Some Examples of the PKCS Standards

Burton S. Kaliski Jr. RSA Data Security, Inc. Redwood City, CA

June 3, 1991

Abstract. This document gives some examples of PKCS. The reader is assumed to be familiar with the members of the PKCS family, or at least to have read the PKCS overview.

1. Introduction

This document illustrates some of the PKCS standards with the following sequence of examples:

- 1. An example user, called "Test User 1," generates an RSA key pair and protects the private key with a password.
- 2. Test User 1 obtains a certificate for his or her public key from the RSA "NOTARY" certification authority. Test User 1 also obtains an extended certificate that certifies an electronic-mail address.
- 3. Test User 1 prepares a digitally signed message.

In the examples, long integers are written in hexadecimal, most significant octet first. As usual, BER encodings are displayed as hexadecimal octet strings. All BER encodings in this document are also DER encodings.

2. Generating a key pair and protecting the private key

The process of generating a key pair and protecting the private key breaks down into five steps:

- Generate an RSA key pair according to PKCS #1.
- Encode values of type RSAPublicKey and RSAPrivateKey according to PKCS #1 to represent the key pair in an algorithm-specific way.

- Encode values of type PrivateKeyInfo according to PKCS #8 to represent the private key in an algorithm-independent way.
- Encrypt the PrivateKeyInfo encoding with a password according to PKCS #5.
- Encode a value of type EncryptedPrivateKeyInfo according to PKCS #8 to represent the encrypted PrivateKeyInfo value in an algorithm-independent way.

2.1 Generate an RSA key pair

Test User 1 generates an RSA key pair according to PKCS #1.

In the example, the modulus n is the following 512-bit integer:

n= d4 b8 b1 7c d3 9e cf a3 ed 86 92 82 28 be 81 8c 94 b4 e0 7d 2c e2 97 40 d8 68 c0 1a 8c 05 b0 d1 e3 56 fe 2a 64 8f 75 66 ff 7e d6 de 95 6a 1c 71 1c c2 c4 53 0f ab ad e2 ca 76 12 c4 db da 26 89

The prime factors p and q of the modulus are:

p =eb 48 76 69 f8 02 72 98 2d 3e c4 cc e1 f9 26 33 9a 5c 68 bc b8 f1 a2 f2 2e d7 c6 c6 73 e6 21 f1

q = e7 73 aa ea e7 0e 5c 3b 7d 4d 8e 68 16 94 1a 37 f2 e4 cd 4c 52 d5 39 a3 c7 be c1 58 3a b8 36 19

The public exponent e is F_4 (65537):

e = 01 00 01

The private exponent d and other private-key parameters are as follows:

d=56 4d 9c 8c d5 5a c0 49 74 cb bd 53 1b 6e b7 26 af 0a 77 6d b1 b1 38 c6 71 46 86 71 fe cf 2c 3e 2e 0e 76 7c 16 8e 1c cc f7 90 f0 c3 f6 c5 50 6a dd 1f 25 f8 77 7d f5 40 a3 51 35 72 58 35 c4 81

 $d \mod p-1 = 34$ cd 8a 65 b5 d7 36 c8 98 3e e2 03 e9 22 f5 43 d7 7c ad b4 cc fc cd c8 f4 62 50 a5 f1 14 d0 c1

 $d \mod q - 1 =$ 02 06 0b df 83 e7 a7 3c c9 ea ac ad 12 4d 77 31 38 09 72 16 11 bf 55 82 ae 13 67 e3 5a 98 13 b9

 $q^{-1} \mod p = 1$ b 62 90 30 89 b4 d1 ce 3d 0a a1 39 81 53 4f a2 47 ca 83 01 b4 e9 62 90 db b3 db f8 f5 a6 59 b1

2.2 Encode RSAPublicKey and RSAPrivateKey values

Test User 1 encodes values of type RSAPublicKey and RSAPrivateKey according to PKCS #1 to represent the key pair in an algorithm-specific way.

The BER-encoded RSAPublicKey value is:

The RSAPublicKey value is later used in a certificate.

