Internet Engineering Task Force (IETF)
Request for Comments: 8018
Obsoletes: 2898
Category: Informational
ISSN: 2070-1721

K. Moriarty, Ed.
Dell EMC
B. Kaliski
Verisign
A. Rusch
RSA

January 2017

PKCS #5: Password-Based Cryptography Specification Version 2.1

Abstract

This document provides recommendations for the implementation of password-based cryptography, covering key derivation functions, encryption schemes, message authentication schemes, and ASN.1 syntax identifying the techniques.

This document represents a republication of PKCS #5 v2.1 from RSA Laboratories' Public-Key Cryptography Standards (PKCS) series. By publishing this RFC, change control is transferred to the IETF.

This document also obsoletes RFC 2898.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc8018.

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 [PKCS8] [RFC5958]. It is expected that application standards and implementation profiles 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 [BELLOV] [JABLON] [WU] are outside the scope of this document. Guidelines for the selection of passwords are also outside the scope. This document supersedes PKCS #5 version 2.0 [RFC2898] but includes compatible techniques.

This document represents a republication of PKCS #5 v2.1 [PKCS5_21] from RSA Laboratories' Public-Key Cryptography Standards (PKCS) series.

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, a positive integer
EM	encoded message, an octet string
Hash	underlying hash function
hLen	length in octets of pseudorandom function output, a positive integer
l	length in blocks of derived key, a positive 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, a positive integer

S salt, an octet string

T message authentication code, an octet string

T_1, ..., T_l, U_1, ..., U_c
: 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 <= i <= 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 [MORRIS] 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 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 v2.0, and are adopted here as well for the various cryptographic operations. Thus, password-based key derivation as defined here is a function of a 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 v2.0, 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 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 [RFC3629] 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 [NISTSP63] 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, one of which 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, or even the most likely keys, corresponding to a dictionary of passwords. 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. The fact that collisions are unlikely addresses some concerns about interactions between multiple uses of the same key that may arise when using 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 an 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 (or a prefix of that key) with the password-based encryption and authentication techniques supported for a given password, then the salt may be generated at random and need not be checked for a particular format by the party receiving the salt. It should be at least eight octets (64 bits) long.

2. Otherwise, the salt should contain data that explicitly distinguishes between different operations and different key lengths, in addition to a random part that is at least eight octets long, and this data should be checked or regenerated by the party receiving the salt. 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. Mathematically, an iteration count of c will increase the security strength of a password by log2(c) bits against trial-based attacks like brute force or dictionary attacks.

Choosing a reasonable value for the iteration count depends on environment and circumstances, and varies from application to application. This document follows the recommendations made in FIPS Special Publication 800-132 [NISTSP132], which says

> The iteration count shall be selected as large as possible, as long as the time required to generate the key using the entered password is acceptable for the users. [...] A minimum iteration count of 1,000 is recommended. For especially critical keys, or for very powerful systems or systems where user-perceived performance is not critical, an iteration count of 10,000,000 may be appropriate.

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 [RFC1319], MD5 [RFC1321], or SHA-1 [NIST180], 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 [PKCS5_15].

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 octet string
	c	iteration count, a positive integer
	dkLen	intended length in octets of derived key, a positive integer, at most 16 for MD2 or MD5 and 20 for SHA-1
Output:	DK	derived key, a dkLen-octet string

Steps:

1. If dkLen > 16 for MD2 and MD5, or dkLen > 20 for SHA-1, output "derived key too long" and stop.

2. 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:

```
T_1 = Hash (P || S) ,
T_2 = Hash (T_1) ,
...
T_c = Hash (T_{c-1}) ,
DK = T_c<0..dkLen-1>
```

3. Output the derived key DK.

5.2. PBKDF2

PBKDF2 applies a pseudorandom function (see Appendix B.1 for an example) to derive keys. The length of the derived key is essentially unbounded. (However, the maximum effective search space for the derived key may be limited by the structure of the underlying pseudorandom function. See Appendix B.1 for further discussion.) 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, a positive integer, at most (2^32 - 1) * hLen
Output:	DK	derived key, a dkLen-octet string

