ASN.1/BER/DER
编码子集入门指南

—— RSA实验室的一份技术笔记

作者：Burton S. Kaliski Jr.

1993年11月1日修订

摘要:

本文简单介绍了OSI 抽象语法符号（ASN.1）的子集——基本编码规则（BER）和可辨别编码规则（DER）。本文的主要目的是为理解和实现PKCS协议族提供足够的背景材料。

翻译：David Zhu
Yang Zhang

译稿版本：1.0版

1、介绍

众所周知，软件开发管理最主要的设计原理就是抽象。通过抽象，设计者可以定义系统的一部分，而不需要关心这部分实际上如何实现或如何表示。这一方法使得实现很 open，它简化了定义过程，使得在实现部件之前可以声明某些“公理”、并且在设计高层部件时假定下层部件是可以实现的。抽象是现代多数软件规范的特点。

作为当今最复杂的系统之一，开放系统互联（OSI，在 X.200 种定义）是一个包含了大量抽象的例子。OSI 是一个国际通用的标准体系，从物理层一直到用户应用层，管理着计算机之间的互联。高层次的对象被抽象定义，并将由底层的对象来实现。比如，某层的一个服务可能需要在计算机之间传递某个抽象对象；某一底层则可能提供关于 0、1 字符串的实现，利用一些编码规则把高层的抽象对象转换成这些字符串。OSI 之所以被称为开放系统是因为它在每一层上支持不同的服务实现。

OSI 定义抽象对象的方法称为 ASN.1(Abstract Syntax Notation One，X.208)，把这些对象转换成“0”和“1”的比特流的一套规则称为 BER(Basic Encoding Rules ，X.209)。ASN.1 是一套灵活的记号，它允许定义多种数据类型，从 integer、bit string 一类的简单类型到结构化类型，如 set 和 sequence，还可以使用这些类型构建复杂类型。BER 说明了如何把每种 ASN.1 类型的值编码为 8bit 的 octet 流。通常每个值有不止一种的 BER 编码方法。一般使用另外一套编码规则 DER，它是 BER 的一个子集，对每个 ASN.1 值只有唯一一种编码方法。

本文档的目的是通过描述 ASN.1、BER 和 DER 的一套子集，以便提供足够的背景知识理解和实现一种基于 OSI 的应用——RSA 公司的 Public-Key Cryptography Standards。本文概括介绍了 ASN.1、BER 和 DER，列举了一部分 ASN.1 类型和它们的 BER/DER 编码。第 2－4 节概括介绍了 ASN.1、BER 和 DER。第 5 节列出了部分 ASN.1 类型，给出了它们的符号、详细的编码规则和举例，并介绍了其在 PKCS 中的应用。第 6 节用一个 X.500 distinguished names 例子作为总结。

本笔记没有谈到 ASN.1 的高级特点（例如宏），因为它们在实现 PKCS 时并不必要。关于其它特征和更多细节，请读者参考 CCITT 的建议文档：X.208、X.209，这两份文档中定义了 ASN.1 和 BER。

术语和符号。本文中，octet（字节）表示一个 8 bit 的无符号整数。Bit8 表示最高位，bit1 表示最低位。

下列元语法用于定义 ASN.1 符号：

`BIT`	类型和值用等宽字体表示。例如，它通常表示一个16进制的字节值。
n_1	粗斜体表示变量
[]	粗的方括号表示该项为可选项
{}	粗体大括号表示一组相关项
|	粗体竖杠表示一组之中的可任选其一
	粗体省略号表示重复出现
=	粗体等号，用一个子项表示该项

2. 抽象语法符号

抽象语法符号一是描述抽象类型和值的符号，缩写为 ASN.1。

在 ASN.1 中，一个类型就是一个值的集合。有些类型有有限个值，有些则有无限多个。一个给定的 ASN.1 类型的值是该类型集合里的一个元素。ASN.1 有四种类型：简单类型，它相当于原子，没有组件；结构类型，有组件；标签类型，由其他类型生成；其他类型，包括 CHOICE 和 ANY 类型。可以使用 ASN.1 的分配符（::=）给类型和值指定名字，这些名字可以用于定义其他类型或值。

除了 CHOICE 和 ANY 类型以外，每种 ASN.1 类型都有一个标签，由一个类和一个非负的标签数组成。标签值可以唯一区分 ASN.1 类型。也就是说，ASN.1 类型的名字并不影响它的抽象含义，只有标签才有这个作用。有四类标签：

Universal：该类型的含义在所有的application中都相同。这种类型只在X.208中定义。

Application：该类型的含义由application决定，如X.500目录服务。两个不同的application中的类型可以具有相同的application-specific标签但是不同的含义。

Private：该类型的含义根据给定的企业而不同。

Context-specific：该类型的含义根据给定的结构类型而不同。Context-specific标签用于在一个给定的结构类型上下文中区分使用相同的下层标签的组件类型。在两个不同的结构类型中组建类型可以具有相同的标签但是含义不同。

具有 universal 标签的类型在 X.208 中定义，X.208 也给出了类型的 universal 标签值。使用其他标签的类型在很多地方都有定义，通常是通过 implicit 或 explicit 标签获得。表一列出了部分 ASN.1 类型及其 universal-class 标签。

Type	Tag number (decimal)	Tag number (hexadecimal)
INTEGER	2	02
BIT STRING	3	03
OCTET STRING	4	04
NULL	5	05
OBJECT IDENTIFIER	6	06
SEQUENCE and SEQUENCE OF	16	10
SET and SET OF	17	11
PrintableString	19	13
T61String	20	14
IA5String	22	16
UTCTime	23	17

ASN.1 类型和值使用一种灵活的、类似编程语言的符号表示，规则如下：

- 分层（换行）无特殊意义；多个空格和多个空行相当于一个空格。
- 注释由一对连字符（--）开头，或者一对连字符和一个空行
- 标识符（值或字段的名字）和类型索引（类型的名字）由大小写字母、数字、连字符和空格组成；标识符由小写字母开头，类型索引由大写字母开头。

下面的四个子节概括介绍了简单类型、结构类型、隐式和显式标签类型，及其他类型。第 5 节定义了类型的更多细节。

2.1 简单类型

简单类型没有子组件，是“原子级”的类型。ASN.1定义了几个简单类型，其中与PKCS标准有关的类型如下：

- **BIT STRING**:由0和1任意组成的比特流
- **IA5String**:由IA5(ASCII)字符任意组成的字符流
- **INTEGER**:一个任意的整数
- **NULL**:null值
- **OBJECT IDENTIFIER**:对象标识符，有一列整数构成，用于确定对象，如算法或属性类型
- **OCTET STRING**:任意的octet（8 bit值）流
- **PrintableString**:任意可打印字符流
- **T61String**:T.61（8bit）字符的任意流
- **UTCTime**:"coordinated universal time"或者格林威治平均时（GMT）值。

简单类型分为两类：string 类型和 non-string 类型。BIT STRING, IA5String, OCTET STRING, PrintableString, T61String, 和 UTCTime 是 string 类型。

