BÁO CÁO THỰC HÀNH

**Môn học: Nhập môn mạng**

**Tên chủ đề: Lab 1**

*GVHD: Tô Trọng Nghĩa*

**Nhóm: Mệt mỏi**

1. **THÔNG TIN CHUNG:**

Lớp: ATTT2021

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Họ và tên | MSSV | Email |
| 1 | Nguyễn Thị Minh Châu | 21520645 | 21520645@gm.uit.edu.vn |
| 2 | Lưu Thị Huỳnh Như | 21521242 | 21521242@gm.uit.edu.vn |

1. **NỘI DUNG THỰC HIỆN:[[1]](#footnote-1)**

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Nội dung | Tình trạng | Trang |
| 1 | Yêu cầu 1 | 100% | 1 - 3 |
| 2 | Yêu cầu 2 | 100% | 3 – 4 |
| 3 | Yêu cầu 3 | 100% | 4 - 6 |
| 4 | Yêu cầu 4 | 100% | 6 - 12 |
| 5 | Yêu cầu 5 | 80% | 12-28 |
| Điểm tự đánh giá | | | **8/10** |

**Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.**

BÁO CÁO CHI TIẾT

**Task 1:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mode | Description | Typical Application | Strength | Weakness |
| ECB | Each block of plaintext bits is encoded independently using the same key. | * Secure transmission of single values (e.g., and encryption key) | It is simple | Repetitive data contained in the plaintext may show in the cipher, if aligned with blocks. If the same message is encrypted (with the same key) and sent twice, their ciphertext are the same |
| CBC | The input to the encryption algorithm is the XOR of the next block of plaintext and the preceding block of ciphertext. | * General-purpose block-oriented transmission * Authentication | Repeated same plaintext block result different ciphertext block. | Plaintext block can be recovered from two adjacent blocks of ciphertext |
| CFB | Input is processed s bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce text unit of cipher text | * General-purpose stream-oriented transmission * Authentication | A ciphertext segment depends on the current and all preceding plaintext segments. | A corrupted ciphertext segment during transmission will affect the current and next several plaintext segments. |
| OFB | Similar to CFB, except that the input to the encryption algorithm is the preceding encryption output, and full blocks are used. | * Stream-oriented transmission over noisy channel | Avoid error propagation. Pre-compute of forward cipher is possible. | When plaintext is known, the output of the forward cipher function will be known. Easily cover plaintext block with the same IV. |
| CTR | Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block | * General-purpose block-orented transmission * Useful for high-speed requirements. | * Need only the encryption algorithm * Random access to encrypted data blocks * Simple, fast | Synchronous counter needs to be maintained at both receiving and sending ends. Losing track of this counter could lead to the ​incorrect recovery of plaintext. |

In normal CBC mode cannot be parallelized during encryption; that's because CBC mode encryption is defined as: Ci=Ek(Ci−1 XOR Pi)

In addition, there is a variant of CBC mode where you don't xor in the immediately previous ciphertext block, you xor in one n blocks ago, as in: Ci=Ek(Ci−n XOR Pi)

This is effectively running n different CBC mode encryptions in parallel; this can be paralleized n ways.

Similarly, CBC encryption also requires the previous ciphertext block:Ci=EK(Pi XORCi−1)

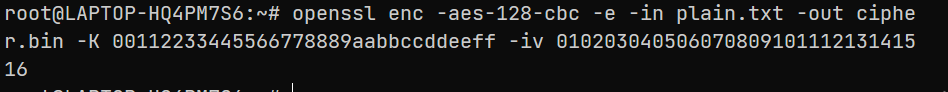
However, when encrypting, the ciphertext blocks are not available yet, and getting the previous block Ci−1 requires the one before that, and so on all the way to the start of the chain and the IV. So encryption in CBC mode cannot be done in parallel.