Steps:

1. If dkLen > (2^32 - 1) * 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 = CEIL (dkLen / hLen)
r = dkLen - (l - 1) * hLen
```

Here, CEIL (x) is the "ceiling" function, i.e., the smallest integer greater than, or equal to, x.

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) ,
...
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) = U_1 \xor U_2 \xor ... \xor U_c
```

where

```
U_1 = PRF (P, S || INT (i)) ,
U_2 = PRF (P, U_1) ,
...
U_c = PRF (P, U_{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 || T_2 ||  ...  || 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 U_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 [NIST46] or RC2 [RFC2268] in cipher block chaining (CBC) mode [NIST81]. PBES1 is compatible with the encryption scheme in PKCS #5 v1.5 [PKCS5_15].

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. PBES1 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 = 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<0..7>
IV  = DK<8..15>
```

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

```
EM = M || 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 = 01, if ||M|| mod 8 = 7 ;
PS = 02 02, if ||M|| mod 8 = 6 ;
...
PS = 08 08 08 08 08 08 08 08, if ||M|| mod 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 [RFC1423].)

5. Encrypt the encoded message EM with the underlying block cipher (DES or RC2) in CBC 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 [NIST46]). 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. PBES1 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 = 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<0..7>
   IV = DK<8..15>

4. Decrypt the ciphertext C with the underlying block cipher (DES or RC2) in CBC 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 || 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. PBES2 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. PBES2 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 MAC 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. PBMAC1 Generation Operation

The MAC generation operation for PBMAC1 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. PBMAC1 Verification Operation

The MAC verification operation for PBMAC1 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".

8. Security Considerations

Password-based cryptography is generally limited in the security that it can provide, particularly for methods such as those defined in this document where offline password search is possible. While the use of salt and iteration count can increase the complexity of attack (see Section 4 for recommendations), it is essential that passwords

are selected well, and relevant guidelines (e.g., [NISTSP63]) should be taken into account. It is also important that passwords be protected well if stored.

In general, different keys should be derived from a password for different uses to minimize the possibility of unintended interactions. For password-based encryption with a single algorithm, a random salt is sufficient to ensure that different keys will be produced. In certain other situations, as outlined in Section 4, a structured salt is necessary. The recommendations in Section 4 should thus be taken into account when selecting the salt value.

For information on security considerations for MD2 [RFC1319], see [RFC6149]; for MD5 [RFC1321], see [RFC6151]; and for SHA-1 [NIST180], see [RFC6194].

9. Normative References

[ANSIX952] ANSI, "Triple Data Encryption Algorithm Modes of Operation", Accredited Standards Committee X9, X9.52-1998, July 1998.

[BELLOV] Bellovin, S. and M. Merritt, "Encrypted Key Exchange: Password-based Protocols Secure against Dictionary Attacks", Proceedings of the IEEE Symposium on Research in Security and Privacy, pages 72-84, IEEE Computer Society, DOI 10.1109/RISP.1992.213269, 1992.

[COCHRAN] Cochran, M., "Notes on the Wang et al. 2^63 SHA-1 Differential Path", Cryptology ePrint Archive: Report 2007/474, August 2008, <http://eprint.iacr.org/2007/474>.

[ISO8824-1] International Organization for Standardization, "Information technology - Abstract Syntax Notation One (ASN.1) - Specification of basic notation", ISO/IEC 8824-1:2008, 2008.

[ISO8824-2] International Organization for Standardization, "Information technology - Abstract Syntax Notation One (ASN.1) - Information object specification", ISO/IEC 8824-2:2008, 2008.

[ISO8824-3]