考虑到编码，String 类型可以视为由组件组成，组件是 substring。这样即使事先不知道值的长度也可以使用结构化的、不定长的编码方式进行编码（例如，从一个 file stream 中输入的 octet string 值）。

String 类型可以指定大小限制，以限制值的长度。

2.2 结构类型

结构类型由组件组成。ASN.1 定义了四种，都与 PKCS 标准有关：

- **SEQUENCE:**一个或多个类型的有序集合
- **SEQUENCE OF:**0个或某个给定类型多次出现的有序集合
- **SET:**一个或多个类型的无序集合
- **SET OF:**0个或某给定类型多次出现的无序集合

结构类型允许有可选组件。可选组件可能有默认值。

2.3 隐式和显式标签类型

在一个 application 中标签（tagging）对于区分类型十分有用，标签通常也用于在一个结构类型中区分组件类型。例如，SET 或 SEQUENCE 类型的可选组件一般都给予不同的 context-specific 标签以避免混淆。

有两种方法可以标记一个类型：隐式(implicitly)和显式(explicitly)。

隐式标签类型是在其它类型基础上通过改变其下层类型的标签而生成的。隐式标签使用 ASN.1 关键词[***class number***] IMPLICIT（见第 5.1 节）表示。

显式标签是在其他类型基础上通过在其下层类型的标签之外添加一个外层标签而生成的。从效果上看，显式标签类型是包含一个组件的结构类型，该组件即下层类型。显式标签由 ASN.1 关键词[***class number***] EXPLICIT（见第 5.2 节）表示。

除非“模块”的 ASN.1 类型默认定义为隐式标签，关键词[***class number***]默认总是与使用显式标签相同。(“模块”属于高级特性，不在本文档描述范围内）

从编码的角度看，隐式标签类型可视为与下层类型相同，只是标签不同。显式标签类型可视为有一个组件的结构类型，该组件即为下层类型。隐式标签可以使编码较短，但是如果下层类型是不确定的，显式标签必须避免含糊不清（例如下层类型是 CHOICE 或 ANY）。

ASN.1 中的其它类型包括 CHOICE 和 ANY 类型。CHOICE 类型表示一个联合体，它具有一个或多个备选项（alternative）；ANY 类型表示任意类型的任意值，其中任意类型可能在使用对象识别符或整数值注册中定义。

3. 基本编码规则（Basic Encoding Rules）

ASN.1 的基本编码规则定义了一种或多种把任意 ASN.1 值表示成字节字符串的方法，缩写为 BER。(当然还有其它的方法，但是 BER 是 OSI 中转换这些值的标准）

使用 BER，一个 ASN.1 的值有三种编码方法，选择哪种取决于值的类型和值的长度是否已知。这三种方法是：简单定长编码，结构化定长编码，及结构化不定长编码。简单的 non-string 类型使用第一种（简单定长编码）；结构化类型可使用任一种结构化的编码方法；简单的 string 类型根据值的长度是否已知可使用任一种方法。隐式标签定义的类型可使用下层类型的方法，显式标签定义的类型使用结构化的编码方法。

每种 BER 编码方法都有三或四部分：

- ***Identifier octets:***定义了 ASN.1 值的类和标签值，指明编码方法是简单的还是结构化的。
- ***Length octets:***对于定长编码方法，它指出了内容字节个数；对于结构化非定长编码方法，它指明长度是不确定的。
- ***Contents octets:***对于简单定长编码方法，它给出了值的具体表示；对于结构化的方法，它给出了值内容的BER编码的串联。
- ***End-of-contents octets:***对于结构化非定长的编码方法，它表示内容结束；对于其它方法，没有该部分。

在下面的章节中介绍了这三种编码方法。

3.1 简单定长方法（Primitive, definite-length method）

这种方法用于简单类型及通过对简单类型使用隐式标签生成的类型。它要求值的长度是事先预知的。BER 编码的部分定义如下：

1. **Identifier octets**，有两种形式：较小的标签值（标签值在 0 和 30 之间）和较大的标签值（标签值大于等于 31）

- ***Low-tag-number form***：一个字节。Bit8和bit7表示类（如表2），bit6值为0，表示编码方法为简单化的。Bit5－1给出了标签值。

Class	Bit 8	Bit 7
universal	0	0
application	0	1
context-specific	1	0
private	1	1

- ***High-tag-number form：*** 两个或多个octet。第一个octet形式如low-tag-number form，但是bit5－1均为1。第二个和以后的字节给出标签值，基于128，最高位在先，以便使用尽可能少的数字，除了最后一个字节以外，每个字节的bit 8都置为1。最后一个字节的为0。

2. **Length octets**：有两种格式：短型（长度在 0 至 127 之间）和长型（长度在 0 至 2^{1008}–1 之间）
   - ***Short form:*** 一个字节，bit8为0，bit 7－1表示长度。
   - ***Long form:*** 2－127个字节。第一个字节的Bit 8为1，bit 7－1表示后面有多少个用于表示实际长度的octet。第二个和随后的octet给出实际长度，基于256，高位数字在先。
3. **Contents octets**：给出了值的具体表示（如果类型是由隐式标签定义的，则给出了下层类型的值），特定类型的细节详见第 5 节。

3.2 结构化定长方法（Constructed, definite-length method）

结构化定长方法适用于简单的 string 类型、结构类型、在二者基础上通过隐式标签生成的类型和在任何类型基础上由显式标签生成的类型。要求值的长度事先已知。BER 编码方法各部分如下：

1. **Identifier octets**：与第 3.1 节介绍的一样，但 bit6 的值为 1，表示编码方法是结构化的。
2. **Length octets**：见第 3.1 节。
3. **Contents octets**，值的组件的 BER 编码的串联
   - 对于简单string类型和在其基础上由隐式标签生成的类型，是值的连续子串的BER编码的串联（隐式标签的下层值）
   - 对于结构类型和在其基础上由隐式标签生成的类型，是值的组件的BER编码的串联（隐式标签的下层值）
   - 对于在任何类型基础上使用显式标签生成的类型，是下层值的BER编码

特定类型的细节见第 5 节。

3.3 结构化非定长方法（Constructed, indefinite-length method）

结构化的、非定长编码用于简单 string 类型、结构类型、在二者基础上使用隐式标签生成的类型和在任何类型基础上使用显式标签生成的类型。不要求事先知道值的长度。BER 编码各部分如下：

- **Identifier octets**，见第 3.2 节
- **Length octets.**一个字节，值为 80
- **Contents octets.**见第 3.2 节。
- **End-of-contents octets** 两个字节，为 00　00。

由于 end-of-contents octet 通常出现在普通 BER 编码出现的位置（例如，在一个 sequence 值的内容 octet 出现的位置），可把 00 和 00 分别视为 identifier 和 length octet。因此 end-of-contents octet 实际上是一个具有 universal class，标签值为 0，长度为 0 的值的简单定长编码。

4. Distinguished Encoding Rules(DER，可辨别编码规则)

DER 是 BER 的子集，它定义了使用一个 octet string 来表示任何 ASN.1 值的编码方法。DER 用于需要使用唯一的 octet string 编码的应用程序，例如根据一个 ASN.1 编码来计算数字签名。DER 在 X.509 的第 8.7 节定义。

