PKCS #5 v2.0: Password-Based Cryptography Standard

RSA Laboratories

THIRD DRAFT— February 2, 1999

Editor's note: This is the third public draft of PKCS #5 v2.0, incorporating discussion from the October 1998 PKCS Workshop and additional comments on the draft published in December 1998. Barring any major technical comments, a final document will be published next month. Accordingly, please send comments and questions, both editorial and technical, to Burt Kaliski (burt@rsa.com), on or before **Friday, March 5, 1999**.

Table of Contents

1. Introduction

This document provides recommendations for the implementation of password-based cryptography, covering the following aspects:

- key derivation functions
- encryption schemes
- message-authentication schemes
- ASN.1 syntax identifying the techniques

The recommendations are intended for general application within computer and communications systems, and as such include a fair amount of flexibility. They are particularly intended for the protection of sensitive information such as private keys, as in PKCS #8 [21]. It is expected that application standards based on these specifications may include additional constraints.

Other cryptographic techniques based on passwords, such as password-based key entity authentication and key establishment protocols [4][5][22] are outside the scope of this document. Guidelines for the selection of passwords are also outside the scope.

This document supersedes PKCS #5 version 1.5 [20], but includes compatible techniques.

2. Notation

C	ciphertext, an octet string
c	iteration count, a positive integer
DK	derived key, an octet string
dkLen	length in octets of derived key, an integer
EM	encoded message, an octet string
Hash	underlying hash function
hLen	output length of hash function, an integer
l	length in blocks of derived key, an integer
IV	initialization vector, an octet string
K	encryption key, an octet string
KDF	key derivation function
M	message, an octet string
P	password, an octet string
PRF	underlying pseudorandom function
PS	padding string, an octet string
psLen	length in octets of padding string, an integer
S	salt, an octet string
T	message authentication code, an octet string
$T_1,\ldots,T_c$, T_l	intermediate values, octet strings
`01`, `02`, …, `08`	octets with value 1, 2, …, 8

\xor	bit-wise exclusive-or of two octet strings
‖ . ‖	octet length operator
‖	concatenation operator
$<i..j>$	substring extraction operator: extracts octets i through j, $0 \le i \le j$

3. Overview

In many applications of public-key cryptography, user security is ultimately dependent on one or more secret text values or *passwords*. Since a password is not directly applicable as a key to any conventional cryptosystem, however, some processing of the password is required to perform cryptographic operations with it. Moreover, as passwords are often chosen from a relatively small space, special care is required in that processing to defend against search attacks.

A general approach to password-based cryptography, as described by Morris and Thompson [8] for the protection of password tables, is to combine a password with a *salt* to produce a key. The salt can be viewed as an index into a large set of keys derived from the password, and need not be kept secret. Although it may be possible for an opponent to construct a table of possible passwords (a so-called “dictionary attack”), constructing a table of possible keys in advance will be difficult, since there will be many possible keys for each password. An opponent will thus be limited to searching through passwords separately for each salt.

Another approach to password-based cryptography is to construct key derivation techniques that are relatively expensive, thereby increasing the cost of exhaustive search. One way to do this is to include an *iteration count* in the key derivation technique, indicating how many times to iterate some underlying function by which keys are derived. A modest number of iterations, say 1000, is not likely to be a burden for legitimate parties when computing a key, but will be a significant burden for opponents.

Salt and iteration count formed the basis for password-based encryption in PKCS #5 v1.5, and adopted here as well for the various cryptographic operations. Thus, password-based key derivation as defined here is a function of password, a salt, and an iteration count, where the latter two quantities need not be kept secret.

From a password-based key derivation function, it is straightforward to define password-based encryption and message authentication schemes. As in PKCS #5 v1.5, the password-based encryption schemes here are based on an underlying, conventional encryption scheme, where the key for the conventional scheme is derived from the password. Similarly, the password-based message authentication scheme is based on an underlying conventional scheme. This two-layered approach makes the password-based techniques modular in terms of the underlying techniques they can be based on.

It is also expected that the password-based key derivation functions may find other applications than just the encryption and message authentication schemes defined here. For instance, one might derive a set of keys with a single application of a key derivation function, rather than derive each key with a separate application of the function. The keys in the set would be obtained as substrings of the output of the key derivation function. This approach might be employed as part of key establishment in a session-oriented protocol. Another application is password checking, where the output of the key derivation function is stored (along with the salt and iteration count) for the purposes of subsequent verification of a password.

Throughout this document, a password is considered to be an octet string of arbitrary length whose interpretation as a text string is unspecified. In the interest of interoperability, however, it is recommended that applications follow some common text encoding rules. ASCII and UTF-8 [23] are two possibilities. (ASCII is a subset of UTF-8.)

Although the selection of passwords is outside the scope of this document, guidelines have been published [11] that may well be taken into account.

4. Salt and iteration count

Inasmuch as salt and iteration count are central to the techniques defined in this document, some further discussion is warranted.