International Organization for Standardization, "Information technology - Abstract Syntax Notation One (ASN.1) - Constraint specification", ISO/IEC 8824-3:2008, 2008.

[ISO8824-4]
International Organization for Standardization, "Information technology - Abstract Syntax Notation One (ASN.1) - Parameterization of ASN.1 specifications", ISO/IEC 8824-4:2008, 2008.

[JABLON] Jablon, D., "Strong Password-Only Authenticated Key Exchange", ACM SIGCOMM Computer Communication Review, Volume 26, Issue 5, DOI 10.1145/242896.242897, October 1996.

[MORRIS] Morris, R. and K. Thompson, "Password security: A case history", Communications of the ACM, Vol. 22, Issue 11, pages 594-597, DOI 10.1145/359168.359172, November 1979.

[NIST46] National Institute of Standards and Technology (NIST), "Data Encryption Standard", FIPS PUB 46-3, October 1999.

[NIST81] National Institute of Standards and Technology (NIST), "DES Modes of Operation", FIPS PUB 81, December 2, 1980.

[NIST180] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015.

[NIST197] National Institute of Standards and Technology (NIST), "Advance Encryption Standard (AES)", FIPS PUB 197, November 2001.

[NIST198] National Institute of Standards and Technology (NIST), "The Keyed - Hash Message Authentication Code (HMAC)", FIPS PUB 198-1, July 2008.

[NISTSP63] National Institute of Standards and Technology (NIST), "Electronic Authentication Guideline", NIST Special Publication 800-63-2, DOI 10.6028/NIST.SP.800-63-2, August 2013.

[NISTSP132] National Institute of Standards and Technology (NIST), "Recommendation for Password-Based Key Derivation, Part 1: Storage Applications", NIST Special Publication 800-132, DOI 10.6028/NIST.SP.800-132, December 2010.

[PKCS5_15] RSA Laboratories, "PKCS #5: Password-Based Encryption Standard Version 1.5", November 1993.

[PKCS5_21] RSA Laboratories, "PKCS #5: Password-Based Encryption Standard Version 2.1", October 2012.

[PKCS8] Kaliski, B., "Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2", RFC 5208, DOI 10.17487/RFC5208, May 2008, <http://www.rfc-editor.org/info/rfc5208>.

[RC5] Rivest, R.L., "The RC5 encryption algorithm", In Proceedings of the Second International Workshop on Fast Software Encryption, pages 86-96, Springer-Verlag, DOI 10.1007/3-540-60590-8_7, 1994.

[RFC1319] Kaliski, B., "The MD2 Message-Digest Algorithm", RFC 1319, DOI 10.17487/RFC1319, April 1992, <http://www.rfc-editor.org/info/rfc1319>.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <http://www.rfc-editor.org/info/rfc1321>.

[RFC1423] Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers", RFC 1423, DOI 10.17487/RFC1423, February 1993, <http://www.rfc-editor.org/info/rfc1423>.

[RFC2040] Baldwin, R. and R. Rivest, "The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms", RFC 2040, DOI 10.17487/RFC2040, October 1996, <http://www.rfc-editor.org/info/rfc2040>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

[RFC2268] Rivest, R., "A Description of the RC2(r) Encryption Algorithm", RFC 2268, DOI 10.17487/RFC2268, March 1998, <http://www.rfc-editor.org/info/rfc2268>.

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, DOI 10.17487/RFC2898, September 2000, <http://www.rfc-editor.org/info/rfc2898>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <http://www.rfc-editor.org/info/rfc5652>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <http://www.rfc-editor.org/info/rfc5958>.

[RFC6149] Turner, S. and L. Chen, "MD2 to Historic Status", RFC 6149, DOI 10.17487/RFC6149, March 2011, <http://www.rfc-editor.org/info/rfc6149>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <http://www.rfc-editor.org/info/rfc6151>.

[RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <http://www.rfc-editor.org/info/rfc6194>.

[WANG] Wang, X., Yao, A.C., and F. Yao, "Cryptanalysis on SHA-1", presented by Adi Shamir at the rump session of CRYPTO 2005, <http://csrc.nist.gov/groups/ST/hash/documents/Wang_SHA1-New-Result.pdf>.

[WU] Wu, T., "The Secure Remote Password protocol", In Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium, pages 97-111, Internet Society, 1998, <https://www.isoc.org/isoc/conferences/ndss/98/wu.pdf>.

Appendix 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. (Various aspects of ASN.1 are specified in several ISO/IEC standards [ISO8824-1] [ISO8824-2] [ISO8824-3] [ISO8824-4].)

The object identifier pkcs-5 identifies the arc of the OID tree from which the OIDs (specific to PKCS #5) 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}
```

