
网络信息安全实验报告

易凯

人工智能与机器人研究所

西安交通大学软件学院

yikai2015@stu.xjtu.edu.cn

Abstract

本次实验共分为三个不同的小实验，其分别为 RC4 算法的设计与实现，OpenSSL 于 CA 的应用以及安全通信软件的设计。其中 RC4 主要通过考察其基本算法以及实现的基本伪代码，基于 Python 完成了其实现。而 OpenSSL 则是在掌握 CA 基本机制的基础之上，对于 OpenSSL 应用于 CA 的基本流程以及业务逻辑有了较为深入的认识。最后关于安全通信软件的设计则是在完成基于 TCP 通信的 IP 电话的基础之上，对其涉及到的安全机制进行了强化。总的来说，三个不同的实验基本设计到了网络信息安全课程上所学习内容的许多重要环节，同时进行了积极的实践。

1 RC4 算法

1.1 实验要求

使用 RC4 算法，对信息进行加密/解密。

1.2 RC4 算法原理

RC4 是一种对称加密算法 [1]，该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)。虽然当前的 RC4 算法存在安全性问题 [2]，但是作为一种极为经典的算法，仍然具有极大的参考学习意义。

RC4 算法包括初始化算法（KSA）和伪随机子密码生成算法（PRGA）两大部分。假设 S-box 的长度为 256，密钥长度为 Len。用 C 代码呈现的 RC4 算法如代码块如下：

```
/* 初始化函数 */  
void rc4_init(unsigned char*s, unsigned char*key, unsigned long Len) {  
    int i=0, j=0;
```

该实验报告由西安交通大学大三本科生易凯完成，其学号为 22151601053，班级为软件 53 班。网络信息安全的指导老师为田暄老师。

```

char k[256]={0};
unsigned char tmp=0;
for (i=0;i<256;i++) {
    s[i]=i;
    k[i]=key[i%Len];
}

for (i=0;i<256;i++) {
    j=(j+s[i]+k[i])%256;
    tmp=s[i];
    s[i]=s[j]; // 交换 s[i] 和 s[j]
    s[j]=tmp;
}
}

```

在初始化的过程中，密钥的主要功能是将 S-box 搅乱，i 确保 S-box 的每个元素都得到处理，j 保证 S-box 的搅乱是随机的。而不同的 S-box 在经过伪随机子密码生成算法的处理后可以得到不同的子密钥序列，将 S-box 和明文进行 xor 运算，得到密文，解密过程也完全相同。

算法加密过程的 C 语言代码为：

```

/* 加解密 */
void rc4_crypt(unsigned char*s, unsigned char*Data, unsigned long Len)
{
    int i=0,j=0,t=0;
    unsigned long k=0;
    unsigned char tmp;
    for (k=0;k<Len;k++)
    {
        i=(i+1)%256;
        j=(j+s[i])%256;
        tmp=s[i];
        s[i]=s[j]; // 交换 s[i] 和 s[j]
        s[j]=tmp;
        t=(s[i]+s[j])%256;
        Data[k]^=s[t];
    }
}

```

1.3 代码实现

此处完整使用 Python 代码实现的结果为:

```
def KSA(key):
    keylength = len(key)

    S = range(256)

    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % keylength]) % 256
        S[i], S[j] = S[j], S[i] # swap

    return S

def PRGA(S):
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i] # swap

        K = S[(S[i] + S[j]) % 256]
        yield K

def RC4(key):
    S = KSA(key)
    return PRGA(S)

if __name__ == '__main__':
    # test vectors are from http://en.wikipedia.org/wiki/RC4

    # ciphertext should be BBF316E8D940AF0AD3
    key = 'Key'
    plaintext = 'Plaintext'

    # ciphertext should be 1021BF0420
    #key = 'Wiki'
    #plaintext = 'pedia'
```

```

# ciphertext should be 45A01F645FC35B383552544B9BF5
#key = 'Secret'
#plaintext = 'Attack_at_dawn'

def convert_key(s):
    return [ord(c) for c in s]
key = convert_key(key)

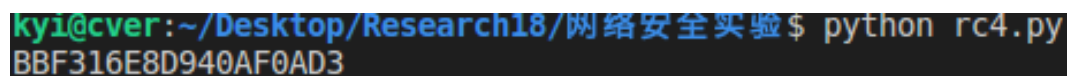
keystream = RC4(key)

import sys
for c in plaintext:
    sys.stdout.write("%02X" % (ord(c) ^ keystream.next()))
print

