ASN.1 key structures in DER and PEM

Section:
Cryptography

Author:
Paul Bakker

Published:
Oct 14, 2012

Last updated:
Apr 14, 2014

Sharing:
G+1

Introduction

Everybody loves PEM and the very documented ASN.1 structures that are used in saving cryptographic keys and certificates in a portable format. Well.. Everybody would if they would actually be documented. But it is rather a big feat to find what the structure is inside each DER or PEM formatted file.

As we need this information, we will share it here as well, to help others in their quest for knowledge and understanding ;)

ASN.1 and DER encoding

Within the RSA, PKCS#1 and SSL/TLS communities the Distinguished Encoding Rules (DER) encoding of ASN.1 is used to represent keys, certificates and such in a portable format. Although ASN.1 is not the easiest to understand representation formats and brings a lot of complexity, it does have its merits. The certificate or key information is stored in the binary DER for ASN.1 and applications providing RSA, SSL and TLS should handle DER encoding to read in the information.

PEM files

Because DER encoding results in a truly binary representation of the encoded data, a format has been devised for being able to send these in an encoding of printable characters so you can actually mail these things. The format I focus on now is the PEM format.

Most PEM formatted files we will see are generated by OpenSSL when generating or exporting an RSA private or public key and X509 certificates.

In essence PEM files are just base64 encoded versions of the DER encoded data. In order to distinguish from the outside what kind of data is inside the DER encoded string, a header and footer are present around the data. An example of a PEM encoded file is:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDMYfnvWtC8Id5bPKae5yXSxQTt
+Zpul6AnnZWfI2TtIarvjHBFUtXRo96y7hoL4VWOPKGCsRqMFDkrbeUjRrx8iL91
4/srnyf6sh9c8Zk04xEOpK1ypvBz+Ks4uZObtjnnitf0NBGdjMKxveTq+VE7BWUI
yQjtQ8mbDOsiLLvh7wIDAQAB
-----END PUBLIC KEY-----
```

The first and last line indicate the DER format that should be expected inside. The data inside is a base64 encoded version of the DER encoded information.

Formats

So that's all nice and well. But what IS the structure you should expect in each different file? Look below for explanation of different formats.

RSA Public Key file (PKCS#1)

The RSA Public key PEM file is specific for RSA keys.

It starts and ends with the tags:

Related articles:
- What external dependencies does mbed TLS rely on?
- How to generate a Certificate Request (CSR)
- How to generate a self-signed certificate
- How to encrypt and decrypt with RSA
- Migrating from PolarSSL-1.2 to the PolarSSL 1.3 branch
- RSA Key Pair generator
- Thread Safety and Multi Threading: concurrency issues
- Using an external RSA private key
- Migrating from mbed TLS 1.3 to mbed TLS 2.0
- How are error codes defined

```
-----BEGIN RSA PUBLIC KEY-----
BASE64 ENCODED DATA
-----END RSA PUBLIC KEY-----
```

Within the base64 encoded data the following DER structure is present:

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

Public Key file (PKCS#8)

Because RSA is not used exclusively inside X509 and SSL/TLS, a more generic key format is available in the form of PKCS#8, that identifies the type of public key and contains the relevant data.

It starts and ends with the tags:

```
-----BEGIN PUBLIC KEY-----
BASE64 ENCODED DATA
-----END PUBLIC KEY-----
```

Within the base64 encoded data the following DER structure is present:

```
PublicKeyInfo ::= SEQUENCE {
  algorithm       AlgorithmIdentifier,
  PublicKey       BIT STRING
}

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

So for an RSA public key, the OID is 1.2.840.113549.1.1.1 and there is a RSAPublicKey as the PublicKey key data bitstring.

RSA Private Key file (PKCS#1)

The RSA private key PEM file is specific for RSA keys.

It starts and ends with the tags:

```
-----BEGIN RSA PRIVATE KEY-----
BASE64 ENCODED DATA
-----END RSA PRIVATE KEY-----
```

Within the base64 encoded data the following DER structure is present:

```
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
}
```

Private Key file (PKCS#8)

Because RSA is not used exclusively inside X509 and SSL/TLS, a more generic key format is available in the form of PKCS#8, that identifies the type of private key and contains the relevant data.

The unencrypted PKCS#8 encoded data starts and ends with the tags:

```
-----BEGIN PRIVATE KEY-----
BASE64 ENCODED DATA
-----END PRIVATE KEY-----
```

Within the base64 encoded data the following DER structure is present:

```
PrivateKeyInfo ::= SEQUENCE {
  version         Version,
  algorithm       AlgorithmIdentifier,
  PrivateKey      BIT STRING
}

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

So for an RSA private key, the OID is 1.2.840.113549.1.1.1 and there is a RSAPrivateKey as the PrivateKey key data bitstring.

The encrypted PKCS#8 encoded data start and ends with the tags:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
BASE64 ENCODED DATA
-----END ENCRYPTED PRIVATE KEY-----
```

Within the base64 encoded data the following DER structure is present:

```
EncryptedPrivateKeyInfo ::= SEQUENCE {
  encryptionAlgorithm  EncryptionAlgorithmIdentifier,
  encryptedData        EncryptedData
}

EncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

EncryptedData ::= OCTET STRING
```

The EncryptedData OCTET STRING is a PKCS#8 PrivateKeyInfo (see above).

Did this help?

G+1

Let's be friends! Privacy Policy