CRYPTOGRAPHY

MESIGI3906
ASN.1

JOHN CAGNOL / AMAURY DARSCH
DER CALCUL SCIENTIFIQUE / DER GÉNIE INFORMATIQUE

AGENDA

- Abstract Syntax Notation 1
  - Basic principles
  - Primitives types
  - Encoding rules
- Applications with RSA
  - PEM and DER key encoding
  - Decoding with OpenSSL

WELCOME

- You can't be a real country unless you have a beer and an airline. It helps if you have some kind of a football team, or some nuclear weapons, but at the very least you need a beer.

  Frank Zappa

- Welcome into the world of ASN.1
  - ASN.1 starts counting at 0, bits starts at 1
  - You can write things in 2 or 3 differents forms
  - There is a standard that standardize a standard...
  - You can't be a real geek if you do not know ASN.1

ASN.1

- Abstract Syntax Notation 1
  - A wonderful IUTF standard X.208
  - Encode a set of types with values
- Provide 2 encoding modes
  - BER : Basic Encoding Rule X.209
  - CER : Canonical Encoding Rule X.690
  - DER : Distinguished encoding Rule
- Supposed to be a unified representation
  - Complex in some cases, not fully understood
  - Used mainly in certificate X.509

ASN.1 TYPES

- ASN.1 is a notation for describing abstract types ans values
  - A value is abstracted with a type
  - Each type is associated with a tag
- ASN.1 defines 4 types
  - Universal X.208
  - Application
  - Private
  - Context specific

UNIVERSAL TYPES

- Universal types have the same meanings across all applications
- The tags are defined in X.208

Type	**Tag**
Integer	2
Bit string	3
Octet string	4
Null	5
Object indentifier	6
Sequence	16

SIMPLE UNIVERSAL TYPES

- Integer — Tag = 2
  - An arbitrary integer
- Bit string — Tag = 3
  - An arbitratry bit string
- Octet string — Tag = 4
  - An arbitrary string of octets
- Null — Tag = 5
  - The null value

STRUCTURED TYPES

- Sequence Tag = 16
  - Ordered collection of one or more types
- Sequence of Tag = 16
  - Ordered collection of zero or more types
- Set Tag = 17
  - Unrdered collection of one or more types
- Set of Tag = 17
  - Unordered collection of zero or more types

ASN.1 RSA PUBLIC KEY

- The RSA public key is a sequence of 2 integers :
  - The modulus
  - The exponent
- The ASN.1 equivalent notation is :

```
RSAPublicKey ::= SEQUENCE {
    modulus           INTEGER, -- n
    publicExponent    INTEGER  -- e
}
```

IMPLICIT & EXPLICIT TAG

- An implicit tag is derived from another type by changing the tag of the underlying type
  - `[[class] number] IMPLICIT Type`
  - `class = UNIVERSAL | APPLICATION | PRIVATE`
- Explicitly tagged types are those derived from other types by adding an outer tag to the underlying type.
  - `[[class] number] EXPLICIT Type`
  - `class = UNIVERSAL | APPLICATION | PRIVATE`

ASN.1 IMPLICIT EXAMPLE

```
PersonnalRecord ::= [APPLICATION 0] IMPLICIT SET {
   name                Name,
   title          [0]  VisibleString,
   number              EmployeeNUmber,
   dateOfHire     [1]  Date
   nameOfSpouse   [2]  Name
   children       [3]  IMPLICIT SEQUENCE OF
                       ChildInformation DEFAULT ()
}
```

AN INTRICATE NOTION

The default tagging for certificates varies depending on which standard you're using. The original X.509v1 definition used the ASN.1 default of explicit tags, with X.509v3 extensions in a separate module with implicit tags. The PKIX definition is quite confusing because the ASN.1 definitions in the appendices use TAGS IMPLICIT but mix in X.509v3 definitions which use explicit tags. Appendix A has such a mixture of implied implicit and implied explicit tags that it's not really possible to tell what tagging you're supposed to use. Appendix B (which first appeared in draft 7, March 1998) is slightly better, but still confusing in that it starts with TAGS IMPLICIT, but tries to partially switch to TAGS EXPLICIT for some sections (for example the TBSCertificate has an 'EXPLICIT' keyword in the definition which is probably intended to signify that everything within it has explicit tagging, except that it's not valid ASN.1). The definitions given in the body of the document use implicit tags, and the definitions of TBSCertificate and and TBSCertList have both EXPLICIT and IMPLICIT tags present. To resolve this, you can either rely entirely on Appendix B with the X.509v1 sections moved into a separate section declared without 'IMPLICIT TAGS', or use the X.509v3 definitions. The SET definitions consistently use implicit tags.