```

1.4 实验结果

上述代码经过测试的实验结果如图 1 所示。



```

ky1@cver:~/Desktop/Research18/网络安全实验$ python rc4.py
BBF316E8D940AF0AD3

```

图 1: RC4 测试结果输出

2 OpenSSL

2.1 实验要求

使用 OpenSSL 工具（或类似的开源 SSL 库），建立 CA（即数字证书认证中心），并为其他用户颁发 SSL 证书。

2.2 SSL 与 OpenSSL 基本介绍

SSL [3]，即 Secure Sockets Layer（安全套接层协议），可以在 Internet 上提供秘密性传输。Netscape 公司在推出第一个 Web 浏览器的同时，提出了 SSL 协议标准。其目标是保证两个应用间通信的保密性和可靠性，可在服务器端和用户端同时实现支持。已经成为 Internet 上保密通讯的工业标准。

OpenSSL [4] 是一个安全套接字层密码库，囊括主要的密码算法、常用的密钥和证书封装管理功能及 SSL 协议，并提供丰富的应用程序供测试或其它目的使用。

2.3 建立 CA 认证中心

2.3.1 创建目录

首先第一步我们为 CA 创建目录，其过程以及最后的结果如图所示：

```
kyi@cver:~/CA$ mkdir newcerts private conf
kyi@cver:~/CA$ chmod g-rwx,o-rwx private
kyi@cver:~/CA$ echo "01" > serial
kyi@cver:~/CA$ touch index.txt
kyi@cver:~/CA$ tree
.
├── conf
├── index.txt
├── newcerts
├── private
└── serial
3 directories, 2 files
```

图 2: CA 目录

其中，private_keys 和 certs 分别为存放私钥和证书的目录。serial 内容写为 01，即接下来将签发证书的第一个用户的 ID。

2.3.2 创建根密钥

接下来为 CA 创建根密钥，其命令行为：

```
openssl genrsa -des3 -out ~/CA/private/cakey.pem 2048
```

该命令的含义是：使用 RSA 算法生成密钥，其加密方式为 DES3，私钥长度为 2048 位，输出到 /CA/private/cakey.pem 文件。

其中，在命令执行过程中会提示输入口令并验证，我们可以随意选择一个好记住的口令，本试验中原则 19961102 作为口令。其实验结果如图 3 所示。

```
kyi@cver:~/CA$ openssl genrsa -des3 -out ~/CA/private/cakey.pem 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for /home/kyi/CA/private/cakey.pem:
Verifying - Enter pass phrase for /home/kyi/CA/private/cakey.pem:
```

图 3: CA 根密钥创建

2.3.3 生成根证书申请文件

然后，我们为 CA 生成证书申请文件，其命令如下所示：

```
openssl req -new -days 365 -key ~/CA/private/cakey.pem -out ~/CA/careq.csr
```

其得到的指令输出为：

```
Enter pass phrase for /root/CA/private_keys/cakey.pem:
```

```
You are about to be asked to enter information
that will be incorporated into your certificate request.
```

```
What you are about to enter is what is called
a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```

```
Country Name (2 letter code) [AN]:CN
State or Province Name (full name) []:Shaanxi
Locality Name (eg, city) [Default City]:Xian
Organization Name (eg, company) [Default Company Ltd]
:Xi'an Jiaotong University
Organizational Unit Name (eg, section) []:Software Engineering
Common Name (eg, your name or your server's hostname)
[]:harper Email Address []: williamyi96@gmail.com
Please enter the following 'extra' attributes to be sent
with your certificate request A challenge password []:
An optional company name []: Xi'an Jiaotong University
.....
```

2.3.4 自签发根证书

接下来，为 CA 自签发根证书，命令如下所示：

