

Practice 4

Secret-key cryptography

Using Crypto++

Session lab 4 | Monday, April 27, 2020

By Joel Harim Hernández Javier

Description

In this session we will start using a cryptographic library of your choice. Please do the following programming exercises on your own. Use only one programming language (C/C++, Java, Python).

It is recommended that you try to solve the exercises on your own. You can discuss the solution to the exercises with your colleagues but you should not copy source code. If copying is detected, that may immediately lead to a grade less than 6.

Programming Exercises

1. Choose a cryptographic library for one of the programming languages mentioned above.
2. Find how to do the following and test it.
 - a. Use a cryptographically secure pseudorandom generator.
 - b. Key generation for secret-key cryptography.
 - c. Encryption and decryption using a stream cipher. Find out which are the stream ciphers available in the cryptographic library of your choice.
 - d. Use of modes of operation. Prove all the traditional modes of operation we studied in class: ECB, CBC, CTR, OFB, CFB.
 - e. Encryption and decryption combining a block cipher and each mode of operation. Use files of different sizes (start at 100kb) to prove this point.

Products

- Your personal information, date of the lab session and the topic that we are studying in this lab session.
- Mention which cryptographic library you chose and briefly explain why you chose it.
- Briefly explain how you implemented each point in exercise 2 in the cryptographic library of your choice. Also describe the problems you had.
- Include screen captures showing your tests for each point of exercise 2.

Crypto++ for C++

Chosen library

I chose Crypto++ because it is one of the few that is built in C++ (the other one I found was Botan, which one has less features and it's newer), **its documentation is very complete, clear and easy to understand**, and has a lot of interesting features.

In my personal opinion, it's one of the most complete.

In my search, I found [this table](#), comparing multiple crypto libraries:

Name	Development language	License
Botan	C++	Simplified BSD
Bouncy Castle	Java / C#	MIT
cryptlib	C	Sleepycat License
Crypto++	C++	Boost Software License
GnuTRS	C	GNU LGPL v2.1+
Libgcrypt	C	GNU LGPL v2.1+
libsodium	C	ISC license
NaCl	C	Public domain ?
Nettle	C	GNU GPL v2+ or GNU LGPL v3
NSS	C	MPL 2.0
OpenSSL	C	Apache Licence 1.0 and 4-Clause BSD Licence
RSA BSAFE Crypto-C Micro Edition	C	Proprietary
RSA BSAFE Crypto-J	Java	Proprietary
wolfCrypt	C	GPL v2 or commercial license
mbed TLS	C	Apache Licence 2.0

Most of them are written in C! C is so powerful, but currently **I'm more comfortable using C++ than C.**

Testing Crypto++

Pseudorandom generations

Crypto++ offers an `AutoSeededRandomPool`, which takes entropy from the operative system (On Linux uses `/dev/random` or `/dev/urandom`, on Windows, it uses `CryptGenRandom`).

Because it takes entropy automatically, **its results are not reproducible**.

```
// Create a pseudo random number generator
CryptoPP::AutoSeededRandomPool prng;


// Internally creates a Integer with the pseudorandom generator and the length
for (int i = 0 ; i < 10; ++i)
    std::cout << i << " " << std::hex << CryptoPP::Integer(prng, 64) << "\n";
```

```
Cryptography/P04/code_examples/pseudorandom_generator/build on master [!?]
→ ./pseudorandom_autoseeded
0 4765a028b27f41c8h
1 2d6785f91833d563h
2 5c1c3ee19dcc399h
3 f908466b54c64405h
4 32e9aa31ffd5b395h
5 9e46c6197cb1663fh
6 b69d018eb0b27642h
7 387e2975e699da6h
8 20b4049eca1153ech
9 fa674f7ec15a5526h
```

A reproducible generation of bits is also available through OFB algorithm:

```
CryptoPP::OFB_Mode<CryptoPP::AES>::Encryption prng;
prng.SetKeyWithIV(seed, 32, seed + 32, 16);

for (unsigned int i = 0; i < 10; ++i)
    std::cout << i << " " << std::hex << CryptoPP::Integer(prng, 64) << "\n";
```

Cryptography/P04/code_examples/pseudorandom_generator/build on  master [!?]