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 12}
```

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

```
PBKDF2-params ::= SEQUENCE {
    salt CHOICE {
        specified OCTET STRING,
        otherSource AlgorithmIdentifier {{PBKDF2-SaltSources}}
    },
    iterationCount INTEGER (1..MAX),
    keyLength INTEGER (1..MAX) OPTIONAL,
    prf AlgorithmIdentifier {{PBKDF2-PRFs}} DEFAULT
    algid-hmacWithSHA1 }
```

The fields of type PBKDF2-params have the following meanings:

- salt specifies the salt value or the source of the salt value. It shall either be an octet string or an algorithm ID with an OID in the set PBKDF2-SaltSources, which is reserved for future versions of PKCS #5.

  The salt-source approach is intended to indicate how the salt value is to be generated as a function of parameters in the algorithm ID, application data, or both. For instance, it may indicate that the salt value is produced from the encoding of a structure that specifies detailed information about the derived key as suggested in Section 4.1. Some of the information may be carried elsewhere, e.g., in the encryption algorithm ID. However, such facilities are deferred to a future version of PKCS #5.

  In this version, an application may achieve the benefits mentioned in Section 4.1 by choosing a particular interpretation of the salt value in the specified alternative.

PBKDF2-SaltSources ALGORITHM-IDENTIFIER ::= { ... }

- iterationCount specifies the iteration count. The maximum iteration count allowed depends on the implementation. It is expected that implementation profiles may further constrain the bounds.

- keyLength, an optional field, is the length in octets of the derived key. The maximum key length allowed depends on the implementation; it is expected that implementation profiles may further constrain the bounds. The field is provided for convenience only; the key length is not cryptographically protected. If there is concern about interaction between operations with different key lengths for a given salt (see Section 4.1), the salt should distinguish among the different key lengths.

- 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},
     {NULL IDENTIFIED BY id-hmacWithSHA224},
     {NULL IDENTIFIED BY id-hmacWithSHA256},
     {NULL IDENTIFIED BY id-hmacWithSHA384},
     {NULL IDENTIFIED BY id-hmacWithSHA512},
     {NULL IDENTIFIED BY id-hmacWithSHA512-224},
     {NULL IDENTIFIED BY id-hmacWithSHA512-256},
     ...
   }
```

The default pseudorandom function is HMAC-SHA-1:

```
      algid-hmacWithSHA1 AlgorithmIdentifier {{PBKDF2-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-5.10
SHA-1	RC2	pkcs-5.11

```
   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 10}
   pbeWithSHA1AndRC2-CBC OBJECT IDENTIFIER ::= {pkcs-5 11}
```

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 13}
```