DER 在第 3 节给出的规则基础上增加了如下限制：

1. 如果长度在 0－127 之间，必须使用短型长度表示法。
2. 如果长度大于等于 128，必须使用长型长度表示法。并且长度必须使用尽可能少的字节表示。
3. 对于简单 string 类型和在其基础上使用隐式标签生成的类型，必须使用简单定长编码方法。
4. 对于结构化类型和在其基础上使用隐式标签生成的类型、及在任何类型基础上使用显式标签生成的类型，必须使用结构化定长编码方法。

对于特殊类型（如 BIT STRING、SEQUENCE、SET 和 SET OF）的其他限制见第 5 节。

5. 某些类型的符号和编码

本节给出了部分 ASN.1 类型的符号，并介绍了如何使用 BER 和 DER 对这些类型进行编码。

这里介绍的类型是第 2 节提到的，按字母顺序列在下面。

每个介绍包括 ASN.1 符号、BER 编码和 DER 编码。编码的重点主要在内容字节上，标签和长度字节遵循第 3、4 节的介绍。介绍还解释了每种类型用在 PKCS 中何处及相关的标准。ASN.1 符号主要是针对类型的，但还是针对 OBJECT IDENTIFIER 给出了类型符号和值符号。

5.1 隐式标签类型

隐式标签类型是在其他类型基础上通过改变下层类型的标签生成的类型。

隐式标签在整个 PKCS 中用于可选的 SEQUENCE 组件，其下层类型是除 ANY 以外的任何类型。也用于 PKCS #7 的 ExtendedCertificateOrCertificate 类型的 extendedCertificate 备选项。

ASN.1 表示法:

[[*class*] *number*] IMPLICIT *Type*

class = UNIVERSAL | APPLICATION | PRIVATE

其中，***Type*** 是类型，***class*** 是可选的 class 名，***number*** 是类内的标签值，是一个非负整数。

在默认标签方法为隐式标签的 ASN.1 模块中，也可以使用**[[*class*] *number*]** ***Type*** 符号，关键词 IMPLICIT 是隐含的（见第 2.3 节）。在模块以外的地方进行定义声明，最好使用

关键词 IMPLICIT 以避免语义含混。

如果没有 class 名称，则标签为 context-specific 类。context-specific 类型的标签只能出现在结构类型或 CHOICE 类型的组件中。

例：PKCS #8 的 PrivateKeyInfo 类型有一个可选的 attributes 组件，具有一个隐式的、context-specific 标签：

```
PrivateKeyInfo ::= SEQUENCE {
  version Version,
  privateKeyAlgorithm PrivateKeyAlgorithmIdentifier,
  privateKey PrivateKey,
  attributes [0] IMPLICIT Attributes OPTIONAL }
```

这里，下层类型为Attributes，class是缺省的（即context-specific类），在这个类内的标签值为0。

BER encoding：根据下层类型可以采用简单化的或结构化的编码。内容字节与下层值的BER编码有关。

例：一个PrivateKeyInfo值的attributes组件的BER编码如下：

- 如果下层的Attributes值采用简单化的BER编码（bit 6为0），则identifier octet为80；如果采用结构化的编码（bit 6为1），则identifier octet为a0。
- length和content octet分别与下层Attributes值的BER编码的相同。

DER encoding： 根据下层类型选择简单或结构化编码方法。Content octet 与下层值的 DER 编码有关。

5.2 显式标签类型

显示标签通过在一个类型的外层增加一个标签可以生成一个新的类型。

在整个 PKCS 中，显式标签用于下层类型为 ANY 的、可选的 SEQUENCE 类型组件，及 X.509 的 Certificate 类型的 version 组件。

ASN.1 表示法:

[[*class*] *number*] EXPLICIT *Type*

class = UNIVERSAL | APPLICATION | PRIVATE

Type 表示类型，***class*** 是一个可选的类名，***number*** 是一个非负整数，表示在类内的标签值。

如果没有类名，则标签为 context-specific 类。Context-specific 类标签只能出现在 SEQUENCE、SET 或 CHOICE 类型的组件中。

在默认标签方法为显式标签的 ASN.1 模块中，也可以使用 **[[*class*] *number*] *Type*** 符号，关键词 EXPLICIT 是隐含的（见第 2.3 节）。在模块以外的地方进行定义声明，最好显式写明关键词 EXPLICIT 以避免语义不清。

例: PKCS #7 中的 ContentInfo 类型有一个可选的 content 组件，具有一个显式的、context-specific 类的标签：

```
ContentInfo ::= SEQUENCE {
  contentType ContentType,
```

```
content
  [0] EXPLICIT ANY DEFINED BY contentType OPTIONAL }
```

这里下层的类型为 ANY DEFINED BY contentType，类名省略了（即为 context-specific 类），类内的标签值为 0。

例2： X.509 的 Certificate 类型有一个 version 组件，具有显式的 context-specific 类标签，其 EXPLICIT 关键词省略了：

```
Certificate ::= ...
  version [0] Version DEFAULT v1988,
...
```

这个标签是显式的，因为在 X.509 中，ASN.1 模块的默认标签方法定义 Certificate 类型为显式标签。

BER encoding. 结构化编码。Contents octets 与下层值的 BER 编码有关。

例： ContentInfo 值的 content 组件的 BER 编码如下：

- identifier octets为a0
- length octets为下层ANY DEFINED BY contentType值的BER编码长度
- contents octets为下层ANY DEFINED BY contentType值的BER编码

DER encoding. 结构化编码。Contents octets 与下层值的 BER 编码有关。

5.3 ANY

ANY 类型用于表示任意类型的一个任意值，其中任意类型可能在对象描述符的注册中定义，或与一个整数值相关。

ANY 类型用于 PKCS #7 中的 ContentInfo 类型中特定的 content 值，或 X.509's AlgorithmIdentifier 类型中特定算法的参数，或 X.501 中 Attribute 类型和 AttributeValueAssertion 类型的属性值。Attribute 类型广泛应用在 PKCS #6, #7, #8, #9 and #10 中，AttributeValueAssertion 类型用于 X.501 DN 中。

ASN.1 表示法:

ANY [DEFINED BY *identifier*]

这里 ***identifier*** 是可选的识别符。

在 ANY 形式下，实际的类型是不确定的。

ANY DEFINED BY ***identifier*** 形式只能出现在 SEQUENCE 或 SET 类型的组件中，对于这些组件 ***identifier*** 定义了一些其他的组件，其类型为 INTEGER 或 OBJECT IDENTIFIER（或者是在这两者基础上添加标签生成的类型）。这种形式下，实际的类型由其他组件的值决定，其他组件的值可能由对象描述符注册也可能在一个整数表里。

例： X.509 的 AlgorithmIdentifier 类型有一个 ANY 类型的组件：

```
AlgorithmIdentifier ::= SEQUENCE {
  algorithm OBJECT IDENTIFIER,
  parameters ANY DEFINED BY algorithm OPTIONAL }
```