The BER-encoded RSAPrivateKey value is:

```
30 82 01 39
  02 01 00
                                                              version = 0
  02 41
                                                              modulus = n
      00 d4 b8 b1 7c d3 9e cf a3 ed 86 92 82 28 be 81
     8c 94 b4 e0 7d 2c e2 97 40 d8 68 c0 1a 8c 05 b0
     d1 e3 56 fe 2a 64 8f 75 66 ff 7e d6 de 95 6a 1c
      71 1c c2 c4 53 0f ab ad e2 ca 76 12 c4 db da 26
     89
   02 03 01 00 01
                                                       publicExponent = e
   02 40
                                                      privateExponent = d
     56 4d 9c 8c d5 5a c0 49 74 cb bd 53 1b 6e b7 26
     af 0a 77 6d b1 b1 38 c6 71 46 86 71 fe cf 2c 3e
      2e 0e 76 7c 16 8e 1c cc f7 90 f0 c3 f6 c5 50 6a
     dd 1f 25 f8 77 7d f5 40 a3 51 35 72 58 35 c4 81
                                                               prime1 = p
     00 eb 48 76 69 f8 02 72 98 2d 3e c4 cc e1 f9 26
      33 9a 5c 68 bc b8 f1 a2 f2 2e d7 c6 c6 73 e6 21
      f1
   02 21
                                                               prime2 = q
      00 e7 73 aa ea e7 0e 5c 3b 7d 4d 8e 68 16 94 1a
     37 f2 e4 cd 4c 52 d5 39 a3 c7 be c1 58 3a b8 36
     19
```

2.3 Encode PrivateKeyInfo value

Test User 1 encodes a value of type PrivateKeyInfo according to PKCS #8 to represent the private key in an algorithm-independent way.

In this example, the private key is identified by PKCS #1's rsaEncryption, which has the object identifier value {1 2 840 113549 1 1 1}. There are no attributes in the private-key information.

The BER-encoded PrivateKeyInfo value is the following 343-octet string:

2.4 Encrypt PrivateKeyInfo encoding

Test User 1 encrypts the PrivateKeyInfo encoding with a password according to PKCS #5.

In this example, the selected password-based encryption algorithm is "MD2 with DES-CBC." There are three steps to this algorithm: a DES key and initializing vector are derived from the password with MD2, given a salt value and an iteration count; the PrivateKeyInfo encoding is padded to a multiple of eight bytes; and the padded PrivateKeyInfo encoding is encrypted under DES.

The message M is the PrivateKeyInfo encoding.

The password *P* is the ASCII string "password":

```
P = 70 61 73 73 77 6f 72 64
```

The salt value S (which happens to be derived deterministically from the MD2 message digest of the octet string $P \parallel M$) is:

```
S = 76 58 e9 08 dd a0 d8 2b
```

The iteration count c is 1000.

The result of 1000 iterations of MD2 on the octet string $P \parallel S$ is the following 16-octet string:

```
77 78 8a b2 b1 8f 8c d1 11 b9 fd 82 2e dc 6a 24
```

The DES key *K* (with odd parity) and the initializing vector *IV* derived from the message digest are:

```
K = 76 79 8a b3 b0 8f 8c d0
```

```
IV = 11 b9 fd 82 2e dc 6a 24
```

The padding string *PS* for the message *M* is

```
PS = 01
```

since the length of the message M is 343 octets, which is one less than a multiple of eight.

The ciphertext C resulting from encrypting the octet string $M \parallel PS$ under DES with key K and initializing vector IV is the following 344-octet string:

```
        fb
        00
        82
        09
        02
        23
        18
        1c
        47
        d5
        25
        ce
        6b
        1b
        1c
        cf
        05
        33
        34
        0f
        af
        09
        da
        24

        12
        04
        a6
        f9
        41
        af
        c6
        cd
        d1
        2a
        22
        a5
        95
        e7
        ca
        29
        95
        ed
        c5
        7e
        c3
        34
        d2

        61
        98
        e6
        b9
        33
        2c
        65
        31
        f8
        9a
        0e
        3a
        41
        2c
        f9
        8f
        b7
        75
        7e
        f8
        f8
        97
        8d
        6c

        00
        e9
        f3
        9a
        6b
        5e
        9a
        7d
        21
        8e
        46
        1c
        a6
        f6
        44
        4f
        a2
        ad
        f8
        ac
        90
        ba
        c6
        ga
        ac
        ac
        ba
        ac</td
```

2.5 Encode EncryptedPrivateKeyInfo value

Test User 1 encodes a value of type EncryptedPrivateKeyInfo according to PKCS #8 to represent the encrypted PrivateKeyInfo value in an algorithm-independent way.

In this example, the encryption algorithm is identified by PKCS #5's md2WithDES-CBC, which has the object identifier value {1 2 840 113549 1 5 1}.

The BER-encoded EncryptedPrivateKeyInfo value is:

Test User 1 can now store this encoding and transfer it from one computer system to another. The private key is obtained by reversing steps 3, 4 and 5.