→ ./ofb_random

0 a20471f751581822h

1 5027caba22e7fb39h

2 d956b2c8bac01d24h

3 90553ef7a1562211h

4 f0ea487095114237h

5 e1d92237cd165892h

6 77cda90e3f8e61d7h

7 aeee3c6b793f1b52h

8 691bd4ea45d2c4f5h

9 5ff70b1cf4909867h

Key generation

Key generation of keys is the same as the generation of random numbers because at the end are string of bits.

These 'strings' or blocks of bits has its own structure in Crypto++: `SecByteBlock`, and, to generate the random bits, it's used `OS_GenerateRandomBlock` as following example:

```
// We declare a byte block of AES::DEFAULT_KEYLENGTH (16 bytes)
CryptoPP::SecByteBlock key(CryptoPP::AES::DEFAULT_KEYLENGTH);
// Where will be storage
std::string k;

// Fill with OS entropy
CryptoPP::OS_GenerateRandomBlock(true, key, key.size());

// Hex encoder to k
CryptoPP::HexEncoder hex(new CryptoPP::StringSink(k));
hex.Put(key, key.size());
hex.MessageEnd();

// Now with the iv
CryptoPP::SecByteBlock iv(CryptoPP::AES::BLOCKSIZE);
CryptoPP::OS_GenerateRandomBlock(false, iv, iv.size());
std::string v;
hex.Detach(new CryptoPP::StringSink(v));
hex.Put(iv, iv.size());
hex.MessageEnd();

std::cout << "Key: " << k << "\n";
std::cout << "IV: " << v << "\n";
```

```
Cryptography/P04/code_examples/secret_key_generation/build on  master [!?]
→ ./secretKeyGeneration
Key: 1BE224E6D5C81A6A26CC93334146F3ED
IV: 862115C79B604F2A68FAC85DE2CC6924
```

Stream ciphers

Crypto++ implements the following stream ciphers:

- | | |
|---------------------|----------------|
| → XChaCha20Poly1305 | → HC-256 |
| → ChaCha20Poly1305 | → HC-128 |
| → XChaCha20 | → Rabbit |
| → XChaCha20 | → ARC4 |
| → ChaChaTLS | → SEAL |
| → ChaCha20 | → WAKE |
| → Salsa20 | → WAKE-OFB |
| → Sosemanuk | → BlumBlumShub |
| → Panama | |

The following example shows its usage:

```
// Message
std::string message = "Hello world";
// Could be random generated with SecBlock and OS_GenerateRandomBlock
std::string key = "123";

// Initialization
CryptoPP::ARC4 arc4((CryptoPP::byte*)key.data(), key.size());

// Encryption
arc4.ProcessData((CryptoPP::byte*)message.data(),
(CryptoPP::byte*)message.data(), message.size());
std::cout << "Encrypted message: " << message << "\n";

// Reset
arc4.SetKey((CryptoPP::byte*)key.data(), key.size());

// Decryption
arc4.ProcessData((CryptoPP::byte*)message.data(),
(CryptoPP::byte*)message.data(), message.size());
std::cout << "Decrypted message: " << message << "\n";
```

```
Cryptography/P04/code_examples/stream_ciphers/build on master [!?]
→ ./arc4
Encrypted message: 00Ts4Wg
Decrypted message: Hello world
```

Encryption and decryption using block ciphers

All the following examples uses CBC mode and likely the same implementation of `encrypt` and `decrypt` functions:

```
// Encrypting AES / CBC mode
std::string encrypt(
    std::string input,
    CryptoPP::CBC_Mode<CryptoPP::AES>::Encryption encryptor
) {
    std::string result;

    CryptoPP::StringSource ss (input, true,
        new CryptoPP::StreamTransformationFilter(encryptor,
            new CryptoPP::StringSink( result )));
    return result;
}

// Decrypting AES / CBC mode
std::string decrypt(std::string input,
    CryptoPP::CBC_Mode<CryptoPP::AES>::Decryption decryptor
) {
    std::string result;

    CryptoPP::StringSource ss (input, true,
        new CryptoPP::StreamTransformationFilter(decryptor,
            new CryptoPP::StringSink( result )));

    return result;
}
```