这里 parameter 组件的实际类型由 algorithm 组件的值决定。实际类型在组件的对象描述符的注册中定义。

BER encoding. 与实际值的 BER 编码相同。

例： `parameter` 组件的 `BER` 编码值是实际类型值的 `BER` 编码。实际类型在 `algorithm` 组件的对象描述符值的注册中定义。

DER encoding. 与实际值的 DER 编码相同。

5.4 BIT STRING

`BIT STRING` 类型表示任意的 `0` 和 `1` 的比特流。一个 `BIT STRING` 值的长度可以是任意值，包括 `0`。该类型为 `string` 类型。

`BIT STRING` 类型用于 PKCS #6 `ExtendedCertificate` 类型中的 extended certificates 的 digital signatures，或 X.509 `Certificate` 类型的 certificates 的 digital signatures，或 X.509 `SubjectPublicKeyInfo` 类型的 certificates 的 public keys。

ASN.1 表示法:

```
BIT STRING
```

例： X.509 的 `SubjectPublicKeyInfo` 类型有一个 `BIT STRING` 类型的组件：

```
SubjectPublicKeyInfo ::= SEQUENCE {
  algorithm AlgorithmIdentifier,
  publicKey BIT STRING }
```

BER encoding. 简单或结构化的编码。在简单编码中，第一个内容字节指出了该 bit string 凑成 8 的倍数所缺少的 bit 数（这些 bit 称为无用的比特）。第二个和随后的内容字节指出了转换成字节字符串的 bit string 的值。转换过程如下：

1. 在 bit string 的结尾填充 0 至 7 个任意值的比特，使整个 bit string 的长度是 8 的倍数。如果比特流的长度已经是 8 的倍数，则不需要填充
2. 填充后的比特流被分成 octet。前八个 bit 作为第一个 octet，分别为 bit8 至 bit1；这样一直分割下去，直到最后 8 个 bit 组成 octet。

在结构化编码中，contents octets 是 bit string 连续子串的 BER 编码的串联。除了最后一个子串，每个子串都是 8bit 的整数倍。

例： `BIT STRING` "011011100101110111"的 BER 编码可以是下列任意一种，区别在于填充比特的选择、长度字节的格式、及编码是简单还是结构化的。

```
03 04 06 6e 5d c0                       DER encoding

03 04 06 6e 5d e0                       padded with "100000"

03 81 04 06 6e 5d c0                    long form of length octets

23 09                   constructed encoding: "0110111001011101" + "11"
   03 03 00 6e 5d
   03 02 06 c0
```

DER encoding. 简单编码。内容字节与 BER 简单编码类似，只是填充比特全部为 0 bit。

例： `BIT STRING` 的"011011100101110111" DER 编码为：

```
03 04 06 6e 5d c0
```

5.5 CHOICE

CHOICE 类型表示一个或多个备选项的联合体。

CHOICE 类型用于表示扩展证书的联合体和 PKCS #7 的 ExtendedCertificateOrCertificate 类型中的 X.509 证书。

ASN.1 表示法:

CHOICE {

[*identifier*$_1$] *Type*$_1$,

□□,

[*identifier*$_n$] *Type*$_n$ }

这里 ***identifier***$_1$，……，***identifier***$_n$ 是可选的，不同的 identifiers 表示不同的备选项。***Type***$_1$，……，***Type***$_n$ 是备选项的类型。Identifiers 主要是为了书面表示，并不影响类型的值或类型的编码。

类型必须具有不同的标签值。可以在备选项上添加显式或隐式标签来满足此要求。

例: PKCS #7 的 ExtendedCertificateOrCertificate 类型是一个 CHOICE 类型：

```
ExtendedCertificateOrCertificate ::= CHOICE {
  certificate Certificate, -- X.509
  extendedCertificate [0] IMPLICIT ExtendedCertificate }
```

这里备选项的 identifiers 是 certificate 和 extendedCertificate，备选项的类型是 Certificate 和[0] IMPLICIT ExtendedCertificate。

BER encoding. 与被选中的选项的 BER 编码相同。由于备选项的标签值不同，所以可以区分出不同的 BER 编码。

例: 如果选择 certificate 选项，则 BER 编码的 identifier octets 是 30，如果选择的是 extendedCertificate 选项则为 a0。

DER encoding. 与被选中的选项的 DER 编码相同。

5.6 IA5String

IA5String 类型表示由 IA5 字符组成的任意流。IA5 代表 International Alphabet 5，与 ASCII 相同。该字符集包括不可打印的控制字符。一个 IA5String 值长度可以为任意值，包括 0。该类型属于 string 类型。

IA5String 类型用于 PKCS #9 的 electronic-mail address, unstructured-name, and unstructured-address 属性。

ASN.1 表示法:

IA5String

BER encoding. 简单或结构化的编码方式。使用简单编码方式时，内容字节为 IA5 string 格式的字符，用 ASCII 编码。如果使用结构化编码方式，内容字节为 IA5 string 格式的连续子串的 BER 编码的串联。

例： 根据长度字节的形式和采用简单还是结构化编码方式，IA5String "test1@rsa.com" 的 BER 编码可以是下列任一种：

```
16 0d 74 65 73 74 31 40 72 73 61 2e 63 6f 6d   DER encoding

16 81 0d                                       long form of length octets
   74 65 73 74 31 40 72 73 61 2e 63 6f 6d

36 13                          constructed encoding: "test1" + "@" + "rsa.com"
   16 05 74 65 73 74 31
   16 01 40
   16 07 72 73 61 2e 63 6f 6d
```

DER encoding. 简单化编码。Contents octets 与 BER 编码相同。

例： IA5String 类型值"test1@rsa.com" is 的 DER 编码为：

```
16 0d 74 65 73 74 31 40 72 73 61 2e 63 6f 6d
```

5.7 INTEGER

INTEGER 类型表示任意的整数。INTEGER 值可以为正数、负数或 0，具有任意大小。

在整个 PKCS 中 INTEGER 类型用于表示版本号；也用于表示密码值，如在 PKCS #1 的 RSAPublicKey 和 RSAPrivateKey 类型及 PKCS #3 的 DHParameter 类型中的模数、指数或素数等；在 PKCS #5 的 PBEParameter 类型中表示信息摘要重复次数；在 X.509 Certificate 类型中表示版本号和序列号。

ASN.1 表示法:

INTEGER [{ $identifier_1(value_1)$ $identifier_n(value_n)$ }]

这里 $identifier_1$，……，$identifier_n$ 是可选的不同识别符，$value_1$，……，$value_n$ 是可选的整数值。识别符如果出现就与类型值相关联。

例： X.509 的 Version 类型是一个 INTEGER 类型，其可识别的值为：

```
Version ::= INTEGER { v1988(0) }
```

识别符 v1988 与值 0 相关联。X.509 的 Certificate 类型使用识别符 v1988 给 version 组件指定了默认值 0：

```
Certificate ::= ...
  version Version DEFAULT v1988,
...
```