```
openssl ca -selfsign -in ~/CA/careq.csr -out ~/CA/cacert.cer -config ~/CA/private/cakey
```

该命令的含义为：签署证书，使用签名请求的密钥进行签名，使用 v3_ca 扩展，表示该证书是颁发给 CA 的证书。执行结果如下所示：

```
Using configuration from /etc/pki/tls/openssl.cnf
```

```
Enter pass phrase for /root/CA/private/cakey.pem:
```

Check that the request matches the signature

Signature ok

Certificate Details:

Serial Number: 1 (0x1)

Validity

Not Before: Jun 25 11:23:42 2018 GMT

Not After : Jun 25 11:23:42 2019 GMT

Subject:

countryName = CN

stateOrProvinceName = Shaanxi

organizationName = Xi'an Jiaotong University

organizationalUnitName = Software Engineering

commonName = KaiYi

emailAddress = williamyi96@gmail.com

X509v3 extensions:

X509v3 Subject Key Identifier:

D6:27:FA:2D:BF:66:E8:E0:EC:34:00:A8:15:F0:39:97:8D:08:F5:A5

X509v3 Authority Key Identifier:

keyid:D6:27:FA:2D:BF:66:E8:E0:EC:34:00:A8:15:F0:39:97:8D:08:F5:A5

X509v3 Basic Constraints:

CA:TRUE

Certificate is to be certified until Jan 3 09:58:29 2019 GMT (365 days)

Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y

Write out database with 1 new entries Data Base Updated

2.4 签发证书

2.4.1 用户生成 RSA 密钥

首先生成用户的 RSA 密钥，命令如下：

```
openssl genrsa -des3 -out ~/CA/userkey.pem 2048
```

用户的 2048 位私钥将被存入 CA 下的 userkey.pem 文件中。结果如下：

```
2048 Generating RSA private key, 2048 bit long modulus
```

.....+++

.....+++

e is 65537 (0x10001)

Enter pass phrase **for** /root/CA/userkey.pem:

Verifying - Enter pass phrase **for** /root/CA/userkey.pem:

此处，我们使用 `username` 作为口令进行测试。

2.4.2 用户生成签发证书请求

接下来，我们为用户生成签发证书请求，其对应口令如下所示：

```
openssl req -new -days 365 -key ~/CA/userkey.pem -out ~/CA/userreq.csr
```

该口令的含义与之前的 CA 证书请求生成命令基本相同。执行结果为：

Enter pass phrase for /root/CA/userkey.pem:

You are about to be asked to enter information that

will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value ,

If you enter '.', the field will be left blank.

Country Name (2 letter code) [AN]:CN

State or Province Name (full name) []:Shaanxi

Locality Name (eg, city) [Default City]:Xian

Organization Name (eg, company) [Default Company Ltd]:

Xi'an Jiaotong University

Organizational Unit Name (eg, section) []:Software Engineering


```
Common Name (eg, your name or your server's
hostname) []: WilliamYi Email Address []: williamyi96@gmail.com
```

```
Please enter the following 'extra' attributes to be sent
with your certificate request A challenge password []:
```

```
An optional company name []:
```

2.4.3 为用户签发证书

最后，我们可以为该用户签发证书了。命令如下：

```
openssl ca -in ~/CA/userreq.csr -out ~/CA/usercert.cer -extensions v3_req
```

该命令的含义为针对用户的请求文件 userreq.csr 签发对应的证书。执行结果如下：

