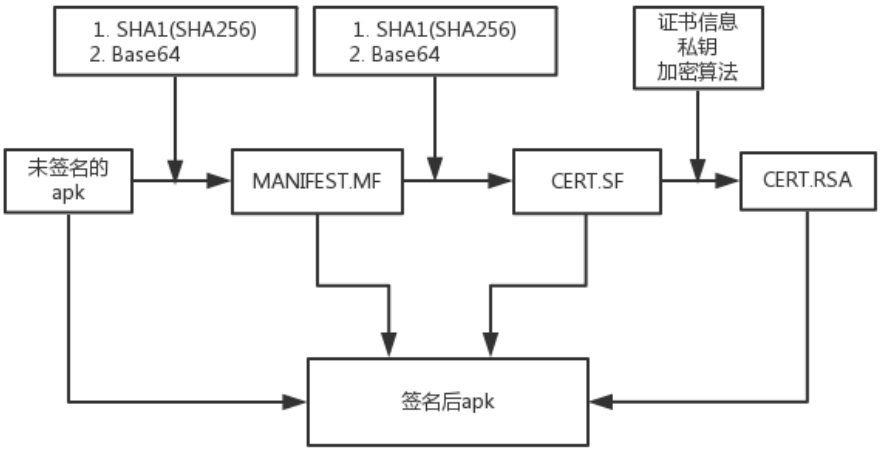
android 签名证书文件的解析和签名校验的加强

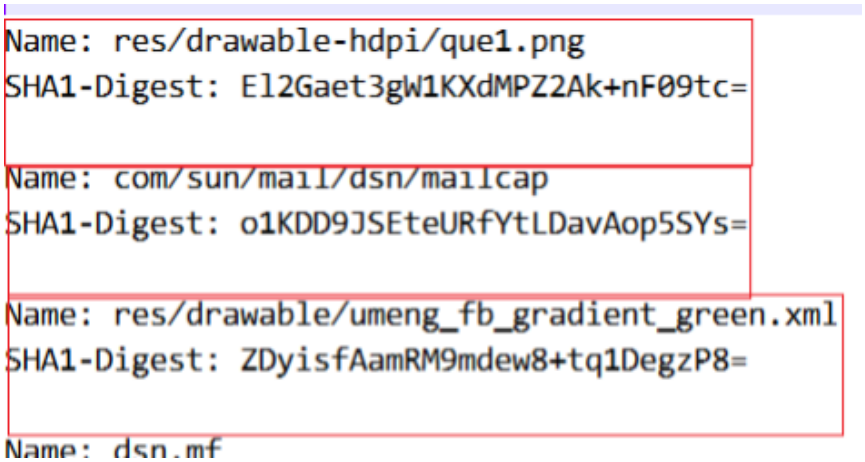
目前 android 签名证书文件解析有三种途径：1）android 提供的解析类；2）java 提供的解析类；3）openssl 提供的解析。我这篇文章介绍了签名证书文件格式、提供了解析代码以及签名检验加强的方法。

1. android 签名机制

大家都知道，android 系统中只有签名过的 apk 才能被安装，签名机制用于 1）apk 中文件的数字签名，保证文件的完整性；2）将 apk 与签名者建立对应关系，在程序中也会对签名进行验证防止被重新打包、盗版等。Android 签名过程如下图所示，



- 首先使用哈希算法（SHA1 或者 SHA256）对未签名 apk 的每一个文件进行 hash 计算，将值以 Base64 的格式保存在 MANIFEST.MF 文件中，这是生成每个文件的消息摘要；
- 对 MANIFEST 文件生成消息摘要，在对 MANIFEST.MF 中每个文件的内容再生成消息摘要，并将结果保存在 CERT.SF，如下图，以每个框为内容进行计算；



- 使用证书（保存有私钥）对 CERT.SF 文件生成数字签名，即先生成消息摘要，在用私钥进行非对称加密，加密后的数据放入到 CERT.RSA 文件中，该文件还包括有签名者的证书信息（没有私钥）。
- 最后将生成的三个文件放入到 META-INF 目录下，并打包成 apk，这样就完成了应用程序的签名。

下面主要对 CERT.RSA 签名证书文件进行介绍，该文件是属于 pkcs7 标准格式(rfc2315)，里面的证书部分是属于 X.509 标准(rfc3280)，有兴趣想深入了解的话可以找这两个标准看。

