

---

# Some Examples of the PKCS Standards

An RSA Laboratories Technical Note

Burton S. Kaliski Jr.

Revised November 1, 1993\*

*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 according to PKCS #1 and protects the private key with a password according to PKCS #5 and #8.
2. The user prepares a PKCS #10 certification request and obtains a certificate.
3. The user prepares a digitally signed message according to PKCS #7.

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:

1. Generating an RSA key pair according to PKCS #1.
2. Encoding values of type `RSAPublicKey` and `RSAPrivateKey` according to PKCS #1 to represent the key pair in an algorithm-specific way.

---

\*Supersedes June 3, 1991 version, which was also published as NIST/OSI Implementors' Workshop document SEC-SIG-91-25. PKCS documents are available by electronic mail to <pkcs@rsa.com>.

Copyright © 1991–1993 RSA Laboratories, a division of RSA Data Security, Inc. License to copy this document is granted provided that it is identified as "RSA Data Security, Inc. Public-Key Cryptography Standards (PKCS)" in all material mentioning or referencing this document.

003-903017-110-000-000

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

## 2.1 Generating 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 508-bit integer:

```
n = 0a 66 79 1d c6 98 81 68 de 7a b7 74 19 bb 7f b0 c0 01 c6 27 10 27 00 75 14 29 42 e1 9a 8d 8c 51 d0
53 b3 e3 78 2a 1d e5 dc 5a f4 eb e9 94 68 17 01 14 a1 df e6 7c dc 9a 9a f5 5d 65 56 20 bb ab
```

The prime factors  $p$  and  $q$  of the modulus are:

```
p = 33 d4 84 45 c8 59 e5 23 40 de 70 4b cd da 06 5f bb 40 58 d7 40 bd 1d 67 d2 9e 9c 14 6c 11 cf 61
q = 33 5e 84 08 86 6b 0f d3 8d c7 00 2d 3f 97 2c 67 38 9a 65 d5 d8 30 65 66 d5 c4 f2 a5 aa 52 62 8b
```

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 = 01 23 c5 b6 1b a3 6e db 1d 36 79 90 41 99 a8 9e a8 0c 09 b9 12 2e 14 00 c0 9a dc f7 78 46 76 d0 1d
23 35 6a 7d 44 d6 bd 8b d5 0e 94 bf c7 23 fa 87 d8 86 2b 75 17 76 91 c1 1d 75 76 92 df 88 81
```

```
d mod p-1 = 04 5e c9 00 71 52 53 25 d3 d4 6d b7 96 95 e9 af ac c4 52 39 64 36 0e 02 b1 19 ba a3 66 31
62 41
```

```
d mod q-1 = 15 eb 32 73 60 c7 b6 0d 12 e5 e2 d1 6b dc d9 79 81 d1 7f ba 6b 70 db 13 b2 0b 43 6e 24 ea
da 59
```

```
q-1 mod p = 2c a6 36 6d 72 78 1d fa 24 d3 4a 9a 24 cb c2 ae 92 7a 99 58 af 42 65 63 ff 63 fb 11 65 8a 46
1d
```

## 2.2 Encoding 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:

```
30 47
```

```
02 40 modulus = n
```

```
0a 66 79 1d c6 98 81 68 de 7a b7 74 19 bb 7f b0
c0 01 c6 27 10 27 00 75 14 29 42 e1 9a 8d 8c 51
d0 53 b3 e3 78 2a 1d e5 dc 5a f4 eb e9 94 68 17
01 14 a1 df e6 7c dc 9a 9a f5 5d 65 56 20 bb ab
```

```
02 03 01 00 01 publicExponent = e
```

The RSAPublicKey value is later used in a certificate.

The BER-encoded RSAPrivateKey value is:

