计算机网络安全实验报告

RC4 加密实验

姓名	田丰瑞
班级	软件 73 班
学号	2172213528
电话	18744296191
Email	tianfr@stu.xjtu.edu.cn
日期	2020-5-3

声明

网络安全实验的所有代码已经全部开源于我的 GitHub（仅开源代码部分），项目地址：

https://github.com/tianfr/Internet-Security-ExpCode/

RC4 实验

实验要求

实现 rc4 加解密运算，要求：源代码与运行效果（加解密包含学生信息如姓名班级学号等不少于 200 字）

RC4 加解密原理

RC4 简介

在密码学中，Rivest Cipher 4 是一种流加密算法，密钥长度可变。它加解密使用相同的密钥，因此也属于对称加密算法。

RC4 于 1987 年提出，和 DES 算法一样，是一种对称加密算法，也就是说使用的密钥为单钥（或称为私钥）。但不同于 DES 的是，RC4 不是对明文进行分组处理，而是字节流的方式依次加密明文中的每一个字节，解密的时候也是依次对密文中的每一个字节进行解密。

RC4 算法的特点是算法简单，运行速度快，而且密钥长度是可变的，可变范围为 1-256 字节(8-2048 比特)，在如今技术支持的前提下，当密钥长度为 128 比特时，用暴力法搜索密钥已经不太可行，所以可以预见 RC4 的密钥范围任然可以在今后相当长的时间里抵御暴力搜索密钥的攻击。

RC4 算法原理

在介绍 RC4 算法原理之前，先看看算法中的几个关键变量：

密钥流：RC4 算法的关键是根据明文和密钥生成相应的密钥流，密钥流的长度和明文的长度是对应的，也就是说明文的长度是 500 字节，那么密钥流也是 500 字节。当然，加密生成的密文也是 500 字节，因为密文第 i 字节=明文第 i 字节^密钥流第 i 字节。

