2e 6a 23 d9 5e c7 87-05 87 05 5e 38 99 82 04   >.j#.^.....^8...
0020 - b6 32 d0 92 a4 b2 2f 2d-55 5e 5c a8 2a a0 2d 65   .2..../-U^\.*.-e
0030 - 8c 1e 80 68 ba 32 1c 24-1e c0 eb 11 2f a2 fe 87   ...h.2.$..../...

[ - ]   Decrypted text is:      College of Engineering, Guindy - Anna University

┌──(bhuvan㉿Bhuvaneshwar)-[/mnt/e/clg 6th sem/Crypto&Net Security/assignment/openssl]
└─$
```

Here, it display the cipher text for given plain text using key and IV.