Palestine Polytechnic University

College of IT and Computer Engineering

Introduction to Cryptography

AES Encryption Project

CBC Mode

Suhair Shareef

Dr. Mousa Frajallah

Executive Summary

This project represents AES encryption using CryptoPP C++ library. Using CBC mode and RC4 random number generator for generating the key. Then we evaluated the time performance of the project.

List of Errors

- Libraries and headers not seen: The libraries compiled weren't seen by the compiler.
- Wrong build configurations: The CryptoPP was compiled using Debug 64x and I encountered an error when the settings got back to defaults.
- The wrong bath to the library: had to add the headers and source files from zero.
- The wrong version of the used code: had to reset all setup from scratch.
- Byte not being recognized by the compiler: had to convert the output of **RC4** to byte.
- Couldn't convert the image to a string: had to install opencv to do that
- Had to reinstall the whole library multiple times.
- **Opencv** library didn't work in the same program environment, so the default for it was on the **debug 64x** setup, and the **cryptoPP** words on **release 64x,** so I had to change the library from **opencv_world420d.lib** to **opencv_world420.lib**
- The approach I went with at the beginning wasn't the correct way, so I redid the whole project from the beginning. I was converting the image into a string stream and encoding it into base64, but I discovered that the project had to be done on binary data bit by bit. I went with another approach.

Methodology

Steps to configure CryptoPP *(before changing the approach)*:

1. Download CryptoPP library
2. Download open cv
3. download base64 library
4. Create a project
5. Run the cryptest to create a release version of the library
6. Add the library to the project
7. Configure the path to the library in the project
8. Build the project and check if the path is correct for the headers

Steps to configure CryptoPP & Opencv *(after changing the approach)*:

1. Download **CryptoPP** library
2. Create a project
3. Run the cryptest to create a release version of the library
4. Add the library to the project
5. Configure the path to the library **cryptlib.lib** in the project
6. Build the project and check if the path is correct for the headers
7. Download **OpenCV**
8. Add the path to the "**..\opencv\build\x64\vc15\bin**" folder in the environment variable **PATH**
9. Add the include files and library to the project settings
10. Add the library to the linker **opencv_world420.lib**

Steps to create the project:

1. Define the functions used:
    - **init_key**(key, key_len): function to initialize a byte key randomly (I used it for IV)
    - **AES_enc_dec**(dynamic_key, plain, row, col, channel): main function for the encryption and decryption process, it has the generating of RC4 keys, and function call of AES encryption and decryption
    - **AES_encryption**(plain, row, col, channel): takes a block of plain text with a starting point in the image cipher matrix. Returns the ciphertext
    - **AES_decryption**(cipher, row, col, channel): takes the ciphertext and decrypts it, and recovers the plain text into the resulted image matrix

2. In the main function, it read the bitmap image into a matrix and initialized two matrices for the cipher image and the recovered image.
3. For every 128 bytes(chars), the function AES_enc_dec is called with the dynamic key and the start of the current block.
4. In the function AES_enc_dec, the RC4 takes the dynamic key and generates 8 keys for each round, which means that every 16 bytes we get a new key from the dynamic key.
5. When the encryption function is called, it takes the plain text and encrypts it into the AES using the generated key in CBC mode, then it adds the values of the ciphertext into the cipher image, then it returns the ciphertext.
6. when the description function is called, it takes the cipher text and decrypts it using the AES in CBC mode, then it adds the values of the recovered plain text into the recovered image.
7. After both operations are done, the program displays the plain image, the cipher image and the recovered image.

Time Performance

- Image size = 512 * 512 * 3 bytes
- Encryption time = 1.578636 seconds
- **ET** = 7.4725776 * 10^6 byte/second
- **CPU speed** = 1.8 * 10^9 Hz
- **Number of cycles** per byte = CPU speed / ET = **240.8807** cycle/byte