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. Examples of 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 14}
```

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 keyDerivationFunc 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 ::=
   { {PBKDF2-params 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. Examples of underlying encryption schemes are given in Appendix B.3.

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

Appendix 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

Examples of pseudorandom function for PBKDF2 (Section 5.2) include HMAC with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256. Applications may employ other schemes as well.

B.1.1. HMAC-SHA-1

HMAC-SHA-1 is the pseudorandom function corresponding to the HMAC message authentication code [RFC2104] based on the SHA-1 hash function [NIST180]. The pseudorandom function is the same function by which the message authentication code is computed, with a full-length output. (The first argument to the pseudorandom function PRF serves as HMAC's "key", and the second serves as HMAC's "text". In the case of PBKDF2, the "key" is thus the password and the "text" is the salt.) HMAC-SHA-1 has a variable key length and a 20-octet (160-bit) output value.

Although the length of the key to HMAC-SHA-1 is essentially unbounded, the effective search space for pseudorandom function outputs may be limited by the structure of the function. In particular, when the key is longer than 512 bits, HMAC-SHA-1 will first hash it to 160 bits. Thus, even if a long derived key consisting of several pseudorandom function outputs is produced from a key, the effective search space for the derived key will be at most 160 bits. Although the specific limitation for other key sizes depends on details of the HMAC construction, one should assume, to be conservative, that the effective search space is limited to 160 bits for other key sizes as well.

(The 160-bit limitation should not generally pose a practical limitation in the case of password-based cryptography, since the search space for a password is unlikely to be greater than 160 bits.)

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. A hash function may also meet the requirements of a pseudorandom function under certain assumptions. For instance, the direct

application of a hash function to the concatenation of the "key" and the "text" may be appropriate, provided that "text" has appropriate structure to prevent certain attacks. HMAC-SHA-1 is preferable, however, because it treats "key" and "text" as separate arguments and does not require "text" to have any structure.

During 2004 and 2005, there were a number of attacks on SHA-1 that reduced its perceived effective strength against collision attacks to 62 bits instead of the expected 80 bits (e.g., Wang et al. [WANG], confirmed by M. Cochran [COCHRAN]). However, since these attacks centered on finding collisions between values, they are not a direct security consideration here because the collision-resistant property is not required by the HMAC authentication scheme.

B.1.2. HMAC-SHA-2

HMAC-SHA-2 refers to the set of pseudorandom functions corresponding to the HMAC message authentication code (now a FIPS standard [NIST198]) based on the new SHA-2 functions (FIPS 180-4 [NIST180]). HMAC-SHA-2 has a variable key length and variable output value depending on the hash function chosen (SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, or SHA-512/256) -- that is, 28, 32, 48, or 64 octets.

Using the new hash functions extends the search space for the produced keys. Where SHA-1 limits the search space to 20 octets, SHA-2 sets new limits of 28, 32, 48, and 64 octets.

Object identifiers for HMAC are defined as follows:

```
id-hmacWithSHA224 OBJECT IDENTIFIER ::= {digestAlgorithm 8}
id-hmacWithSHA256 OBJECT IDENTIFIER ::= {digestAlgorithm 9}
id-hmacWithSHA384 OBJECT IDENTIFIER ::= {digestAlgorithm 10}
id-hmacWithSHA512 OBJECT IDENTIFIER ::= {digestAlgorithm 11}
id-hmacWithSHA512-224 OBJECT IDENTIFIER ::= {digestAlgorithm 12}
id-hmacWithSHA512-256 OBJECT IDENTIFIER ::= {digestAlgorithm 13}
```

B.2. Encryption Schemes

An example encryption scheme for PBES2 (Section 6.2) is AES-CBC-Pad. The schemes defined in PKCS #5 v2.0 [RFC2898], DES-CBC-Pad, DES-EDE3-CBC-Pad, RC2-CBC-Pad, and RC5-CBC-Pad, are still supported, but DES-CBC-Pad, DES-EDE3-CBC-Pad, RC2-CBC-Pad are now considered legacy and should only be used for backwards compatibility reasons.

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 [NIST46] in CBC mode [NIST81] with the padding operation specified in RFC 1423 [RFC1423] (see Section 6.1.1 of this document). 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 (defined in the NIST/OSI Implementors' Workshop agreements) 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 (SIZE(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 [ANSIX952] with the padding operation specified in RFC 1423 [RFC1423]. 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 (SIZE(8)), specifying the initialization vector for CBC mode.

Note: An OID for DES-EDE3-CBC without padding is given in ANSI X9.52 [ANSIX952]; the one given here is preferred since it specifies padding.

B.2.3. RC2-CBC-Pad

RC2-CBC-Pad is the RC2 encryption algorithm [RFC2268] in CBC mode with the padding operation specified in RFC 1423 [RFC1423]. 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 (SIZE(8)) }
```