4.1 Salt

A salt in password-based cryptography has traditionally served the purpose of producing a large set of keys corresponding to a given password, among which one is selected at random according to the salt. An individual key in the set is selected by applying a key derivation function *KDF*, as

$$DK = KDF\ (P, S)$$

where *DK* is the derived key, *P* is the password, and *S* is the salt. This has two benefits:

1. It is difficult for an opponent to precompute all the keys corresponding to a dictionary of passwords, or even the most likely keys. If the salt is 64 bits long, for instance, there will be as many as 2^{64} keys for each password. An opponent is thus limited to searching for passwords after a password-based operation has been performed and the salt is known.

2. It is unlikely that the same key will be selected twice. Again, if the salt is 64 bits long, the chance of “collision” between keys does not become significant until about 2^{32} keys have been produced, according to the Birthday Paradox. This

addresses concerns about interactions between multiple uses of the same key, which may apply for some encryption and authentication techniques.

In password-based encryption, the party encrypting a message can gain assurance that these benefits are realized simply by selecting a large and sufficiently random salt when deriving an encryption key from a password. A party generating a message authentication code can gain such assurance in a similar fashion.

The party decrypting a message or verifying a message authentication code, however, cannot be sure that a salt supplied by another party has actually been generated at random. It is possible, for instance, that the salt may have been copied from another password-based operation, in an attempt to exploit interactions between multiple uses of the same key. For instance, suppose two legitimate parties exchange a encrypted message, where the encryption key is an 80-bit key derived from a shared password with some salt. An opponent could take the salt from that encryption and provide it to one of the parties as though it were for a 40-bit key. If the party reveals the result of decryption with the 40-bit key, the opponent may be able to solve for the 40-bit key. In the case that 40-bit key is the first half of the 80-bit key, the opponent can then readily solve for the remaining 40 bits of the 80-bit key.

To defend against such attacks, either the interaction between multiple uses of the same key should be carefully analyzed, or the salt should contain data that explicitly distinguishes between different operations. For instance, the salt might have an additional, non-random octet that specifies whether the derived key is for encryption, for message authentication, or for some other operation.

Based on this, the following is recommended for salt selection:

1. If there is no concern about interactions between multiple uses of the same key with the password-based encryption and authentication techniques supported for a given password, then the salt may be generated at random. It should be at least eight octets (64 bits) long.

2. Otherwise, the salt should contain data that explicitly distinguishes between different operations, in addition to a random part that is at least eight octets long. For instance, the salt could have an additional non-random octet that specifies the purpose of the derived key. Alternatively, it could be the encoding of a structure that specifies detailed information about the derived key, such as the encryption or authentication technique and a sequence number among the different keys derived from the password. The particular format of the additional data is left to the application.

Note. If a random number generator or pseudorandom generator is not available, a deterministic alternative for generating the salt (or the random part of it) is to apply a password-based key derivation function to the password and the message M to be processed. For instance, the salt could be computed with a key derivation function as $S =$

KDF (*P*, *M*). This approach is not recommended if the message *M* is known to belong to a small message space (e.g., “Yes” or “No”), however, since then there will only be a small number of possible salts.

4.2 Iteration count

An iteration count has traditionally served the purpose of increasing the cost of producing keys from a password, thereby also increasing the difficulty of attack. For the methods in this document, a minimum of 1000 iterations is recommended. This will increase the cost of exhaustive search for passwords significantly, without a noticeable impact in the cost of deriving individual keys.

5. Key derivation functions

A *key derivation function* produces a *derived key* from a *base key* and other parameters. In a *password-based key derivation function*, the base key is a password and the other parameters are a salt value and an iteration count, as outlined in Section 3.

The primary application of the password-based key derivation functions defined here is in the encryption schemes in Section 6 and the message authentication scheme in Section 7. Other applications are certainly possible, hence the independent definition of these functions.

Two functions are specified in this section: PBKDF1 and PBKDF2. PBKDF2 is recommended for new applications; PBKDF1 is included only for compatibility with existing applications, and is not recommended for new applications.

A typical application of the key derivation functions defined here might include the following steps:

1. Select a salt *S* and an iteration count *c*, as outlined in Section 4.

2. Select a length in octets for the derived key, *dkLen*.

3. Apply the key derivation function to the password, the salt, the iteration count and the key length to produce a derived key.

4. Output the derived key.

Any number of keys may be derived from a password by varying the salt, as described in Section 3.

5.1 PBKDF1

PBKDF1 applies a hash function, which shall be MD2 [6], MD5 [15] or SHA-1 [12], to derive keys. The length of the derived key is bounded by the length of the hash function output, which is 16 octets for MD2 and MD5 and 20 octets for SHA-1. PBKDF1 is compatible with the key derivation process in PKCS #5 v1.5.

PBKDF1 is recommended only for compatibility with existing applications since the keys it produces may not be large enough for some applications.

PBKDF1 (P, S, c, $dkLen$)

Options:	*Hash*	underlying hash function
Input:	*P*	password, an octet string
	S	salt, an eight-octet string
	c	iteration count, a positive integer
	dkLen	intended length in octets of derived key, at most 16 for MD2 or MD5 and 20 for SHA-1
Output:	*DK*	derived key, a *dkLen*-octet string

Steps:

1. Apply the underlying hash function *Hash* for c iterations to the concatenation of the password P and the salt S, then extract the first *dkLen* octets to produce a derived key *DK*:

$$\begin{gathered} T_1 = Hash\ (P \parallel S)\ , \\ T_2 = Hash\ (T_1)\ , \\ \ldots \\ T_c = Hash\ (T_{c-1})\ , \\ DK = T_c\langle 0..dkLen-1\rangle\ . \end{gathered}$$

2. Output the derived key *DK*.

5.2 PBKDF2

PBKDF2 applies a pseudorandom function (see Appendix 0 for an example) to derive keys. The length of the derived key is essentially unbounded.