3. Obtaining a certificate and an extended certificate

The process of obtaining a certificate and an extended certificate breaks down into six steps:

- Encode a value of type CertificateInfo according to X.509 (as revised by RFC 1114) to represent certificate information.
- Sign the CertificateInfo encoding.
- Encode a value of type Certificate according to X.509 from the CertificateInfo value and its signature.
- Encode a value of type ExtendedCertificateInfo according to PKCS #6 to represent the X.509 certificate and extended attributes.
- Sign the ExtendedCertificateInfo encoding.
- Encode a value of type ExtendedCertificate according to PKCS #6 from the ExtendedCertificateInfo value and its signature.

3.1 Encode CertificateInfo value

Test User 1 (or the "NOTARY" unit) encodes a value of type CertificateInfo according to X.509 (as revised by RFC 1114) to represent certificate information. (The CertificateInfo value, as explained in PKCS #6, is the "to-be-signed" component of the Certificate value.)

In this example, the certificate information is as follows:

- The version is 0.
- The serial number is 1.
- The signature algorithm is NIST/OIW's md2WithRSA, which has the object identifier value {1 3 14 7 2 3 1}.
- The issuer's distinguished name is the organizational unit "NOTARY" within the organizational unit "Beta 1" within the organization "RSA Data Security, Inc." within the country "US."
- The subject's distinguished name is the common name "Test User 1" within the organization "RSA Data Security, Inc." within the country "US."
- The validity period is from 8:00 a.m. GMT, October 1, 1990 to 7:59:59 a.m. GMT, October 1, 1992.
- the public key algorithm is X.509's rsa, which has the object identifier value {2 5 8 1 1}. The public key could be identified by PKCS #1's rsaEncryption instead, but X.509's rsa is chosen since it is presently better known and therefore gives the resulting certificate a wider audience. The public key is Test User 1's public key.

The BER-encoded CertificateInfo value is:

```
30 82 01 24
                                                          certificateInfo
   02 01 01
                                                          serialNumber = 1
   30 0a
                                                                 signature
      06 06 2b 0e 07 02 03 01
                                                   algorithm = md2WithRSA
      05 00
                                                        parameters = NULL
   30 51
                                                                issuerName
      31 0b
         30 09
            06 03 55 04 06
                                              attributeType = countryName
            13 02 55 53
                                                     attributeValue = "US"
      31 20
         30 1e
```

```
06 03 55 04 0a
                                     attributeType = organizationName
         13 17
                                 attributeValue = "RSA Data Security, Inc."
            52 53 41 20 44 61 74 61 20 53 65 63 75 72 69 74
            79 2c 20 49 6e 63 2e
  31 Of
     30 0d
         06 03 55 04 0b attributeType = organizationalUnitName
         13 06 42 65 74 61 20 31 attributeValue = "Beta 1"
  31 Of
     30 0d
         06 03 55 04 0b
                             attributeType = organizationalUnitName
         13 06 4e 4f 54 41 52 59 attributeValue = "NOTARY"
30 le
                                                             validity
                                     start = October 1, 1990, 8:00:00am GMT
  17 0d
     39 30 31 30 30 31 30 38 30 30 30 30 5a
                                       end = October 1, 1992, 7:59:59am GMT
      39 32 31 30 30 31 30 37 35 39 35 39 5a
30 45
                                                          subjectName
  31 0b
     30 09
         06 03 55 04 06
                                         attributeType = countryName
         13 02 55 53
                                                 attributeValue = "US"
  31 20
     30 1e
                           attributeType - Class
attributeValue = "RSA Data Security, Inc."
         06 03 55 04 0a
         13 17
            52 53 41 20 44 61 74 61 20 53 65 63 75 72 69 74
            79 2c 20 49 6e 63 2e
  31 14
     30 12
         06 03 55 04 03 attributeType = organizationalUnitName
                                           attributeValue = "Test User 1"
         13 0b
            54 65 73 74 20 55 73 65 72 20 31
30 59
                                                 subjectPublicKeyInfo
  30 0a
                                                             algorithm
                                                      algorithm = rsa
      06 04 55 08 01 01
                                                 parameters = 512 (bits)
      02 02 02 00
                               subjectPublicKey = RSAPublicKey encoding
   03 4b
      00 30 48 02 41 ... 03 01 00 01
```

3.2 Sign CertificateInfo encoding

The "NOTARY" unit signs the CertificateInfo encoding with its private key.

In this example, the selected signature algorithm is NIST/OIW's md2WithRSA. There are three steps to this algorithm: An MD2 message digest is computed on the CertificateInfo encoding, then the message digest is encoded as an OCTET STRING value, then the OCTET STRING encoding is encrypted under RSA with the "NOTARY" unit's private key.