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

- rc2ParameterVersion is a proprietary RSA 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 >= 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 [RC5] in CBC mode with the padding operation specified in RFC 5652 [RFC5652], which is a generalization of the padding operation specified in RFC 1423 [RFC1423]. The scheme is fully specified in [RFC2040]. 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.

Note: For RC5 with a 64-bit block size, the padding string is as defined in RFC 1423 [RFC1423]. 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).

The object identifier rc5-CBC-PAD [RFC2040] identifies the 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)} (v1-0),
   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.

B.2.5. AES-CBC-Pad

AES-CBC-Pad is the AES encryption algorithm [NIST197] in CBC mode with the padding operation specified in RFC 5652 [RFC5652]. AES-CBC-Pad has a variable key length of 16, 24, or 32 octets and has a 16-octet block size. It has a 16-octet initialization vector.

Note: For AES, the padding string consists of 16-(||M|| mod 16) octets each with value 16-(||M|| mod 16).

For AES, object identifiers are defined depending on key size and operation mode. For example, the 16-octet (128-bit) key AES encryption scheme in CBC mode would be aes128-CBC-Pad identifying the AES-CBC-PAD encryption scheme using a 16-octet key:

```
aes128-CBC-PAD OBJECT IDENTIFIER ::= {aes 2}
```

The AES object identifier is defined in Appendix C.

The parameters field associated with this OID in an AlgorithmIdentifier shall have type OCTET STRING (SIZE(16)), specifying the initialization vector for CBC mode.

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 [RFC2104] based on the SHA-1 hash function [NIST180]. 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).

B.3.2. HMAC-SHA-2

HMAC-SHA-2 refers to the set of HMAC message authentication schemes [NIST198] based on the SHA-2 functions [NIST180]. HMAC-SHA-2 has a variable key length and a message authentication code whose length is based on the hash function chosen (SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, or SHA-512/256) -- that is, 28, 32, 48, or 64 octets.

The object identifiers id-hmacWithSHA224, id-hmacWithSHA256, id-hmacWithSHA384, id-hmacWithSHA512, id-hmacWithSHA512-224, and id-hmacWithSHA512-256 (see Appendix B.1.2) identify the HMAC-SHA-2 schemes. The object identifiers are the same for both the pseudorandom functions and the message authentication schemes; the distinction is to be understood by context. These object identifiers are intended to be employed in the object set PBMAC1-Macs (Appendix A.5).