密钥 K：长度一般为 256 字节，注意密钥的长度与明文长度、密钥流的长度没有必然关系，本次实验将密匙保存在**“密匙.TXT”**文件中。

S 盒：S 盒的目的是利用密匙中的符号将一个顺序数组打乱。其输入是一个整数（本次实验为 256），输出为一个被密匙打乱的数组。

RC4 的总流程如下：

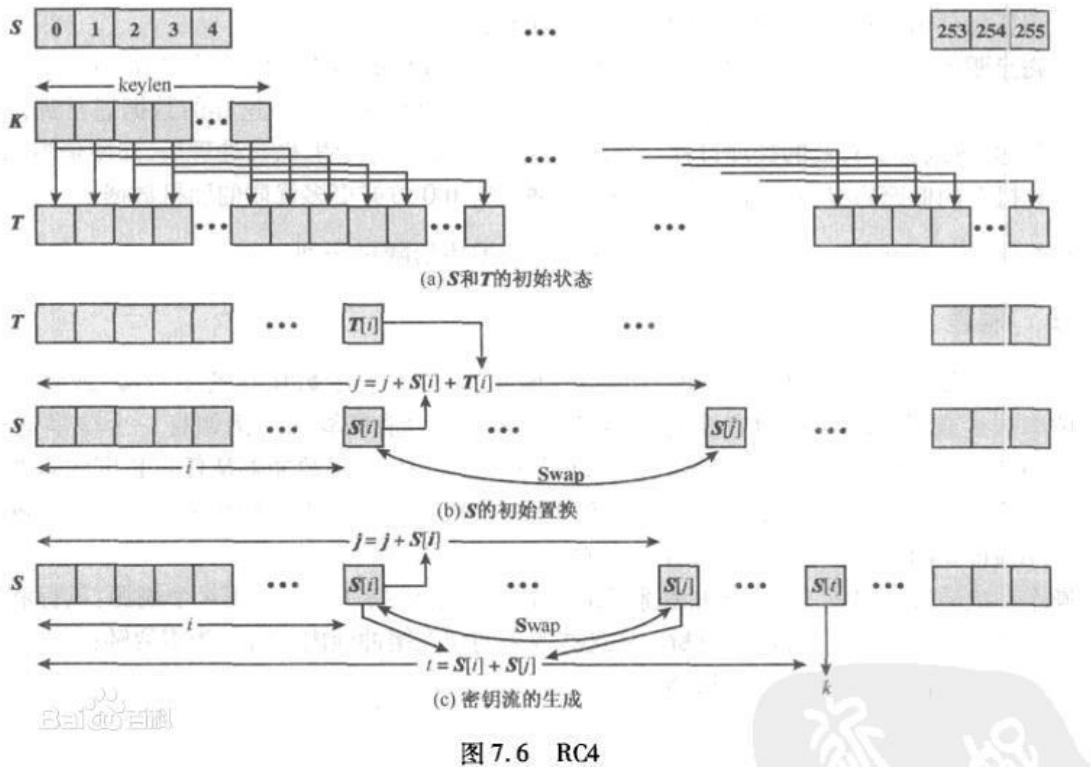


图 7.6 RC4

算法首先在 S 盒中初始化一个 0-255 的有序数组 S[256]，维护一个指针 i,将密匙中的内容读入 key 的变量后，对 i 从 0 遍历到 255，执行如下操作:

- 令 j=(j + S [i] + key[i % len(key)]) % 256
- 交换 i，j 指向的 S 数组中的内容

执行操作后，输出的 S 中的每一项的数字被彻底打乱。类似的，将输入的明文读入到 plain 变量中，对 plain 中每一个字符执行**异或**操作。

RC4 的加密解密的算法过程完全相同，**因为对一个数字进行两次异或操作后该数字保存不变。**

RC4 算法实现

本次实验我使用 Python3.8 开发环境，为了让加密后的密文可读，我将加密后的字符串利用 Base64 库将其转换为 Base64 子节码，使其可读。

S 盒的实现

```
def init_box(key):
    """
    S 盒
    """
    s_box = list(range(256)) #我这里没管秘钥小于 256 的情况，小于 256 应该不断重复填充即可
    j = 0
    for i in range(256):
        j = (j + s_box[i] + ord(key[i % len(key)])) % 256
        s_box[i], s_box[j] = s_box[j], s_box[i]
```

```
    #print(type(s_box)) #for_test
    return s_box
```

RC4 加解密实现

```
def ex_encrypt(plain,box,mode):
    """
    利用 PRGA 生成秘钥流并与密文字节异或，加解密同一个算法
    """

    if mode == '2':
        while True:
            # c_mode = input("输入你的解密模式:Base64 or ordinary\n")
            c_mode = 'Base64'
            if c_mode == 'Base64':
                plain = base64.b64decode(plain)
                plain = bytes.decode(plain)
                break
            elif c_mode == 'ordinary':
                plain = plain
                break
            else:
                print("Something Wrong,请重新新输入")
                continue

    res = []
    i = j =0
    for s in plain:
        i = (i + 1) %256
        j = (j + box[i]) %256
        box[i], box[j] = box[j], box[i]
        t = (box[i] + box[j])% 256
        k = box[t]
        res.append(chr(ord(s)^k))

    cipher = "".join(res)
    #print(cipher)
    if  mode == '1':
        # 化成可视字符需要编码
        print("部分加密后的输出(没经过任何编码):")
        print(cipher[:10])
        with open (u"d:/Cpp/test/密文.txt",'r+',encoding='utf-8') as f:
            f.write(str(base64.b64encode(cipher.encode('utf-8')),'utf-
8'))
```

```
        # base64 的目的也是为了变成可见字符
        print("部分 base64 后的编码:")
        print(str(base64.b64encode(cipher.encode('utf-8')),'utf-
8')[:50])
    if mode == '2':
        print("部分解密后的密文: ")
        print(cipher[:50])
        with open (u"d:/Cpp/test/解密文件.txt",'r+',encoding='utf-8')
as f:
            f.write(cipher)
```