PBKDF2 is recommended for new applications.

PBKDF2 (P, S, c, $dkLen$)

Options:	*PRF*	underlying pseudorandom function (*hLen* denotes the length in octets of the pseudorandom function output)
Input:	*P*	password, an octet string
	S	salt, an octet string
	c	iteration count, a positive integer
	dkLen	intended length in octets of the derived key, at most $(2^{32} - 1) \cdot hLen$
Output:	*DK*	derived key, a *dkLen*-octet string

Steps:

1. If $dkLen > (2^{32} - 1) \cdot hLen$, output "derived key too long" and stop.

2. Let l be the number of *hLen*-octet blocks in the derived key, rounding up, and let r be the number of octets in the last block:

$$l = \lceil dkLen / hLen \rceil \,,$$
$$r = dkLen - (l - 1) \cdot hLen \,.$$

3. For each block of the derived key apply the function F defined below to the password P, the salt S, the iteration count c, and the block index to compute the block:

$$T_1 = F\,(P, S, c, 1)\,,$$
$$T_2 = F\,(P, S, c, 2)\,,$$
$$\ldots$$
$$T_l = F\,(P, S, c, l)\,,$$

where the function F is defined as the exclusive-or sum of the first c iterates of the underlying pseudorandom function *PRF* applied to the password P and the concatenation of the salt S and the block index i:

$$F\,(P, S, c, i) = T_1 \text{ \xor } T_2 \text{ \xor } \cdots \text{ \xor } T_c$$

where

$$T_1 = PRF\,(P, S \,\|\, \textsc{Int}\,(i))\,,$$
$$T_2 = PRF\,(P, T_1)\,,$$
$$\ldots$$
$$T_c = PRF\,(P, T_{c-1})\,.$$

Here, Int (i) is a four-octet encoding of the integer i, most significant octet first.

4. Concatenate the blocks and extract the first *dkLen* octets to produce a derived key *DK*:

$$DK = T_1 \parallel T_2 \parallel \cdots \parallel T_l<0..r-1> .$$

5. Output the derived key *DK*.

Note. The construction of the function *F* follows a "belt-and-suspenders" approach. The iterates T_i are computed recursively to remove a degree of parallelism from an opponent; they are exclusive-ored together to reduce concerns about the recursion degenerating into a small set of values.

6. Encryption schemes

An *encryption scheme*, in the symmetric setting, consists of an *encryption operation* and a *decryption operation*, where the encryption operation produces a ciphertext from a message under a key, and the decryption operation recovers the message from the ciphertext under the same key. In a *password-based encryption scheme*, the key is a password.

A typical application of a password-based encryption scheme is a private-key protection method, where the message contains private-key information, as in PKCS #8. The encryption schemes defined here would be suitable encryption algorithms in that context.

Two schemes are specified in this section: PBES1 and PBES2. PBES2 is recommended for new applications; PBES1 is included only for compatibility with existing applications, and is not recommended for new applications.

6.1 PBES1

PBES1 combines the PBKDF1 function (Section 5.1) with an underlying block cipher, which shall be either DES [9] or RC2^{TM} [17] in CBC mode [10]. PBES1 is compatible with the encryption scheme in PKCS #5 v1.5.

PBES1 is recommended only for compatibility with existing applications, since it supports only two underlying encryption schemes, each of which has a key size (56 or 64 bits) that may not be large enough for some applications.

6.1.1 Encryption operation

The encryption operation for PBES1 consists of the following steps, which encrypt a message *M* under a password *P* to produce a ciphertext *C*:

1. Select an eight-octet salt *S* and an iteration count *c*, as outlined in Section 4.

2. Apply the PBKDF1 key derivation function (Section 5.1) to the password P, the salt S, and the iteration count c to produce a derived key DK of length 16 octets:

$$DK = \mathrm{PBKDF1}\ (P, S, c, 16)\ .$$

3. Separate the derived key DK into an encryption key K consisting of the first eight octets of DK and an initialization vector IV consisting of the next eight octets:

$$K = DK\langle 0..7\rangle\ ,$$
$$IV\ = DK\langle 8..15\rangle\ .$$

4. Concatenate M and a padding string PS to form an encoded message EM:

$$EM = M \parallel PS\ ,$$

where the padding string PS consists of 8-($\|M\|$ mod 8) octets each with value 8-($\|M\|$ mod 8). The padding string PS will satisfy one of the following statements:

$$PS = \texttt{01} \text{ — if } \|M\| \bmod 8 = 7\ ;$$
$$PS = \texttt{02 02} \text{ — if } \|M\| \bmod 8 = 6\ ;$$
$$\ldots$$
$$PS = \texttt{08 08 08 08 08 08 08 08} \text{ — if } \|M\| \bmod 8 = 0.$$

The length in octets of the encoded message will be a multiple of eight and it will be possible to recover the message M unambiguously from the encoded message. (This padding rule is taken from RFC 1423 [3].)