```
Using configuration from /etc/pki/tls/openssl.cnf
```

```
Enter pass phrase for /root/CA/private_keys/cakey.pem:
```

```
Check that the request matches the signature
```

```
Signature ok
```

```
Certificate Details:
```

```
Serial Number: 2 (0x2)
```

```
Validity
```

```
Not Before: Jun 25 21:12:19 2018 GMT
```

```
Not After : Jun 25 21:12:19 2019 GMT
```

```
Subject:
```

```
countryName = CN
```

```
stateOrProvinceName = Shaanxi
```

```
organizationName = Xi'an Jiaotong University
```

```

organizationalUnitName = Software Engineering

commonName             = WilliamYi

emailAddress            = williamyi96@gmail.com

X509v3 extensions:

X509v3 Basic Constraints:

CA:FALSE

X509v3 Key Usage:

Digital Signature, Non Repudiation, Key Encipherment
Certificate is to be certified until Jan 3 10:07:03 2019 GMT (365 days)

Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries Data Base Updated

```

至此为止，使用 OpenSSL 框架模拟认证中心为用户签署证书的过程已实验完毕。

3 安全通信软件

3.1 实验要求

完成一个即时通信程序的设计，要求其提供至少包括机密性、完整性、可用性、认证等安全服务。可在之前做过的通信程序的基础上改进。

3.2 实验思路

3.2.1 IP 电话的基本思路

由于之前接触过 IP 电话的应用程序，因此此处想以此为依据进行安全通信软件的设计与实现。该程序可以实现通过 IP 和端口呼叫、接听、挂断、拒接等基本功能。该程序为 P2P 模式，无需设置服务器。程序分为空闲、呼叫、通话、来电四种状态 IP 电话的有限状态机如图所示。通话状态下，呼叫方和接听方会不断向对方发送 UDP 报文（内容为很小的一段时间内的录音），对方则接受报文进行播放以听取该段声音。报文以明文方式发送，若被截

获则可被很轻易地恢复出录音的内容。因此，本次实验中将对该程序进行修改以达到安全通信的要求。

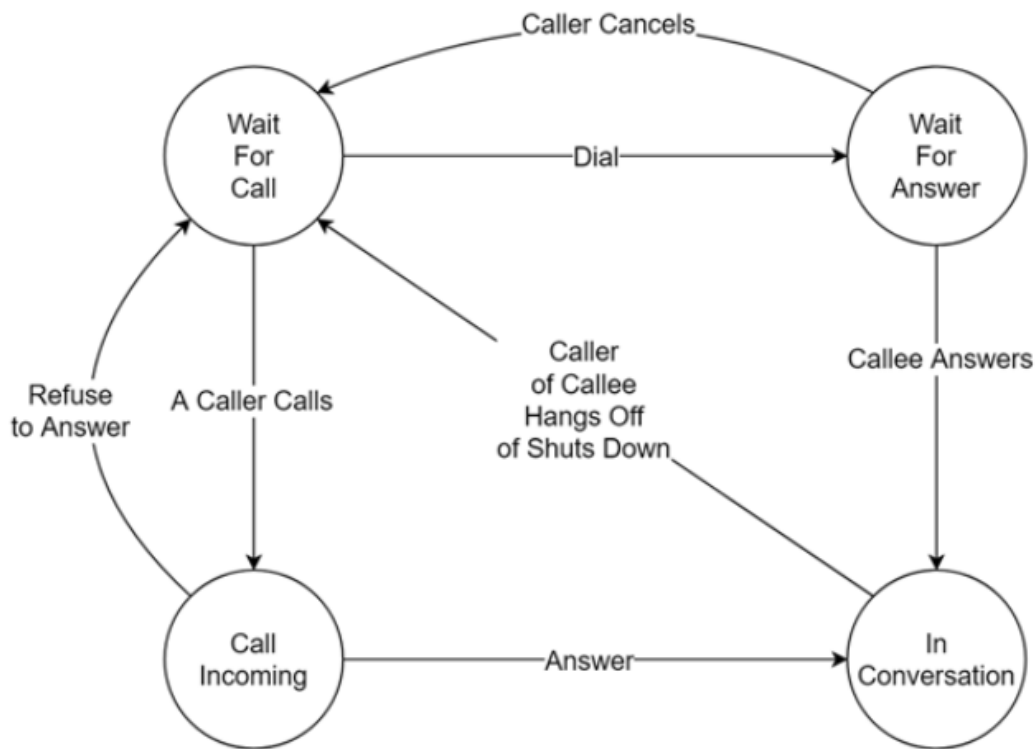


图 4: IP 电话的有限状态机

以下将从 CIA 的不同层面对可靠性通信的多个层面内容进行分析，其中可靠性包括机密性、完整性、认证性三个不同层面的内容。

3.2.2 机密性

为实现机密性，需对双方用户发送的报文进行加密并使用合适的方式分配密钥。本实验将采用 RSA 算法生成密钥并加解密。因为程序未设置服务器，所以密钥分配不设第三方、不设安全信道，而是仅采用公钥加密的方式。

3.2.3 完整性

完整性要求消息报文不被篡改。因此，在每次发送报文之前，先将消息内容使用 MD5 算法获取摘要，再将消息和摘要一并使用会话密钥加密传送。

3.2.4 认证性

要保证根据消息可鉴别对方身份，需要对摘要使用报文发送方的私钥进行加密（即数字签名），这样接收方可用对方的公钥进行解密从而验证对方的身份。

3.3 代码实现

该部分基于 TCP 通信的 IP 电话文件传输可以参看计算机网络相关资料，此处重点关注与信息安全有关的部分。工程中与信息安全相关的部分说明如下。

3.3.1 RSA 密钥对产生

RSA 密钥产生过程使用了 JDK 自带的 `KeyPairGenerator` 类以及现有的 `bouncycastle.jce.provider` 包。产生的密钥长度为 1024 位。`KeyPair` 类对象为公钥和私钥的集合。