The message digest is:

```
6b d9 7b f9 4f 23 5a f9 a6 2d fb 56 60 fa 06 9a
```

The OCTET STRING encoding is:

```
04 10 6b d9 7b f9 4f 23 5a f9 a6 2d fb 56 60 fa 06 9a
```

The resulting signature is:

```
05 9f 66 02 57 b2 e9 2f 10 3b 70 57 06 5a 81 ea 07 cd 0a 84 98 60 da c4 10 16 65 19 2b 76 19 52 da a2 09 b6 fd 0e 36 82 65 63 61 ef 5a 11 f3 55 1d 04 95 c0 25 b3 c3 93 1c 73 4b c8 69 d7 f4 36 e7 9d 45 1a eb 5b 96 98 03 46 11 7e cc e1 dc c2 0a 17 7b 03 f0 4a 20 fe
```

We do not give the private key for the "NOTARY" unit here, but the signature can be verified with the "NOTARY" unit's public key. The BER-encoded RSAPublicKey value for that public key is:

3.3 Encode Certificate value

The "NOTARY" unit encodes a value of type Certificate according to X.509 from the CertificateInfo value and its signature.

The BER-encoded Certificate value is:

```
30 82 01 8f
30 82 01 24 ... 03 01 00 01 certificateInfo
30 0a signatureAlgorithm
```

```
06 06 2b 0e 07 02 03 01 algorithm = md2WithRSA 05 00 parameters = NULL 03 59 signature 00 05 9f 66 02 ... f0 4a 20 fe
```

3.4 Encode ExtendedCertificateInfo value

Test User 1 (or the "NOTARY" unit) encodes a value of type ExtendedCertificateInfo to represent the X.509 certificate and extended attributes.

In this example, there is one extended attribute, PKCS #9's emailAddress, which has the object identifier value {1 2 840 113549 1 9 1}. The attribute has one value, "test1@rsa.com."

The BER-encoded ExtendedCertificateInfo value is:

```
30 82 01 b6
   02 01 00
                                                              version = 0
   30 82 01 8f ... f0 4a 20 fe
                                                              certificate
   31 1e
                                                               attributes
     30 1c
         06 09
                                            attributeType = emailAddress
            2a 86 48 86 f7 0d 01 09 01
         31 Of
                                                          attributeValues
            16 0d
                                                        = "test1@rsa.com"
               74 65 73 74 31 40 72 73 61 2e 63 6f 6d
```

3.5 Sign ExtendedCertificateInfo encoding

The "NOTARY" unit signs the ExtendedCertificateInfo encoding with its private key.

As in Section 3.2, the selected signature algorithm is NIST/OIW's md2WithRSA. There are three steps to this algorithm: An MD2 message digest is computed on the ExtendedCertificateInfo encoding, then the message digest is encoded as an OCTET STRING value, then the OCTET STRING encoding is encrypted under RSA with the "NOTARY" unit's private key.

The message digest is:

```
93 le 05 la b6 f6 e2 ed eb c7 cb e7 19 0c f8 35
```

The OCTET STRING encoding is:

```
04 10 93 1e 05 1a b6 f6 e2 ed eb c7 cb e7 19 0c f8 35
```

The resulting signature is:

```
01 bf eb a8 a7 28 49 21 cc 67 c8 fa 1f 9b db 2a 39 4e 37 94 78 1e 48 7f 59 3c a0 70 54 38 c5 e5 0b 49 e2 b6 71 3b 9f 8e 80 18 56 05 82 23 72 32 5e 0f be 0e c6 b4 ce 89 70 06 ee 15 7b 56 49 f7 f2 e3 54 15 de 00 fa 77 59 d2 a6 d7 a6 a4 ee 96 c6 df 9c 46 f6 6d ed a4
```

The signature can be verified with the "NOTARY" unit's public key, which is given in Section 3.2.

3.6 Encode ExtendedCertificate value

The "NOTARY" unit encodes a value of type ExtendedCertificate according to PKCS #6 from the ExtendedCertificateInfo value and its signature.

The BER-encoded ExtendedCertificate value is:

```
30 82 02 21
30 82 01 b6 ... 2e 43 4f 4d extendedCertificateInfo
30 0a signatureAlgorithm
06 06 2b 0e 07 02 03 01 algorithm = md2WithRSA
05 00 parameters = NULL
03 59 signature
00 01 bf eb a8 ... f6 6d ed a4
```

4. Preparing a digitally signed message

The process of preparing a digitally signed message breaks down into five steps:

- Encode a value of type ContentInfo according to PKCS #7 for the data to be signed.
- Digest the data to be signed according to PKCS #7.
- Encrypt the message digest with a private key according to PKCS #1 and PKCS #7.
- Encode a value of type SignedData according to PKCS #7 from the first ContentInfo value, the encrypted message digest, and other information.