5. Encrypt the encoded message EM with the underlying block cipher (DES or RC2) in cipher block chaining mode under the encryption key K with initialization vector IV to produce the ciphertext C. For DES, the key K shall be considered as a 64-bit encoding of a 56-bit DES key with parity bits ignored (see [9]). For RC2, the “effective key bits” shall be 64 bits.

6. Output the ciphertext C.

The salt S and the iteration count c may be conveyed to the party performing decryption in an `AlgorithmIdentifier` value (see Appendix A.3).

6.1.2 Decryption operation

The decryption operation for PBES1 consists of the following steps, which decrypt a ciphertext C under a password P to recover a message M:

1. Obtain the eight-octet salt S and the iteration count c.

2. Apply the PBKDF1 key derivation function (Section 5.1) to the password *P*, the salt *S*, and the iteration count *c* to produce a derived key *DK* of length 16 octets:

   $$DK = \text{PBKDF1}\ (P, S, c, 16)\ .$$

3. Separate the derived key *DK* into an encryption key *K* consisting of the first eight octets of *DK* and an initialization vector *IV* consisting of the next eight octets:

   $$K = DK\langle 0..7\rangle\ ,$$
   $$IV = DK\langle 8..15\rangle\ .$$

4. Decrypt the ciphertext *C* with the underlying block cipher (DES or RC2) in cipher block chaining mode under the encryption key *K* with initialization vector *IV* to recover an encoded message *EM*. If the length in octets of the ciphertext *C* is not a multiple of eight, output “decryption error” and stop.

5. Separate the encoded message *EM* into a message *M* and a padding string *PS*:

   $$EM = M \parallel PS\ ,$$

   where the padding string *PS* consists of some number *psLen* octets each with value *psLen*, where *psLen* is between 1 and 8. If it is not possible to separate the encoded message *EM* in this manner, output “decryption error” and stop.

6. Output the recovered message *M*.

6.2 PBES2

PBES2 combines a password-based key derivation function, which shall be PBKDF2 (Section 5.2) for this version of PKCS #5, with an underlying encryption scheme (see Appendix B.2 for examples). The key length and any other parameters for the underlying encryption scheme depend on the scheme.

PBES2 is recommended for new applications.

6.2.1 Encryption operation

The encryption operation for PBES2 consists of the following steps, which encrypt a message *M* under a password *P* to produce a ciphertext *C*, applying a selected key derivation function *KDF* and a selected underlying encryption scheme:

1. Select a salt *S* and an iteration count *c*, as outlined in Section 4.

2. Select the length in octets, *dkLen*, for the derived key for the underlying encryption scheme.

3. Apply the selected key derivation function to the password *P*, the salt *S*, and the iteration count *c* to produce a derived key *DK* of length *dkLen* octets:

$$DK = KDF\ (P, S, c, dkLen)\ .$$

4. Encrypt the message *M* with the underlying encryption scheme under the derived key *DK* to produce a ciphertext *C*. (This step may involve selection of parameters such as an initialization vector and padding, depending on the underlying scheme.)

5. Output the ciphertext *C*.

The salt *S*, the iteration count *c*, the key length *dkLen*, and identifiers for the key derivation function and the underlying encryption scheme may be conveyed to the party performing decryption in an `AlgorithmIdentifier` value (see Appendix A.4).

6.2.2 Decryption operation

The decryption operation for PBES2 consists of the following steps, which decrypt a ciphertext *C* under a password *P* to recover a message *M*:

1. Obtain the salt *S* for the operation.

2. Obtain the iteration count *c* for the key derivation function.

3. Obtain the key length in octets, *dkLen*, for the derived key for the underlying encryption scheme.

4. Apply the selected key derivation function to the password *P*, the salt *S*, and the iteration count *c* to produce a derived key *DK* of length *dkLen* octets:

$$DK = KDF\ (P, S, c, dkLen)\ .$$

5. Decrypt the ciphertext *C* with the underlying encryption scheme under the derived key *DK* to recover a message *M*. If the decryption function outputs "decryption error," then output "decryption error" and stop.

6. Output the recovered message *M*.

7. Message authentication schemes

A *message authentication scheme* consists of a *MAC (message authentication code) generation operation* and a *MAC verification operation*, where the MAC generation operation produces a message authentication code from a message under a key, and the MAC verification operation verifies the message authentication code under the same key. In a *password-based message authentication scheme*, the key is a password.

One scheme is specified in this section: PBMAC1.

7.1 PBMAC1

PBMAC1 combines a password-based key derivation function, which shall be PBKDF2 (Section 5.2) for this version of PKCS #5, with an underlying message authentication scheme (see Appendix B.3 for an example). The key length and any other parameters for the underlying message authentication scheme depend on the scheme.

7.1.1 MAC generation

The MAC generation operation for PBMAC2 consists of the following steps, which process a message *M* under a password *P* to generate a message authentication code *T*, applying a selected key derivation function *KDF* and a selected underlying message authentication scheme:

1. Select a salt *S* and an iteration count *c*, as outlined in Section 4.

2. Select a key length in octets, *dkLen*, for the derived key for the underlying message authentication function.

3. Apply the selected key derivation function to the password *P*, the salt *S*, and the iteration count *c* to produce a derived key *DK* of length *dkLen* octets:

   $$DK = KDF\ (P, S, c, dkLen)\ .$$

4. Process the message *M* with the underlying message authentication scheme under the derived key *DK* to generate a message authentication code *T*.