2. PKCS7 格式

```
contentInfo : SEQUENCE
    contentType    :    ObjectIdentifier
    content[optinal]        #类型由 contentType 决定
```

contentInfo 用于表示该结构，冒号后面表示类型，缩进表示该项属于上面非缩进的内容，即 contentType 和 content 是 contentInfo 的内容。contentType 决定 content 的内容和格式。contentType 包括有 data、signedData、envelopedData、signedAndEnvelopedData、digestedData 和 encryptedData 六种类型，不同类型只是 content 部分不同。

数字签名只涉及到 data 和 signedData 类型，下面就之说这两种类型。

2.1 DER 编码

pkcs7 格式保存在文件中是采用 DER(Distinguished Encoding Rules)编码的，该编码是对每一项用{tag, length, data}来表示，tag 表示该项的类型，length 用于指定 data 的长度，data 为数据部分，长度为 length。

- tag

tag 用一个字节表示，下面是常用的 tag:

Tag	Value	description
INTEGER	0x02	整数类型
BITSTRING	0x03	位字符串
OCTETSTRING	0x04	字节字符串
ObjectIdentifier	0x06	该类型可以理解为整数数组
SEQUENCE	0x30	类似于 C 语言的 struct,表示该项是一个结构
SET	0x31	类似于 C 语言的 union,表示该项是某集合中一个

上述 tag 中除了 ObjectIdentifier 都比较理解，ObjectIdentifier 是类似于整数序列，如下图

```
// pkcs7 pre-defined content types
private static int[]  pkcs7 = {1, 2, 840, 113549, 1, 7};
private static int[]   data = {1, 2, 840, 113549, 1, 7, 1};
private static int[]  sdata = {1, 2, 840, 113549, 1, 7, 2};
private static int[]  edata = {1, 2, 840, 113549, 1, 7, 3};
private static int[] sedata = {1, 2, 840, 113549, 1, 7, 4};
private static int[]  ddata = {1, 2, 840, 113549, 1, 7, 5};
private static int[] crdata = {1, 2, 840, 113549, 1, 7, 6};
private static int[] nsdata = {2, 16, 840, 1, 113730, 2, 5};
// timestamp token (id-ct-TSTInfo) from RFC 3161
```

data 和 sdata 是是属于下面要讲的 data 和 signedData 类型的 contentType 值，contentType 类型就是 ObjectIdentifier。这些是属于标准已经定义好的。

- length 的表示

length 的表示首先需要一个字节的标记信息，用 lenbyte 表示，紧随在 tag 的后面。若 lenbyte 最高位为 1，则后 7 位表示 length 所占用的字节长度，采用大端存储方式紧随在 lenbyte 后面；若最高位是 0，lenbyte 就等于 length。

例如：0x82 0x34 0x56 0x12... 0x82 是 lenbyte，最高位为 1， 则 0x82 & 0x7f 等于 2，说明后面两个字节表示长度，采用的是大端存储方式，则 length 等于 0x3456 ((0x34 << 8) | 0x56)。

- data

length 后面紧挨就是 length 长度字节的 data 数据了。

2.2 data 类型

```
contentInfo : SEQUENCE
    contentType : ObjectIdentifier    {data}
    content : OCTETSTRING
```

data 类型比较简单，只包含有数据部分，数据类型是字节字符串。

2.3 signedData 类型

apk 中的签名证书文件是属于该类型，是对 CERT.SF 文件内容进行数字签名，数字签名的目的不是用于加密，而是保证数据的完整性。接收者校验数字签名过程：1）用签名者的公钥将加密数据进行解密，得到数据的消息摘要；2）对数据使用消息摘要算法计算消息摘要，然后对比，若相等则说明数据完整，相反则数据被修改了。因此在该类型文件中必须包括有以下内容：1）消息摘要算法；2）签名者信息，公钥以及非对称加密的算法；3）加密后的数据等。下面介绍该类型所包含的部分。

```
contentInfo :   SEQUENCE
    contentType : ObjectIdentifier    {signedData}
    content[optional] : SEQUENCE
        version : INTEGER
        digestAlgorithms : SET : DigestAlgorithmIdentifier
        contentInfo : SEQUENCE
        certificates[optional] : SEQUENCE
        crls[optional] : SET
        signerInfos : SET
```

- version　版本信息，是属于整数类型，当前版本为 1
- digestAlgorithms 消息摘要算法，类型是 SET，表示是多个消息摘要算法中的一个

- contentInfo 内部中 pkcs7 结构，该部分保存的是要签名的数据，后面会向西介绍
- certificates 证书部分，后面详细介绍
- crls 证书吊销列表
- signerInfos 签名者信息，后面详细介绍

