信息安全技术作业（二）

AES 加密

中山大学计算机学院　计算机科学与技术

19335174　施天予

目录

1　问题描述 **2**

2　实验原理 **2**

3　解决方案 **3**

3.1　初始化 3

3.2　密匙和 IV 3

3.3　工作模式 3

3.4　填充模式 4

3.5　加密过程 4

3.6　解密过程 4

4　实验结果 **5**

A　AES 加密解密源代码 **5**

一、问题描述

学习一个能够实现典型分组密码，如 DES 和 AES 的密码软件库（任何编程语言皆可），简单介绍它的功能，以及它在分组密码的工作模式和填充模式上的设置方法，以 AES 加密为例，给出实现不同工作模式和填充模式的加解密源代码。

二、实验原理

本人学习的是 AES 密码软件库。高级加密标准 (AES,Advanced Encryption Standard) 为最常见的对称加密算法 (微信小程序加密传输就是用这个加密算法的)。对称加密算法也就是加密和解密用相同的密钥，具体的加密流程如下图：

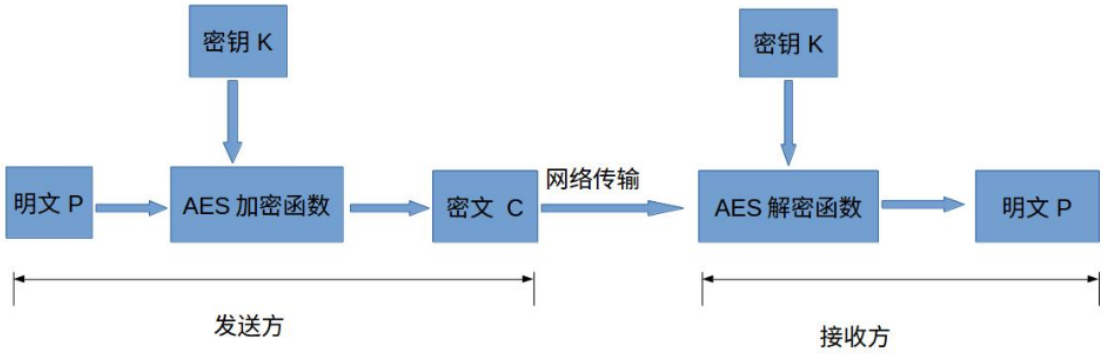

图 1: AES 工作原理

各个部分的作用与意义

- **明文 P**：没有经过加密的数据。
- **密钥 K**：用来加密明文的密码，在对称加密算法中，加密与解密的密钥是相同的。密钥为接收方与发送方协商产生，但不可以直接在网络上传输，否则会导致密钥泄漏，通常是通过非对称加密算法加密密钥，然后再通过网络传输给对方，或者直接面对面商量密钥。密钥是绝对不可以泄漏的，否则会被攻击者还原密文，窃取机密数据。
- **AES 加密函数**：设 AES 加密函数为 E，则 C = E(K, P), 其中 P 为明文，K 为密钥，C 为密文。也就是说，把明文 P 和密钥 K 作为加密函数的参数输入，则加密函数 E 会输出密文 C。
- **密文 C**：经加密函数处理后的数据
- **AES 解密函数**：设 AES 解密函数为 D，则 P = D(K, C), 其中 C 为密文，K 为密钥，P 为明文。也就是说，把密文 C 和密钥 K 作为解密函数的参数输入，则解密函数会输出明文 P。

对称加密算法与非对称加密算法的区别

- **对称加密算法**：加密和解密用到的密钥是相同的，这种加密方式加密速度非常快，适合经常发送数据的场合。缺点是密钥的传输比较麻烦。
- **非对称加密算法**：加密和解密用的密钥是不同的，这种加密方式是用数学上的难解问题构造的，通常加密解密的速度比较慢，适合偶尔发送数据的场合。优点是密钥传输方便。常见的非对称加密算法为 RSA、ECC 和 EIGamal。

实际中，一般是通过 RSA 加密 AES 的密钥，传输到接收方，接收方解密得到 AES 密钥，然后发送方和接收方用 AES 密钥来通信。

三、解决方案

1. 初始化

首先需要初始化一个随机密匙池，并初始化一个 encoder

```
AutoSeededRandomPool prng;
HexEncoder encoder(new FileSink(std::cout));
```

2. 密匙和 IV

初始化密匙,可以选择不同的密匙长度DEFAULT_KEYLENGTH,MIN_KEYLENGTH,MAX_KEYLENGTH。

```
SecByteBlock key(AES::DEFAULT_KEYLENGTH);
// AES::MIN_KEYLENGTH 来生成192位的密钥
// AES::MAX_KEYLENGTH 来生成256位的密钥
prng.GenerateBlock(key, key.size());
```

初始化 IV

```
SecByteBlock iv(AES::BLOCKSIZE);
prng.GenerateBlock(iv, iv.size());
```

3. 工作模式

分组密码有五种工作体制：

1. 电码本模式（Electronic Codebook Book (ECB)）
2. 密码分组链接模式（Cipher Block Chaining (CBC)
3. 计算器模式（Counter (CTR)
4. 密码反馈模式（Cipher FeedBack (CFB)）
5. 输出反馈模式（Output FeedBack (OFB)

```
CBC_Mode<AES>::Encryption e;
// ECB_Mode<AES>::Encryption e;
```

```
// CTR_Mode<AES>::Encryption e;
// CFB_Mode<AES>::Encryption e;
// OFB_Mode<AES>::Encryption e;
e.SetKeyWithIV(key, key.size(), iv, iv.size()); // 设置加密程序的密钥和IV
```