5. Output the message authentication code *T*.

The salt *S*, the iteration count *c*, the key length *dkLen*, and identifiers for the key derivation function and underlying message authentication scheme may be conveyed to the party performing verification in an `AlgorithmIdentifier` value (see Appendix A.5).

7.1.2 MAC verification

The MAC verification operation for PBMAC2 consists of the following steps, which process a message *M* under a password *P* to verify a message authentication code *T*:

1. Obtain the salt *S* and the iteration count *c*.

2. Obtain the key length in octets, *dkLen*, for the derived key for the underlying message authentication scheme.

3. Apply the selected key derivation function to the password P, the salt S, and the iteration count c to produce a derived key DK of length $dkLen$ octets:

$$DK = KDF\ (P, S, c, dkLen)\ .$$

4. Process the message M with the underlying message authentication scheme under the derived key DK to verify the message authentication code T.

5. If the message authentication code verifies, output “correct”; else output “incorrect.”

A. ASN.1 syntax

This section defines ASN.1 syntax for the key derivation functions, the encryption schemes, the message authentication scheme, and supporting techniques. The intended application of these definitions includes PKCS #8 and other syntax for key management, encrypted data, and integrity-protected data.

The object identifier `pkcs-5` identifies the arc of the OID tree from which the PKCS #5-specific OIDs in this section are derived:

```
rsadsi OBJECT IDENTIFIER ::=
  {iso(1) member-body(2) us(840) 113549}
pkcs OBJECT IDENTIFIER ::= {rsadsi 1}
pkcs-5 OBJECT IDENTIFIER ::= {pkcs 5}
```

In the following, all tags (i.e., `[0]` or `[1]`) are explicit tags.

A.1 PBKDF1

No object identifier is given for PBKDF1, as the object identifiers for PBES1 are sufficient for existing applications and PBKDF2 is recommended for new applications.

A.2 PBKDF2

The object identifier `id-PBKDF2` identifies the PBKDF2 key derivation function (Section 5.2).

```
id-PBKDF2 OBJECT IDENTIFIER ::= {pkcs-5 9}
```

The `parameters` field associated with this OID in an `AlgorithmIdentifier` shall have type `PBKDF2-params`:

```
PBKDF2-params ::= SEQUENCE {
  saltSource AlgorithmIdentifier {{PBKDF2-saltSources}},
  iterationCount INTEGER (1..MAX),
  keyLength [0] INTEGER (1..MAX) DEFAULT 16,
  prf [1] AlgorithmIdentifier {{PBKDF2-PRFs}}
    DEFAULT hmacSHA1Identifier }
```

The fields of type `PKDF2-params` have the following meanings:

- `saltSource` specifies the source (and typically the value) of the salt. It shall be an algorithm ID with an OID in the set `PBKDF2-SaltSources`, which for this version of PKCS #5 shall consist of `id-SaltSpecified`, indicating that the salt value is specified explicitly. The `parameters` field for `id-saltSpecified` shall have type `OCTET STRING`, containing the salt value.

```
PBKDF2-saltSources ALGORITHM-IDENTIFIER ::=
  { {OCTET STRING IDENTIFIED BY id-saltSpecified}, ... }
```

```
id-saltSpecified OBJECT IDENTIFIER ::= {pkcs-5 12}
```

- `iterationCount` specifies the iteration count, a positive integer.

- `keyLength` is the length in octets of the derived key. The default is 16 octets.

- `prf` identifies the underlying pseudorandom function. It shall be an algorithm ID with an OID in the set `PBKDF2-PRFs`, which for this version of PKCS #5 shall consist of `id-hmacWithSHA1` (see Appendix B.1.1) and any other OIDs defined by the application.

```
PBKDF2-PRFs ALGORITHM-IDENTIFIER ::=
  { {NULL IDENTIFIED BY id-hmacWithSHA1}, ... }
```

The default pseudorandom function is HMAC-SHA-1:

```
hmacSHA1Identifier AlgorithmIdentifier {{PBDKF2-PRFs}}
  ::= {algorithm id-hmacWithSHA1, parameters NULL : NULL}
```

A.3 PBES1

Different object identifiers identify the PBES1 encryption scheme (Section 6.1) according to the underlying hash function in the key derivation function and the underlying block cipher, as summarized in the following table:

Hash Function	**Block Cipher**	**OID**
MD2	DES	`pkcs-5.1`
MD2	RC2	`pkcs-5.4`
MD5	DES	`pkcs-5.3`
MD5	RC2	`pkcs-5.6`
SHA-1	DES	`pkcs-5.7`
SHA-1	RC2	`pkcs-5.8`

```
pbeWithMD2AndDES-CBC OBJECT IDENTIFIER ::= {pkcs-5 1}
pbeWithMD2AndRC2-CBC OBJECT IDENTIFIER ::= {pkcs-5 4}
pbeWithMD5AndDES-CBC OBJECT IDENTIFIER ::= {pkcs-5 3}
pbeWithMD5AndRC2-CBC OBJECT IDENTIFIER ::= {pkcs-5 6}
pbeWithSHA1AndDES-CBC OBJECT IDENTIFIER ::= {pkcs-5 7}
pbeWithSHA1AndRC2-CBC OBJECT IDENTIFIER ::= {pkcs-5 8}
```

For each OID, the `parameters` field associated with the OID in an `AlgorithmIdentifier` shall have type `PBEParameter`:

```
PBEParameter ::= SEQUENCE {
  salt OCTET STRING SIZE(8),
  iterationCount INTEGER }
```

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

- `salt` specifies the salt value, an eight-octet string.