AES:

```
// Rand generator
CryptoPP::AutoSeededRandomPool rand;

// Generating a random key
CryptoPP::SecByteBlock key(0x00, CryptoPP::AES::DEFAULT_KEYLENGTH);
rand.GenerateBlock(key, key.size());
std::cout << "Using key: " << byteBlockToString(key) << "\n";

// Generating a random IV
CryptoPP::SecByteBlock iv(0x00, CryptoPP::AES::BLOCKSIZE);
rand.GenerateBlock(iv, iv.size());
```

```

std::cout << "Using IV: " << byteBlockToString(iv) << "\n";

// Input message
std::string input = "Hello world!, this is a very long message to prove the AES
encryption with CBC mode provided by the Crypto++ library!";
std::string cipher, encoded, recovered;

// CBC Encryptor and Decryptor
CryptoPP::CBC_Mode<CryptoPP::AES>::Encryption encryptor(key, key.size(), iv);
CryptoPP::CBC_Mode<CryptoPP::AES>::Decryption decryptor(key, key.size(), iv);

// Encrypting
cipher = encrypt(input, encryptor);
std::cout << "Encrypted message: " << toHex(cipher) << "\n";

// Decrypting
recovered = decrypt(cipher, decryptor);
std::cout << "Decrypted message: " << recovered << "\n";

```

```

Cryptography/P04/code_examples/block_ciphers/build on master [!?]
→ ./aes
Using key: BB60364F6FE1BA2B2F8D747C59D07B70
Using IV: 4108E15B50FEB8506D1B7F3D213E4410
Encrypted message: 3DD7B7BA4E86CE655FAC6D342070C3D5EBFD73EBC84FD84CA280A73FDC2B3D5925FB78E7AAA48EA
7E13612A45E1401579887700AC656ED60438D03C91FE05A3B8C80B7E398
Decrypted message: Hello world!, this is a very long message to prove the AES encryption with CBC

```

3DES

```

// Rand generator
CryptoPP::AutoSeededRandomPool rand;

// Generating a random key
CryptoPP::SecByteBlock key(0x00, CryptoPP::DES_EDE3::DEFAULT_KEYLENGTH);
rand.GenerateBlock(key, key.size());
std::cout << "Using key: " << byteBlockToString(key) << "\n";

// Generating a random IV
CryptoPP::SecByteBlock iv(0x00, CryptoPP::DES_EDE3::BLOCKSIZE);
rand.GenerateBlock(iv, iv.size());
std::cout << "Using IV: " << byteBlockToString(iv) << "\n";

// Input message
std::string input = "Hello world!, this is a very long message to prove the
TripleDES encryption with CBC mode provided by the Crypto++ library!";

```



```

std::string cipher, encoded, recovered;

// CBC Encryptor and Decryptor
CryptoPP::CBC_Mode<CryptoPP::DES_EDE3>::Encryption encryptor(key, key.size(),
iv);
CryptoPP::CBC_Mode<CryptoPP::DES_EDE3>::Decryption decryptor(key, key.size(),
iv);

// Encrypting
cipher = encrypt(input, encryptor);
std::cout << "Encrypted message: " << toHex(cipher) << "\n";

// Decrypting
recovered = decrypt(cipher, decryptor);
std::cout << "Decrypted message: " << recovered << "\n";

```

```

Cryptography/P04/code_examples/block_ciphers/build on master [!?]
→ ./tripleDes
Using key: 098B4EF9654B2AFE0F54BE792908B121189B828E06254293
Using IV: B8144C589E7635B5
Encrypted message: 04153FB47958F806ACAF68B66ABA0FF38E8C469F89F34B1CD5B9FD9A8A9F6984EB81E4CF62800428865F513072A5CD46CB
CC3D5FC735E525BEDA71D35A79E87125FADD9ABBFEE8042B95DEB17B07
Decrypted message: Hello world!, this is a very long message to prove the TripleDES encryption with CBC mode provided