```
public static KeyPair generateKeyPair() throws EncryptException {
    try {
        KeyPairGenerator keyPairGen =
            KeyPairGenerator.getInstance("RSA",
                new org.bouncycastle.jce.provider.BouncyCastleProvider());
        final int KEY_SIZE = 1024;
        keyPairGen.initialize(KEY_SIZE, new SecureRandom());
        KeyPair keyPair = keyPairGen.genKeyPair();
        return keyPair;
    } catch (Exception e) {
        throw new EncryptException(e.getMessage());
    }
}
```

3.3.2 RSA 的加解密

加解密的函数实严格按照 RSA 算法的规定实现，代码如下：

```
public static byte[] encrypt(Key key, byte[] data) throws
EncryptException {
    try {
        Cipher cipher = Cipher.getInstance("RSA", new
            org.bouncycastle.jce.provider.BouncyCastleProvider());
        cipher.init(Cipher.ENCRYPT_MODE, key);
        int blockSize = cipher.getBlockSize();
        int outputSize = cipher.getOutputSize(data.length);
        int leavedSize = data.length % blockSize;
        int blocksSize = leavedSize != 0 ? data.length / blockSize +
            1 : data.length / blockSize;
        byte[] raw = new byte[outputSize * blocksSize];
        int i = 0;
        while (data.length - i * blockSize > 0) {
            if (data.length - i * blockSize > blockSize)
```

```

cipher.doFinal(data, i * blockSize, blockSize, raw, i *
outputSize);
else
cipher.doFinal(data, i * blockSize, data.length - i *
blockSize, raw, i * outputSize);
i++;
}
return raw;
} catch (Exception e) {
throw new EncryptException(e.getMessage());
}
}

public static byte[] decrypt(Key key, byte[] raw) throws
EncryptException {
try {
Cipher cipher = Cipher.getInstance("RSA", new
org.bouncycastle.jce.provider.BouncyCastleProvider());
cipher.init(cipher.DECRYPT_MODE, key);
int blockSize = cipher.getBlockSize();
ByteArrayOutputStream bout = new ByteArrayOutputStream(64);
int j = 0;
while (raw.length - j * blockSize > 0) {
bout.write(cipher.doFinal(raw, j * blockSize, blockSize));
j++;
}
return bout.toByteArray();
} catch (Exception e) {
throw new EncryptException(e.getMessage());
}
}
}

```

3.3.3 发送报文之前的摘要、签名与加密

录音机对象获取声音数据（长度 1024 的 byte 类型数组）后，首先使用 MD5 算法进行摘要 digest，再将摘要内容使用发送者自己的私钥进行加密从而得到数字签名 signature。将声音信息 record 和签名 signature 连接后使用接收方的公钥进行加密，得到密文 message。代码如下：