- `iterationCount` specifies the iteration count.

A.4 PBES2

The object identifier `id-PBES2` identifies the PBES2 encryption scheme (Section 6.2).

```
id-PBES2 OBJECT IDENTIFIER ::= {pkcs-5 10}
```

The `parameters` field associated with this OID in an `AlgorithmIdentifier` shall have type `PBES2-params`:

```
PBES2-params ::= SEQUENCE {
  keyDerivationFunc AlgorithmIdentifier {{PBES2-KDFs}}
  encryptionScheme AlgorithmIdentifier {{PBES2-Encs}} }
```

The fields of type `PBES2-params` have the following meanings:

- `keyDerivationFunc` identifies the underlying key derivation function. It shall be an algorithm ID with an OID in the set `PBES2-KDFs`, which for this version of PKCS #5 shall consist of `id-PBKDF2` (Appendix A.2).

```
PBES2-KDFs ALGORITHM-IDENTIFIER ::=
  { {PBKDF2-params IDENTIFIED BY id-PBKDF2}, ... }
```

- `encryptionScheme` identifies the underlying encryption scheme. It shall be an algorithm ID with an OID in the set `PBES2-Encs`, whose definition is left to the application. Example underlying encryption schemes are given in Appendix B.2.

```
PBES2-Encs ALGORITHM-IDENTIFIER ::= { ... }
```

A.5 PBMAC1

The object identifier `id-PBMAC1` identifies the PBMAC1 message authentication scheme (Section 7.1).

```
id-PBMAC1 OBJECT IDENTIFIER ::= {pkcs-5 11}
```

The `parameters` field associated with this OID in an `AlgorithmIdentifier` shall have type `PBMAC1-params`:

```
PBMAC1-params ::=  SEQUENCE {
  keyDerivationFunc AlgorithmIdentifier {{PBMAC1-KDFs}},
  messageAuthScheme AlgorithmIdentifier {{PBMAC1-MACs}} }
```

The `keyDerivationFunct` field has the same meaning as the corresponding field of `PBES2-params` (Appendix A.4) except that the set of OIDs is `PBMAC1-KDFs`.

```
PBMAC1-KDFs ALGORITHM-IDENTIFIER ::=
  { {PBKDF2Params IDENTIFIED BY id-PBKDF2}, ... }
```

The `messageAuthScheme` field identifies the underlying message authentication scheme. It shall be an algorithm ID with an OID in the set `PBMAC1-MACs`, whose definition is left to the application. Example underlying encryption schemes are given in Appendix B.3.

```
PBMAC1-MACs ALGORITHM-IDENTIFIER ::= { ... }
```

B. Supporting techniques

This section gives several examples of underlying functions and schemes supporting the password-based schemes in Sections 5, 6 and 7. While these supporting techniques are appropriate for applications to implement, none of them is required to be implemented. It is expected, however, that profiles for PKCS #5 will be developed that specify particular supporting techniques.

This section also gives object identifiers for the supporting techniques.

The object identifiers `digestAlgorithm` and `encryptionAlgorithm` identify the arcs from which certain algorithm OIDs referenced in this section are derived:

```
digestAlgorithm OBJECT IDENTIFIER ::= {rsadsi 2}
encryptionAlgorithm OBJECT IDENTIFIER ::= {rsadsi 3}
```

B.1 Pseudorandom functions

An example pseudorandom function for PBKDF2 (Section 5.2) is HMAC-SHA-1.

B.1.1 HMAC-SHA-1

HMAC-SHA-1 is the pseudorandom function corresponding to the HMAC message authentication code [7] based on the SHA-1 hash function [12]. The pseudorandom function is the same function by which the message authentication code is computed, with a full-length output. HMAC-SHA-1 has a variable key length and a 20-octet (160-bit) output value.

The object identifier `id-hmacWithSHA1` identifies the HMAC-SHA-1 pseudorandom function:

```
id-hmacWithSHA1 OBJECT IDENTIFIER ::= {digestAlgorithm 7}
```

The `parameters` field associated with this OID in an `AlgorithmIdentifier` shall have type `NULL`. This object identifier is employed in the object set `PBKDF2-PRFs` (Appendix A.2).

Note. Although HMAC-SHA-1 was designed as a message authentication code, its proof of security is readily modified to accommodate requirements for a pseudorandom function, under stronger assumptions.

B.2 Encryption schemes

Example pseudorandom functions for PBES2 (Section 6.2) are DES-CBC-Pad, DES-EDE2-CBC-Pad, RC2-CBC-Pad, and RC5-CBC-Pad.