1. File plain.txt

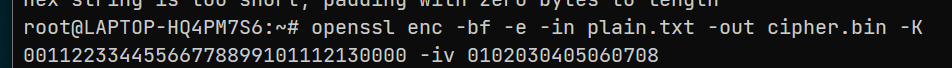
Ảnh có chứa văn bản

Mô tả được tạo tự động

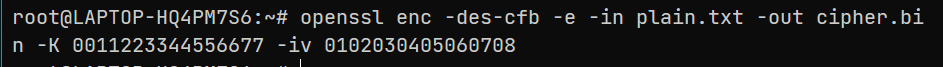
* Encrypt with AES-CBC



* Encrypt with BF



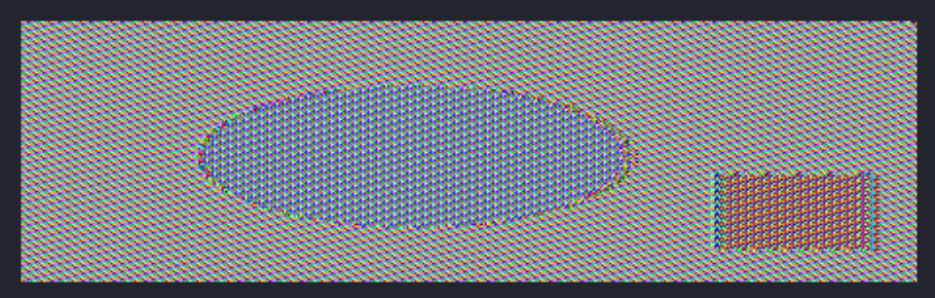
* Encrypt with DES-CFB



Task 2:

1. Encryption Mode – ECB vs CBC

– For ECB: although the image is encrypted, the basic shape of the objects in the image can still be seen. Since ecb is an independent block-by-block encryption, each block will be encrypted with the same key and same algorithm, inputting the same plaintext yields the same ciphertext.



- For CBC: the image is completely encrypted, nothing can be seen from the image. Because for cbc mode, the encryption result of the previous block will be combined with the next data block before performing the encryption. So even if the input is the same, the output can be different, depending on the IV or previous encryption results.

- Random selected image:

Ảnh có chứa biểu đồ

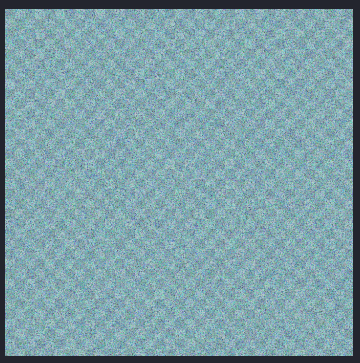
Mô tả được tạo tự động

* Encrypted with ECB

A close-up of a brain

Description automatically generated with low confidence

* Encrypted with CBC



Task 3:

Modes have padding: ECB, CBC

Modes don’t have padding: CFB, OFB

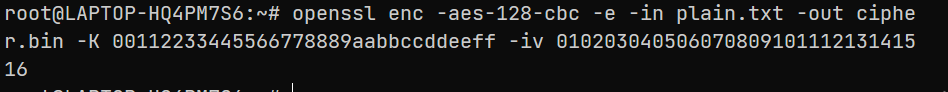
ECB or CBC requires their input to be an exact multiple of the block size. If the plaintext to be encrypted is not an exact multiple, you need to pad before encrypting by adding a padding string. If you encrypt using CFB or OFB modes then the ciphertext will be the same size as the plaintext and so padding is not required.