4. 填充模式

填充有六种

1. **NoPadding**：不填充。缺点就是只能加密长为 128bits 倍数的信息，一般不会使用。
2. **PKCS5**：缺几个字节就填几个缺的字节。
3. **PKCS7**：缺几个字节就填几个缺的字节。
4. **ISO 10126**：最后一个字节是填充的字节数（包括最后一字节），其他全部填随机数。
5. **NANSI X9.23**：跟 ISO 10126 很像，只不过 ANSI X9.23 其他字节填的都是 0 而不是随机数。
6. **ZerosPadding**：全部填充 0x00，无论缺多少全部填充 0x00，已经是 128bits 倍数仍要填充。

在 StreamTransformationFilter 中，可以设置填充模式的参数

```
StreamTransformationFilter(StreamTransformation &c, BufferedTransformation *
    ↪ attachment=NULL, BlockPaddingScheme padding=DEFAULT_PADDING)
// BlockPaddingScheme设置填充模式
// 1.NO_PADDING 不填充
// 2.ZEROS_PADDING 用‘0’填充
// 3.PKCS_PADDING
// 4.ONE_AND_ZEROS_PADDING
// 5.允许用户指定填充的字符
```

5. 加密过程

加密明文，返回密文和密文长度

```
StringSource s(plainText, true, new StreamTransformationFilter(e, new StringSink(
    ↪ cipherText)));
```

6. 解密过程

解密过程与加密过程差不多，注意要设置相同的工作模式

```
CBC_Mode<AES>::Decryption d;
d.SetKeyWithIV(key, key.size(), iv);
StringSource s(cipherText, true, new StreamTransformationFilter(e, new StringSink(
    ↪ recoverdText))
```

四、实验结果

编译并运行程序

```
g++ test.cpp -o test -lcryptopp
./test
```

得到结果如图2所示，发现恢复的明文正确。

```
sty@ubuntu:~$ g++ test.cpp -o test -lcryptopp
sty@ubuntu:~$ ./test
plain text: This is John speaking
key: 4BF87DE99C9C4E5A0CDCCCF1E927DAA2ADFFE0D0FB83BED447FAE1902609357C
iv: B32084DCBA9A0FC588D2106761882033
cipher text: 9014174FC3F44CEEE6E69E9A79A5290141FAC4C227D3412ACE5478C6E144F404
recovered text: This is John speaking
```

图 2: 运行结果

参考文献

[1] AES 加密.https://cryptopp.com/wiki/Advanced_Encryption_Standard,2021.

[2] AES 加密算法的详细介绍与实现.https://blog.csdn.net/qq_28205153/article/details/55798628,2017.

附录 A. AES 加密解密源代码

```
#include <bits/stdc++.h>
#include <cryptopp/files.h>
#include <cryptopp/aes.h>
#include <cryptopp/filters.h>
#include <cryptopp/modes.h>
#include <cryptopp/hex.h>
#include <cryptopp/osrng.h>

using namespace std;

int main(int argc, char *argv[])
{
    using namespace CryptoPP;

    AutoSeededRandomPool prng;
    HexEncoder encoder(new FileSink(std::cout));

    // 可以选择DEFAULT_KEYLENGTH、MIN_KEYLENGTH、MAX_KEYLENGTH
    SecByteBlock key(AES::MAX_KEYLENGTH);
```

```
20      SecByteBlock iv(AES::BLOCKSIZE);
21
22      // 初始化密匙和IV
23      prng.GenerateBlock(key, key.size());
24      prng.GenerateBlock(iv, iv.size());
25
26      std::string plain = "This is John speaking"; // 明文
27      std::string cipher, recovered;          // 密文和恢复的明文
28      std::cout << "plain text: " << plain << std::endl;
29
30      try
31      {
32          CBC_Mode<AES>::Encryption e;       // 选择不同的加密模式
33          e.SetKeyWithIV(key, key.size(), iv); // 设置密匙和IV
34
35          StringSource s(plain, true,
36              new StreamTransformationFilter(e,
37                  new StringSink(cipher))     // StreamTransformationFilter
38          );                                  // StringSource
39      }
40      catch (const Exception &e)
41      {
42          std::cerr << e.what() << std::endl;
43          exit(1);
44      }
45
46      std::cout << "key: ";
47      encoder.Put(key, key.size());
48      encoder.MessageEnd();
49      std::cout << std::endl;
50
51      std::cout << "iv: ";
52      encoder.Put(iv, iv.size());
53      encoder.MessageEnd();
54      std::cout << std::endl;
55
56      std::cout << "cipher text: ";
57      encoder.Put((const byte *)&cipher[0], cipher.size());
58      encoder.MessageEnd();
59      std::cout << std::endl;
60
61      try
62      {
63          CBC_Mode<AES>::Decryption d;       // 解密模式需要对应之前的加密模式
64          d.SetKeyWithIV(key, key.size(), iv);
```

```
65
66          StringSource s(cipher, true,
67              new StreamTransformationFilter(d,
68                  new StringSink(recovered)) // StreamTransformationFilter
69          );                                 // StringSource
70
71          std::cout << "recovered text: " << recovered << std::endl;
72      }
73      catch (const Exception &e)
74      {
75          std::cerr << e.what() << std::endl;
76          exit(1);
77      }
78      return 0;
79  }
```