X.509 解析
实验报告

姓名：宋晓彤

学号：16340192

方向：嵌入式软件与系统

2018. 12. 12

一、 X.509 证书结构

x.509 标准规定了证书可以包含什么信息，并说明了记录信息的方法。

X.509 结构中包括版本号（integer）、序列号（integer）、签名算法（object）、颁布者（set）、有效期（utc_time）、主体（set）、主体公钥（bit_string）、主体公钥算法（object）、签名值（bit_string）。

使用 ASN.1 描述，我们可以将其抽象为以下结构

```
Certificate::=SEQUENCE{
      tbsCertificate      TBSCertificate,
      signatureAlgorithm  AlgorithmIdentifier,
      signatureValue      BIT STRING
}
TBSCertificate::=SEQUENCE{
    version           [0]   EXPLICIT Version DEFAULT v1,
    serialNumber            CertificateSerialNumber,
    signature               AlgorithmIdentifier,
    issuer                  Name,
    validity                Validity,
    subject                 Name,
    subjectPublicKeyInfo    SubjectPublicKeyInfo,
    issuerUniqueID    [1]   IMPLICIT UniqueIdentifier OPTIONAL,
    subjectUniqueID   [2]   IMPLICIT UniqueIdentifier OPTIONAL,
    extensions        [3]   EXPLICIT Extensions OPTIONAL
}
```

而本次实验，我选择使用从 chrome 上直接下载证书，此时我们可以看到，证书结构如下：

类	结构	信息	备注
TBSCertificate	版本信息	证书的使用版本	整数格式，0-V1，1-V2，2-V3
TBSCertificate	序列号	每个证书都有一个唯一的证书序列号	整数格式
TBSCertificate	签名算法	得到签名时使用的算法	有 OID 与之对应
TBSCertificate	颁发者	命名命规则一般采用 X.500 格式	Name
TBSCertificate	有效期	通用的证书一般采用 UTC 时间格式，计时范围为 1950-2049	Format: yymmddhhmssZ
TBSCertificate	使用者	使用证书的主体	Name
TBSCertificate	主体密钥	证书所有人的公开密钥	
Certificate	公钥签名算法	证书公钥的加密算法	有 OID 与之对应
Certificate	签名值	得到的签名结果	

二、数据结构

【编码方法】

X509 的编码方法为 TLV 结构，使用 T 记录当前数据的类型（type），使用 L 记录当前数据的长度（length），使用 V 记录当前数据的取值（value），其中，不同的 type 值对应不同的数据类型

Type	数据类型	编码格式
01	Boolean	01；01；FF/00
02	Integer	长度大于 7f 时，长度 n 与 0x80 进行"位或"运算的结果赋给 length 的第一个字节
03	Bit string	填充 0 成为 8 的倍数，Value 的第一个字节记录填充数
04	Ectet string	04；len；val
05	Null	value 部分为空，一共两字节
06	Object Identifier	V1.V2.V3.V4.V5....Vn (1)计算 40*V1+V2 作为第一字节；(2)将 Vi(i>=3)表示为 128 进制，每一个 128 进制位作为一个字节，再将除最后一个字节外的所有字节的最高位置 1；(3)依次排列，就得到了 value 部分
19	ASCII string	13；len；val
23	UTCtime	yymmddhhmssZ
24	Generalize time	yyyymmddhhmssZ

48	Sequence constructer	序列内所有项目的编码的依次排列
49	Set constructer	集合内所有项目的编码
160	Tag	对于简单类型，type=80+tag 序号；对于构造类型，type=A0+tag 序号。length 和 value 不变

【数据结构】

类的声明：均使用 string 类型记录数据，数据具体的内容已经在注释中标出

```
class TbsCertificate{
    public:
        string version;
        string serialNumber;
        string signature[2];// algorithm parameters
        string issuer_[6][2];
        string validity[2];
        string subject_[6][2];
        string subjectPublicKeyInfo[3];// algorithm parameters Pkey
        string issuerUniqueID;
        string subjectUniqueID;
        string extensions;
};

class X509cer{
    public:
        TbsCertificate cat;
        string casa[2];// algorithm parameters
        string casv;
};
```

三、算法流程

1. 打开.cer 文件，选择按字节读取，即每次读取一个字符
2. 读取的整体流程如下：
   a) 读取一个字节的 type
   b) 读取一个字节的 length
   c) 对 type 进行判断：如果 type 是非标签的：直接根据类型判断当前的数据类型是什么
   d) 对 real length 进行判断：根据 type 决定读取的数据长度，如 integer 区分长短数据，减去 0x80 后才是真正的长度值，同时换算出真正的长度
   e) 对 value 进行记录：根据长度读入实际的数据，并转换成自己需要的格式，如 06 的格式为 V11.V2.V3….
3. 每次读取到 value 后，直接赋值给证书中的内容，此时，我们首先需要对当前的赋值对象进行判断
   a) 根据证书的结构，我们对读取的过程划分为以下阶段： "ver", "seq", "sigalg", "iss", "starttime","endtime", "usr", "keyalg","sigalg"

b) 此时，我们可以根据当前的赋值阶段 n 和此时读取的数据类型 type 来判断当前的赋值对象究竟是什么，避免产生因为发布者数目不统一而无法读取所有.cer 文件的问题

P.S, 参考了 https://www.cnblogs.com/jiu0821/p/4598352.html 并进行了很大程度上的优化

四、 实现源码

【函数流程】

```
int main(){
    init();// file
    tlv();// read->fill
    output();// print
}
```

【读取 type 和长度】

```
    bool b =true;
    unsigned char type=fgetc(fp);// type
    unsigned char len0=fgetc(fp);// len
    int len = len0;
    s = "";
```