```

IDEA:

```

// Rand generator
CryptoPP::AutoSeededRandomPool rand;

// Generating a random key
CryptoPP::SecByteBlock key(0x00, CryptoPP::IDEA::DEFAULT_KEYLENGTH);
rand.GenerateBlock(key, key.size());
std::cout << "Using key: " << byteBlockToString(key) << "\n";

// Generating a random IV
CryptoPP::SecByteBlock iv(0x00, CryptoPP::IDEA::BLOCKSIZE);
rand.GenerateBlock(iv, iv.size());
std::cout << "Using IV: " << byteBlockToString(iv) << "\n";


// Input message
std::string input = "Hello world!, this is a very long message to prove the IDEA
encryption with CBC mode provided by the Crypto++ library!";
std::string cipher, encoded, recovered;

// CBC Encryptor and Decryptor
CryptoPP::CBC_Mode<CryptoPP::IDEA>::Encryption encryptor(key, key.size(), iv);
CryptoPP::CBC_Mode<CryptoPP::IDEA>::Decryption decryptor(key, key.size(), iv);

```

```
// Encrypting
cipher = encrypt(input, encryptor);
std::cout << "Encrypted message: " << toHex(cipher) << "\n";

// Decrypting
recovered = decrypt(cipher, decryptor);
std::cout << "Decrypted message: " << recovered << "\n";
```

Cryptography/P04/code_examples/block_ciphers/build on  master [!?]

→ ./idea

Using key: 49182C1AEB249BFDE41924740328FC0E

Using IV: 058901FDF6739D86

Encrypted message: A96844C1F16374794B8584EFB0008DAC7CAB7F73304CB97C915D8A73BA4A2EBC5BE05957CB0625E424D3D14D76C670F08705BC3C6E33BB9AC021A1C2777F188F8B27808156

Decrypted message: Hello world!, this is a very long message to prove the IDEA encryption with CBC mode provided b

Modes of operation

The same code and logic used as above, all modes uses AES as cipher algorithm (showing just captures)

```
Cryptography/P04/code_examples/modes_of_operation/build on master [!?]
→ ls
cbc cfb ctr ecb ofb

Cryptography/P04/code_examples/modes_of_operation/build on master [!?]
→ ./ecb
Using key: 114070D87131B4F73F6239360FC613CE
Encrypted message: 3AB8B52DD34EA6D311B87298E064D4E12F4328D7A51A3ED37EEB6813B96A8CB789B43AC4CE31107BB7C001B0CCDB7079BCB07F0
0455F0E7287EBB1CF72B3D7F534
Decrypted message: Hello world!, this is a very long message to test ECB mode with AES provided by the Crypto++ library!

Cryptography/P04/code_examples/modes_of_operation/build on master [!?]
→ ./cbc
Using key: 6F4E80468779EFA06230838C0332B737
Using IV: B4AC921BFCAEF2DF2A0433FCDA45E12F
Encrypted message: 4836690BF6EEC71AF9CF3DDAFC1AB6268ABF432495DED8E28A13E6A0AB37C72D5F4E4C4F4281976E420FD7F048895774E9D870D
6505F145FA8C763B1C629EF7E83
Decrypted message: Hello world!, this is a very long message to test CBC mode with AES provided by the Crypto++ library!

Cryptography/P04/code_examples/modes_of_operation/build on master [!?]
→ ./ctr
Using key: 795E7A04AD01886DD3053E5D724397BD
Using IV: F33356CF5F03F822290B46D2EC9B29E3
Encrypted message: 412AE22A68D904D6CD9777FC0D338616ACF18BD427A845E6BAE2AB24B18FE655C85D1975A9CA5364F94D19A7C712AE46E9DC5A7
21673
Decrypted message: Hello world!, this is a very long message to test CTR mode with AES provided by the Crypto++ library!

Cryptography/P04/code_examples/modes_of_operation/build on master [!?]
→ ./ofb
Using key: DCD8A56FC59271496A6DD890509988AD
Using IV: B28DCC18B1D1DC5AE414B8EFC709155
Encrypted message: 659611A6EDF4EE3136543BB5C80CC0758AA52A8AAE32B070771F31C1124AC31499672FC1E1B743F49EBF82E1CD799B9FC65CBD5
76E7A
Decrypted message: Hello world!, this is a very long message to test OFB mode with AES provided by the Crypto++ library!

Cryptography/P04/code_examples/modes_of_operation/build on master [!?]
→ ./cfb
Using key: 85A02532FFA5AA9CE03A11AD6FD8D924
Using IV: 9F5BD79241FAF3B6997BB276B27DE451
Encrypted message: E37EA1EB55AFA247C124BCB62E5A5D84DF61A638D1B46A14B3655E5198F9361894D1277F5E8C1DDE4F756956284D3E619869127
EA2CB
Decrypted message: Hello world!, this is a very long message to test CFB mode with AES provided by the Crypto++ library!
```