The object identifiers given in this section are intended to be employed in the object set `PBES2-Encs` (Appendix A.4).

B.2.1 DES-CBC-Pad

DES-CBC-Pad is single-key DES [9] in CBC mode [10] with the RFC 1423 padding operation (see Section 6.1.1). DES-CBC-Pad has an eight-octet encryption key and an eight-octet initialization vector. The key is considered as a 64-bit encoding of a 56-bit DES key with parity bits ignored.

The object identifier `desCBC` [13] identifies the DES-CBC-Pad encryption scheme:

```
desCBC OBJECT IDENTIFIER ::=
  {iso(1) identified-organization(3) oiw(14) secsig(3)
    algorithms(2) 7}
```

The `parameters` field associated with this OID in an `AlgorithmIdentifier` shall have type `OCTET STRING(8)`, specifying the initialization vector for CBC mode.

B.2.2 DES-EDE3-CBC-Pad

DES-EDE3-CBC-Pad is three-key triple-DES in CBC mode [1] with the RFC 1423 padding operation. DES-EDE3-CBC-Pad has a 24-octet encryption key and an eight-octet initialization vector. The key is considered as the concatenation of three eight-octet keys, each of which is a 64-bit encoding of a 56-bit DES key with parity bits ignored.

The object identifier `des-EDE3_CBC` identifies the DES-EDE3-CBC-Pad encryption scheme:

```
des-EDE3_CBC OBJECT IDENTIFIER ::= {encryptionAlgorithm 7}
```

The `parameters` field associated with this OID in an `AlgorithmIdentifier` shall have type `OCTET STRING(8)`, specifying the initialization vector for CBC mode.

Note. An OID for DES-EDE3-CBC without padding is given in ANSI X9.52 [1]; the one given here is preferred since it specifies padding.

B.2.3 RC2-CBC-Pad

RC2-CBC-Pad is the RC2™ encryption algorithm [17] in CBC mode with the RFC 1423 padding operation. RC2-CBC-Pad has a variable key length, from one to 128 octets, a separate "effective key bits" parameter from one to 1024 bits that limits the effective search space independent of the key length, and an eight-octet initialization vector.

The object identifier `rc2CBC` identifies the RC2-CBC-Pad encryption scheme:

```
rc2CBC OBJECT IDENTIFIER ::= {encryptionAlgorithm 2}
```

`The parameters` field associated with OID in an `AlgorithmIdentifier` shall have type `RC2-CBC-Parameter`:

```
RC2-CBC-Parameter ::= SEQUENCE {
  rc2ParameterVersion INTEGER OPTIONAL,
  iv OCTET STRING (8)}
```

The fields of type `RC2-CBCParameter` have the following meanings:

- `rc2ParameterVersion` is a proprietary RSA Data Security, Inc. encoding of the "effective key bits" for RC2. The following encodings are defined:

Effective key bits	Encoding
40	160
64	120
128	58
$b \geq 256$	b

  If the `rc2ParameterVersion` field is omitted, the "effective key bits" defaults to 32. (This is for backward compatibility with certain very old implementations.)

- `iv` is the eight-octet initialization vector.

B.2.4 RC5-CBC-Pad

RC5-CBC-Pad is the RC5™ encryption algorithm [16] in CBC mode with a generalization of the RFC 1423 padding operation[1]. This scheme is fully specified in [2]. RC5-CBC-Pad has a variable key length, from 0 to 256 octets, and supports both a 64-bit block size and a 128-bit block size. For the former, it has an eight-octet initialization vector, and for the latter, a 16-octet initialization vector. RC5-CBC-Pad also has a variable number of "rounds" in the encryption operation, from 8 to 127.

The object identifier `rc5_CBC_PAD` [2] identifies RC5-CBC-Pad encryption scheme:

```
rc5_CBC_PAD OBJECT IDENTIFIER ::= {encryptionAlgorithm 9}
```

The `parameters` field associated with this OID in an `AlgorithmIdentifier` shall have type `RC5_CBC_Parameters`:

```
RC5_CBC_Parameters ::= SEQUENCE {
  version INTEGER (v1_0(16)),
  rounds INTEGER (8..127),
  blockSizeInBits INTEGER (64, 128),
  iv OCTET STRING OPTIONAL  }
```

The fields of type `RC5_CBC_Parameters` have the following meanings:

- `version` is the version of the algorithm, which shall be `v1_0`.
- `rounds` is the number of rounds in the encryption operation, which shall be between 8 and 127.
- `blockSizeInBits` is the block size in bits, which shall be 64 or 128.
- `iv` is the initialization vector, an eight-octet string for 64-bit RC5 and a 16-octet string for 128-bit RC5. The default is a string of the appropriate length consisting of zero octets.

[1] The generalization of the padding operation is as follows. For RC5 with a 64-bit block size, the padding string is as defined in RFC 1423. For RC5 with a 128-bit block size, the padding string consists of 16-($||M||$ mod 16) octets each with value 16-($||M||$ mod 16).

B.3 Message authentication schemes

An example message authentication scheme for PBMAC1 (Section 7.1) is HMAC-SHA-1.