Code for AES and RC4

```
// Libraries includes
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <chrono>
#include <opencv2/opencv.hpp>
#include "opencv2/imgcodecs.hpp"
#include "opencv2/core/saturate.hpp"


// Header files includes
#include "osrng.h"
#include "cryptlib.h"
#include "hex.h"
#include "aes.h"
#include "ccm.h"
#include "eax.h"
#include "filters.h"


// Namespaces
using namespace std;
using namespace CryptoPP;
using namespace chrono;
using namespace cv;


ifstream img;
ofstream cipher_img;
ofstream recovered_img;
ofstream key_file;

// Initialize images data
Mat original_image = imread("lena_gray.bmp", IMREAD_UNCHANGED);
Mat cipher_image = original_image.clone();
Mat restored_image = original_image.clone();
int rows = original_image.rows;
int cols = original_image.cols;
byte key[AES::DEFAULT_KEYLENGTH];   // 128 bit
byte iv[AES::BLOCKSIZE];

void AES_enc_dec(int dynamic_key, string plain, int row, int col, int
channel);
string AES_encryption(string plain, int row, int col, int channel);
```

```
void AES_decryption(string cipher, int row, int col, int channel);


void Init_key(byte* genkey, size_t size);


int main(int argc, char* argv[])
{
    // initializing timer
    auto start = high_resolution_clock::now();

    cout << "rows: " << rows << endl;
    cout << "cols: " << cols << endl;

    if (original_image.empty())
    {
        cout << "Cannot load image!" << endl;
        return -1;
    }

    // encrypt every 128 block
    string plain = "";
    int dynamic_key = rand();
    int row_start, col_start, channel_start;
    int rounds = 0;


    for (int row = 0; row < rows; row++)
    {
        for (int col = 0; col < cols; col++)
        {
            for (int channel = 0; channel < 3; channel++) {
                char ch =
saturate_cast<char>(original_image.at<Vec3b>(row, col)[channel]);
                if (plain == "") {
                    row_start = row;
                    col_start = col;
                    channel_start = channel;
                }
                plain += ch;
                if (plain.length() == 128) {
                    AES_enc_dec(dynamic_key, plain, row_start, col_start,
channel_start);
                    rounds++;
                    plain = "";
                }
            }
```

```
        }
    }

    // print the duration of the encryption and dycreption
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "Excution Duration: " << duration.count() << " microsecond" <<
endl;

    imshow("Original image", original_image);

    imshow("Decrypted image", cipher_image);

    imshow("Restored image", restored_image);
    waitKey(0);


    return 0;
}

void AES_enc_dec(int dynamic_key, string plain, int row, int col, int
channel)
{
    // Plain recieved is 128 bytes or less which will go through 16 rounds
    // Define the random key generator
    byte key[AES::DEFAULT_KEYLENGTH];   // 128 bit
    byte iv[AES::BLOCKSIZE];

    // Initializing RC4
    int key_length = sizeof(key);
    int* s = new int[256];
    int* T = new int[256];
    for (int i = 0; i < 256; i++)
    {
        s[i] = i;
        T[i] = dynamic_key;
    }
    int temp = 0;
    for (int j = 0; j < 256; j++)
    {
        temp = (temp + s[j] + T[j]) % 256;
        swap(s[j], s[temp]);
    }

    int rounds = plain.length() / 16;
    int plain_start = 0;
```

```
int* val = new int[key_length];

while (rounds) {
    // the beginning of the current block
    string curr_plain = plain.substr(plain_start, 16);

    // initialive IV
    Init_key(iv, sizeof(iv));

    int i = 0;
    int j = 0;
    int z = 0;
    int pp = key_length;

    // Construct key from the dynamic key
    while (pp)
    {
        i = (i + 1) % key_length;
        j = (j + s[i]) % key_length;
        swap(s[i], s[j]);
        int t = (s[i] + s[j]) % key_length;
        val[z] = s[t];
        key[z] = (char)val[z];
        z++;
        pp -= 1;
    }

    string cipher = AES_encryption(curr_plain, row, col, channel);
    AES_decryption(cipher, row, col, channel);

    // Move row to the next 16 byte block
    int count = 0;
    while (count < 16) {
        channel++;
        if (channel > 2) {
            channel = 0;
            col++;
        }
        if (col > cols) {
            col = 0;
            row++;
        }
        count++;
    }

    rounds--;
    plain_start += 16;
```

```
    }
}

string AES_encryption(string plain, int row, int col, int channel) {
    string cipher = "";
    try
    {
        // Encrypting every 8 bytes
        CBC_Mode<AES>::Encryption e;
        e.SetKeyWithIV(key, sizeof(key), iv);

        // The StreamTransformationFilter removes padding as required.
        StringSource s(plain, true, new StreamTransformationFilter(e, new
StringSink(cipher)));

        // Add to cipher matrix
        int count = 0;
        while (count < 16) {
            //unsigned char ch = (uchar)cipher[count];
            cipher_image.at<Vec3b>(row, col)[channel] =
(uchar)cipher[count];
            channel++;
            if (channel > 2) {
                channel = 0;
                col++;
            }
            if (col > cols) {
                col = 0;
                row++;
            }
            count++;
        }

#if 0
        StreamTransformationFilter filter(e);
        filter.Put((const byte*)plain.data(), plain.size());
        filter.MessageEnd();
        const size_t ret = filter.MaxRetrievable();
        cipher.resize(ret);
        filter.Get((byte*)cipher.data(), cipher.size());
#endif

    }
    catch (const CryptoPP::Exception& e)
    {
        cerr << e.what() << endl;
```

```
        cout << "error here" << endl;
        exit(1);
    }
    return cipher;
}

void AES_decryption(string cipher, int row, int col, int channel) {
    string recovered;
    try
    {
        // Decryption
        CBC_Mode<AES>::Decryption d;
        d.SetKeyWithIV(key, sizeof(key), iv);
        // The StreamTransformationFilter removes
        // padding as required.
        StringSource s(cipher, true, new StreamTransformationFilter(d, new
StringSink(recovered)));

        // Recovery
        // Add to restored matrix
        int count = 0;
        while (count < 16) {
            //unsigned char ch = (uchar)cipher[count];
            restored_image.at<Vec3b>(row, col)[channel] =
(uchar)recovered[count];
            channel++;
            if (channel > 2) {
                channel = 0;
                col++;
            }
            if (col > cols) {
                col = 0;
                row++;
            }
            count++;
        }
#if 0
        StreamTransformationFilter filter(d);
        filter.Put((const byte*)cipher.data(), cipher.size());
        filter.MessageEnd();
        const size_t ret = filter.MaxRetrievable();
        recovered.resize(ret);
        filter.Get((byte*)recovered.data(), recovered.size());
#endif

    }
```

```
    catch (const CryptoPP::Exception& e)
    {
        cerr << e.what() << endl;
        exit(1);
    }

}

void Init_key(byte* genkey, size_t size) {
    // Initialization of key data
    for (size_t i = 0; i < size; ++i) {
        genkey[i] = rand();
    }
}
```

References

- CryptoPP Wiki Crypto++ Wiki (cryptopp.com)
- OpenCV documentation OpenCV Documentation (vovkos.github.io)
- StackOverFlow c++ - Loading & Saving Bitmap Images In OpenCV (OpenCV Version = 3.1.0) - Stack Overflow
- Youtube How To Install OpenCV C++ and Use It With Microsoft Visual Studio - YouTube
- Youtube Compiling Crypto++ in Microsoft Visual Studio 2019/2017 (with Cryptopp-PEM) - YouTube