```

byte[] record = recorder.record();
InetSocketAddress socketAddress = (InetSocketAddress)
socket.getRemoteSocketAddress();

```

```

System.out.println("Remote Address: " + socketAddress.getAddress());
byte[] digest = MD5Util.md5(record);
try {
byte[] signature = RSAUtil.encrypt(keyPair.getPrivate(), digest);
byte[] message = ByteArrayUtil.cat(record, signature);
byte[] encrypted = RSAUtil.encrypt(peerPublicKey, message);
DatagramPacket datagramPacket = new DatagramPacket(encrypted,
encrypted.length, socketAddress.getAddress(), remoteDatagramPort);
try {
datagramSocket.send(datagramPacket);
System.out.println("Datagram Sent to " +
socketAddress.getAddress() + " : " + remoteDatagramPort);
} catch (IOException e) {
e.printStackTrace();
}
} catch (EncryptException e) {
e.printStackTrace();
}
}

```

3.3.4 接受报文之后的解密与验证

任何一个通话方具有一个 UDP 报文接收线程。在收到报文后，将依次执行发送报文之前的处理过程的反过程——即解密、验证签名、验证完整性。代码如下：

```

byte[] buffer = new byte[1280];
DatagramPacket datagramPacket = new DatagramPacket(buffer,
buffer.length);
try {
datagramSocket.receive(datagramPacket);
System.out.println("Datagram Received from " +
datagramPacket.getSocketAddress());
byte[] message = datagramPacket.getData();
byte[] decrypted = RSAUtil.decrypt(keyPair.getPrivate(), message);
byte[] record = new byte[1024];
for (int i = 0; i < 1024; i++) {
record[i] = decrypted[i];
}
player.play(record);
} catch (EncryptException e) {
e.printStackTrace();
}
}

```

3.4 实验结果

本程序的实际运行情况已在第一部分进行过展示，与信息安全相关的部分并未对程序 UI 造成影响。由于 Java 和 RSA 的效率问题，通话过程出现了明显的卡顿与延迟。

4 总结

本次实验共分为三个不同的小实验，其分别为 RC4 算法的设计与实现，OpenSSL 于 CA 的应用以及安全通信软件的设计。其中 RC4 主要通过考察其基本算法以及实现的基本伪代码，基于 Python 完成了其实现。而 OpenSSL 则是在掌握 CA 基本机制的基础之上，对于 OpenSSL 应用于 CA 的基本流程以及业务逻辑有了较为深入的认识。最后关于安全通信软件的设计则是在完成基于 TCP 通信的 IP 电话的基础之上，对其涉及到的安全机制进行了强化。总的来说，三个不同的实验基本设计到了网络信息安全课程上所学习内容的许多重要环节，同时进行了积极的实践。

致谢

该实验报告的顺利开展以及完成，少不了多方面的帮助。感谢田老师在实验过程中提供的指导以及课堂上循循善诱而又有趣的教学，感谢各位同学们不辞辛苦和我讨论交流相关问题，感谢乐乐一直依赖提供的支持与理解，感谢开源社区能够进行宝贵的经验分享。

参考文献

- [1] R. Rivest. The rc4 encryption algorithm. *Rsa Data Security Inc Document No*, 20(1):86–96, 1992.
- [2] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of rc4. *Sac*, 2259:1–24, 2001.
- [3] A Freier. The ssl 3.0 protocol. *Netscape Communications*, 1996.
- [4] J. Viega, M. Messier, and P. Chandra. Network security with openssl. *Oreily*, (4), 2002.