B.3.1 HMAC-SHA-1

HMAC-SHA-1 is the HMAC message authentication scheme [7] based on the SHA-1 hash function [12]. HMAC-SHA-1 has a variable key length and a 20-octet (160-bit) message authentication code.

The object identifier `id-hmacWithSHA1` (see Appendix B.1.1) identifies the HMAC-SHA-1 message authentication scheme. (The object identifier is the same for both the pseudorandom function and the message authentication scheme; the distinction is to be understood by context.) This object identifier is intended to be employed in the object set `PBMAC1-Macs` (Appendix A.5).

C. Intellectual property considerations

RSA Data Security makes no patent claims on the general constructions described in this document, although specific underlying techniques may be covered. Among the underlying techniques, the RC5 encryption algorithm (Appendix B.2.4) is protected by U.S. Patents 5,724,428 [18] and 5,835,600 [19].

RC2 and RC5 are trademarks of RSA Data Security.

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.

RSA Data Security makes no representations regarding intellectual property claims by other parties. Such determination is the responsibility of the user.

D. 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 was part of the June 3, 1991 initial public release of PKCS. Version 1.4 was published as NIST/OSI Implementors' Workshop document SEC-SIG-91-20.

Version 1.5

Version 1.5 incorporated several editorial changes, including updates to the references and the addition of a revision history.

Version 2.0

Version 2.0 incorporates major editorial changes in terms of the document structure, and introduces the PBES2 encryption scheme, the PBMAC1 message authentication scheme, and independent password-based key derivation functions. This version continues to support the encryption process in version 1.5.

E. References

[1] *American National Standard X9.52 - 1998, Triple Data Encryption Algorithm Modes of Operation.* Working draft, Accredited Standards Committee X9, July 27, 1998.

[2] R. Baldwin and R. Rivest. *RFC 2040: The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms.* IETF, October 1996.

[3] D. Balenson. *RFC 1423: Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers.* IETF, February 1993.

[4] S.M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Computer Society Conference on Research in Security and Privacy,* pages 72–84, IEEE Computer Society, 1992.

[5] D. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communications Review,* October 1996.

[6] B. Kaliski. *RFC 1319: The MD2 Message-Digest Algorithm.* IETF, April 1992.

[7] H. Krawczyk, M. Bellare, and R. Canetti. *RFC 2104: HMAC: Keyed-Hashing for Message Authentication.* IETF, February 1997.

[8] Robert Morris and Ken Thompson. Password security: A case history. *Communications of the ACM,* 22(11):594–597, November 1979.

[9] National Institute of Standards and Technology (NIST). *FIPS PUB 46-2: Data Encryption Standard.* December 30, 1993.

[10] National Institute of Standards and Technology (NIST). *FIPS PUB 81: DES Modes of Operation.* December 2, 1980.

[11] National Institute of Standards and Technology (NIST). *FIPS PUB 112: Password Usage.* May 30, 1985.

[12] National Institute of Standards and Technology (NIST). *FIPS Publication 180-1: Secure Hash Standard.* April 1994.

[13] [[OIW Implementors' Agreements.]]

[14] Blake Ramsdell. *S/MIME Version 3 Message Specification.* Internet-Draft `draft-ietf-smime-msg-05.txt`, August 6, 1998 (expires February 6, 1999).

[15] R. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm.* IETF, April 1992.

[16] R.L. Rivest. The RC5 encryption algorithm. *In Proceedings of the Second International Workshop on Fast Software Encryption*, pages 86-96, Springer-Verlag, 1994.

[17] R. Rivest. *RFC 2268: A Description of the RC2(r) Encryption Algorithm.* IETF, March 1998.

[18] R.L. Rivest. *Block-Encryption Algorithm with Data-Dependent Rotations.* U.S. Patent No. 5,724,428, March 3, 1998.

[19] R.L. Rivest. *Block Encryption Algorithm with Data-Dependent Rotations.* U.S. Patent No. 5,835,600, November 10, 1998.

[20] RSA Laboratories. *PKCS #5: Password-Based Encryption Standard.* Version 1.5, November 1993.

[21] RSA Laboratories. *PKCS #8: Private-Key Information Syntax Standard.* Version 1.2, November 1993.

[22] T. Wu. The Secure Remote Password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97-111, Internet Society, 1998.

[23] F. Yergeau. *RFC 2279: UTF-8, a Transformation Format of ISO 10646.* IETF, January 1998.

F. About PKCS

The *Public-Key Cryptography Standards* are specifications produced by RSA Laboratories in cooperation with secure systems developers worldwide for the purpose of accelerating the deployment of public-key cryptography. First published in 1991 as a result of meetings with a small group of early adopters of public-key technology, the PKCS documents have become widely referenced and implemented. Contributions from the PKCS series have become part of many formal and *de facto* standards, including ANSI X9 documents, PKIX, SET, S/MIME, and SSL.

Further development of PKCS occurs through mailing list discussions and occasional workshops, and suggestions for improvement are welcome. For more information, contact:

PKCS Editor
RSA Laboratories
2955 Campus Drive, Suite 400
San Mateo, CA 94403-2507 USA
(650) 295-7600
(650) 295-7700 (fax)
`pkcs-editor@rsa.com`
`http://www.rsa.com/rsalabs/pubs/PKCS`