```

30 82 01 36
02 01 00      version = 0
02 40 modulus = n
0a 66 79 1d c6 98 81 68 de 7a b7 74 19 bb 7f b0
c0 01 c6 27 10 27 00 75 14 29 42 e1 9a 8d 8c 51
d0 53 b3 e3 78 2a 1d e5 dc 5a f4 eb e9 94 68 17
01 14 a1 df e6 7c dc 9a 9a f5 5d 65 56 20 bb ab
02 03 01 00 01 publicExponent = e
02 40 privateExponent = d
01 23 c5 b6 1b a3 6e db 1d 36 79 90 41 99 a8 9e
a8 0c 09 b9 12 2e 14 00 c0 9a dc f7 78 46 76 d0
1d 23 35 6a 7d 44 d6 bd 8b d5 0e 94 bf c7 23 fa
87 d8 86 2b 75 17 76 91 c1 1d 75 76 92 df 88 81
02 20 prime1 = p
33 d4 84 45 c8 59 e5 23 40 de 70 4b cd da 06 5f
bb 40 58 d7 40 bd 1d 67 d2 9e 9c 14 6c 11 cf 61
02 20 prime2 = q
33 5e 84 08 86 6b 0f d3 8d c7 00 2d 3f 97 2c 67
38 9a 65 d5 d8 30 65 66 d5 c4 f2 a5 aa 52 62 8b
02 20 exponent1 = d mod p-1
04 5e c9 00 71 52 53 25 d3 d4 6d b7 96 95 e9 af
ac c4 52 39 64 36 0e 02 b1 19 ba a3 66 31 62 41
02 20 exponent2 = d mod q-1
15 eb 32 73 60 c7 b6 0d 12 e5 e2 d1 6b dc d9 79
81 d1 7f ba 6b 70 db 13 b2 0b 43 6e 24 ea da 59
02 20 coefficient = q-1 mod p
2c a6 36 6d 72 78 1d fa 24 d3 4a 9a 24 cb c2 ae
92 7a 99 58 af 42 65 63 ff 63 fb 11 65 8a 46 1d

```

### 2.3 Encoding a 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 340-octet string:

```

30 82 01 50
02 01 00      version = 0
30 0d privateKeyAlgorithmIdentifier
06 09      algorithm = rsaEncryption
2a 86 48 86 f7 0d 01 01 01
05 00      parameters = NULL
04 82 01 3a privateKey = RSAPrivateKey encoding
30 82 01 36 ... 65 8a 46 1d

```

## 2.4 Encrypting the 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:

$S = 53\ 7c\ 94\ 2e\ 8a\ 96\ 04\ 4b$

The iteration count  $c$  is 1.

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

$13\ 1a\ 55\ 51\ fe\ 1f\ d2\ a4\ 3a\ d9\ 95\ 74\ 66\ 6b\ 67\ ce$

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

$K = 13\ 1a\ 54\ 51\ fe\ 1f\ d3\ a4$

$IV = 3a\ d9\ 95\ 74\ 66\ 6b\ 67\ ce$

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

$PS = 04\ 04\ 04\ 04$

since the length of the message  $M$  is 340 octets, which is four 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:

$20\ d4\ dd\ 6a\ 50\ 5f\ 0d\ ea\ e3\ da\ a6\ 98\ 22\ a0\ 10\ 0e\ 70\ ef\ e2\ 8f\ 4b\ 07\ ff\ ee\ 77\ b2\ 34\ 3f\ c7\ ee\ 61\ 25\ 84\ b3\ 7e\ 13\ c3\ d8\ fd\ ad\ 83\ 94\ 0c\ a3\ 5b\ 70\ 67\ 2d\ 48\ 9c\ 10\ 23\ 57\ 31\ 77\ b1\ 48\ 2a\ c2\ 65\ 40\ ce\ 10\ 33\ 40\ 87\ cf\ f8\ 7b\ 2a\ 05\ 0e\ 3f\ 3a\ 9e\ c7\ 4e\ a1\ 08\ 7f\ 02\ 9e\ a9\ 06\ 7a\ a5\ 9a\ 7e\ 64\ cd\ 03\ 1a\ 49\ 6b\ 47\ b0\ 64\ 6d\ 04\ 65\ 8b\ 31\ d7\ 3a\ 12\ 58\ 24\ 80\ da\ 44\ 73\ 0a\ c4\ 0f\ af\ 4a\ 00\ 8e\ 8f\ d3\ 5b\ 22\ 1e\ 84\ 1c\ 54\ 20\ 37\ 50\ b3\ c2\ 94\ 74\ 60\ 64\ 51\ 65\ a1\ 41\ ca\ a7\ 34\ 68\ a1\ c1\ e3\ 59\ be\ 9b\ 42\ 54\ 14\ 06\ ae\ 17\ b4\ f4\ f3\ 75\ 9f\ 6d\ 29\ 96\ ef\ 3e\ 5c\ aa\ 6d\ 61\ 4d\ d8\ 5d\ d3\ b5\ 7d\ fd\ c4\ 54\ c8\ 63\ 0e\ a1\ 22\ 90\ 28\ a9\ 11\ a6\ e6\ dd\ 41\ 93\ 75\ 76\ f1\ b3\ e5\ 6a\ 0f\ 85\ 7b\ 19\ 95\ a2\ 94\ 9b\ 25\ 3c\ e2\ fe\ 27\ aa\ d6\ 1e\ f2\ d7\ bb\ 00\ cb\ 62\ fa\ b7\ 87\ c9\ bd\ 6a\ fa\ 5c\ ce\ 22\ b7\ 2b\ 6c\ 8c\ 29\ 4b\ e3\ f2\ 2b\ be\ fa\ 44\ 42\ dc\ 31\ 11\ 0a\ f2\ 6d\ ad\ 82\ 9c\ c3\ 2a\ 15\ ca\ 1f\ 00\ c3\ 93\ e8\ 1a\ fc\ 4b\ 5d\ 99\ 75\ 77\ f4\ f7\ fd\ 17\ 65\ 9e\ 6e\ 7f\ a0\ 66\ 05\ b0\ 28\ b3\ ef\ c0\ 65\ 4e\ bb\ ea\ 34\ 78\ 36\ cf\ d3\ ae\ 38\ dd\ 79\ 45\ f7\ f0\ b8\ 99\ cb\ 71\ 27\ 64\ c5\ c7\ d3\ 61\ 9d\ fb\ 6e\ ba\ 4c\ e6\ a4\ 22\ dd\ 11\ 8d\ e8\ 88\ 63\ 77\ 4a\ 4a\ 8f\ 88\ 40\ b5\ 1d\ 01\ 12\ e5\ ea\ fe\ 71\ b6\ b3\ 7e\ 71\ c8\ cf$

### 2.5 Encoding the 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:

```

30 82 01 78
 30 1a encryptionAlgorithm
 06 09      algorithm = pbeWithMD2AndDES-CBC
 2a 86 48 86 f7 0d 01 05 01
 30 0d      parameter
 04 08 53 7c 94 2e 8a 96 04 4b      salt value
 02 01 01      iteration count = 1
 04 82 01 58      encryptedData
 20 d4 dd 6a ... 7e 71 c8 cf

```

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

The process of obtaining a certificate breaks down into three steps:

1. Encoding a value of type CertificationRequestInfo according to PKCS #10 from Test User 1's name and public key.
2. Signing the CertificationRequestInfo encoding.
3. Encoding a value of type CertificationRequest according to PKCS #10 from the CertificationRequestInfo value and the signature.

The certification authority's steps are not considered, although the resulting certificate is described.

### 3.1 Encoding a CertificationRequestInfo value

Test User 1 encodes a value of type CertificationRequestInfo according to PKCS #10 from its name and public key. In this example, the name is the common name "Test User 1" within the organization "Example Organization" within the country "US." The public key is identified by PKCS #1's rsaEncryption. There are no attributes in the certification request information.

The BER-encoded CertificationRequestInfo value is:

```

30 81 a4
  02 01 00      version = 0
  30 42 subject
    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
          30 5b subjectPublicKeyInfo
            30 0d      algorithm
              06 09      algorithm = rsaEncryption
              2a 86 48 86 f7 0d 01 01 01
              05 00      parameters = NULL
            03 4a      subjectPublicKey = RSAPublicKey encoding
            00 30 47 02 40 ... 03 01 00 01

```

### 3.2 Signing the CertificationRequestInfo encoding

Test User 1 signs the CertificationRequestInfo encoding with its private key.

In this example, the selected signature algorithm is PKCS #1's md2WithRSAEncryption. There are three steps to this algorithm: An MD2 message digest is computed on the CertificateInfo encoding, then the message digest is encoded as a DigestInfo value, then the DigestInfo encoding is encrypted under RSA with Test User 1's private key. (Section 4.4 gives a more detailed example of RSA private-key encryption.)

The message digest is:

```
dc a9 ec f1 c1 5c 1b d2 66 af f9 c8 79 93 65 cd
```

In this example, the message-digest algorithm in the DigestInfo value is identified by RFC 1321's md2, which has the object identifier value {1 2 840 113549 2 2}.

The BER-encoded DigestInfo value is:

```

30 20
  30 0c digestAlgorithm
    06 08 2a 86 48 86 f7 0d 02 02  algorithm = md2
    05 00      parameters = NULL
  04 10 digest
    dc a9 ec f1 ... 79 93 65 cd

```

RSA encryption according to PKCS #1 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:

```
00
01      block type
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff      padding string
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
00      00 separator
30 20 30 0c ... 79 93 65 cd      data D
```

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

```
06 db 36 cb 18 d3 47 5b 9c 01 db 3c 78 95 28 08
02 79 bb ae ff 2b 7d 55 8e d6 61 59 87 c8 51 86
3f 8a 6c 2c ff bc 89 c3 f7 5a 18 d9 6b 12 7c 71
7d 54 d0 d8 04 8d a8 a0 54 46 26 d1 7a 2a 8f be
```

### 3.3 Encoding a CertificationRequest value

Test User 1 encodes a value of type *CertificationRequest* according to PKCS #10 from the *CertificationRequestInfo* value and its signature.

The BER-encoded *CertificationRequest* value is:

```
30 81 f9
 30 81 a4 02 ... 03 01 00 01      certificationRequestInfo
 30 0d signatureAlgorithm
   06 09      algorithm = md2WithRSAEncryption
   2a 86 48 86 f7 0d 01 01 02
   05 00      parameters = NULL
 03 41 signature
   00 06 db 36 cb ... 7a 2a 8f be
```

Test User 1 can now send the encoding to a certification authority to obtain a certificate.

### 3.4 The certificate

In this example, the certificate is as follows:

- The version is 0. (Since this is the default, the field is not included in the encoding.)
- The serial number is 14000029 (hexadecimal).
- The signature algorithm is PKCS #1's *md5WithRSAEncryption*, which has the object identifier value {1 2 840 113549 1 1 5}.
- The issuer's distinguished name is the organization "Example Organization" within the country "US."
- The name is Test User 1's name from the certification request.
- The validity period is from 10:18:06pm GMT, September 9, 1992 to 10:18:05pm GMT, September 9, 1994.
- The public key is Test User 1's public key from the certification request.

The BER-encoded Certificate value is:

```

30 82 01 5a
  30 82 01 04  certificateInfo
    02 04 14 00 00 29  serialNumber = 14000029
    30 0d  signature
      06 09  algorithm = md2WithRSAEncryption
        2a 86 48 86 f7 0d 01 01 02
      05 00  parameters = NULL
    30 2c  issuer
      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
            30 1e  validity
              17 0d  start = September 9, 1992, 10:18:06pm GMT
                39 32 30 39 30 39 32 32 31 38 30 36 5a
              17 0d  end = September 9, 1994, 10:18:05pm GMT
                39 34 30 39 30 39 32 32 31 38 30 35 5a
            30 42 31 0b ... 65 72 20 31  subject
            30 5b 30 0d ... 03 01 00 01  subjectPublicKeyInfo
    30 0d  signatureAlgorithm
      06 09  algorithm = md2WithRSAEncryption
        2a 86 48 86 f7 0d 01 01 02
      05 00  parameters = NULL
    03 41  signature
      00 45 1a a1 e1 aa 77 20 4a 5f cd f5 76 06 9d
      02 f7 32 c2 6f 36 7b 0d 57 8a 6e 64 f3 9a 91
      1f 47 95 df 09 94 34 05 11 a0 d1 df 4a 20 b2
      6a 77 4c ca ef 75 fc 69 2e 54 c2 a1 93 7c 07
      11 26 9d 9b 16

```

The BER-encoded RSAPublicKey value for Example Organization's key is:

```

30 48
  02 41  modulus
    00 c5 59 65 91 f6 70 ac 38 7f ba bf f6 c0 d0 83
    e5 93 22 0e 0b c3 a0 0f 97 5b 9e b7 ad 51 4c 77
    7b ae 14 25 2d f8 30 58 74 7f 17 7e ac 9c 1d a5
    39 4d 33 eb d8 de 92 16 a4 1c 69 d2 d4 83 8f a4
    51
  02 03 01 00 01  publicExponent

```

The certificate signature can be verified with Example Organization's public key. Example Organization's private key is stored in secure hardware at RSA Data Security.

## 4. Preparing a digitally signed message

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

1. Encoding a value of type ContentInfo according to PKCS #7 for the data to be signed.
2. Digesting the data to be signed according to PKCS #7.
3. Encoding a value of type DigestInfo according to PKCS #7 from the message digest.
4. Encrypting the DigestInfo encoding with a private key according to PKCS #1.
5. Encoding a value of type SignedData according to PKCS #7 from the first ContentInfo value, the encrypted message digest, and other information.
6. Encoding a value of type ContentInfo according to PKCS #7 from the SignedData value.

### 4.1 Encoding the 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 encoding does not specify ASCII; such an interpretation is left to the application.)

The BER-encoded inner ContentInfo value is:

```
30 28
 06 09 contentType = data
 2a 86 48 86 f7 0d 01 07 01
a0 1b [0] EXPLICIT
 04 19 content = OCTET STRING value: "Everyone gets Friday off."
 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
```

### 4.2 Digesting the data

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

The selected message-digest algorithm is MD2, as described in RFC 1321.

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 Encoding a DigestInfo value

Test User 1 encodes a value of type DigestInfo according to PKCS #7 from the message digest.

The BER-encoded DigestInfo value is:

```
30 20
 30 0c digestAlgorithm
   06 08 2a 86 48 86 f7 0d 02 02 algorithm = md2
   05 00 parameters = NULL
 04 10 digest
   1d 32 de 00 9f 9c 56 ea 46 36 d3 9a af fd ae a1
```

### 4.4 Encrypting the DigestInfo encoding

Test User 1 encrypts the DigestInfo encoding with his (or her) private key according to PKCS #1. Section 3.2 describes the RSA encryption process in detail. The resulting encrypted message digest is the following 64-octet string:

```
05 fa 6a 81 2f c7 df 8b f4 f2 54 25 09 e0 3e 84
6e 11 b9 c6 20 be 20 09 ef b4 40 ef bc c6 69 21
69 94 ac 04 f3 41 b5 7d 05 20 2d 42 8f b2 a2 7b
5c 77 df d9 b1 5b fc 3d 55 93 53 50 34 10 c1 e1
```

### 4.5 Encoding a 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 and the serial number of Test User 1's certificate (see Section 3.4);
- Test User 1's certificate;
- a message-digest algorithm identifier (RFC 1321's md2); and
- a message-digest encryption algorithm identifier (PKCS #1's rsaEncryption).

The BER-encoded SignedData value is:

```
30 82 02 3d
 02 01 01
 31 0e
  30 0c
    06 08 2a 86 48 86 f7 0d 02 02
    05 00
 30 28 06 09 ... 6f 66 66 2e
a0 82 01 5e
  30 82 01 5a ... 26 9d 9b 16
31 81 9b
 30 81 98
  02 01 01
```

```

30 34
  30 2c 31 0b ... 74 69 6f 6e
  02 04 14 00 00 29
30 0c
  06 08 2a 86 48 86 f7 0d 02 02
  05 00
30 0d
  06 09
  2a 86 48 86 f7 0d 01 01 01
  05 00
04 40
  05 fa 6a 81 2f c7 df 8b f4 f2 54 25 09 e0 3e 84
  6e 11 b9 c6 20 be 20 09 ef b4 40 ef bc c6 69 21
  69 94 ac 04 f3 41 b5 7d 05 20 2d 42 8f b2 a2 7b
  5c 77 df d9 b1 5b fc 3d 55 93 53 50 34 10 c1 e1

```

#### 4.6 Encoding a 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:

```

30 82 02 50
  06 09 2a 86 48 86 f7 0d 01 07 02
  a0 82 02 41
  30 82 02 3d ... 34 10 c1 e1

```

(The full BER encoding is reproduced in Appendix A for reference.)

## References

- PKCS #1        RSA Laboratories. *PKCS #1: RSA Encryption Standard*. Version 1.5, November 1993.
- PKCS #5        RSA Laboratories. *PKCS #5: Password-Based Encryption 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 #10       RSA Laboratories. *PKCS #10: Certification Request Syntax Standard*. Version 1.0, November 1993.

## Appendix A: Example signed-data encoding

The full BER encoding for the example signed-data value in Section 4.6 is as follows:

```
30 82 02 50 06 09 2a 86 48 86 f7 0d 01 07 02 a0
82 02 41 30 82 02 3d 02 01 01 31 0e 30 0c 06 08
2a 86 48 86 f7 0d 02 02 05 00 30 28 06 09 2a 86
48 86 f7 0d 01 07 01 a0 1b 04 19 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 a0 82 01 5e 30 82 01 5a 30 82 01 04
02 04 14 00 00 29 30 0d 06 09 2a 86 48 86 f7 0d
01 01 02 05 00 30 2c 31 0b 30 09 06 03 55 04 06
13 02 55 53 31 1d 30 1b 06 03 55 04 0a 13 14 45
78 61 6d 70 6c 65 20 4f 72 67 61 6e 69 7a 61 74
69 6f 6e 30 1e 17 0d 39 32 30 39 30 39 32 32 31
38 30 36 5a 17 0d 39 34 30 39 30 39 32 32 31 38
30 35 5a 30 42 31 0b 30 09 06 03 55 04 06 13 02
55 53 31 1d 30 1b 06 03 55 04 0a 13 14 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 13 0b 54 65 73 74
20 55 73 65 72 20 31 30 5b 30 0d 06 09 2a 86 48
86 f7 0d 01 01 01 05 00 03 4a 00 30 47 02 40 0a
66 79 1d c6 98 81 68 de 7a b7 74 19 bb 7f b0 c0
01 c6 27 10 27 00 75 14 29 42 e1 9a 8d 8c 51 d0
53 b3 e3 78 2a 1d e5 dc 5a f4 eb e9 94 68 17 01
14 a1 df e6 7c dc 9a 9a f5 5d 65 56 20 bb ab 02
03 01 00 01 30 0d 06 09 2a 86 48 86 f7 0d 01 01
02 05 00 03 41 00 45 1a a1 e1 aa 77 20 4a 5f cd
f5 76 06 9d 02 f7 32 c2 6f 36 7b 0d 57 8a 6e 64
f3 9a 91 1f 47 95 df 09 94 34 05 11 a0 d1 df 4a
20 b2 6a 77 4c ca ef 75 fc 69 2e 54 c2 a1 93 7c
07 11 26 9d 9b 16 31 81 9b 30 81 98 02 01 01 30
34 30 2c 31 0b 30 09 06 03 55 04 06 13 02 55 53
31 1d 30 1b 06 03 55 04 0a 13 14 45 78 61 6d 70
6c 65 20 4f 72 67 61 6e 69 7a 61 74 69 6f 6e 02
04 14 00 00 29 30 0c 06 08 2a 86 48 86 f7 0d 02
02 05 00 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01
05 00 04 40 05 fa 6a 81 2f c7 df 8b f4 f2 54 25
09 e0 3e 84 6e 11 b9 c6 20 be 20 09 ef b4 40 ef
bc c6 69 21 69 94 ac 04 f3 41 b5 7d 05 20 2d 42
8f b2 a2 7b 5c 77 df d9 b1 5b fc 3d 55 93 53 50
34 10 c1 e1
```

## Revision history

### June 3, 1991 version

The June 3, 1991 version is part of the initial public release of PKCS. It was published as NIST/OSI Implementors' Workshop document SEC-SIG-91-25.

### November 1, 1993 version

The November 1, 1993 version incorporates several editorial changes, including the addition of a references section, Appendix A: Example signed-data encoding, and a revision history. It is updated to be consistent with the following versions of the PKCS documents:

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

*PKCS #5: Password-Based Encryption 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 #10: Certification Request Syntax Standard. Version 1.0, November 1993.*

The following substantive changes were made:

General: Names and keys are changed, consistent with external PKCS examples.

Section 3: Certification request examples are added. Extended certificate examples are removed.

## 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