Peter Gutman
X.509 Style Guide

BER ENCODING (I)

- Basic Encoding Rule X.209
  - Represent an ASN.1 value as an octet string
- Three methods for encoding
  - primitive, definite-length encoding;
  - constructed, definite-length encoding;
  - constructed, indefinite-length encoding
- The simple type, except string uses the primitive encding. Other encoding are used depending on the type structure and the knowledge of the value size.

BER ENCODING (II)

- Identifier octets
  - The class and tag numbers of the ASN.1 value.
  - Indicates if the value is primitive or constructed.
- Length octets
  - For the definite-length method, the number of content octets.
  - For the constructed, indefinite-length method, indicates that the length is indefinite.

BER ENCODING (III)

- Contents octets
  - For the primitive, definite-length method, the concrete representation of the value.
  - For the constructed methods, the concatenation of the BER encodings of the components of the value.
- End-of-contents octets
  - For the constructed, indefinite-length method, these denote the end of the contents. For the other methods, these are absent.
  - Two octets: 00 00

PRIMITIVE METHOD (I)

- Used for simple type and type derived from simple type by implicit tagging
- Identifier octet
  - Low tag number – tag between 0 and 30
    - **Bit 8-7 : class value**
    - **Bit 6 : 0 (i.e primitive)**
    - **Bit 5-1 : tag number**
  - High tag number – tag above 30
    - Two or more octets
    - First octet is the low-tag number with bits 5-1 set to 1, followed by the tag number

PRIMITIVE METHOD (II)

- Primitive class encoding

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

- Example
  - 0x01: universal, primitive integer
  - 0x30: universal, primitive sequence

LENGTH ENCODING

- Short form
  - One octet
  - Length between 0 and 127
- Long form
  - Two to 127 octets
  - Bit 8 of first octet has value "1" and bits 7–1 gives the number of additional length octets. Second and following octets gives the length, base 256, most significant digit first.

CER MODEL

- The Canonical Encoding Rule model restricts the use of some BER features :
  - Force the indefinite length form for constructed encoding
  - Recommend primitive encoding for bit string and octet string

DER MODEL

- The Distinguished Encoding Rule model restricts the use of some BER features :
  - Force the definite length form with minimum number of octets
  - Do not use the constructed form of encoding for bit string and octet string

RSA WITH OPENSSL

- Generating the key
  - `openssl genrsa [options] [bits]`
- Example
  - `openssl –out rsa.pem 1024`
- The output file is in base 64 encoding as an ASN.1 sequence described in PKCS V2.1
  - PEM: Privacy Enhanced Mail
  - Both public and private keys are present in the form of an ASN.1 sequence

RSA ASN.1 NOTATION

```
RSAPrivateKey ::= SEQUENCE {
   version             Version,
   modulus             INTEGER, -- n
   publicExponent      INTEGER, -- e
   privateExponent     INTEGER, -- d
   prime1              INTEGER, -- p
   prime2              INTEGER, -- q
   exponent1           INTEGER, -- d mod (p-1)
   exponent2           INTEGER, -- d mod (q-1)
   coefficient         INTEGER, -- (inverse of q) mod p
   otherPrimeInfos   OtherPrimeInfos OPTIONAL
 }
```

DECODING THE RSA KEY

- PEM format
  - `asn1parse -in rsa.pem`
- PEM to DER form
  - `enc -d -base64 -in rsa.pem -out rsa.der`
- DER format
  - `asn1parse -inform DER -in rsa.der`