BER encoding. 简单编码。内容字节以 2 的补码形式给出了整数值，基于 256，最高位在先，以使用最少的字节。值 0 编码为一个 00 字节。

表 3 列出了一些 BER 编码的例子（同时也是 DER 编码的例子）：

Integer value	BER encoding
0	02 01 00
127	02 01 7F
128	02 02 00 80
256	02 02 01 00
-128	02 01 80
-129	02 02 FF 7F

DER encoding. 简单化编码。内容字节与 BER 编码相同。

5.8 NULL

NULL 类型表示一个 null 值。

在 PKCS 的几个地方 NULL 类型用于算法参数。

ASN.1 表示法:

```
NULL
```

BER encoding. 简单编码。内容字节为空。

例: 根据长度字节的形式，NULL 值的 BER 编码可以为下列任意一种或其他：

```
05 00

05 81 00
```

DER encoding. 简单编码。内容为空。NULL 值的 DER 编码固定为 05 00。

5.9 OBJECT IDENTIFIER

OBJECT IDENTIFIER 类型表示一个对象识别符，由一列整数组件组成，用于识别一个对象，如算法、属性类型、或定义了其他对象识别符的注册机构。一个 OBJECT IDENTIFIER 值可以有任意个组件，组件通常可以为任意非负值。该类型是 non-string 类型。

OBJECT IDENTIFIER 值由注册机构指定其含义。每个注册机构负责由一个指定序列开始的所有组件的序列。通常，注册机构会把序列的子集委派给下属的其他注册机构，或者指定给对象的特殊类型。通常至少有两个组件。

OBJECT IDENTIFIER 类型在 PKCS #7 的 ContentInfor 类型中用于识别 content，在 X.509 的 AlgorithmIdenifier 类型中用于识别算法。在 X.501 的 Attribute 和 AttributeValueAssertion 类型中用于识别属性。Attribute 类型广泛用于 PKCS #6, #7, #8, #9, and #10，AttributeValueAssertion 类型用于 X.501 distinguished names。OBJECT IDENTIFIER 值在整个 PKCS 中定义。

ASN.1 表示法:

```
OBJECT IDENTIFIER
```

OBJECT IDENTIFIER 类型值的 ASN.1 符号为：

$$\{\ [identifier]\ component_1\ \square\square\ component_n\ \}$$

$$component_i = identifier_i \mid identifier_i\ (value_i) \mid value_i$$

这里 ***identifier***, ***identifier***$_1$，……，***identifier***$_n$ 是识别符，***value***$_1$，……，***value***$_n$ 是可选的整数值。

没有 ***identifier*** 的形式是由组件组成的完整的值，有 ***identifier*** 的形式使用另一个对象识别符的值简化了开始的组件。识别符 ***identifier***$_1$，……，***identifier***$_n$ 主要用于书写，但是当与整数值同时出现时必须对应。只有当识别符在 X.208 定义的一小部分识别符范围内时才可以省略整数值。

例： 下列值都是分配给 RSA Data Security, Inc.的对象识别符：

```
{ iso(1) member-body(2) 840 113549 }
{ 1 2 840 113549 }
```

（在下面给出了 ASN.1 值的符号的例子中，对象识别符的值为 10 进制而不是 16 进制）表 4 列举了一些对象识别符的值和它们的含义。

Object identifier value	Meaning
{ 1 2 }	ISO member bodies
{ 1 2 840 }	US (ANSI)
{ 1 2 840 113549 }	RSA Data Security, Inc.
{ 1 2 840 113549 1 }	RSA Data Security, Inc. PKCS
{ 2 5 }	directory services (X.500)
{ 2 5 8 }	directory services—algorithms

Table 4. Some object identifier values and their meanings.

BER encoding. 简单编码。内容字节如下，其中 *value*$_1$，……，*value*$_n$ 表示在整个对象描述符中组件的整数值：

1. 第一个字节值为 `40` ×`value1 + value2.` (这是唯一的，由于 `value1` 的值限制为 `0, 1,`和 `2`；当 `value1` 为 `0` 或 `1` 时 `value2` 限制在 `0` 至 `39` 之间；根据 `X.208`，`n` 总是至少为 `2.`)
2. 如果有后续的字节，编码为 `value3`，……，`valuen`。每个值基于 `128` 编码，最高位在先以保证使用尽可能少的数字，除了最后一个字节外，每个 `octet` 的最高位都置为 `1`。

例： RSA Data Security, Inc.的对象描述符的 BER 编码的第一个字节是 $40\times1 + 2 = 42 = 2a_{16}$。840 的编码为 $6\times128 + 48_{16}$ 即 `86 48`， 113549 的编码为 $6\cdot128^2 + 77_{16}\cdot128 + d_{16}$ 是 `86 f7 0d`。最后的 BER 编码为：

```
06 06 2a 86 48 86 f7 0d
```

DER encoding. 简单编码。内容字节与简单的 BER 编码相同。

5.10 OCTET STRING

`OCTET STRING` 类型表示任意的字节流。一个 `OCTET STRING` 值可以为任意长度，包括 `0`。该类型为 `string` 类型。

`OCTET STRING` 类型用于 `PKCS #5` 的 `PBEParmeter` 类型的 salt 值，也用于 PKCS #7 的信息摘要、加密信息、加密信息摘要和加密内容，还用于 PKCS #8 的私钥和加密私钥。

ASN.1 表示法:

```
OCTET STRING [SIZE ({size | size1..size2})]
```

这里 ***size***, $size_1$，和 $size_2$ 是可选的尺寸限制。在 `OCTET STRING SIZE (`***size***`)`形式下，octet string 必须包含 ***size*** 个 octets。在 `OCTET STRING SIZE (`$size_1$`..`$size_2$`)` 形式下，octet string 必须包含 size1 至 size2 个 octets。在 `OCTET STRING` 形式下，octet string 大小任意。

例: PKCS #5 的 `PBEParameter` 类型有一个类型为 `OCTET STRING` 的组件：

```
PBEParameter ::= SEQUENCE {
  salt OCTET STRING SIZE(8),
  iterationCount INTEGER }
```

这里组件 `salt` 的大小总是 8 个字节。

BER encoding. 简单或结构化编码。如果使用简单化编码，内容字节给出了 octet string 的值，从第一个字节到最后一个。如果使用结构化编码，内容字节给出了 `OCTET STRING` 值子串的 BER 编码的串联。

例: 根据长度字节的形式和采用简单化编码还是结构化编码，`OCTET STRING` 类型值 `01 23 45 67 89 ab cd ef` 的 `BER` 编码可以为下列任一种：

```
04 08 01 23 45 67 89 ab cd ef                    DER encoding

04 81 08 01 23 45 67 89 ab cd ef      long form of length octets

24 0c                 constructed encoding: 01 ... 67 + 89 ... ef
   04 04 01 23 45 67
   04 04 89 ab cd ef
```

DER encoding. 简单化编码。Contents octets 与 BER 编码相同。

例: `OCTET STRING` 类型值 `01 23 45 67 89 ab cd ef` 的 DER 编码为：

```
04 08 01 23 45 67 89 ab cd ef
```

5.11 PrintableString

`PrintableString` 类型表示由可打印字符组成的任意流。可打印字符列于下列集合：

A, B, □□, Z

a, b, □□, z

0, 1, □□, 9

(space) ' ( ) + , - . / : = ?