• Encode a value of type ContentInfo according to PKCS #7 from the SignedData value.

4.1 Encode inner ContentInfo value

Test User 1 encodes a value of type ContentInfo according to PKCS #7 for the data to be signed.

In this example, the content type is PKCS #7's data, which has the object identifier value {1 2 840 113549 1 7 1}. The content is an OCTET STRING value containing the ASCII string "Everyone gets Friday off."

The BER-encoded inner ContentInfo value is:

4.2 Digest data

Test User 1 digests the data to be signed according to PKCS #7.

The selected message-digest algorithm is NIST/OIW's md2.

The input to the message-digest algorithm is the original ASCII string:

```
45 76 65 72 79 6f 6e 65 20 67 65 74 73 20 46 72 69 64 61 79 20 6f 66 66 2e
```

The resulting message digest is:

```
1d 32 de 00 9f 9c 56 ea 46 36 d3 9a af fd ae a1
```

4.3 Encrypt message digest

Test User 1 encrypts the message digest with his (or her) private key according to PKCS #1 and PKCS #7.

The message digest is first prefixed with the object identifier encoding for NIST/OIW's md2, for PEM compatibility. NIST/OIW's md2 has the object identifier value {1 3 14 7 2 2 1}.

The input to the message-digest encryption algorithm is:

```
06 06 2b 0e 07 02 02 01 1d 32 de 00 9f 9c 56 ea 46 36 d3 9a af fd ae a1
```

In this example, the selected message-digest encryption algorithm is PKCS #1's rsaEncryption, which has two general steps: An encryption block is constructed from a block type, a padding string, and the prefixed message digest; then the encryption block is exponentiated with Test User 1's private exponent.

The encryption block *EB* is the following 64-octet string:

The resulting encrypted message digest is the following 64-octet string:

```
3d b4 38 4a 67 f6 85 8f 6f 1f 33 36 77 9c 11 40 b1 f8 81 7c 76 17 ac c1 e1 fe 20 4f 32 e9 ad 59 5f b3 7a 7a 20 0b 79 e3 17 bb dc ef 93 83 9e c2 03 d8 1b 74 7a ea c0 43 5d f1 b7 80 a3 8b d5 83
```

4.4 Encode SignedData value

Test User 1 encodes a value of type SignedData according to PKCS #7 from the inner ContentInfo value, the encrypted message digest, and other information. The other information includes:

- the issuer's distinguished name (the "NOTARY" unit—see Section 3.1) and the serial number of Test User 1's certificate
- a message-digest algorithm identifier (NIST/OIW's md2, which has the object identifier value {1 3 14 7 2 2 1})

• a message-digest encryption algorithm identifier (PKCS #1's rsaEncryption, which has the object identifier value {1 2 840 113549 1 1 1})

The BER-encoded SignedData value is:

```
30 81 ed
  02 01 00
                                                             version = 0
  31 08
                                                        digestAlgorithms
     06 06 2b 0e 07 02 02 01
                                                                   = md2
  30 28 06 09 ... 6f 66 66 2e
                                       content = inner ContentInfo
  31 81 b3
                                                             signerInfos
     30 81 b0
        02 01 00
                                                             version = 0
        30 56
                                                  issuerAndSerialNumber
           30 51 31 0b ... 54 41 52 59
                                                             issuerName
           02 01 01
                                                            serialNumber
        06 06 2b 0e 07 02 02 01
                                                  digestAlgorithm = md2
                              digestEncryptionAlgorithm = rsaEncryption
           2a 86 48 86 f7 0d 01 01 01
                                                        encryptedDigest
           3d b4 38 4a 67 f6 85 8f 6f 1f 33 36 77 9c 11 40
           b1 f8 81 7c 76 17 ac c1 e1 fe 20 4f 32 e9 ad 59
           5f b3 7a 7a 20 0b 79 e3 17 bb dc ef 93 83 9e c2
           03 d8 1b 74 7a ea c0 43 5d f1 b7 80 a3 8b d5 83
```

4.5 Encode ContentInfo value

Test User 1 encodes a value of type ContentInfo according to PKCS #7 from the SignedData value.

The content type is PKCS #7's signedData, which has the object identifier value {1 2 840 113549 1 7 2}.

The BER-encoded ContentInfo value is: