PKCS #1: RSA Encryption Standard

An RSA Laboratories Technical Note
Version 1.5
Revised November 1, 1993[*]

1. Scope

This standard describes a method for encrypting data using the RSA public-key cryptosystem. Its intended use is in the construction of digital signatures and digital envelopes, as described in PKCS #7:

- For digital signatures, the content to be signed is first reduced to a message digest with a message-digest algorithm (such as MD5), and then an octet string containing the message digest is encrypted with the RSA private key of the signer of the content. The content and the encrypted message digest are represented together according to the syntax in PKCS #7 to yield a digital signature. This application is compatible with Privacy-Enhanced Mail (PEM) methods.

- For digital envelopes, the content to be enveloped is first encrypted under a content-encryption key with a content-encryption algorithm (such as DES), and then the content-encryption key is encrypted with the RSA public keys of the recipients of the content. The encrypted content and the encrypted content-encryption key are represented together according to the syntax in PKCS #7 to yield a digital envelope. This application is also compatible with PEM methods.

The standard also describes a syntax for RSA public keys and private keys. The public-key syntax would be used in certificates; the private-key syntax would be used typically in PKCS #8 private-key information. The public-key syntax is

[*]Supersedes June 3, 1991 version, which was also published as NIST/OSI Implementors' Workshop document SEC-SIG-91-18. PKCS documents are available by electronic mail to <pkcs@rsa.com>.

identical to that in both X.509 and Privacy-Enhanced Mail. Thus X.509/PEM RSA keys can be used in this standard.

The standard also defines three signature algorithms for use in signing X.509/PEM certificates and certificate-revocation lists, PKCS #6 extended certificates, and other objects employing digital signatures such as X.401 message tokens.

Details on message-digest and content-encryption algorithms are outside the scope of this standard, as are details on sources of the pseudorandom bits required by certain methods in this standard.

2. References

FIPS PUB 46–1	National Bureau of Standards. *FIPS PUB 46–1: Data Encryption Standard.* January 1988.
PKCS #6	RSA Laboratories. *PKCS #6: Extended-Certificate Syntax 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.
RFC 1319	B. Kaliski. *RFC 1319: The MD2 Message-Digest Algorithm.* April 1992.
RFC 1320	R. Rivest. *RFC 1320: The MD4 Message-Digest Algorithm.* April 1992.
RFC 1321	R. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm.* April 1992.
RFC 1423	D. Balenson. *RFC 1423: Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers.* February 1993.
X.208	CCITT. *Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1).* 1988.
X.209	CCITT. *Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).* 1988.
X.411	CCITT. *Recommendation X.411: Message Handling Systems: Message Transfer System: Abstract Service Definition and Procedures.* 1988.
X.509	CCITT. *Recommendation X.509: The Directory—Authentication Framework.* 1988.
[dBB92]	B. den Boer and A. Bosselaers. An attack on the last two rounds of MD4. In J. Feigenbaum, editor, *Advances in Cryptology—CRYPTO '91 Proceedings,* volume

576 of *Lecture Notes in Computer Science,* pages 194–203. Springer-Verlag, New York, 1992.

[dBB93] B. den Boer and A. Bosselaers. Collisions for the compression function of MD5. Presented at EUROCRYPT '93 (Lofthus, Norway, May 24–27, 1993).

[DO86] Y. Desmedt and A.M. Odlyzko. A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes. In H.C. Williams, editor, *Advances in Cryptology—CRYPTO '85 Proceedings* , volume 218 of *Lecture Notes in Computer Science,* pages 516–521. Springer-Verlag, New York, 1986.

[Has88] Johan Hastad. Solving simultaneous modular equations. *SIAM Journal on Computing,* 17(2):336–341, April 1988.

[IM90] Colin I'Anson and Chris Mitchell. Security defects in CCITT Recommendation X.509—The directory authentication framework. *Computer Communications Review,* :30–34, April 1990.

[Mer90] R.C. Merkle. Note on MD4. Unpublished manuscript, 1990.

[Mil76] G.L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and Systems Sciences,* 13(3):300–307, 1976.

[QC82] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters,* 18(21):905–907, October 1982.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM,* 21(2):120–126, February 1978.