* File plain.txt

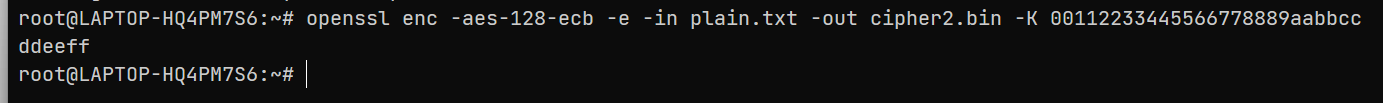
Ảnh có chứa văn bản

Mô tả được tạo tự động

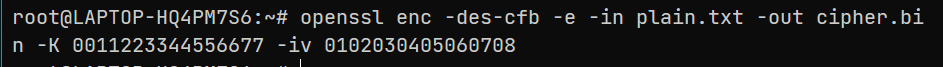
* Encrypt with AES-CBC



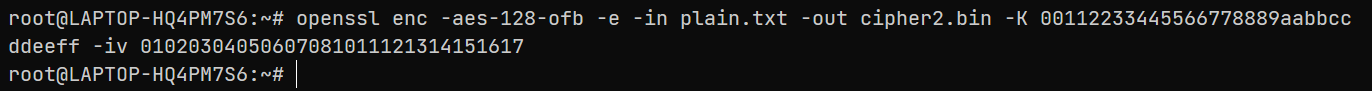
* Encrypt with AES-ECB



* Encrypt with DES-CFB



* Encrypt with AES-OFB



2.

Three plaintext files:

**Ảnh có chứa văn bản

Mô tả được tạo tự động**

* Encryption

**Ảnh có chứa văn bản

Mô tả được tạo tự động**

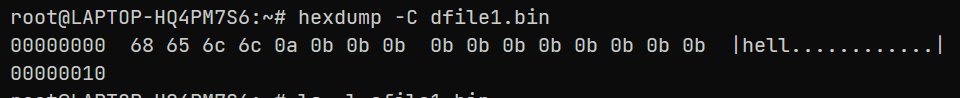
* Encrypted files’ size

**Ảnh có chứa văn bản

Mô tả được tạo tự động**

After encryption the file size becomes [1 block (16 bytes) IV + original file size + padding length]

* Decryption files



Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

Task 4:

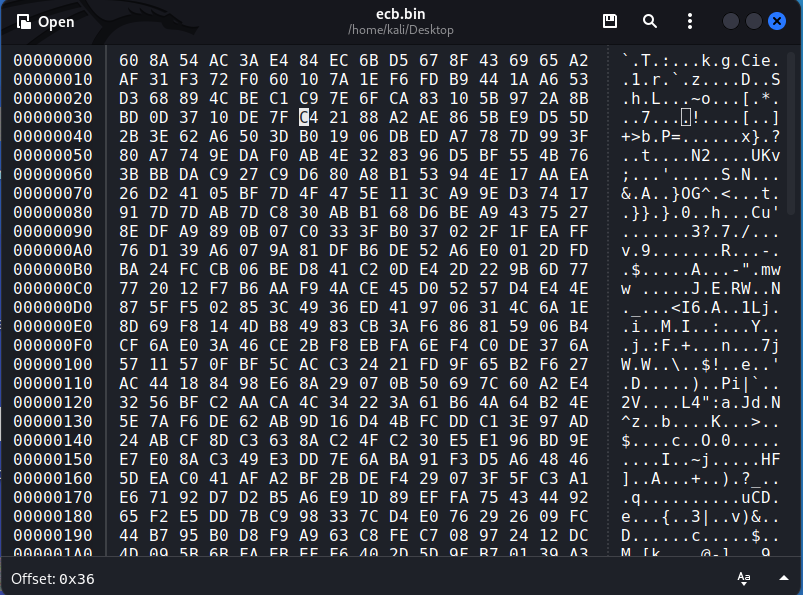
- In ECB, if a bit is corrupted, only the block containing that bit is affected, the rest of the blocks are decoded independently.

- In CBC, if a bit is broken, it affects the block containing that bit and the block that follows the block containing the corrupted bit.

- In OFB, if a bit is corrupted, it will only corrupt the character containing that bit.

* ECB:

Before:



After:

A screen shot of a computer

Description automatically generated with low confidence

File decrypt:

Ảnh có chứa văn bản, ảnh chụp màn hình, bình phong

Mô tả được tạo tự động

-> Since ECB is a bit-block independent encryption and decryption mode, the failed block does not affect other blocks.