完整代码实现

```
# -*- coding: utf-8 -*-
# Author: CalmCat
# Date: 2020-5-1
# Blog: tianfr.github.io
# Content: 计算机网络安全 RC4 作业

import base64

def get_encoder_message():
    # print("输入你的信息: ")
    # s = input()
    with open (u"d:/Cpp/test/明文.txt",'r+', encoding='utf-8') as f:
        s = f.read()
    print("部分明文: \n",s[:50])
    return s

def get_decoder_message():
    with open (u"d:/Cpp/test/密文.txt",'r+',encoding='utf-8') as f:
        s = f.read()
    print("部分密文: \n",s[:50])
    return s

def get_key():
    # print("输入你的秘钥: ")
    # key = input()
    # if key == '':
    #     key = 'none_public_key'
    with open(u"d:/Cpp/test/密钥流.txt",'r+',encoding='utf-8') as f:
        key = f.read()
    print("部分秘钥: \n",key[:50])
    return key

def init_box(key):
    """
```

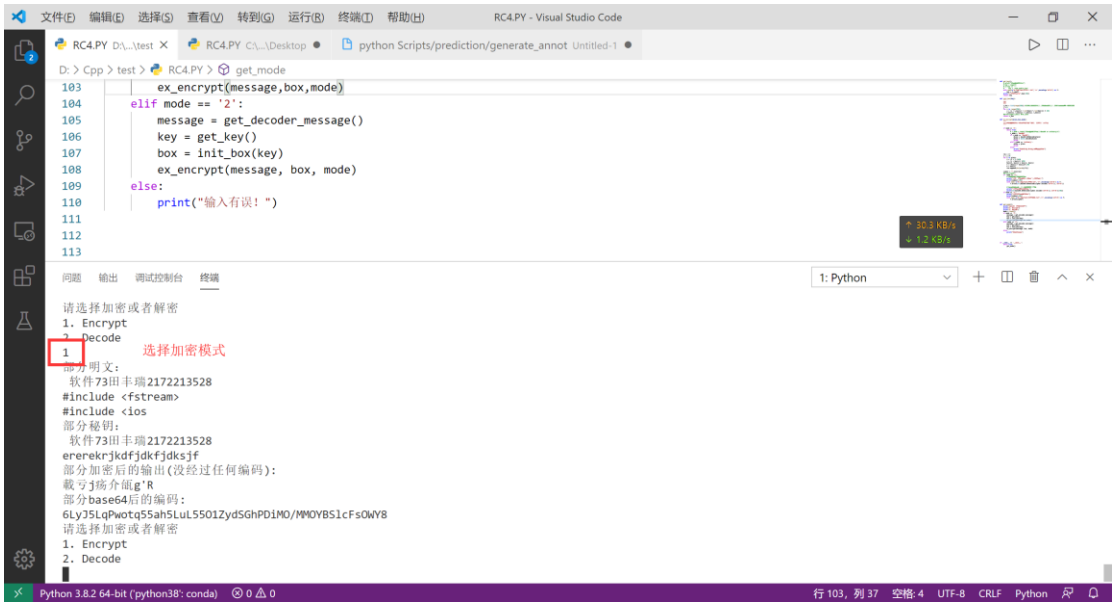
```
    S 盒
    """
    s_box = list(range(256)) #我这里没管秘钥小于 256 的情况，小于 256 应该不断
重复填充即可
    j = 0
    for i in range(256):
        j = (j + s_box[i] + ord(key[i % len(key)])) % 256
        s_box[i], s_box[j] = s_box[j], s_box[i]
    #print(type(s_box)) #for_test
    return s_box

def ex_encrypt(plain,box,mode):
    """
    利用 PRGA 生成秘钥流并与密文字节异或，加解密同一个算法
    """

    if mode == '2':
        while True:
            # c_mode = input("输入你的解密模式:Base64 or ordinary\n")
            c_mode = 'Base64'
            if c_mode == 'Base64':
                plain = base64.b64decode(plain)
                plain = bytes.decode(plain)
                break
            elif c_mode == 'ordinary':
                plain = plain
                break
            else:
                print("Something Wrong,请重新新输入")
                continue

    res = []
    i = j =0
    for s in plain:
        i = (i + 1) %256
        j = (j + box[i]) %256
        box[i], box[j] = box[j], box[i]
        t = (box[i] + box[j])% 256
        k = box[t]
        res.append(chr(ord(s)^k))

    cipher = "".join(res)
    #print(cipher)
    if  mode == '1':
        # 化成可视字符需要编码
        print("部分加密后的输出(没经过任何编码):")
```