3. Definitions

For the purposes of this standard, the following definitions apply.

`AlgorithmIdentifier`: A type that identifies an algorithm (by object identifier) and associated parameters. This type is defined in X.509.

ASN.1: Abstract Syntax Notation One, as defined in X.208.

BER: Basic Encoding Rules, as defined in X.209.

DES: Data Encryption Standard, as defined in FIPS PUB 46-1.

MD2: RSA Data Security, Inc.'s MD2 message-digest algorithm, as defined in RFC 1319.

MD4: RSA Data Security, Inc.'s MD4 message-digest algorithm, as defined in RFC 1320.

MD5: RSA Data Security, Inc.'s MD5 message-digest algorithm, as defined in RFC 1321.

modulus: Integer constructed as the product of two primes.

PEM: Internet Privacy-Enhanced Mail, as defined in RFC 1423 and related documents.

RSA: The RSA public-key cryptosystem, as defined in [RSA78].

private key: Modulus and private exponent.

public key: Modulus and public exponent.

4. Symbols and abbreviations

Upper-case italic symbols (e.g., *BT*) denote octet strings and bit strings (in the case of the signature *S*); lower-case italic symbols (e.g., *c*) denote integers.

ab	hexadecimal octet value	c	exponent
BT	block type	d	private exponent
D	data	e	public exponent
EB	encryption block	k	length of modulus in octets
ED	encrypted data	n	modulus
M	message	p, q	prime factors of modulus
MD	message digest	x	integer encryption block
MD'	comparative message digest	y	integer encrypted data
PS	padding string	mod n	modulo n
S	signature	$X \parallel Y$	concatenation of X, Y
$\|X\|$	length in octets of X		

5. General overview

The next six sections specify key generation, key syntax, the encryption process, the decryption process, signature algorithms, and object identifiers.

Each entity shall generate a pair of keys: a public key and a private key. The encryption process shall be performed with one of the keys and the decryption process shall be performed with the other key. Thus the encryption process can be either a public-key operation or a private-key operation, and so can the decryption process. Both processes transform an octet string to another octet string. The processes are inverses of each other if one process uses an entity's public key and the other process uses the same entity's private key.

The encryption and decryption processes can implement either the classic RSA transformations, or variations with padding.

6. Key generation

This section describes RSA key generation.

Each entity shall select a positive integer e as its public exponent.

Each entity shall privately and randomly select two distinct odd primes p and q such that $(p-1)$ and e have no common divisors, and $(q-1)$ and e have no common divisors.

The public modulus n shall be the product of the private prime factors p and q:

$$n = pq \ .$$

The private exponent shall be a positive integer d such that $de-1$ is divisible by both $p-1$ and $q-1$.

The length of the modulus n in octets is the integer k satisfying

$$2^{8(k-1)} \leq n < 2^{8k} \ .$$

The length k of the modulus must be at least 12 octets to accommodate the block formats in this standard (see Section 8).

Notes.

1. The public exponent may be standardized in specific applications. The values 3 and F_4 (65537) may have some practical advantages, as noted in X.509 Annex C.

2. Some additional conditions on the choice of primes may well be taken into account in order to deter factorization of the modulus. These security conditions fall outside the scope of this standard. The lower bound on the length k is to accommodate the block formats, not for security.

7. Key syntax

This section gives the syntax for RSA public and private keys.

7.1 Public-key syntax

An RSA public key shall have ASN.1 type `RSAPublicKey`:

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

(This type is specified in X.509 and is retained here for compatibility.)

The fields of type `RSAPublicKey` have the following meanings:

- `modulus` is the modulus *n*.
- `publicExponent` is the public exponent *e*.

7.2 Private-key syntax

An RSA private key shall have ASN.1 type `RSAPrivateKey`:

```
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 }

Version ::= INTEGER
```

The fields of type `RSAPrivateKey` have the following meanings:

- `version` is the version number, for compatibility with future revisions of this standard. It shall be 0 for this version of the standard.
- `modulus` is the modulus *n*.
- `publicExponent` is the public exponent *e*.