* CBC:

Before:

Ảnh có chứa văn bản, đồ điện tử, máy tính

Mô tả được tạo tự động

After:

A computer screen capture

Description automatically generated with low confidence

File decrypt:

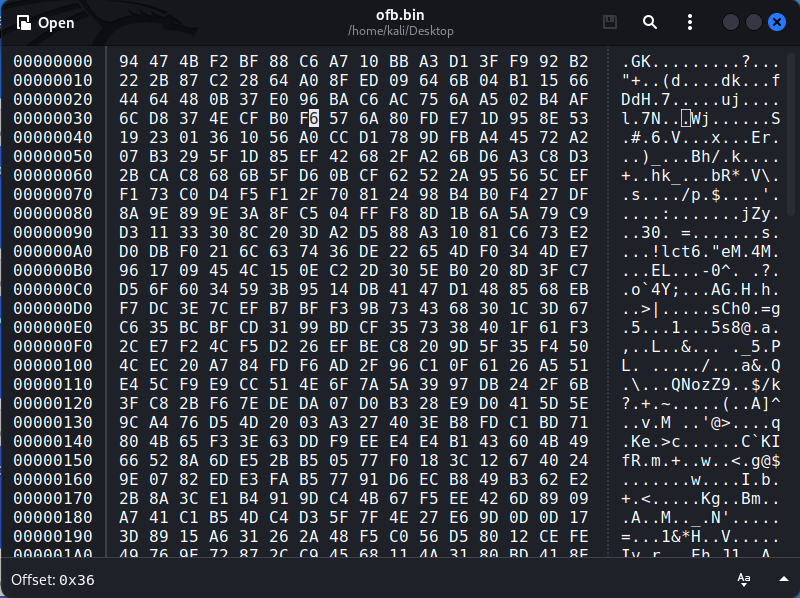
Graphical user interface, text, application

Description automatically generated

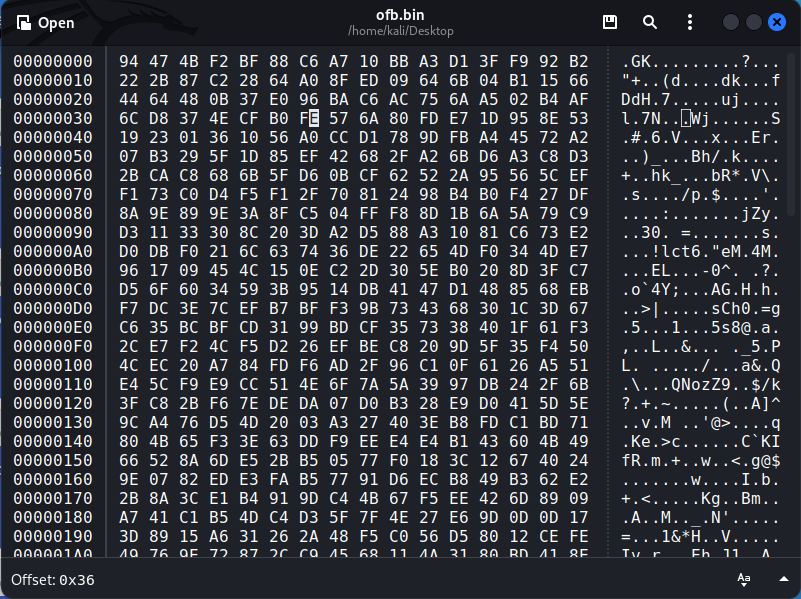
-> Because when decrypting, the cipher text of the following block must be xor to the cipher text of the previous block to complete the decryption process, so if the previous block fails, the next block will also fail.

* OFB:

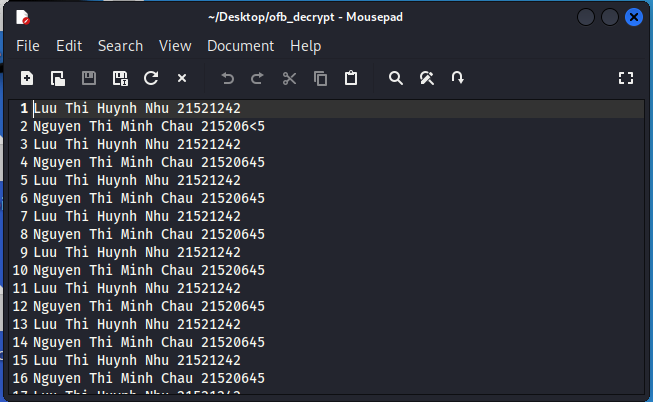
Before:



After:



File decrypt:



-> because when decoding, we will take the value of the previous output (this output will not make an error, because it has taken iv as input at the beginning, then it continues to take the result just done. into the next block, even if the cipher text is wrong doesn't affect this result), xor with the cipher text, so the bit error is not propagated, and only gets the wrong character immediately.