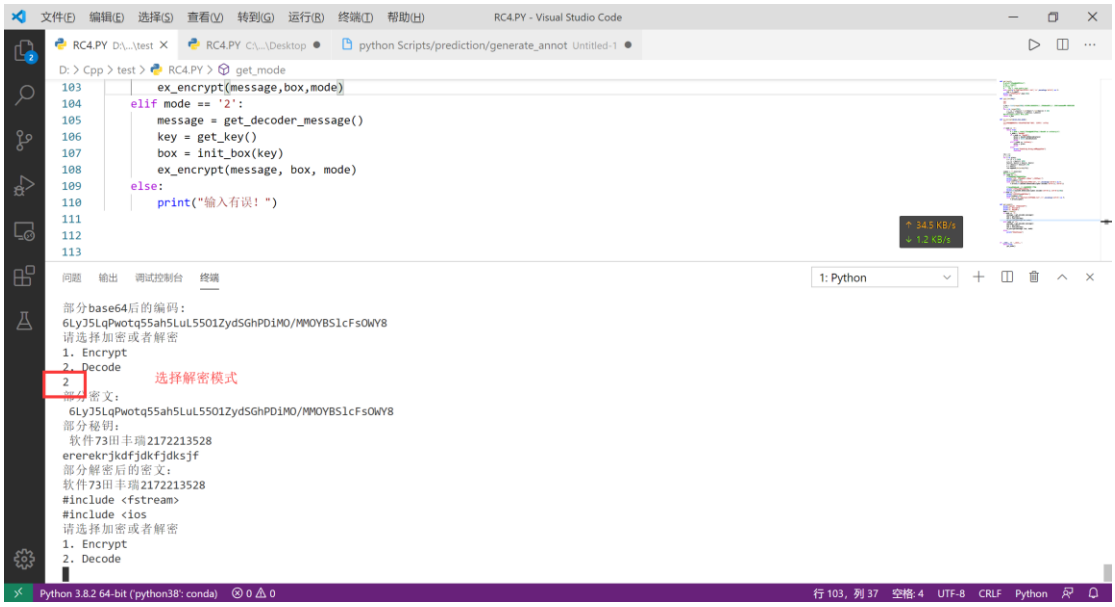
```
        print(cipher[:10])
        with open (u"d:/Cpp/test/密文.txt",'r+',encoding='utf-8') as f:
            f.write(str(base64.b64encode(cipher.encode('utf-8')),'utf-
8'))

        # base64 的目的也是为了变成可见字符
        print("部分 base64 后的编码:")
        print(str(base64.b64encode(cipher.encode('utf-8')),'utf-
8')[:50])
    if mode == '2':
        print("部分解密后的密文: ")
        print(cipher[:50])
        with open (u"d:/Cpp/test/解密文件.txt",'r+',encoding='utf-8')
as f:
            f.write(cipher)


def get_mode():
    print("请选择加密或者解密")
    print("1. Encrypt")
    print("2. Decode")
    mode = input()
    if mode == '1':
        message = get_encoder_message()
        key = get_key()
        box = init_box(key)
        ex_encrypt(message,box,mode)
    elif mode == '2':
        message = get_decoder_message()
        key = get_key()
        box = init_box(key)
        ex_encrypt(message, box, mode)
    else:
        print("输入有误! ")




if __name__ == '__main__':
    while True:
        get_mode()
```

