密码学实验七实验报告

16337183 孟衍璋

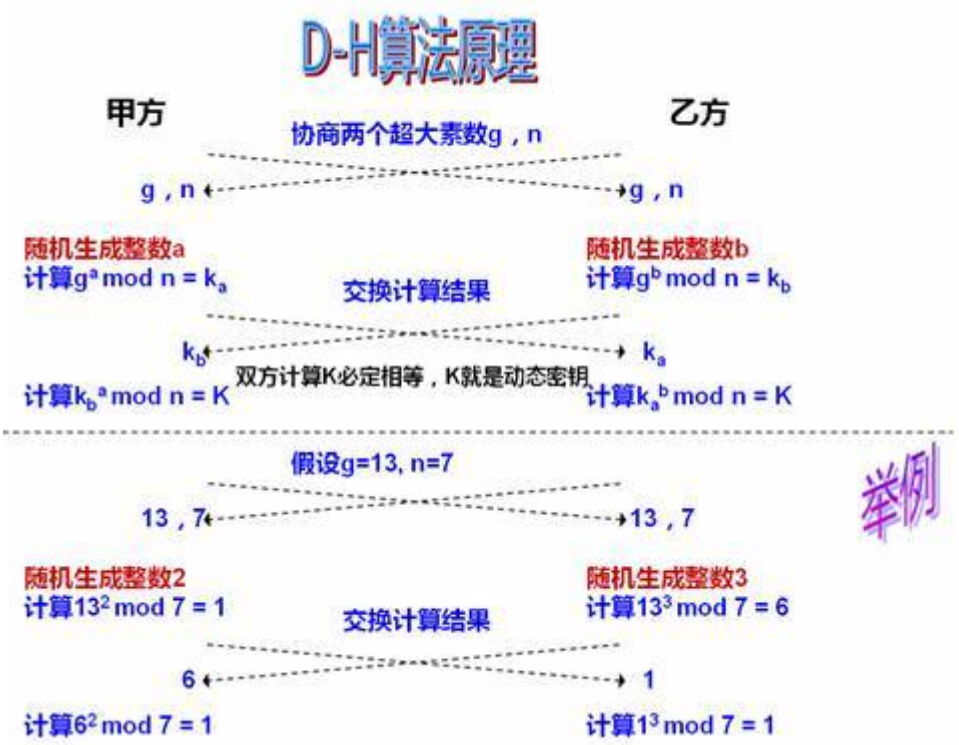
实验要求

1. 实现DH密钥交换协议（算法参数，p是128bit的任意一个素数， g 为 2）
2. 利用1.实现的算法，设计一个服务端和客户端，进行通信，在通信的过程中，消息要使用协商好的密钥加密消息。（加密算法是AES的cbc模式, 密钥长度为128）

实验步骤

实现DH密钥交换协议

DH密钥交换协议的原理如下：



以图中的甲方与乙方为例，首先双方先要协商两个超大素数g，n。然后分别独立生成一个128bits的大素数，生成素数的方法在文件`generate_prime.py`中，主要原理是先随机生成128位的01串（末尾为1保证其为奇数），然后再进行素性检测，不成功则持续+2进行判断，直到找到一个素数。

然后甲乙两方分别计算：

$$k_a = g^a mod\, n \qquad k_b = g^b mod\, n$$

再将计算出来的结果发给对方，收到之后分别计算：

$$K = k_b^a \ mod\, n \quad K = k_a^b \ mod\, n$$

上述计算需要用到平方乘算法，实现如下：

```
# 平方-乘算法，计算(base ^ exponent) mod n
def square_and_multiply(base, exponent, n):
    # 将exponent化为二进制存入bin_exponent中。
    # 切片[2:]是因为bin()函数作用后前面会有0b
    bin_exponent = bin(exponent)[2:]
    # print("exponent:", exponent)
    # print("bin_exponent:", bin_exponent)
    length = len(bin_exponent)

    z = 1
    for i in range(length):
        z = (z ** 2) % n
        # 从高位访问到低位
        if int(bin_exponent[i]) == 1:
            # print("true")
            z = (z * base) % n
        # print(z)
    return z
```

上面的式子计算出来K即为动态密钥。

实现AES加解密

使用`pycryptodome`库中的AES算法进行加解密，主要步骤为：

加密

```
aes = AES.new(key, AES.MODE_CBC, add_to_16('IV'))  # 初始化加密器

ciphertext = str(base64.encodebytes(aes.encrypt(add_to_16(message))),
encoding='utf8').replace('\n', '')  # 加密
```