Task 5:

#include <iostream>

#include "cryptopp\aes.h"

using CryptoPP::AES;

#include "cryptopp\files.h"

using CryptoPP::FileSource;

#include "cryptopp\filters.h"

using CryptoPP::StringSink;

using CryptoPP::StringSource;

using CryptoPP::StreamTransformationFilter;

#include "cryptopp\hex.h"

using CryptoPP::HexEncoder;

using CryptoPP::HexDecoder;

#include "cryptopp\osrng.h"

#include "cryptopp\modes.h"

#include <string>

using namespace std;

string PrettyPrint(string text) {

    string encoded = "";

    encoded.clear();

    StringSource ss(text, true, new HexEncoder(new StringSink(encoded)));

    return encoded;

}

void GetString(string &str) {

    cout << "Enter string: ";

    getline(cin, str);

}

void GetStringFromFile(string &str) {

    string fileName;

    cout << "Enter file name: ";

    getline(cin, fileName);

    FileSource(fileName.c\_str(), true, new StringSink(str));

    cout << "Readed: " << str << endl;

}

void GetCipherString(string &str) {

    string temp;

    cout << "Enter cipher: ";

    temp.clear();

    getline(cin, temp);

    str.clear();

    StringSource(temp, true, new HexDecoder(new StringSink(str)));

    cout << "Cipher: " << PrettyPrint(str) << endl;

}

void ReadKeyAndIVFromKeyBoard(byte \*&key, int &keySize, byte \*&iv, bool &ivSet) {

    string temp;

    bool cont = false;

    // get key

    GetString(temp);

    if (temp.length() < 1 || temp.length() / 2 > AES::MAX\_KEYLENGTH) {

        cout << "Invalid key size\n";

        exit(-1);

    }

    key = new byte[temp.length() / 2];

    keySize = temp.length() / 2;

    StringSource(temp, keySize, true,

        new HexDecoder(

            new ArraySink(key, keySize)

        )

    );

    cout << "Key: " << PrettyPrint(key, keySize) << endl;

    // get iv

    cout << "Enter IV? (1/0): ";

    cin >> ivSet;

    cin.ignore();

    if (!ivSet)

        return;

    GetString(temp);

    if (temp.length() < 1 || temp.length() > 32) {

        cout << "Invalid iv size\n";

        exit(-1);

    }

    iv = new byte[temp.length() / 2];

    StringSource(temp, true,

                new HexDecoder(new CryptoPP::ArraySink(iv, temp.length() / 2)));

    cout << "IV: " << PrettyPrint(iv, temp.length() / 2) << endl;

}

void GenRandomByte(byte \*&out\_byte, const int size) {

    AutoSeededRandomPool prng;

    out\_byte = new byte[size];

    prng.GenerateBlock(out\_byte, size);

}

void GenKey(byte \*&key, const int &keySize, byte \*&iv, bool &genIV) {

    genIV = false;

    // keySize = DEFAULT\_KEYSIZE;

    GenRandomByte(key, keySize);

    cout << "Gen IV? (1/0): ";

    cin >> genIV;

    if (genIV)

        GenRandomByte(iv, AES::BLOCKSIZE);

    cout << "Key: " << PrettyPrint(key, keySize) << endl;

    cout << "Key write down to AES\_key.key\n";

    FileSink("AES\_key.key", true).Put(key, keySize);

    if (genIV) {

        cout << "IV: " << PrettyPrint(iv, AES::BLOCKSIZE) << endl;

        cout << "IV write down to AES\_iv.iv\n";

        FileSink("AES\_iv.iv", true).Put(iv, keySize);

    }

    genIV = true;

