PKI Data Formats

Contents

Introduction

This document describes the most common Public Key Infrastructure (PKI) data formats and encodings.

Prerequisites

Requirements

Cisco recommends that you have knowledge of these topics:

- public-key cryptography (basic concepts).
- public-key infrastructure (basic concepts).

Components Used

This document is not restricted to specific software and hardware versions.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, make sure that you understand the potential impact of any command.

Conventions

Refer to Cisco Technical Tips Conventions for information on document conventions.

ASN.1 Notation

Abstract Syntax Notation One (ASN.1) is a formal language for the definition of data types and values, and how those data types and values are used and combined in various data structures. The goal of the standard is to define the abstract syntax of information without constraining how the information is encoded for transmission.

Here is an example excerpted from the *X.509 RFC*:

```
   Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }
CertificateSerialNumber ::= INTEGER
Validity ::= SEQUENCE {
notBefore Time,
notAfter Time }
Time ::= CHOICE {
utcTime UTCTime,
generalTime GeneralizedTime }
```

Refer to these documents from the International Telecommunication Union (ITU-T) standards sites:

- X.680 ASN.1: Specification of basic notation
- X.681 ASN.1: Information object specification
- X.682 ASN.1: Constraint specification
- X.683 ASN.1: Parameterization of ASN.1 specifications

ITU-T recommendations search - Search for **X.509** in **Rec. or standard** with **Language** set to **ASN.1**.

BER/CER/DER Encodings

The ITU-T has defined a standard way of encoding data structures described in ASN.1 into binary data. X.690 defines Basic Encoding Rules (BER) and its two subsets, Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). All three are based on **type-length-value** data fields packed in a hierarchical structure, which is built from **SEQUENCEs**, **SETs**, and **CHOICEs**, with these differences:

- BER provides multiple ways of encoding the same data, which is not suited for crypto operations.
- CER provides unequivocal encoding and uses indefinite length data, with an end-of-data marker in specific cases.
- DER provides unequivocal encoding and uses explicit length tags in specific cases.
- Among the three, DER is the one that is usually encountered when dealing with PKI and crypto payloads.

Example: In DER, the 20-bit value `1010 1011 1100 1101 1110` is encoded as :

- **tag**: `0x03` (bitstring)
- **length**: `0x04` (bytes)
- **value**: `0x04ABCDE0`
- **complete DER encoding**: `0x030404ABCDE0`

The leading **`04`** means that the last 4 bits (equals the trailing **`0`** digit) of the encoded value must be discarded because the encoded value does not end on a byte boundary.

Refer to these documents from the TU-T standards site:

- X.690 ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

From the Wikipedia site, refer to these documents:

- Basic Encoding Rules
- Canonical Encoding Rules
- Distinguished Encoding Rules

DER hex Dump

Cisco IOS, Adaptive Security Appliance (ASA), and other devices display DER content as a **hex dump** with the **show running-config** command. Here is the output:

```
crypto pki certificate chain root
certificate ca 01
30820213 3082017C A0030201 02020101 300D0609 2A864886 F70D0101 04050030
1D310C30 0A060355 040B1303 54414331 0D300B06 03550403 1304726F 6F74301E
170D3039 30373235 31313436 33325A17 0D313230 37323431 31343633 325A301D
...
```

This kind of hex dump can be converted back to DER in various ways. For instance, you can remove the space characters and pipe it to the **xxd program**:

```
$ cat ca.hex | tr -d ' ' | xxd -r -p -c 32 | openssl x509 -inform der -text -noout
```

Another easy way is to use this Perl script :

```
#!/usr/bin/perl
foreach (<>) {
s/[^a-fA-F0-9]//g;
print join("", pack("H*", $_));
} $ perl hex2der.pl < hex-file.txt > der-file.der
```

In addition, a compact way to convert **cert dumps**, each one previously manually copied to a file with extension **.hex**, from a **bash** command line as shown here:

```
for hex in *.hex; do
b="${hex%.hex}"
hex2der.pl < "$hex" > "$b".der
openssl x509 -inform der -in "$b".der > "$b".pem
openssl x509 -in "$b".pem -text -noout > "$b".txt
done
```

Each file results in:

- **file.hex -** The original file (must contain hex digits only).
- **file.der -** Certificate in DER (binary) format.
- **file.pem -** Certificate in PEM (Base64 + header/footer) format.

- **file.txt -** User-friendly, readable version of the certificate.

Base64 Encoding

Base64 encoding represents the binary data with only 64 printable characters (`A-Za-z0-9+/`) similarly to **uuencode**. In the conversion from binary to Base64, every 6-bit block of the original data is encoded into an 8-bit printable ASCII character with a translation table. Therefore, the size of the data after encoding has increased by 33 percent (data times 8 divided by 6 bits, equals 1.333).