解密

```
aes = AES.new(key, AES.MODE_CBC, add_to_16('IV'))  # 初始化加密器

message = str(aes.decrypt(base64.decodebytes(bytes(ciphertext,
encoding='utf8'))).rstrip(b'\0').decode("utf8"))  # 解密
```

如果使用库的话，加密解密都直接调用函数就可以了，但是也还是有需要注意的问题，比如密钥必须是128bits，即16bytes。我的程序计算出的密钥是128位的二进制串，如果要传入加密器的话，需要转化为bytes。这方面的转化开始不是很熟悉，所以花费了很长时间， 最后使用了`bytes.fromhex`函数才成功完成转化，实现了加解密功能。

用TCP实现服务端与客户端通信

服务端

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import socket
import threading
import time
import generate_prime
import base64
from Crypto.Cipher import AES

# str不是16的倍数那就补足为16的倍数
def add_to_16(text):
    while len(text) % 16 != 0:
        text += '\0'
    return str.encode(text)  # 返回bytes

def Padding_128(m):
    for i in range(128 - len(m)):
```

```
        m += '0'
    return m

def Bin2Hex_str(s):
    # print(s)
    transform = {'0000':'0', '0001':'1', '0010':'2', '0011':'3',
'0100':'4', '0101':'5', '0110':'6',
        '0111':'7', '1000':'8', '1001':'9', '1010':'a', '1011':'b',
'1100':'c', '1101':'d', '1110':'e', '1111':'f'}
    new_s = ''
    for i in range(len(s) // 4):
        new_s += transform[s[4*i:4*i+4]]
    return new_s

# 平方-乘算法，计算(base ^ exponent) mod n
def square_and_multiply(base, exponent, n):
    # 将exponent化为二进制存入bin_exponent中。
    # 切片[2:]是因为bin()函数作用后前面会有0b
    bin_exponent = bin(exponent)[2:]
    # print("exponent:", exponent)
    # print("bin_exponent:", bin_exponent)
    length = len(bin_exponent)

    z = 1
    for i in range(length):
        z = (z ** 2) % n
        # 从高位访问到低位
        if int(bin_exponent[i]) == 1:
            # print("true")
            z = (z * base) % n
        # print(z)
    return z

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 监听端口:
s.bind(('127.0.0.1', 9999))

s.listen(5)
print('Waiting for connection...')

def tcplink(sock, addr):
```

```
        print('Accept new connection from %s:%s...' % addr)
        sock.send(b'Welcome!')

        # 发送yb
        xb = generate_prime.proPrime(160)
        g = 2
        p = 2957086538844258193032226934752757551 43
        yb = square_and_multiply(g, xb, p)
        print('g =', g)
        print('p =', p)
        print('yb =', yb)
        yb = str(yb)
        sock.send(yb.encode('utf8'))

        # 接受ya
        ya = sock.recv(1024).decode('utf8')
        ya = int(ya)
        print('ya =', ya)

        # 计算密钥
        key = square_and_multiply(ya, xb, p)
        print('key =', key)
        key = bin(key)[2:]
        key = Padding_128(key)
        key = Bin2Hex_str(key)
        key = bytes.fromhex(key)

        aes = AES.new(key, AES.MODE_CBC, add_to_16('IV'))  # 初始化加密器

        # 接受密文
        ciphertext = sock.recv(1024).decode('utf8')
        print('ciphertext =', ciphertext)
        # print(type(ciphertext))
        message = str(aes.decrypt(base64.decodebytes(bytes(ciphertext,
encoding='utf8'))).rstrip(b'\0').decode("utf8"))  # 解密
        print('ciphertext after Decryption =', message)

        sock.close()
        print('Connection from %s:%s closed.\n' % addr)

    while True:
        # 接受一个新连接:
```

```
    sock, addr = s.accept()
    # 创建新线程来处理TCP连接:
    t = threading.Thread(target=tcplink, args=(sock, addr))
    t.start()
```

客户端

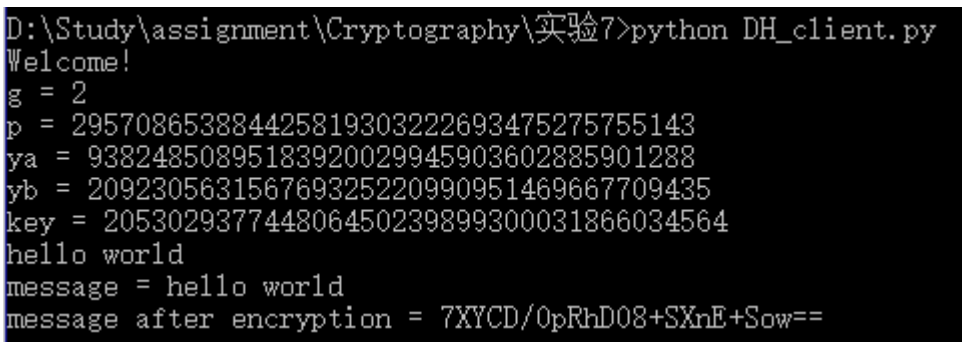
```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import socket
import generate_prime
# import AESencode
import base64
from Crypto.Cipher import AES