}

void GetKeyAndIVFromFile(byte \*&key, int &keySize, byte \*&iv, bool &ivSet) {

    string filename;

    cout << "Enter key file name: ";

    getline(cin, filename);

    keySize = AES::DEFAULT\_KEYLENGTH \* 2; // \* 2 for xts

    key = new byte[keySize];

    FileSource(filename.c\_str(), true, new ArraySink(key, keySize));

    cout << "Key: " << PrettyPrint(key, keySize) << endl;

    cout << "Enter IV? (1/0): ";

    cin >> ivSet;

    if (!ivSet)

        return;

    cout << "Enter iv file name: ";

    cin.ignore();

    getline(cin, filename);

    iv = new byte[AES::BLOCKSIZE];

    FileSource(filename.c\_str(), true, new ArraySink(iv, AES::BLOCKSIZE));

    cout << "IV: " << PrettyPrint(iv, AES::BLOCKSIZE) << endl;

}

const char \*menuMode = "1. ECB\n2. CBC\n3. OFB\n4. CFB\n5. CTR\n6. XTS\n7. CCM\n";

const char \*menuGetKey = "1. Generate key\n2. Use key\n3. Enter key\n";

const char \*menuGetText = "Get plain/cipher text\n1. Enter from console\n2. Use file\n";

void SwitchGetString(string &str) {

    int choice;

    cout << menuGetText;

    cin >> choice;

    cin.ignore();

    switch (choice) {

        case 1: {

            GetString(str);

            break;

        }

        case 2: {

            GetStringFromFile(str);

            break;

        }

    }

}

void SwitchGetStringCipher(string &str) {

    int choice;

    cout << menuGetText;

    cin >> choice;

    cin.ignore();

    switch (choice) { // get cipher

        case 1: {

            GetCipherString(str);

            break;

        }

        case 2: {

            GetStringFromFile(str);

            break;

        }

    }

}

void SwitchGetKeyAndIV(byte \*&key, int &keySize, byte \*&iv, bool &ivSet) {

    int choice;

    cout << menuGetKey;

    cin >> choice;

    cin.ignore();

    switch (choice) { // get key

        case 1: {

            keySize = DEFAULT\_KEYSIZE;

            GenKey(key, keySize, iv, ivSet);

            break;

        }

        case 2: {

            GetKeyAndIVFromFile(key, keySize, iv, ivSet);

            break;

        }

        case 3: {

            ReadKeyAndIVFromKeyBoard(key, keySize, iv, ivSet);

            break;

        }

    }

}

using byte = unsigned char;

using std::string;

constexpr int tagSize = 8;

string ECBMode\_Encrypt(string plain, byte key[], int keySize) {

    string cipher = "";

    try {

        ECB\_Mode<AES>::Encryption e;

        e.SetKey(key, keySize);

        StringSource ss(plain, true,

                        new StreamTransformationFilter(e, new StringSink(cipher)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return cipher;

}

string ECBMode\_Decrypt(string cipher, byte key[], int keySize) {

    string recovered = "";

    try {

        ECB\_Mode<AES>::Decryption d;

        d.SetKey(key, keySize);

        StringSource ss(

            cipher, true,

            new StreamTransformationFilter(d, new StringSink(recovered)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return recovered;

}

string CBCMode\_Encrypt(string plain, byte key[], int keySize, byte iv[]) {

    string cipher = "";

    try {

        CBC\_Mode<AES>::Encryption e;

        e.SetKeyWithIV(key, keySize, iv);

        StringSource ss(plain, true,

                        new StreamTransformationFilter(e, new StringSink(cipher)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return cipher;

}

string CBCMode\_Decrypt(string cipher, byte key[], int keySize, byte iv[]) {

    string recovered = "";

    try {

        CBC\_Mode<AES>::Decryption d;

        d.SetKeyWithIV(key, keySize, iv);

        StringSource ss(

            cipher, true,

            new StreamTransformationFilter(d, new StringSink(recovered)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return recovered;

}

string OFBMode\_Encrypt(string plain, byte key[], int keySize, byte iv[]) {

    string cipher = "";

    try {

        OFB\_Mode<AES>::Encryption e;

        e.SetKeyWithIV(key, keySize, iv);