- `privateExponent` is the private exponent d.
- `prime1` is the prime factor p of n.
- `prime2` is the prime factor q of n.
- `exponent1` is d mod $(p–1)$.
- `exponent2` is d mod $(q–1)$.
- `coefficient` is the Chinese Remainder Theorem coefficient q^{-1} mod p.

Notes.

1. An RSA private key logically consists of only the modulus n and the private exponent d. The presence of the values p, q, d mod $(p–1)$, d mod $(p–1)$, and q^{-1} mod p is intended for efficiency, as Quisquater and Couvreur have shown [QC82]. A private-key syntax that does not include all the extra values can be converted readily to the syntax defined here, provided the public key is known, according to a result by Miller [Mil76].
2. The presence of the public exponent e is intended to make it straightforward to derive a public key from the private key.

8. Encryption process

This section describes the RSA encryption process.

The encryption process consists of four steps: encryption-block formatting, octet-string-to-integer conversion, RSA computation, and integer-to-octet-string conversion. The input to the encryption process shall be an octet string D, the data; an integer n, the modulus; and an integer c, the exponent. For a public-key operation, the integer c shall be an entity's public exponent e; for a private-key operation, it shall be an entity's private exponent d. The output from the encryption process shall be an octet string ED, the encrypted data.

The length of the data D shall not be more than $k–11$ octets, which is positive since the length k of the modulus is at least 12 octets. This limitation guarantees that the length of the padding string PS is at least eight octets, which is a security condition.

Notes.

1. In typical applications of this standard to encrypt content-encryption keys and message digests, one would have $\|D\| \leq 30$. Thus the length of the RSA modulus will need to be at least 328 bits (41 octets), which is reasonable and consistent with security recommendations.

2. The encryption process does not provide an explicit integrity check to facilitate error detection should the encrypted data be corrupted in transmission. However, the structure of the encryption block guarantees that the probability that corruption is undetected is less than 2^{-16}, which is an upper bound on the probability that a random encryption block looks like block type `02`.

3. Application of private-key operations as defined here to data other than an octet string containing a message digest is not recommended and is subject to further study.

4. This standard may be extended to handle data of length more than k-11 octets.

8.1 Encryption-block formatting

A block type *BT*, a padding string *PS*, and the data *D* shall be formatted into an octet string *EB*, the encryption block.

$$EB = \texttt{00} \parallel BT \parallel PS \parallel \texttt{00} \parallel D \,. \qquad (1)$$

The block type *BT* shall be a single octet indicating the structure of the encryption block. For this version of the standard it shall have value `00`, `01`, or `02`. For a private-key operation, the block type shall be `00` or `01`. For a public-key operation, it shall be `02`.

The padding string *PS* shall consist of k–3–$\|D\|$ octets. For block type `00`, the octets shall have value `00`; for block type `01`, they shall have value `FF`; and for block type `02`, they shall be pseudorandomly generated and nonzero. This makes the length of the encryption block *EB* equal to k.

Notes.

1. The leading `00` octet ensures that the encryption block, converted to an integer, is less than the modulus.

2. For block type `00`, the data *D* must begin with a nonzero octet or have known length so that the encryption block can be parsed unambiguously. For block types `01` and `02`, the encryption block can be parsed unambiguously since the padding string *PS* contains no octets with value `00` and the padding string is separated from the data *D* by an octet with value `00`.

3. Block type `01` is recommended for private-key operations. Block type `01` has the property that the encryption block, converted to an integer, is guaranteed to be large, which prevents certain attacks of the kind proposed by Desmedt and Odlyzko [DO86].

4. Block types `01` and `02` are compatible with PEM RSA encryption of content-encryption keys and message digests as described in RFC 1423.

5. For block type `02`, it is recommended that the pseudorandom octets be generated independently for each encryption process, especially if the same data is input to more than one encryption process. Hastad's results [Has88] motivate this recommendation.

6. For block type `02`, the padding string is at least eight octets long, which is a security condition for public-key operations that prevents an attacker from recoving data by trying all possible encryption blocks. For simplicity, the minimum length is the same for block type `01`.