2.3.1 证书格式

```
certificates[optional] : SEQUENCE
        tbsCertificate : SEQUENCE
                version : INTEGER
                serialNumber : INTEGER
                signature ： SEQUENCE : AlgorithmIdentifier
                issuer : SET
                validity : SEQUENCE
                subject : SET
                subjectPublicKeyInfo : SEQUENCE
                issuerUniqueID[optional] : BITSTRING
                subjectUniqueID[optional] : BITSTRING
                extensions[optional] : SEQUENCE
    signatureAlgorithm : AlgorithmIdentifier
    signatureValue : BITSTRING
```

证书部分包含有三项 tbsCertificate、signatureAlgorithm 和 signatureValue。tbsCerticate 是证书的核心部分，而后两项是对 tbsCerticate 进行数字签名，保证该核心部分不被修改。signatureAlgorithm 是签名加密算法，常用的有 SHA256withRSA ，signatureValue 加密后的数据部分。下面详细介绍 tbsCerticate 部分：

- Version 版本信息，整数类型
- serialNumber 证书的序列号，是整数类型，是由证书颁布者分配
- signature 对 tbsCerticate 进行数字签名算法，和上述 signatureAlgorithm 的值一致
- issuer 证书颁布者信息，issuer 和上面的 serialNumber 可以唯一确定证书
- validity 证书的有效期，有开始时间和截止时间

- subject 证书主体的信息
- subjectPublicKeyInfo 证书公钥信息，包含有加密算法和公钥
- issuerUniqueID 和 subjectUniqueID 主要是对证书颁布者或证书主体多个引用时使用，可选的

可见证书中主要包括有：证书的序列号、证书颁布者和主体、证书有效期和公钥等信息。

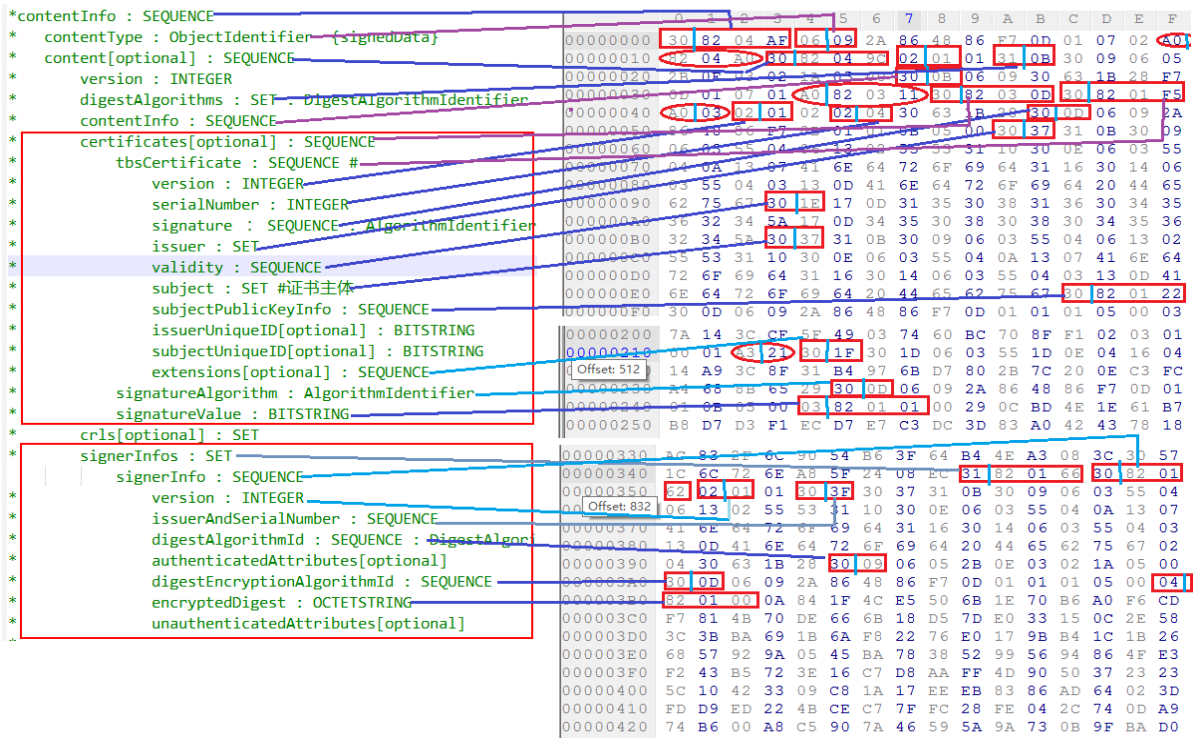
2.3.2 签名者格式

```
signerInfo : SEQUENCE
        version : INTEGER
        issuerAndSerialNumber : SEQUENCE
        digestAlgorithmId : SEQUENCE : DigestAlgorithmIdentifier
        authenticatedAttributes[optional]
        digestEncryptionAlgorithmId : SEQUENCE
        encryptedDigest : OCTETSTRING
        unauthenticatedAttributes[optional]
```