        StringSource ss(plain, true,

                        new StreamTransformationFilter(e, new StringSink(cipher)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return cipher;

}

string OFBMode\_Decrypt(string cipher, byte key[], int keySize, byte iv[]) {

    string recovered = "";

    try {

        OFB\_Mode<AES>::Decryption d;

        d.SetKeyWithIV(key, keySize, iv);

        StringSource ss(

            cipher, true,

            new StreamTransformationFilter(d, new StringSink(recovered)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return recovered;

}

string CFBMode\_Encrypt(string plain, byte key[], int keySize, byte iv[]) {

    string cipher = "";

    try {

        CFB\_Mode<AES>::Encryption e;

        e.SetKeyWithIV(key, keySize, iv);

        StringSource ss(plain, true,

                        new StreamTransformationFilter(e, new StringSink(cipher)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return cipher;

}

string CFBMode\_Decrypt(string cipher, byte key[], int keySize, byte iv[]) {

    string recovered = "";

    try {

        CFB\_Mode<AES>::Decryption d;

        d.SetKeyWithIV(key, keySize, iv);

        StringSource ss(

            cipher, true,

            new StreamTransformationFilter(d, new StringSink(recovered)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return recovered;

}

string CTRMode\_Encrypt(string plain, byte key[], int keySize, byte iv[]) {

    string cipher = "";

    try {

        CTR\_Mode<AES>::Encryption e;

        e.SetKeyWithIV(key, keySize, iv);

        StringSource ss(plain, true,

                        new StreamTransformationFilter(e, new StringSink(cipher)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return cipher;

}

string CTRMode\_Decrypt(string cipher, byte key[], int keySize, byte iv[]) {

    string recovered = "";

    try {

        CTR\_Mode<AES>::Decryption d;

        d.SetKeyWithIV(key, keySize, iv);

        StringSource ss(

            cipher, true,

            new StreamTransformationFilter(d, new StringSink(recovered)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return recovered;

}

string XTSMode\_Encrypt(string plain, byte key[], int keySize, byte iv[]) {

    string cipher = "";

    try {

        XTS\_Mode<AES>::Encryption e;

        e.SetKeyWithIV(key, keySize, iv);

        StringSource ss(plain, true,

                        new StreamTransformationFilter(e, new StringSink(cipher)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return cipher;

}

string XTSMode\_Decrypt(string cipher, byte key[], int keySize, byte iv[]) {

    string recovered = "";

    try {

        XTS\_Mode<AES>::Decryption d;

        d.SetKeyWithIV(key, keySize, iv);

        StringSource ss(

            cipher, true,

            new StreamTransformationFilter(d, new StringSink(recovered)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return recovered;

}

string CCMMode\_Encrypt(string plain, byte key[], int keySize, byte iv[]) {

    string cipher = "";

    try {

        CCM<AES, tagSize>::Encryption e;

        e.SetKeyWithIV(key, keySize, iv);

        e.SpecifyDataLengths(0, plain.size(), 0);

        StringSource ss(

            plain, true,

            new AuthenticatedEncryptionFilter(e, new StringSink(cipher)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return cipher;

}

string CCMMode\_Decrypt(string cipher, byte key[], int keySize, byte iv[]) {

    string recovered = "";

    try {

        CCM<AES, tagSize>::Decryption d;

        d.SetKeyWithIV(key, keySize, iv);

        d.SpecifyDataLengths(0, cipher.size() - tagSize, 0);

        StringSource ss(

            cipher, true,

            new AuthenticatedDecryptionFilter(d, new StringSink(recovered)));

    } catch (const CryptoPP::Exception &e) {

        cerr << e.what() << endl;

        exit(1);

    }

    return recovered;