该类型为 string 类型。

`PrintableString` 用于 PKCS #9 的 challenge-password 和 unstructuerd-address 属性，也用于几个 X.521 distinguished names attributes.

ASN.1 表示法:

```
PrintableString
```

BER encoding. 简单或结构化编码。如果使用简单编码，内容字节给出了用 ASCII 编码的可打印字符流。如果使用结构化编码，内容字节给出了流的连续子串的 BER 编码的串联。

例: 根据长度字节的形式和采用哪种编码方法，可打印字符串"Test User 1"的 BER 编码可以为下列任一种：

```
13 0b 54 65 73 74 20 55 73 65 72 20 31          DER encoding

13 81 0b 54 65 73 74 20 55 73 65 72 20 31
                                        long form of length octets

33 0f                                  constructed encoding: "Test " + "User 1"
   13 05 54 65 73 74 20
   13 06 55 73 65 72 20 31
```

DER encoding. 简单编码。内容字节与 BER 编码相同。

例: 可打印字符流"Test User 1" 的 DER 编码为：

```
13 0b 54 65 73 74 20 55 73 65 72 20 31
```

5.12 SEQUENCE

SEQUENCE 类型表示一个或多个类型的有序集合。

SEQUENCE 类型用于整个 PKCS 和相关标准。

ASN.1 表示法:

SEQUENCE {

[*identifier*$_1$] *Type*$_1$ [{OPTIONAL | DEFAULT *value*$_1$}],

□□

[*identifier*$_n$] *Type*$_n$ [{OPTIONAL | DEFAULT *value*$_n$}]}

这里 ***identifier*$_1$** ，……， ***identifier*$_n$** 是可选的、针对不同组件不同的识别符，***Type*$_1$**，……，***Type*$_n$** 是组件的类型，***value*$_1$**，……，***value*$_n$** 是组件的可选默认值。识别符主要用于书写，并不影响类型的值或编码。

OPTIONAL 限定词表示组件的值是可选的，不一定出现在序列里。DEFAULT 限定词也表示该值是可选的，如果未指定组件则分配一个默认值给它。

任何带有 OPTIONAL 或 DEFAULT 限定词的组件类型，及其后的组件都必须具有不同的标签。为了满足这一要求，最好在一些组件前加上隐式或显式的标签。

例: X.509 的 Validity 类型是一个带有两个组件的 SEQUENCE 类型：

```
Validity ::= SEQUENCE {
  start UTCTime,
  end UTCTime }
```

这里组件的识别符是 start 和 end，组件类型都是 UTCTime。

BER encoding. 结构化编码。内容字节是序列组件的 BER 编码按照定义的顺序的串联，对于带有 OPTIONAL 和 DEFAULT 识别符的组件需遵守下列规则：

- 如果序列中带有 OPTIONAL 或 DEFAULT 限定词的组件的值未给出，则内容字节中不应该包含该组件的编码。
- 如果带有 DEFAULT 限定词的组件值为默认值，则内容字节中可以包含、也可以不包含该组件的编码。

DER encoding. 结构化编码。内容字节与 BER 编码相同，但是使用 DEFAULT 限定词的组件的值如果是默认值，则内容字节不包含该组件的编码。

5.13 EQUENCE OF

SEQUENCE OF 类型表示某个给定类型出现 0 次或多次的有序集合。

SEQUENCE OF 类型用于 X.501 distinguished names。

ASN.1 表示法:

```
SEQUENCE OF Type
```

这里 ***Type*** 即为类型。

例: X.501 的 RDNSequence 类型包括 0 个或多个 RelativeDistinguishedName 类型，意义最重大的出现在前面：

```
RDNSequence ::= SEQUENCE OF RelativeDistinguishedName
```

BER encoding. 结构化的编码。Contents octets 是按照集合中类型值出现顺序的 BER 编码的串联。

DER encoding.结构化的编码。Contents octets 是按照集合中类型值出现顺序排列的 DER 编码的串联。

5.14 SET

SET 类型表示一个或多个类型的无序集合。

SET 类型在 PKCS 无任何应用。

ASN.1 表示法:

```
SET {
  [identifier1] Type1 [{OPTIONAL | DEFAULT value1}],
  □□,
  [identifiern] Typen [{OPTIONAL | DEFAULT valuen}]}
```

这里 $\boldsymbol{identifier_1}$，……，$\boldsymbol{identifier_n}$ 是可选的、不同的组件识别符。$\boldsymbol{Type_1}$，……，$\boldsymbol{Type_n}$ 是组件的类型。$\boldsymbol{value_1}$，……，$\boldsymbol{value_n}$ 是可选的组建默认值。识别符主要用于书写，不影响类型的值或编码。

OPTIONAL 限定词表示组件的值是可选的，不一定在 set 中出现。DEFAULT 限定词也表示组件值是可选的，如果该组件缺省则使用默认值。

类型必须具有不同的标签。可以在一些组件前添加隐式或显式标签满足这一要求。

BER encoding. 结构化编码。Contents octets 是 set 组件值的 BER 编码的串联，可以是任意顺序的。带有 OPTIONAL 和 DEFAULT 限定词的组件需满足下列规则：

- 如果 set 中带有 OPTIONAL 或 DEFAULT 限定词的组件值未指定，则内容字节中不需要包含该组件的 BER 编码。
- 如果带有 DEFAULT 限定词的组件值为默认值，则内容字节中可以包含、也可以不包含该组件的 BER 编码。

DER encoding. 结构化的编码。除以下两点外，内容字节与 BER 编码相同：

1. 如果带有DEFAULT限定词的组件值为默认值，则不包含该组件的编码。
2. 组件是有顺序的，按照标签升序排列。

5.15 SET OF

SET OF 类型表示一个给定类型出现 0 次或多次的无序集合。

SET OF 类型用于在 PKCS #6, #7, #8, #9 and #10 中表示属性集，或信息摘要算法识别符、签发者信息和 PKCS #7 中重复信息的集合，或 X.501 distinguished names。

ASN.1 表示法:

```
SET OF Type
```

这里 ***Type*** 即为类型。

例： X.501 的 RelativeDistinguishedName 类型由 0 个或多个 AttributeValueAssertion 类型组成，但不关心其出现的顺序：

```
RelativeDistinguishedName ::=
  SET OF AttributeValueAssertion
```

BER encoding.结构化编码。内容字节是集合中出现的值的 BER 编码按任意顺序的串联。

DER encoding. 结构化编码。内容字节与 BER 编码相同，但是 BER 编码是按照字典升序的顺序排列的。所谓字典升序是按如下步骤进行的：在较短的 BER 编码的最后一个字节后面逻辑上填充伪字节，其值小于任何正常的字节。从左至右比较 BER 编码，直到出现差别。不同点字节值较小的那个即为较小的 BER 编码。