# str不是16的倍数那就补足为16的倍数
def add_to_16(text):
    while len(text) % 16 != 0:
        text += '\0'
    return str.encode(text)  # 返回bytes

def Padding_128(m):
    for i in range(128 - len(m)):
        m += '0'
    return m

# 平方-乘算法，计算(base ^ exponent) mod n
def square_and_multiply(base, exponent, n):
    # 将exponent化为二进制存入bin_exponent中。
    # 切片[2:]是因为bin()函数作用后前面会有0b
    bin_exponent = bin(exponent)[2:]
    # print("exponent:", exponent)
    # print("bin_exponent:", bin_exponent)
    length = len(bin_exponent)

    z = 1
    for i in range(length):
        z = (z ** 2) % n
        # 从高位访问到低位
        if int(bin_exponent[i]) == 1:
```

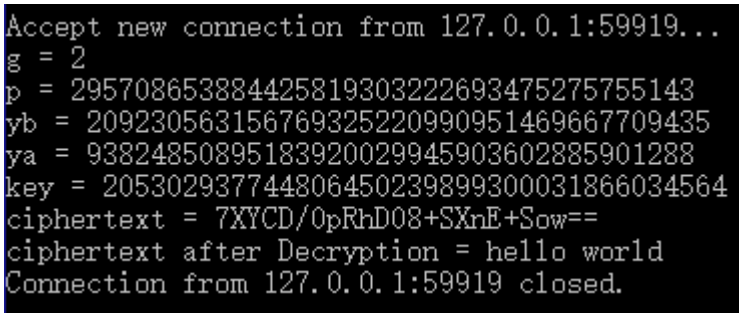
```
            # print("true")
            z = (z * base) % n
        # print(z)
    return z

def Bin2Hex_str(s):
    # print(s)
    transform = {'0000':'0', '0001':'1', '0010':'2', '0011':'3',
'0100':'4', '0101':'5', '0110':'6',
        '0111':'7', '1000':'8', '1001':'9', '1010':'a', '1011':'b',
'1100':'c', '1101':'d', '1110':'e', '1111':'f'}
    new_s = ''
    for i in range(len(s) // 4):
        new_s += transform[s[4*i:4*i+4]]
    return new_s

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 建立连接:
s.connect(('127.0.0.1', 9999))

# 接收欢迎消息:
print(s.recv(1024).decode('utf8'))

# 发送ya:
xa = generate_prime.proPrime(160)
p = 29570865388442581930322693475275755143
g = 2
ya = square_and_multiply(g, xa, p)
print('g =', g)
print('p =', p)
print('ya =', ya)
ya = str(ya)
s.send(ya.encode('utf8'))

# 接受yb
yb = s.recv(1024).decode('utf8')
yb = int(yb)
print('yb =', yb)

# 计算密钥
key = square_and_multiply(yb, xa, p)
```

```
print('key =', key)
key = bin(key)[2:]
key = Padding_128(key)
key = Bin2Hex_str(key)
key = bytes.fromhex(key)

aes = AES.new(key, AES.MODE_CBC, add_to_16('IV'))  # 初始化加密器

# 加密消息
message = input()
print('message =', message)
ciphertext = str(base64.encodebytes(aes.encrypt(add_to_16(message))),
encoding='utf8').replace('\n', '')  # 加密
print('message after encryption =', ciphertext)

# 发送密文
s.send(ciphertext.encode('utf-8'))

s.send(b'exit')
s.close()
```

实验结果

首先，打开`DH_server.py`文件启动服务端程序，再打开`DH_client.py`文件启动客户端程序，在客户端程序中输入消息后加密，将密文发送给服务端程序，服务端程序解密之后输出：





实验心得

这次实验相对而言还是比较轻松，但是也相应地有遇到一些问题。比如在实现AES加解密的时候想直接用`pycrypto`库实现，但是在安装的时候出现了以下问题：

```
error: Unable to find vcvarsall.bat
```

经过查询之后才发现`pycrypto`库只支持到**python3.4**，而我使用的是**python3.6**，后来下载了`pycryptodome`库才实现了相应功能。然后其他的内容，除了TCP服务端与客户端通信的部分，都是之前已经实现过的比较熟悉的东西，所以相对而言比较轻松。