Appendix C. ASN.1 Module

For reference purposes, the ASN.1 syntax in the preceding sections is presented as an ASN.1 module here.

```
-- PKCS #5 v2.1 ASN.1 Module
-- Revised October 27, 2012

-- This module has been checked for conformance with the
-- ASN.1 standard by the OSS ASN.1 Tools

PKCS5v2-1 {
   iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-5(5)
   modules(16) pkcs5v2-1(2)
}

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- ========================
-- Basic object identifiers
-- ========================

nistAlgorithms OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) country(16)
                                      us(840) organization(1)
                                      gov(101) csor(3) 4}
oiw    OBJECT IDENTIFIER ::= {iso(1) identified-organization(3) 14}
rsadsi OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) 113549}
pkcs   OBJECT IDENTIFIER ::= {rsadsi 1}
pkcs-5 OBJECT IDENTIFIER ::= {pkcs 5}

-- ========================
-- Basic types and classes
-- ========================

AlgorithmIdentifier { ALGORITHM-IDENTIFIER:InfoObjectSet } ::=
  SEQUENCE {
    algorithm ALGORITHM-IDENTIFIER.&id({InfoObjectSet}),
    parameters ALGORITHM-IDENTIFIER.&Type({InfoObjectSet}
    {@algorithm}) OPTIONAL
}

ALGORITHM-IDENTIFIER ::= TYPE-IDENTIFIER

-- ======
-- PBKDF2
-- ======
```

```
   PBKDF2Algorithms ALGORITHM-IDENTIFIER ::= {
      {PBKDF2-params IDENTIFIED BY id-PBKDF2},
      ...
   }

   id-PBKDF2 OBJECT IDENTIFIER ::= {pkcs-5 12}

   algid-hmacWithSHA1 AlgorithmIdentifier {{PBKDF2-PRFs}} ::=
      {algorithm id-hmacWithSHA1, parameters NULL : NULL}

   PBKDF2-params ::= SEQUENCE {
       salt CHOICE {
         specified OCTET STRING,
         otherSource AlgorithmIdentifier {{PBKDF2-SaltSources}}
       },
       iterationCount INTEGER (1..MAX),
       keyLength INTEGER (1..MAX) OPTIONAL,
       prf AlgorithmIdentifier {{PBKDF2-PRFs}} DEFAULT
       algid-hmacWithSHA1
   }

   PBKDF2-SaltSources ALGORITHM-IDENTIFIER ::= { ... }

   PBKDF2-PRFs ALGORITHM-IDENTIFIER ::= {
     {NULL IDENTIFIED BY id-hmacWithSHA1},
     {NULL IDENTIFIED BY id-hmacWithSHA224},
     {NULL IDENTIFIED BY id-hmacWithSHA256},
     {NULL IDENTIFIED BY id-hmacWithSHA384},
     {NULL IDENTIFIED BY id-hmacWithSHA512},
     {NULL IDENTIFIED BY id-hmacWithSHA512-224},
     {NULL IDENTIFIED BY id-hmacWithSHA512-256},
     ...
   }

   -- =====
   -- PBES1
   -- =====

   PBES1Algorithms ALGORITHM-IDENTIFIER ::= {
      {PBEParameter IDENTIFIED BY pbeWithMD2AndDES-CBC}  |
      {PBEParameter IDENTIFIED BY pbeWithMD2AndRC2-CBC}  |
      {PBEParameter IDENTIFIED BY pbeWithMD5AndDES-CBC}  |
      {PBEParameter IDENTIFIED BY pbeWithMD5AndRC2-CBC}  |
      {PBEParameter IDENTIFIED BY pbeWithSHA1AndDES-CBC} |
      {PBEParameter IDENTIFIED BY pbeWithSHA1AndRC2-CBC},
      ...
   }
```

```
   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 10}
   pbeWithSHA1AndRC2-CBC OBJECT IDENTIFIER ::= {pkcs-5 11}

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

   -- =====
   -- PBES2
   -- =====

   PBES2Algorithms ALGORITHM-IDENTIFIER ::= {
      {PBES2-params IDENTIFIED BY id-PBES2},
      ...
   }

   id-PBES2 OBJECT IDENTIFIER ::= {pkcs-5 13}

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

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

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

   -- ======
   -- PBMAC1
   -- ======

   PBMAC1Algorithms ALGORITHM-IDENTIFIER ::= {
      {PBMAC1-params IDENTIFIED BY id-PBMAC1},
      ...
   }

   id-PBMAC1 OBJECT IDENTIFIER ::= {pkcs-5 14}

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

```
        messageAuthScheme AlgorithmIdentifier {{PBMAC1-MACs}}
    }

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

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

    -- =====================
    -- Supporting techniques
    -- =====================