7. This standard may be extended in the future to include other block types.

8.2 Octet-string-to-integer conversion

The encryption block *EB* shall be converted to an integer *x*, the integer encryption block. Let $EB_1, \ldots, EB_k$ be the octets of *EB* from first to last. Then the integer *x* shall satisfy

$$x = \sum_{i=1}^{k} 2^{8(k-i)} EB_i \,. \tag{2}$$

In other words, the first octet of *EB* has the most significance in the integer and the last octet of *EB* has the least significance.

Note. The integer encryption block *x* satisfies $0 \le x < n$ since $EB_1 = \texttt{00}$ and $2^{8(k-1)} \le n$.

8.3 RSA computation

The integer encryption block x shall be raised to the power c modulo n to give an integer y, the integer encrypted data.

$$y = x^c \bmod n, \;\; 0 \le y < n \, .$$

This is the classic RSA computation.

8.4 Integer-to-octet-string conversion

The integer encrypted data y shall be converted to an octet string ED of length k, the encrypted data. The encrypted data ED shall satisfy

$$y = \sum_{i=1}^{k} 2^{8(k-i)} ED_i \, . \qquad (3)$$

where ED_1, …, ED_k are the octets of ED from first to last.

In other words, the first octet of ED has the most significance in the integer and the last octet of ED has the least significance.

9. Decryption process

This section describes the RSA decryption process.

The decryption process consists of four steps: octet-string-to-integer conversion, RSA computation, integer-to-octet-string conversion, and encryption-block parsing. The input to the decryption process shall be an octet string ED, the encrypted data; an integer n, the modulus; and an integer c, the exponent. For a public-key operation, the integer c shall be an entity's public exponent e; for a private-key operation, it shall be an entity's private exponent d. The output from the decryption process shall be an octet string D, the data.

It is an error if the length of the encrypted data ED is not k.

For brevity, the decryption process is described in terms of the encryption process.

9.1 Octet-string-to-integer conversion

The encrypted data *ED* shall be converted to an integer y, the integer encrypted data, according to Equation (3).

It is an error if the integer encrypted data y does not satisfy $0 \leq y < n$.

9.2 RSA computation

The integer encrypted data y shall be raised to the power c modulo n to give an integer x, the integer encryption block.

$$x = y^c \bmod n, \ \ 0 \leq x < n \ .$$

This is the classic RSA computation.

9.3 Integer-to-octet-string conversion

The integer encryption block x shall be converted to an octet string *EB* of length k, the encryption block, according to Equation (2).

9.4 Encryption-block parsing

The encryption block *EB* shall be parsed into a block type *BT*, a padding string *PS*, and the data *D* according to Equation (1).

It is an error if any of the following conditions occurs:

- The encryption block *EB* cannot be parsed unambiguously (see notes to Section 8.1).
- The padding string *PS* consists of fewer than eight octets, or is inconsistent with the block type *BT*.
- The decryption process is a public-key operation and the block type *BT* is not `00` or `01`, or the decryption process is a private-key operation and the block type is not `02`.

10. Signature algorithms

This section defines three signature algorithms based on the RSA encryption process described in Sections 8 and 9. The intended use of the signature algorithms is in signing X.509/PEM certificates and certificate-revocation lists, PKCS #6 extended certificates, and other objects employing digital signatures such as X.401 message tokens. The algorithms are not intended for use in constructing digital signatures in PKCS #7. The first signature algorithm (informally, "MD2 with RSA") combines the MD2 message-digest algorithm with RSA, the second (informally, "MD4 with RSA") combines the MD4 message-digest algorithm with RSA, and the third (informally, "MD5 with RSA") combines the MD5 message-digest algorithm with RSA.

This section describes the signature process and the verification process for the two algorithms. The "selected" message-digest algorithm shall be either MD2 or MD5, depending on the signature algorithm. The signature process shall be performed with an entity's private key and the verification process shall be performed with an entity's public key. The signature process transforms an octet string (the message) to a bit string (the signature); the verification process determines whether a bit string (the signature) is the signature of an octet string (the message).

Note. The only difference between the signature algorithms defined here and one of the the methods by which signatures (encrypted message digests) are constructed in PKCS #7 is that signatures here are represented here as bit strings, for consistency with the X.509 `SIGNED` macro. In PKCS #7 encrypted message digests are octet strings.

10.1 Signature process

The signature process consists of four steps: message digesting, data encoding, RSA encryption, and octet-string-to-bit-string conversion. The input to the signature process shall be an octet string M, the message; and a signer's private key. The output from the signature process shall be a bit string S, the signature.

10.1.1 Message digesting

The message M shall be digested with the selected message-digest algorithm to give an octet string MD, the message digest.

10.1.2 Data encoding

The message digest MD and a message-digest algorithm identifier shall be combined into an ASN.1 value of type `DigestInfo`, described below, which shall be BER-encoded to give an octet string D, the data.

```
DigestInfo ::= SEQUENCE {
  digestAlgorithm DigestAlgorithmIdentifier,
  digest Digest }

DigestAlgorithmIdentifier ::= AlgorithmIdentifier

Digest ::= OCTET STRING
```

The fields of type `DigestInfo` have the following meanings:

- `digestAlgorithm` identifies the message-digest algorithm (and any associated parameters). For this application, it should identify the selected message-digest algorithm, MD2, MD4 or MD5. For reference, the relevant object identifiers are the following:

```
md2 OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) US(840) rsadsi(113549)
      digestAlgorithm(2) 2 }
md4 OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) US(840) rsadsi(113549)
      digestAlgorithm(2) 4 }
md5 OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) US(840) rsadsi(113549)
      digestAlgorithm(2) 5 }
```

  For these object identifiers, the `parameters` field of the `digestAlgorithm` value should be `NULL`.

- `digest` is the result of the message-digesting process, i.e., the message digest *MD*.

Notes.

1. A message-digest algorithm identifier is included in the `DigestInfo` value to limit the damage resulting from the compromise of one message-digest algorithm. For instance, suppose an adversary were able to find messages with a given MD2 message digest. That adversary might try to forge a signature on a message by finding an innocuous-looking message with the same MD2 message digest, and coercing a signer to sign the innocuous-looking message. This attack would succeed only if the signer used MD2. If the `DigestInfo` value contained only the message digest, however, an adversary could attack signers that use any message digest.

2. Although it may be claimed that the use of a `SEQUENCE` type violates the literal statement in the X.509 `SIGNED` and `SIGNATURE`

macros that a signature is an `ENCRYPTED OCTET STRING` (as opposed to `ENCRYPTED SEQUENCE`), such a literal interpretation need not be required, as I'Anson and Mitchell point out [IM90].

3. No reason is known that MD4 would not be sufficient for very high security digital signature schemes, but because MD4 was designed to be exceptionally fast, it is "at the edge" in terms of risking successful cryptanalytic attack. A message-digest algorithm can be considered "broken" if someone can find a collision: two messages with the same digest. While collisions have been found in variants of MD4 with only two digesting "rounds" [Mer90][dBB92], none have been found in MD4 itself, which has three rounds. After further critical review, it may be appropriate to consider MD4 for very high security applications.

   MD5, which has four rounds and is proportionally slower than MD4, is recommended until the completion of MD4's review. The reported "pseudocollisions" in MD5's internal compression function [dBB93] do not appear to have any practical impact on MD5's security.

   MD2, the slowest of the three, has the most conservative design. No attacks on MD2 have been published.

10.1.3 RSA encryption

The data *D* shall be encrypted with the signer's RSA private key as described in Section 7 to give an octet string *ED*, the encrypted data. The block type shall be `01`. (See Section 8.1.)

10.1.4 Octet-string-to-bit-string conversion

The encrypted data *ED* shall be converted into a bit string *S*, the signature. Specifically, the most significant bit of the first octet of the encrypted data shall become the first bit of the signature, and so on through the least significant bit of the last octet of the encrypted data, which shall become the last bit of the signature.

Note. The length in bits of the signature *S* is a multiple of eight.

10.2 Verification process

The verification process for both signature algorithms consists of four steps: bit-string-to-octet-string conversion, RSA decryption, data decoding, and message digesting and comparison. The input to the verification process shall be an octet string *M*, the message; a signer's public key; and a bit string *S*, the signature. The output from the verification process shall be an indication of success or failure.

10.2.1 Bit-string-to-octet-string conversion

The signature *S* shall be converted into an octet string *ED*, the encrypted data. Specifically, assuming that the length in bits of the signature *S* is a multiple of eight, the first bit of the signature shall become the most significant bit of the first octet of the encrypted data, and so on through the last bit of the signature, which shall become the least significant bit of the last octet of the encrypted data.

It is an error if the length in bits of the signature *S* is not a multiple of eight.

10.2.2 RSA decryption

The encrypted data *ED* shall be decrypted with the signer's RSA public key as described in Section 8 to give an octet string *D*, the data.

It is an error if the block type recovered in the decryption process is not `01`. (See Section 9.4.)

10.2.3 Data decoding

The data *D* shall be BER-decoded to give an ASN.1 value of type `DigestInfo`, which shall be separated into a message digest *MD* and a message-digest algorithm identifier. The message-digest algorithm identifier shall determine the "selected" message-digest algorithm for the next step.

It is an error if the message-digest algorithm identifier does not identify the MD2, MD4 or MD5 message-digest algorithm.

10.2.4 Message digesting and comparison

The message *M* shall be digested with the selected message-digest algorithm to give an octet string *MD'*, the comparative message digest. The verification process shall succeed if the comparative message digest *MD'* is the same as the message digest *MD*, and the verification process shall fail otherwise.

11. Object identifiers

This standard defines five object identifiers: `pkcs-1`, `rsaEncryption`, `md2WithRSAEncryption`, `md4WithRSAEncryption`, and `md5WithRSAEncryption`.

The object identifier `pkcs-1` identifies this standard.

```
pkcs-1 OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) US(840) rsadsi(113549)
      pkcs(1) 1 }
```

The object identifier `rsaEncryption` identifies RSA public and private keys as defined in Section 7 and the RSA encryption and decryption processes defined in Sections 8 and 9.

```
rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }
```

The `rsaEncryption` object identifier is intended to be used in the `algorithm` field of a value of type `AlgorithmIdentifier`. The `parameters` field of that type, which has the algorithm-specific syntax `ANY DEFINED BY algorithm`, would have ASN.1 type `NULL` for this algorithm.

The object identifiers `md2WithRSAEncryption`, `md4WithRSAEncryption`, `md5WithRSAEncryption`, identify, respectively, the "MD2 with RSA," "MD4 with RSA," and "MD5 with RSA" signature and verification processes defined in Section 10.

```
md2WithRSAEncryption OBJECT IDENTIFIER ::= { pkcs-1 2 }
md4WithRSAEncryption OBJECT IDENTIFIER ::= { pkcs-1 3 }
md5WithRSAEncryption OBJECT IDENTIFIER ::= { pkcs-1 4 }
```

These object identifiers are intended to be used in the `algorithm` field of a value of type `AlgorithmIdentifier`. The `parameters` field of that type, which has the algorithm-specific syntax `ANY DEFINED BY algorithm`, would have ASN.1 type `NULL` for these algorithms.

Note. X.509's object identifier `rsa` also identifies RSA public keys as defined in Section 7, but does not identify private keys, and identifies different encryption and decryption processes. It is expected that some applications will identify public keys by `rsa`. Such public keys are compatible with this standard; an `rsaEncryption` process under an `rsa` public key is the same as the `rsaEncryption` process under an `rsaEncryption` public key .

Revision history

Versions 1.0–1.3

Versions 1.0–1.3 were distributed to participants in RSA Data Security, Inc.'s Public-Key Cryptography Standards meetings in February and March 1991.

Version 1.4

Version 1.4 is part of the June 3, 1991 initial public release of PKCS. Version 1.4 was published as NIST/OSI Implementors' Workshop document SEC-SIG-91-18.

Version 1.5

Version 1.5 incorporates several editorial changes, including updates to the references and the addition of a revision history. The following substantive changes were made:

- Section 10: "MD4 with RSA" signature and verification processes are added.
- Section 11: `md4WithRSAEncryption` object identifier is added.

Author's address

RSA Laboratories	(415) 595-7703
100 Marine Parkway	(415) 595-4126 (fax)
Redwood City, CA 94065 USA	`pkcs-editor@rsa.com`