- Version 版本信息
- issuerAndSerialNumber 证书颁布者和序列号
- digestAlgorithmId 生成消息摘要的算法
- digestEncryptionAlgorithmId 签名算法
- encryptedDigest 私钥加密后的数据

里面所包含的信息都介绍完了，名字后有 optional 表示该项是可选的，上面有些可选的不是很重要就没有介绍。

2.4 格式解析图



上图中红色框中是每项的的 tag 和 length 部分，用蓝色线分隔。由于有些项是可选的，但怎样知道是否包含该项呢？在这里使用了 0xA0 作为标记字节，在上图中椭圆中就是该可选标记，而在 certificates 中包含有多个可选项，分别使用 0xA1，0xA2，0xA3 来作为标记字节。

3. 签名校验加强

一般签名校验要么是在 java 层进行要么是通过反射在 so 中，所使用的是获取 PackageInfo 中 signatures 字段，然后计算 MD5 进行对比，或其他比较方式。signatures[0]实质上保存的是上述证书部分信息。通过 PackageInfo 类会有很明显的特征，下面会介绍如何增强签名校验增强。

3.1 修改签名证书文件

签名证书文件是通过签名工具自动生成的，我想大多数都不会修改这个文件。那这个文件是否可以修改呢？能不能修改就需要看在 1）android 安装 apk 过程中是如何使用该文件的，以及 2）通过 JarFile 类对 jar 包验证如何使用该文件的。（keytool 和 jarsigner –verify 底层都是 JarFile）。

- **文件大小**

通过实验发现，虽然可以通过 contentInfo 中的 length 来确定文件大小，但是在解析的过程中都没有对文件大小进行强制验证，即可以在文件末尾任意增加字符而不会影响使用。因此可以在文件末尾增加任意大小的字符，若被重新签名可以通过验证文件大小来确定。

- **内部 contentInfo 部分**

通过对上述各个项分析，首先证书部分是有自己的数字签名的，因此不能修改，而签名者部分保存有摘要算法、签名算法和加密后的数据部分，因此也是不能修改的。因此目光就落在了之前还没有介绍的内部 contentInfo 部分。

内部 contentInfo 实际上是可以保存进行数字签名的数据，这样在一个文件中即传输了数据部分又有数字签名部分。但目前 android 中进行数字签名的数据是独立的文件，因此该部分就没有使用。由于 android 和 JarFile 类对这部分解析的不同的，因此分别介绍。

■ JarFile 类的解析就比较严格了，1）对 contentInfo 中 contentType 类型有限制；

```
public byte[] getData() throws IOException {
if (contentType.equals(DATA_OID) ||
    contentType.equals(OLD_DATA_OID) ||
    contentType.equals(TIMESTAMP_TOKEN_INFO_OID)) {
    if (content == null)
    return null;
    else
    return content.getOctetString();
}
throw new IOException("content type is not DATA: " + contentType);
}
```

2）若 content 部分包含数据，则会将该内容作为数字签名的数据部分，实际上我们是对 CERT.SF 才是数据部分。

```
if (sfv.needSignatureFileBytes()) {
    // see if we have already parsed an external .SF file
    byte[] bytes = (byte[]) sigFileData.get(key);

    if (bytes == null) {
        // put this block on queue for later processing
        // since we don't have the .SF bytes yet
        // (uname, block);
        if (debug != null) {
            debug.println("adding pending block");
        }
        pendingBlocks.add(sfv);
        return;
    } else {
        sfv.setSignatureFile(bytes);
    }
}
```