5.16 T61String

T61String 类型表示由 T.61 字符组成的任意流。T.61 是 ASCII 字符集的 8bit 扩展。特殊的"escape"序列定义了随后字符作为其他语言（如日语）的解释。初始解释为拉丁语。该字符集包括不可打印的控制字符。T61String 类型只允许拉丁语和日语的解释，及实现者对不支持的控制字符目录名的声明[NIST92]。一个 T61String 值可以为任意长度，包括 0。该类型属于 string 类型。

T61String 类型用于 PKCS #9 的 unstructured-address 和 challenge-password attributes,，及几个 X.521 attributes。

ASN.1 表示法:

```
T61String
```

BER encoding. 简单化或结构化编码。如果使用简单化编码，内容字节为 T.61 流形式的字符，按 ASCII 编码。如果使用结构化编码，内容字节为 T.61 流子串的 BER 编码的串联。*例：：* 根据长度字节的形式和采用哪种编码方法，T61String 类型值"cl s publiques" (French for "public keys")的 BER 编码可以为下列任一种：

```
14 0f                                                DER encoding
   63 6c c2 65 73 20 70 75 62 6c 69 71 75 65 73

14 81 0f                                    long form of length octets
   63 6c c2 65 73 20 70 75 62 6c 69 71 75 65 73

34 15                    constructed encoding: "cl   s" + " " + "publiques"
   14 05 63 6c c2 65 73
   14 01 20
   14 09 70 75 62 6c 69 71 75 65 73
```

8 比特字符 c2 是一个 T.61 前缀，在下一个字符前添加一个重音符。

DER encoding. 简单化编码。内容字节同简单 BER 编码。

例： T61String 类型值"cl s publiques"的 DER 编码为：

```
14 0f 63 6c c2 65 73 20 70 75 62 6c 69 71 75 65 73
```

5.17 UTCTime

UTCTime 类型表示"coordinated universal time"或格林威治平均时间值。一个 UTCTime 值包括精确到分钟或秒的本地时间，及相对于 GMT 的偏移（单位为小时和分钟）。可以采用下列任一种形式：

YYMMDDhhmmZ
YYMMDDhhmm+hh'mm'
YYMMDDhhmm-hh'mm'
YYMMDDhhmmssZ
YYMMDDhhmmss+hh'mm'
YYMMDDhhmmss-hh'mm'

其中：

YY 是年份的末两位

MM 是月份(01 to 12)

DD 是日 (01 to 31)

hh 是小时(00 to 23)

mm 是分钟(00 to 59)

ss 是秒 (00 to 59)

Z 表示本地时间是 GMT，+ 表示本地时间落后于GMT，- 表示本地时间提前于 GMT

hh' 是与GMT相差的绝对小时值

mm' 是与GMT相差的绝对分钟值

该类型为 string 型。

UTCTime 类型用 PKCS #9 的 signing-time attribute 的签发时间和 X.509 的 Validity 类型的证书有效期。

ASN.1 表示法:

```
UTCTime
```

BER encoding. 简单化或结构化编码。如果使用简单化编码，内容字节以流的形式给出字符，按 ASCII 编码。如果使用结构化编码，内容字节为流的子串的 BER 编码的串联。（一般不使用结构化编码，因为 UTCTime 值都很短，但是允许使用结构化编码。）

例： 写本句的时间为 4:45:40 p.m. Pacific Daylight Time on May 6, 1991，可以表示成下列任一种形式：

"910506164540-0700"

"910506234540Z"

其 BER 编码可以为：

```
17 0d 39 31 30 35 30 36 32 33 34 35 34 30 5a

17 11 39 31 30 35 30 36 31 36 34 35 34 30 2D 30 37 30
      30
```

DER encoding. 简单编码。内容字节与 BER 简单编码同。

6. 一个实例

本节介绍了一个关于 ASN.1 符号和 DER 编码的例子：X.501 type Name。

6.1 抽象表示法

本节给出了 X.501 类型 Name 的 ASN.1 表示法：

```
Name ::= CHOICE {  RDNSequence }

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::=SET OF AttributeValueAssertion

AttributeValueAssertion ::= SEQUENCE {
  AttributeType,
  AttributeValue }

AttributeType ::= OBJECT IDENTIFIER

AttributeValue ::= ANY
```

Name 类型用来标明一个 X.500 目录中的对象。Name 是由一个备选项组成的 CHOICE 类型： RDNSequence。（X.500 未来修订本可能会有其他备选项。）

RDNSequence 类型给出了一个在 X.500 目录中相对于根的路径。RDNSequence 是一个由 RelativeDistinguishedName 类型 0 次或多次出现组成的 SEQUENCE OF 类型。

RelativeDistinguishedName 类型为目录树中的一个对象指定了相对于其上级的唯一的名字。RelativeDistinguishedName 是由 AttributeValueAssertion0 次或多次出现组成的 SET OF 类型。

AttributeValueAssertion 类型为具有 relative distinguished name 的属性分配值，例如 country name 或 common name。AttributeValueAssertion 是由两个组件组成的 SEQUENCE 类型：AttributeType 类型和 AttributeValue 类型。

AttributeType 类型通过对象标识符来识别一个属性。AttributeValue 类型指定任一个属性值。属性值的实际类型由属性类型决定。

6.2 DER 编码

本节给出了一个 Name 类型值的 DER 编码的例子，由下至上方式（working from the bottom up.）

名字是 PKCS 例子中的 Test User 1 [Kal93]，由下列路径表示：

(root)
|
countryName = "US"
|
organizationName = "Example Organization"
|
commonName = "Test User 1"

每层对应一个 RelativeDistinguishedName，每层中出现的每项合起来组成了一个 AttributeValueAssertion 值。等号前为 AttributeType 值，等号后为 AttributeValue 值（指定属性类型的可打印字符串）。

CountryName、organizationName 合 commonUnitName 是 X.520 中定义的属性类型：

```
attributeType OBJECT IDENTIFIER ::=
  { joint-iso-ccitt(2) ds(5) 4 }

countryName OBJECT IDENTIFIER ::= { attributeType 6 }
organizationName OBJECT IDENTIFIER ::=
  { attributeType 10 }
commonUnitName OBJECT IDENTIFIER ::=
  { attributeType 3 }
```

6.2.1 AttributeType

三个 AttributeType 值是 OCTET STRING 类型值，所以 DER 编码为简单定长编码方法：

```
06 03 55 04 06                                    countryName

06 03 55 04 0a                                  organizationName

06 03 55 04 03                                     commonName
```

由于 OBJECT IDENTIFIER 类型的标签为 6，所以标识符字节部分采用小标签形式。Bits 8 和 7 为 0，表示为 universal 类， bit 6 为"0"表示采用的是简单编码。长度字节部分采用短型。内容字节部分是由子识别符（十进制）形成的三个字节流的串联：$40 \cdot 2 + 5 = 85 = 55_{16}$; 4; and 6, 10, or 3.