运行结果

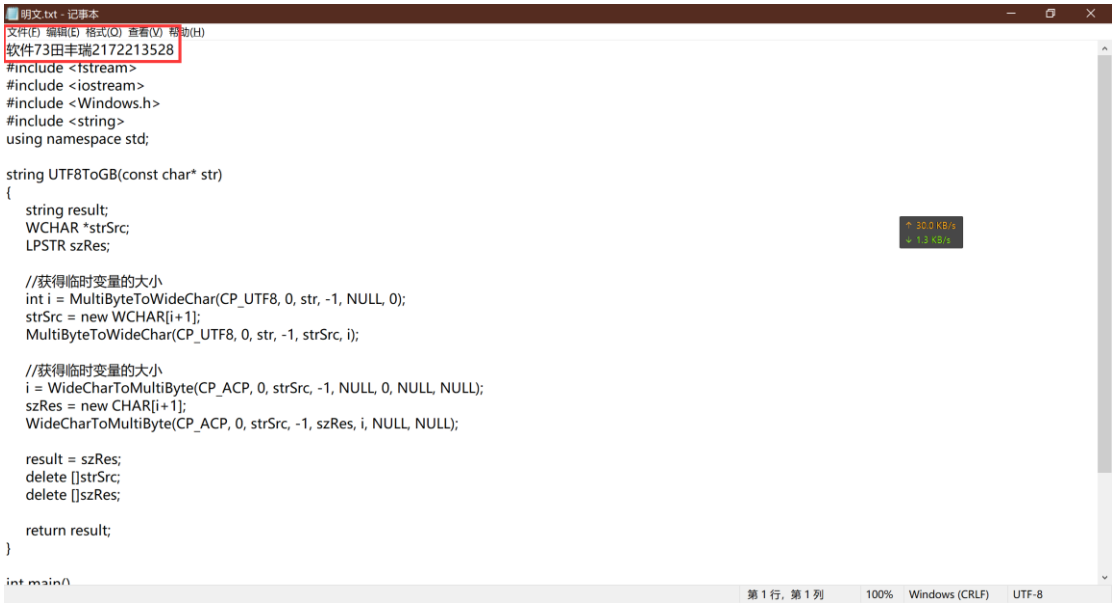
本次使用的明文为我之前写过的一段程序，明文开头有我的姓名学号标识。

程序执行结果

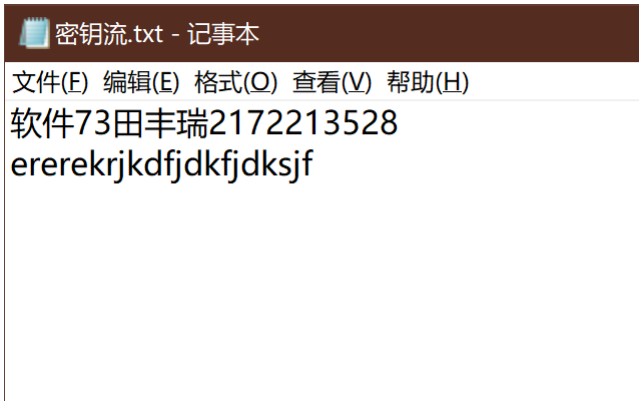




明文

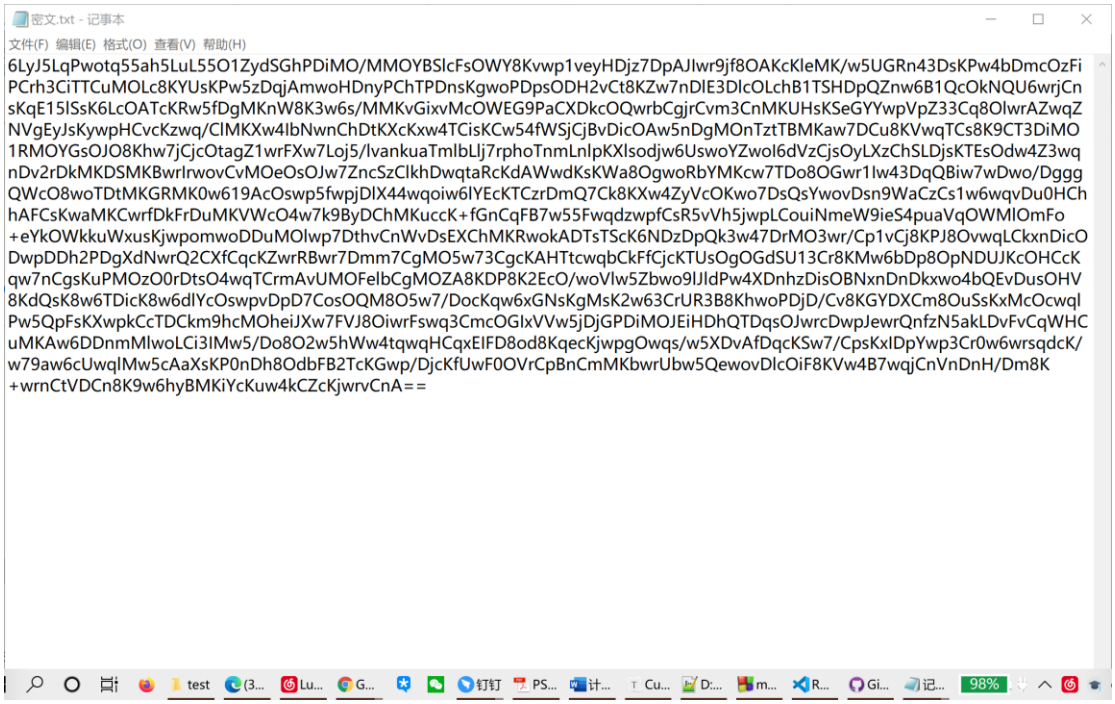


密钥流

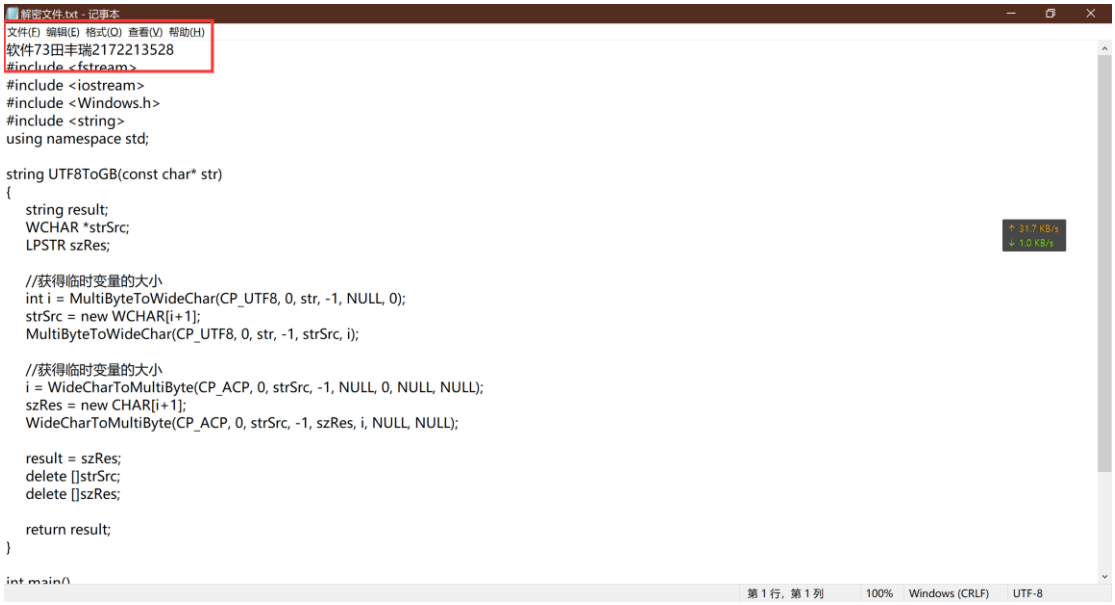


密文

密文已经转换为 base64 编码格式保证可读性。



解密文件



可见已经解密成功。

总结

本次实验加深了我对 RC4 的理解。我个人认为 RC4 最有意思的地方在于其可通过简单的交换实现加密和解密，其没有用到很深的数学知识。本次实验是我在网络安全课上做的第一个实验，让我对网络安全领域有一个初步的认识。在实现 RC4 时候由于我搞清楚了 RC4 的整个流程，因此我并没有用网络现成的代码来实现，而是自己写了一个 RC4 的函数来实现其算法过程。