因此，在 content 不能添加内容，由于支持三种 contentType 类型，这三种类型的值如下图，默认的都是 data，由于 OLD_DATA 和 data 整数个数相同，因此可以将 contentType 换成 OLD_DATA，而不用修改其他部分；而 tstInfo 整数个数更多，换的话需要修改其他部分就比较麻烦了。

```
// pkcs7 pre-defined content types
private static int[]  pkcs7 = {1, 2, 840, 113549, 1, 7};
private static int[]  data = {1, 2, 840, 113549, 1, 7, 1};
private static int[]  sdata = {1, 2, 840, 113549, 1, 7, 2};
private static int[]  edata = {1, 2, 840, 113549, 1, 7, 3};
private static int[] sedata = {1, 2, 840, 113549, 1, 7, 4};
private static int[]  ddata = {1, 2, 840, 113549, 1, 7, 5};
private static int[] crdata = {1, 2, 840, 113549, 1, 7, 6};
private static int[] nsdata = {2, 16, 840, 1, 113730, 2, 5};
// timestamp token (id-ct-TSTInfo) from RFC 3161
private static int[] tstInfo = {1, 2, 840, 113549, 1, 9, 16, 1, 4};
// this is for backwards-compatibility with JDK 1.1.x
private static final int[] OLD_SDATA = {1, 2, 840, 1113549, 1, 7, 2};
private static final int[] OLD_DATA = {1, 2, 840, 1113549, 1, 7, 1};
public static ObjectIdentifier PKCS7_OID;
public static ObjectIdentifier DATA_OID;
```

- android 对这部分内容是没有验证的，即可以在 content 部分添加任意内容，也可以修改 contentType 内容，按照一定格式来修改或添加内容，这样可以防止重新签名。

3.2 C++实现解析签名证书文件的代码

理论终究是要用语言来实现，因此就需要自己实现一套解析签名证书文件的代码，使用 C++可以提高逆向者的难度。可以结合其他手段如字符串加密等方式来共同提高难度。

3.3 总结

签名校验的加强的主要手段有：

1） 在 so 中获取解析签名证书文件；

2） 在文件末尾添加自定义内容；

3） 在内部 contentInfo 中添加自定义内容（可以在 android 上安装使用，但不能使用 keytool 或 jarsigner）；

4） 内部 contentInto 修改 contentType 的 data 类型为 OLD_DATA 类型。

4. 代码介绍

上述所讲的我都已经实现，并且进行了测试。代码地址为：
https://github.com/W-WTerDan/pkcs7

```
class pkcs7 {
    public:
        …… 
        bool open_file(char *file_name);
        void print();
        char* get_MD5();
        bool add_data(unsigned char *data, int len, int
         tail = 1, const char * save_name = NULL);
        bool change_contentType(int type = 1);
        …… 
};
```

- bool open_file(char *file_name);
  打开签名证书文件或者包含有签名证书文件压缩文件。
- void print();
  打印解析出的结果，如下图所示，标识出每个项内容的文件偏移和长度。

```
-------------------------------------------------------------------------
name                                      offset        length
========================================= ============== =============
contentType                                 6(0x06)        9(0x09)
content-[optional]                         23(0x17)     1180(0x49c)
    version                                25(0x19)        1(0x01)
    DigestAlgorithms                       28(0x1c)       11(0x0b)
    contentInfo                            41(0x29)       11(0x0b)
    certificates-[optional]                60(0x3c)      781(0x30d)
        tbsCertificate                     64(0x40)      501(0x1f5)
            version                        68(0x44)        1(0x01)
            serialNumber                   71(0x47)        4(0x04)
            signature                      77(0x4d)       13(0x0d)
            issuer                         92(0x5c)       55(0x37)
            validity                      149(0x95)       30(0x1e)
            subject                       181(0xb5)       55(0x37)
            subjectPublicKeyInfo          240(0xf0)      290(0x122)
            extensions-[optional]         532(0x214)      33(0x21)
        signatureAlgorithm                567(0x237)      13(0x0d)
        signatureValue                    584(0x248)     257(0x101)
    signerInfos                           845(0x34d)     358(0x166)
        signerInfo                        849(0x351)     354(0x162)
            version                       851(0x353)       1(0x01)
            issuerAndSerialNumber         854(0x356)      63(0x3f)
            digestAlgorithmId             919(0x397)       9(0x09)
            digestEncryptionAlgorithmId   930(0x3a2)      13(0x0d)
            encryptedDigest               947(0x3b3)     256(0x100)
-------------------------------------------------------------------------
```

- char* get_MD5();

  获取证书的 MD5 值，和 keytool 所打印的一直。

- bool add_data(unsigned char *data, int len, int tail = 1, const char * save_name = NULL);

  添加内容，tail == 1 是在签名证书文件的末尾添加，==0 是在内部 contentInfo 中的 content 项添加。注意：在 content 项添加的话 keytool 或 jarsigner 会验证失败，但可以正常安装到 android 中。

- bool change_contentType(int type = 1);

  修改 contentInfo 中 contentType 类型，默认都是 data 类型，修改成 OLD_DATA。

最后，水平有限，若有错误，欢迎指正。