6.2.2 AttributeValue

三个 AttributeValue 值是 PrintableString 类型值，所以采用简单化、定长编码方法：

```
13 02 55 53                                                    "US"

13 14                                                "Example Organization"
   45 78 61 6d 70 6c 65 20 4f 72 67 61 6e 69 7a 61
   74 69 6f 6e

13 0b                                                        "Test User 1"
   54 65 73 74 20 55 73 65 72 20 31
```

由于的 PrintableString 标签为 19（十进制），介于 0 和 30 之间，所以标识符字节采用小标签形式。由于 PrintableString 属于 universal 类，所以 Bits 8 和 7 值为 0。编码方式为简单的，所以 Bit 6 为 0。长度字节部分采用短型，内容字节是属性值的 ASCII 码表示。

6.2.3 AttributeValueAssertion

三个 AttributeValueAssertion 值是 SEQUENCE 类型值，所以其 DER 编码为结构化的、定长编码方法：

```
30 09                                          countryName = "US"
   06 03 55 04 06
   13 02 55 53

30 1b                  organizationName = "Example Organizaiton"
   06 03 55 04 0a
   13 14 ... 6f 6e

30 12                                  commonName = "Test User 1"
   06 03 55 04 0b
   13 0b ... 20 31
```

由于 SEQUENCE 类型的标签为 16（十进制），介于 0 与 30 之间，所以标识符部分采用小标签形式。因为 SEQUENCE 属于 universal 类，所以 Bits 8 和 7 值为 0。由于采用结构化编码，所以 Bit 6 为 1。长度字节部分采用短型， 内容字节是 attributeType 和 attributeValue 组件 DER 编码的串联。

6.2.4 RelativeDistinguishedName

三个 RelativeDistinguishedName 值为 SET OF 类型值，所以其 DER 编码为结构化定长编码方法：

```
31 0b
   30 09 ... 55 53

31 1d
   30 1b ... 6f 6e

31 14
   30 12 ... 20 31
```

由于 SET OF 的标签为 17（十进制），介于 0 和 30 之间，所以标识符字节采用小标签形式。由于 SET OF 属于 universal 类，所以 Bits 8 和 7 值为 0。采用结构化编码方法所以 Bit 6 为 1。长度字节部分采用短型。由于每个 set 中只有一个值，所以内容字节部分是各个 AttributeValueAssertion 类型值的分别的 DER 编码。

6.2.5 RDNSequence

RDNSequence 值是一个 SEQUENCE OF 类型值，所以它的 DER 编码为结构化的、定长编码方法：

```
30 42
   31 0b ... 55 53
   31 1d ... 6f 6e
   31 14 ... 20 31
```

由于 SEQUENCE OF 类型的标签为十进制的 16，介于 0 和 30 之间，所以标识符字节部分采用小标签形式。因为 SEQUENCE OF 属于 universal 类，所以 Bits 8 和 7 值为"0"。结构化编码所以 Bit 6 为"1"。长度字节部分采用短型，内容字节部分是按照出现顺序排列的三个 RelativeDistinguishedName 值的 DER 编码的串联。

6.2.6 Name

Name 值为 CHOICE 类型值，所以其 DER 编码与 RDNSequence 的值相同：

```
30 42
   31 0b
      30 09
         06 03 55 04 06                attributeType = countryName
         13 02 55 53                         attributeValue = "US"
   31 1d
      30 1b
         06 03 55 04 0a            attributeType = organizationName
         13 14                  attributeValue = "Example Organization"
            45 78 61 6d 70 6c 65 20 4f 72 67 61 6e 69 7a 61
            74 69 6f 6e
   31 14
      30 12
         06 03 55 04 03                attributeType = commonName
         13 0b                         attributeValue = "Test User 1"
            54 65 73 74 20 55 73 65 72 20 31
```

参考文献

PKCS #1 RSA Laboratories. *PKCS #1: RSA Encryption Standard.* Version 1.5, November 1993.

PKCS #3 RSA Laboratories. *PKCS #3: Diffie-Hellman Key-Agreement Standard.* Version 1.4, November 1993.

PKCS #5 RSA Laboratories. *PKCS #5: Password-Based Encryption Standard.* Version 1.5, November 1993.

PKCS #6 RSA Laboratories. *PKCS #6: Extended-Certificate Syntax Standard.* Version 1.5, November 1993.

PKCS #7 RSA Laboratories. *PKCS #7: Cryptographic Message Syntax Standard.* Version 1.5, November 1993.

PKCS #8 RSA Laboratories. *PKCS #8: Private-Key Information Syntax Standard.* Version 1.2, November 1993.

PKCS #9 RSA Laboratories. *PKCS #9: Selected Attribute Types.* Version 1.1, November 1993.

PKCS #10 RSA Laboratories. *PKCS #10: Certification Request Syntax Standard.* Version 1.0, November 1993.

X.200 CCITT. *Recommendation X.200: Reference Model of Open Systems Interconnection for CCITT Applications.* 1984.

X.208 CCITT. *Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1).* 1988.

X.209 CCITT. *Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).* 1988.

X.500 CCITT. *Recommendation X.500: The Directory – Overview of Concepts, Models and Services.* 1988.

X.501 CCITT. *Recommendation X.501: The Directory – Models.* 1988.

X.509 CCITT. *Recommendation X.509: The Directory – Authentication Framework.* 1988.

X.520 CCITT. *Recommendation X.520: The Directory – Selected Attribute Types.* 1988.

[Kal93] Burton S. Kaliski Jr. *Some Examples of the PKCS Standards.* RSA Laboratories, November 1993.

[NIST92] NIST. *Special Publication 500-202: Stable Implementation Agreements for Open Systems Interconnection Protocols.* Part 11 (Directory Services Protocols). December 1992.

Revision history

1991年6月3日版本

1991 年 6 月 3 日版式 PKCS 首次公开发布的一部分。它被发布为 NIST/OSI 实现示范文档 SEC-SIG-91-17。

1993年11月3日版本

1993 年 11 月 3 日版本增加了编辑修订，也增加了版本历史记录，这使得它和下面的 PKCS 文档版本保持一致：

PKCS #1: RSA Encryption Standard. Version 1.5, November 1993.

PKCS #3: Diffie-Hellman Key-Agreement Standard. Version 1.4, November 1993.

PKCS #5: Password-Based Encryption Standard. Version 1.5, November 1993.

PKCS #6: Extended-Certificate Syntax Standard. Version 1.5, November 1993.

PKCS #7: Cryptographic Message Syntax Standard. Version 1.5, November 1993.

PKCS #8: Private-Key Information Syntax Standard. Version 1.2, November 1993.

PKCS #9: Selected Attribute Types. Version 1.1, November 1993.

PKCS #10: Certification Request Syntax Standard. Version 1.0, November 1993.

下面部分作了重大修订:

Section 5: 增加了 T61String 类型的描述。

Section 6: 修改了名称，使其与 PKCS 例子保持一致性。

Author's address

Burton S. Kaliski Jr., Ph.D.
Chief Scientist
RSA Laboratories
100 Marine Parkway
Redwood City, CA 94065 USA

(415) 595-7703
(415) 595-4126 (fax)
burt@rsa.com