}

void SwitchEncrypt(string &cipher, string &plain, byte \*key, int keySize, byte \*iv, bool ivSet) {

    int choice;

    cout << menuMode;

    cin >> choice;

    switch (choice) { // choice mode of operation

    case 1:

        cipher = ECBMode\_Encrypt(plain, key, keySize);

        cout << "Cipher: " << PrettyPrint(cipher) << endl;

        break;

    case 2:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        cipher = CBCMode\_Encrypt(plain, key, keySize, iv);

        cout << "Cipher: " << PrettyPrint(cipher) << endl;

        break;

    case 3:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        cipher = OFBMode\_Encrypt(plain, key, keySize, iv);

        cout << "Cipher: " << PrettyPrint(cipher) << endl;

        break;

    case 4:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        cipher = CFBMode\_Encrypt(plain, key, keySize, iv);

        cout << "Cipher: " << PrettyPrint(cipher) << endl;

        break;

    case 5:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        cipher = CTRMode\_Encrypt(plain, key, keySize, iv);

        cout << "Cipher: " << PrettyPrint(cipher) << endl;

        break;

    case 6:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        cipher = XTSMode\_Encrypt(plain, key, keySize, iv);

        cout << "Cipher: " << PrettyPrint(cipher) << endl;

        break;

    case 7:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        cipher = CCMMode\_Encrypt(plain, key, keySize, iv);

        cout << "Cipher: " << PrettyPrint(cipher) << endl;

        break;

    default:

        break;

    }

}

void SwitchDecrypt(string &plain, string &cipher, byte \*key, int keySize, byte \*iv, bool ivSet) {

    int choice;

    cout << menuMode;

    cin >> choice;

    switch (choice) { // choice mode of operation

    case 1:

        plain = ECBMode\_Decrypt(cipher, key, keySize);

        cout << "Plain: " << plain << endl;

        break;

    case 2:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        plain = CBCMode\_Decrypt(cipher, key, keySize, iv);

        cout << "Plain: " << plain << endl;

        break;

    case 3:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        plain = OFBMode\_Decrypt(cipher, key, keySize, iv);

        cout << "Plain: " << plain << endl;

        break;

    case 4:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        plain = CFBMode\_Decrypt(cipher, key, keySize, iv);

        cout << "Plain: " << plain << endl;

        break;

    case 5:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        plain = CTRMode\_Decrypt(cipher, key, keySize, iv);

        cout << "Plain: " << plain << endl;

        break;

    case 6:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        plain = XTSMode\_Decrypt(cipher, key, keySize, iv);

        cout << "Plain: " << plain << endl;

        break;

    case 7:

        if (!ivSet) {

        cout << "No IV set" << endl;

        break;

        }

        plain = CCMMode\_Decrypt(cipher, key, keySize, iv);

        cout << "Plain: " << plain << endl;

        break;

    default:

        break;

    }

}

void Encrypt() {

    int keySize = 0;

    int choiceGet = 0;

    bool ivSet = false;

    string plain = "";

    string cipher = "";

    byte \*key = nullptr;

    byte \*iv = nullptr;

    SwitchGetString(plain);

    SwitchGetKeyAndIV(key, keySize, iv, ivSet);

    SwitchEncrypt(cipher, plain, key, keySize, iv, ivSet);

}

void Decrypt() {

    int keySize = 0;

    int choiceGet = 0;

    bool ivSet = false;

    string plain = "";

    string cipher = "";

    byte \*key = nullptr;

    byte \*iv = nullptr;

    SwitchGetStringCipher(cipher);

    SwitchGetKeyAndIV(key, keySize, iv, ivSet);

    SwitchDecrypt(plain, cipher, key, keySize, iv, ivSet);

}

int main() {

    cout <<"Encryption(0) or decryption (other): ";

    int x;

    cin >> x;

    time\_t start, end;

    switch (x) {

    case 1: // Encrypt

        time(&start);

        Encrypt();

        time(&end);

         double time\_taken = double(end - start);

        cout << "Time taken by program is : " << fixed

        << time\_taken << setprecision(5);

        cout << " sec " << endl;

        break;

    case 2: // Decrypt

        time(&start);

        Decrypt();

        time(&end);

         double time\_taken = double(end - start);

        cout << "Time taken by program is : " << fixed

        << time\_taken << setprecision(5);

        cout << " sec " << endl;

        break;

    default:

        break;

    }

    return 0;

}

1. Ghi nội dung công việc, các kịch bản trong bài Thực hành [↑](#footnote-ref-1)