File encryption

For file encryption, as there are many modes and many algorithms, I tried to create a general function that just asks for the input string, the algorithm to use, the mode of operation, the key and the initialization vector, but, due to the way the class hierarchy tree of Crypto++ is implemented, I couldn't figure out how to code this general function compactly and elegantly.

I ended up with a gigantic function that repeats code and has a lot of nested if statements with poor readability and null reusability :(

(showing `encrypt` function)

```
// Really ugly implementation, I can't use functions because the hierarchy class  
tree of Crypto++  
inline  
std::string encrypt(std::string & input, std::string algorithm, std::string  
mode, std::string key, std::string iv){  
  
    CryptoPP::SecByteBlock k = key == "" ? ranBlock(getKeySize(algorithm),  
"key") : getHexFromString(key);  
    CryptoPP::SecByteBlock v = (mode != "ecb" && iv == "") ?  
ranBlock(getIvSize(algorithm), "iv") : getHexFromString(iv);  
  
    std::string result;  
  
    std::cout << "Encrypting in mode: " << mode << " with: " << algorithm <<  
".\n";  
  
    if (algorithm == "aes") {  
        auto l = modes<CryptoPP::AES>();  
  
        if (mode == "ecb"){  
  
            auto e = l.ecbE;  
            mode == "ecb" ? e.SetKey(k, k.size()) : e.SetKeyWithIV(k, k.size(),  
v);  
  
            CryptoPP::StringSource ss(input, true, new  
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));  
  
        }  
  
        if (mode == "cbc"){  
  
            auto e = l.cbcE;  
            e.SetKeyWithIV(k, k.size(), v);  

```

```

        CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

    }

    if (mode == "ctr"){

        auto e = l.ctrE;
        e.SetKeyWithIV(k, k.size(), v);
        CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

    }

    if (mode == "ofb"){

        auto e = l.ofbE;
        e.SetKeyWithIV(k, k.size(), v);
        CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

    }

    if (mode == "cfb"){

        auto e = l.cfbE;
        e.SetKeyWithIV(k, k.size(), v);
        CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

    }

}

if (algorithm == "3des"){
    auto l = modes<CryptoPP::DES_EDE3>();

    if (mode == "ecb"){

        auto e = l.ecbE;
        mode == "ecb" ? e.SetKey(k, k.size()) : e.SetKeyWithIV(k, k.size(),
v);

        CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

    }
}

```

```

        if (mode == "cbc"){

            auto e = l.cbcE;
            e.SetKeyWithIV(k, k.size(), v);
            CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

        }

        if (mode == "ctr"){

            auto e = l.ctrE;
            e.SetKeyWithIV(k, k.size(), v);
            CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

        }

        if (mode == "ofb"){

            auto e = l.ofbE;
            e.SetKeyWithIV(k, k.size(), v);
            CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

        }

        if (mode == "cfb"){

            auto e = l.cfbE;
            e.SetKeyWithIV(k, k.size(), v);
            CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

        }
    }

    if (algorithm == "idea") {
        auto l = modes<CryptoPP::IDEA>();

        if (mode == "ecb"){

            auto e = l.ecbE;
            mode == "ecb" ? e.SetKey(k, k.size()) : e.SetKeyWithIV(k, k.size(),
v);