A 24-bit buffer is used for translation of three (3) groups of eight (8) bits into four (4) groups of six (6) bits. Therefore one (1) or two (2) bytes of padding might be required at the end of the input data stream. Padding is indicated at the end of the Base64-encoded data, by one equals (`=`) sign for each group of eight (8) padding bits added to the input during encoding.

Refer to this example from Wikipedia.

Refer to the most recent information in RFC 4648: The Base16, Base32, and Base64 Data Encodings.

PEM Encoding

Privacy Enhanced Mail (PEM) is a full Internet Engineering Task Force (IETF) PKI standard in order to exchange secure messages. It is no longer widely used as such, but its encapsulation syntax has been widely borrowed in order to format and exchange Base64-encoded PKI-related data.

The PEM RFC 1421, section 4.4: Encapsulation Mechanism, defines PEM messages as delimited by Encapsulation Boundaries (EBs), which are based on RFC 934, with this format:

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
Header: value
Header: value
...

Base64-encoded data
...
-----END PRIVACY-ENHANCED MESSAGE-----
```

In practice today, when PEM-formatted data is distributed, this boundary format is used:

```
-----BEGIN type-----
...
-----END type-----
```

type can be with other keys or certificates such as:

- `RSA PRIVATE KEY`
- `ENCRYPTED PRIVATE KEY`
- `CERTIFICATE`
- `CERTIFICATE REQUEST`
- `X509 CRL`

Note: Although the RFCs do not make this mandatory, the number of leading and trailing dashes (-) in the EBs is significant and should always be five (5). Otherwise, some applications, such

as OpenSSL, choke on the input. On the other hand, other applications, such as Cisco IOS, do not require EBs at all.

Refer to these most recent RFCs for further information:

- RFC 1421 : PEM Part I: Message Encryption and Authentication Procedures
- RFC 1422 : PEM Part II: Certificate-Based Key Management
- RFC 1423 : PEM Part III: Algorithms, Modes, and Identifiers
- RFC 1424 : PEM Part IV: Key Certification and Related Services

X.509 Certificates and CRLs

X.509 is a subset of X.500, which is an extended ITU specification about Open Systems Interconnection. It deals specifically with certificates and public keys and has been adapted as an Internet standard by the IETF. X.509 provides a structure and syntax, expressed in the RFC with ASN.1 Notation, in order to store certificate information and certificate revocation lists.

In an X.509 PKI, a CA issues a certificate that binds a public key, for example: a Rivest-Shamir-Adleman (RSA) or Digital Signature Algorithm (DSA) key to a particular Distinguished Name (DN), or to an alternative name such as an email address or fully qualified domain name (FQDN). The DN follows the structure in the X.500 standards. Here is an example:

```
CN=common-name,OU=organizational-
unit,O=organization,L=location,C=country
```

Because of the ASN.1 definition, X.509 data can be encoded into DER in order to be exchanged in binary form, and optionally, converted to Base64/PEM for text-based means of communication, such as copy-paste on a terminal.

- Refer to this ITU-T standards document X.509 Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks.
- Refer to RFC 5280: X.509 Certificate and Certificate Revocation List (CRL) Profile for more information.

PKCS Standards

The Public-Key Cryptography Standards (PKCS) are specifications from RSA Labs that have partly evolved into industry standards. Those most often encountered, deal with these topics; however, not all of them deal with data formats.

PKCS#1 (RFC 3347) - Covers the implementation aspects of RSA-based cryptography (crypto primitives, encryption/signature schemes, ASN.1 syntax).

PKCS#5 (RFC 2898) - Covers password-based key derivation.

PKCS#7 (RFC 2315) and **S/MIME RFC 3852 -** Defines a message syntax to transmit signed and encrypted data and related certificates. Often used simply as a container for X.509 certificates.

PKCS#8- Defines a message syntax to transport cleartext or encrypted RSA keypairs.

PKCS#9 (RFC 2985) - Defines additional object classes and identity attributes.

PKCS#10 (RFC 2986) - Defines a message syntax for Certificate Signing Requests (CSRs). A CSR is sent by an entity to a CA and contains the information to be signed by the CA, such as public key information, identity, and additional attributes.

PKCS#12 - Defines a container for packaging related PKI data (typically, **entity keypair** + **entity cert** + **root and intermediate CA certificates**) within a single file. It is an evolution of Microsoft's Personal Information Exchange (PFX) format.

Refer to these resources:

- Wikipedia article on PKCS
- RSA Labs page on PKCS

Related Information

- **Technical Support & Documentation - Cisco Systems**