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

    SupportingAlgorithms ALGORITHM-IDENTIFIER ::= {
       {NULL IDENTIFIED BY id-hmacWithSHA1}                    |
       {OCTET STRING (SIZE(8)) IDENTIFIED BY desCBC}           |
       {OCTET STRING (SIZE(8)) IDENTIFIED BY des-EDE3-CBC}     |
       {RC2-CBC-Parameter IDENTIFIED BY rc2CBC}                |
       {RC5-CBC-Parameters IDENTIFIED BY rc5-CBC-PAD},         |
       {OCTET STRING (SIZE(16)) IDENTIFIED BY aes128-CBC-PAD}  |
       {OCTET STRING (SIZE(16)) IDENTIFIED BY aes192-CBC-PAD}  |
       {OCTET STRING (SIZE(16)) IDENTIFIED BY aes256-CBC-PAD},
        ...
    }

    id-hmacWithSHA1 OBJECT IDENTIFIER ::= {digestAlgorithm 7}
    id-hmacWithSHA224 OBJECT IDENTIFIER ::= {digestAlgorithm 8}
    id-hmacWithSHA256 OBJECT IDENTIFIER ::= {digestAlgorithm 9}
    id-hmacWithSHA384 OBJECT IDENTIFIER ::= {digestAlgorithm 10}
    id-hmacWithSHA512 OBJECT IDENTIFIER ::= {digestAlgorithm 11}
    id-hmacWithSHA512-224 OBJECT IDENTIFIER ::= {digestAlgorithm 12}
    id-hmacWithSHA512-256 OBJECT IDENTIFIER ::= {digestAlgorithm 13}

    desCBC OBJECT IDENTIFIER ::= {oiw secsig(3) algorithms(2) 7}

    des-EDE3-CBC OBJECT IDENTIFIER ::= {encryptionAlgorithm 7}

    rc2CBC OBJECT IDENTIFIER ::= {encryptionAlgorithm 2}

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

```
rc5-CBC-PAD OBJECT IDENTIFIER ::= {encryptionAlgorithm 9}

RC5-CBC-Parameters ::= SEQUENCE {
   version INTEGER {v1-0(16)} (v1-0),
   rounds INTEGER (8..127),
   blockSizeInBits INTEGER (64 | 128),
   iv OCTET STRING OPTIONAL
}

aes OBJECT IDENTIFIER ::= { nistAlgorithms 1 }
aes128-CBC-PAD OBJECT IDENTIFIER ::= { aes 2 }
aes192-CBC-PAD OBJECT IDENTIFIER ::= { aes 22 }
aes256-CBC-PAD OBJECT IDENTIFIER ::= { aes 42 }

END
```

Appendix D. Revision History of PKCS #5

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.

Version 2.1

This document transfers PKCS #5 into the IETF and includes some minor changes from the authors for this submission.

- Introduces AES/CBC as an encryption scheme for PBES2 and HMAC with the hash functions SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 as pseudorandom functions for PBKDF2 and message authentication schemes for PBMAC1.

- Changes references for PKCS #5 to RFC 2898 and for PKCS #8 to RFCs 5208 and 5898.

- Incorporates corrections of two editorial errata reported on PKCS #5 [RFC2898].

- Added security considerations for MD2, MD5, and SHA-1.

Appendix E. 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, Secure Electronic Transaction (SET), S/MIME, and SSL.

Further development of most PKCS documents occurs through the IETF. Suggestions for improvement are welcome.

Acknowledgements

This document is based on a contribution of RSA Laboratories, the research center of RSA Security Inc.

RC2 and RC5 are trademarks of EMC Corporation.

Authors' Addresses

Kathleen M. Moriarty (editor)
Dell EMC
176 South Street
Hopkinton, MA 01748
United States of America

Email: Kathleen.Moriarty@Dell.com

Burt Kaliski
Verisign
12061 Bluemont Way
Reston, VA 20190
United States of America

Email: bkaliski@verisign.com
URI: http://verisignlabs.com

Andreas Rusch
RSA
345 Queen Street
Brisbane, QLD 4000
Australia

Email: andreas.rusch@rsa.com