【根据长度读取数据】

```
void bitfill(int len){
    s = "";
    int i = 0;
    for(int i = 0;i < len; ++i){
        unsigned char tl = fgetc(fp);
        char ts2[10];
        sprintf(ts2, "%02x", (int)tl);
        s = s + ts2;
    }
}
```

【结构的循环读取】

```
    else if(type == 0x30 || type == 0x31){ // 循环读取后续结构的内容
        b = false;
        if(len0 > 0x80){
            len = 0;
            len0 -= 0x80;
            unsigned char tl;
            for(int i = 0; i < len0; ++i){
                tl = fgetc(fp);
                len = len * 256 + tl;
            }
        }
        int dlen = len;
        while(dlen>0){
            dlen -= tlv();
        }
    }
    else{
        printf("the cer has errors!\n");
        return len;
    }
```

【根据 n 和 type 赋值】

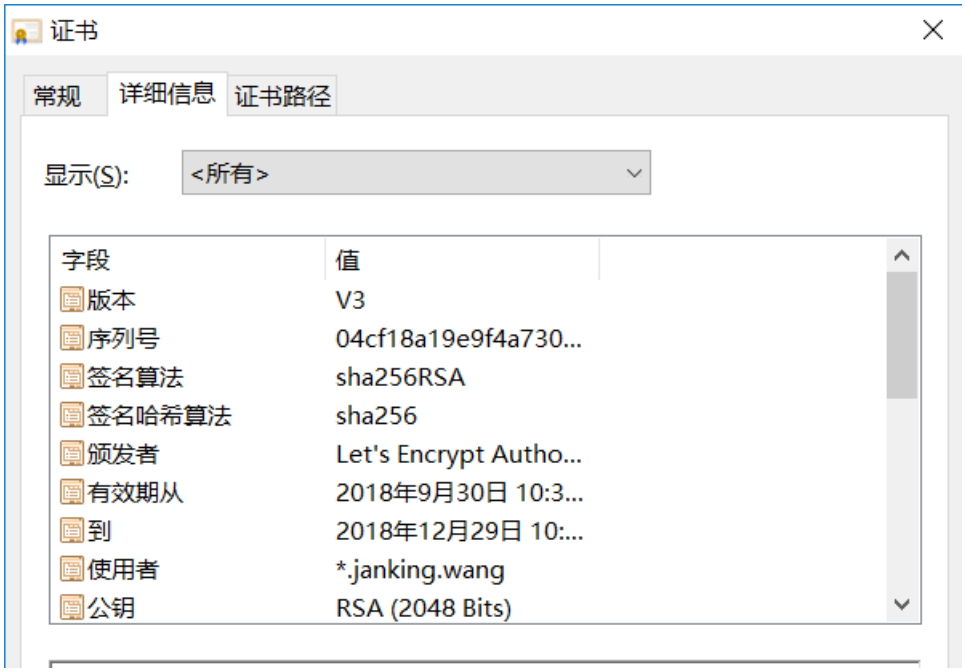
（在主体和主体公钥信息的读取上有冲突，但是可以一起解决）

```
    else if(n == 6 && t == 6){
            for(int i = 0; i < 6; ++i){
                if(s == is[i][0]){
  //                cout << s << " " << i << endl;
                    ca_cer.cat.subject_[i][0] = is[i][1];
                    ca_cer.cat.subject_[i][0] += "of subject:\t";
                    curr=i;
                    break;
                }
            }
    //  cout << s << "****" << endl;
            for(int i = 0; i < 8; ++i){
                if(s == sA[i][0]){
                   ca_cer.cat.subjectPublicKeyInfo[0] = sA[i][1];
                   n++;
                    break;
                }
            }
    }
    else if(n == 6 && (t == 12 || t == 19))
```

五、 实验结果

进入 chrome 并打开一个网页，在安全标识处打开证书，查看内容，并下载 DER 编码格式的.cer 文件，命名为 test.cer

网页上直接查看结果如下



程序运行结果如下

```
【版本】
        V3
【序列号】
        04cf18a19e9f4a730791e0f75bdcd55127d9
【签名算法】
        sha256RSA
【签名算法的参数】
        NULL
【签发者标识信息】
        Country of issuer:       US
        Organization name of issuer:     Let's Encrypt
        Common Name of issuer:   Let's Encrypt Authority X3
【有效期】
        2018.09.30 02:36:10-2018.12.29 02:36:10
【使用者】
        Common Name of subject:  *.janking.wang
【公钥的加密算法】
        RSA
【公钥的加密算法参数】
        NULL
【公钥数据】
        003082010a0282010100ed29962667d7d17572a4ca46dc83d4eee264f2f524d9de13c4184a07bed096859c5d0970d4f616108623b33d7a1f
056f02236ed4283f0483b8b7d0e19b48ca4d4fa55779ff52e8a6a545e3dfac714b9fecee94ff2c6eb4c45fb8eed8a4a37617adc91d5f03e7d3401617
04a4336dd53ac4c681f917b3b75a716262e9f5a2fce44a5974ecb20f18918c5967886e39ca38dfe2be2035a49aa65d7f1c1b96bfce9760675ab57a3e
9baa9d50cb3cc34ef69b63fb95eb8eb30aece3d3420d565361c6f182f3a4ef5eeb16db0d3eb557a7efc4de40b8e8e6f31e4e6efa2ed874e074d636bc
720472953f7e87c1502365576c18c67199faf24cafe23a13c7a97b9ae3b10203010001
【签名算法的参数】
        NULL
【签名结果】
        008a2ebdeaf560a1b945ba946c062473831d99c6e7b17958cfc0ffaecb406f1d90facab1c074192b24504b296b591ecec1e339b988868bac
```

对照后发现，读取成功！