```

```

        CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

    }

    if (mode == "cbc"){

        auto e = l.cbcE;
        e.SetKeyWithIV(k, k.size(), v);
        CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

    }

    if (mode == "ctr"){

        auto e = l.ctrE;
        e.SetKeyWithIV(k, k.size(), v);
        CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

    }

    if (mode == "ofb"){

        auto e = l.ofbE;
        e.SetKeyWithIV(k, k.size(), v);
        CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

    }

    if (mode == "cfb"){

        auto e = l.cfbE;
        e.SetKeyWithIV(k, k.size(), v);
        CryptoPP::StringSource ss(input, true, new
CryptoPP::StreamTransformationFilter(e, new CryptoPP::StringSink( result )));

    }

    }

    return result;

}

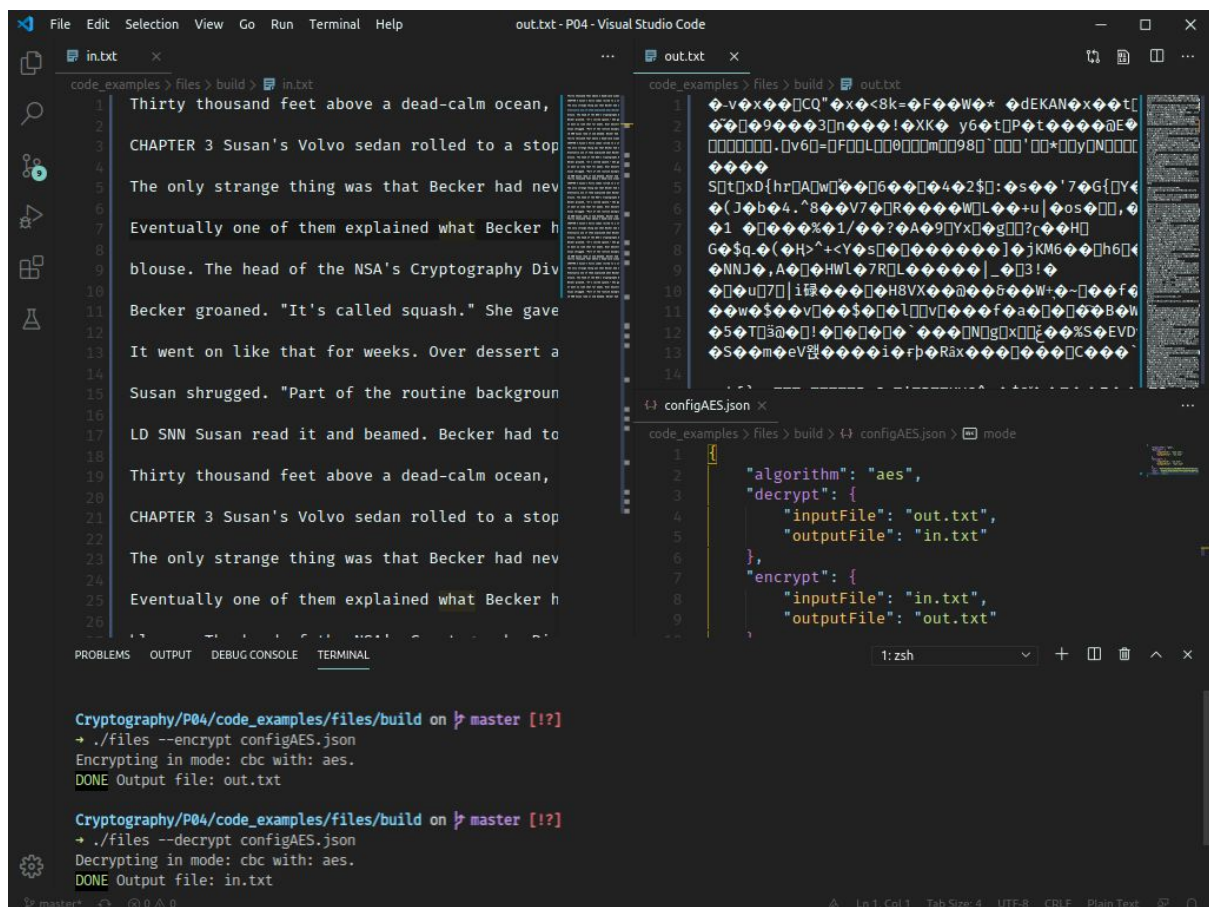
```


That was the only (and more annoying) problem I had using Crypto++.

Encrypting file with AES and CBC mode (Input JSON file configuration for the program):

```
{
  "algorithm": "aes",
  "decrypt": {
    "inputFile": "out.txt",
    "outputFile": "in.txt"
  },
  "encrypt": {
    "inputFile": "in.txt",
    "outputFile": "out.txt"
  },
  "iv": "FA57ECAE32CC502BD91F6CAF2476C151",
  "key": "8C0581E4BCAD5D1FF4FA669F63B8FC81",
  "mode": "ecb"
}
```

Screenshot:



I encrypted and decrypted, and the input text was the same, so, it works fine.

Encrypting file with 3DES and CTR mode (Input JSON file configuration for the program):

```
{
  "algorithm": "3des",
  "decrypt": {
    "inputFile": "out2.txt",
    "outputFile": "in2.txt"
  },
  "encrypt": {
    "inputFile": "in2.txt",
    "outputFile": "out2.txt"
  },
  "iv": "832CA97DC674F612",
  "key": "7926AF742C8FFAF9429D6F238D7ED79E53A27B6D5C91EFAE",
  "mode": "ctr"
}
```

Screenshot:

```
code_examples > files > build > in2.txt
1 CHAPTER 4 The crypto door beeped once, waking
2
3 Pushing through the center of the floor like t
4
5 The only way to unscramble the message was to
6
7 hands make light work. Its three million proce
8
9 CHAPTER 5 "Where is everyone?" Susan wondered
10
11 golden girl. But Strathmore's young cryptograp
12
13 CHAPTER 4 The crypto door beeped once, waking
14
15 Pushing through the center of the floor like t
16
17 The only way to unscramble the message was to
18
19 hands make light work. Its three million proce
20
21 CHAPTER 5 "Where is everyone?" Susan wondered
22
23 golden girl. But Strathmore's young cryptograp
24
25 CHAPTER 4 The crypto door beeped once, waking
26
```

```
code_examples > files > build > out2.txt
1 3s<3Ny'it0s0FX7LEsft?6XI
2 0500K-)0000[000T0o00_01I(X)e--Jp
3 y000>0 000(c0000T0:0D`0@00s
4 000Z0<00.00s0=|^00^0<00'0y0g3034c0A,
5 00ee0~0f8N_0<,00
6 "3s0[0.g0000000fAQ0m0j;0000
7 00-00000>)0'0000f"m0j0A0~000o0000T0000000rV0
8 0_000000010G40<05'
9 0u0Z0=00000a|K0000000000K\02, c0
10 q0[000000W02-?s-0l0.000,0}p0p
11 i0q000d6j0e000G<u080l 0cJ0NH0a?0n0
12 00I0!0P0""0{~0}0000<0R00|V0000-00[
13 0A010j000100i00f0i00000r$[T00000W0000k000#00
14
```

```
config3DES.json
code_examples > files > build > config3DES.json > mode
8   "inputFile": "in2.txt",
9   "outputFile": "out2.txt"
10 },
11 "iv": "832CA97DC674F612",
12 "key": "7926AF742C8FFAF9429D6F238D7ED79E53
13 "mode": "ctr"
14 }
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Cryptography/P04/code_examples/files/build on master [!?]
+ ./files --encrypt config3DES.json
Encrypting in mode: ctr with: 3des.
DONE Output file: out2.txt

Cryptography/P04/code_examples/files/build on master [!?]
+ ./files --decrypt config3DES.json
Decrypting in mode: ctr with: 3des.
DONE Output file: in2.txt
```

I encrypted and decrypted, and the input text was the same, so, it works fine, again.

My program supports AES, 3DES, IDEA with ECB, CBC, CTR, OFB, CFB modes, also generate random key and iv for encryption process if it's necessary.

The complete source code can be found at my personal Github 😊
github.com/JoelHernandez343/Cryptography/tree/master/P04