

## Password-based Encryption for CMS

### Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

### Copyright Notice

Copyright (C) The Internet Society (2001). All Rights Reserved.

### Abstract

This document provides a method of encrypting data using user-supplied passwords and, by extension, any form of variable-length keying material which is not necessarily an algorithm-specific fixed-format key. The Cryptographic Message Syntax data format does not currently contain any provisions for password-based data encryption.

### 1. Introduction

This document describes a password-based content encryption mechanism for CMS. This is implemented as a new RecipientInfo type and is an extension to the RecipientInfo types currently defined in RFC 2630.

The format of the messages are described in ASN.1 [ASN1].

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## 1.1 Password-based Content Encryption

CMS currently defined three recipient information types for public-key key wrapping (KeyTransRecipientInfo), conventional key wrapping (KEKRecipientInfo), and key agreement (KeyAgreeRecipientInfo). The recipient information described here adds a fourth type, PasswordRecipientInfo, which provides for password-based key wrapping.

## 1.2 RecipientInfo Types

The new recipient information type is an extension to the RecipientInfo type defined in section 6.2 of CMS, extending the types to:

```
RecipientInfo ::= CHOICE {  
    ktri KeyTransRecipientInfo,  
    kari [1] KeyAgreeRecipientInfo,  
    kekri [2] KEKRecipientInfo,  
    pwri [3] PasswordRecipientInfo  -- New RecipientInfo type  
}
```

Although the recipient information generation process is described in terms of a password-based operation (since this will be its most common use), the transformation employed is a general-purpose key derivation one which allows any type of keying material to be converted into a key specific to a particular content-encryption algorithm. Since the most common use for password-based encryption is to encrypt files which are stored locally (rather than being transmitted across a network), the term "recipient" is somewhat misleading, but is used here because the other key transport mechanisms have always been described in similar terms.

### 1.2.1 PasswordRecipientInfo Type

Recipient information using a user-supplied password or previously agreed-upon key is represented in the type PasswordRecipientInfo. Each instance of PasswordRecipientInfo will transfer the content-encryption key (CEK) to one or more recipients who have the previously agreed-upon password or key-encryption key (KEK).

```
PasswordRecipientInfo ::= SEQUENCE {  
    version CMSVersion,  -- Always set to 0  
    keyDerivationAlgorithm  
        [0] KeyDerivationAlgorithmIdentifier OPTIONAL,  
    keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,  
    encryptedKey EncryptedKey }
```

The fields of type PasswordRecipientInfo have the following meanings:

version is the syntax version number. It MUST be 0. Details of the CMSVersion type are discussed in CMS [RFC2630], section 10.2.5.

keyDerivationAlgorithm identifies the key-derivation algorithm, and any associated parameters, used to derive the KEK from the user-supplied password. If this field is absent, the KEK is supplied from an external source, for example a crypto token such as a smart card.

keyEncryptionAlgorithm identifies the key-encryption algorithm, and any associated parameters, used to encrypt the CEK with the KEK.

encryptedKey is the result of encrypting the content-encryption key with the KEK.

### 1.2.2 Rationale

Password-based key wrapping is a two-stage process, a first stage in which a user-supplied password is converted into a KEK if required, and a second stage in which the KEK is used to encrypt a CEK. These two stages are identified by the two algorithm identifiers. Although the PKCS #5v2 standard [RFC2898] goes one step further to wrap these up into a single algorithm identifier, this design is particular to that standard and may not be applicable for other key wrapping mechanisms. For this reason the two steps are specified separately.

The current format doesn't provide any means of differentiating between multiple password recipient infos, which would occur for example if two passwords are used to encrypt the same data. Unfortunately there is a lack of existing practice in this area, since typical applications follow the model of encrypting data such as a file with a single password obtained from the user. Without any clear requirements, an appropriate multiple password mechanism would be difficult (perhaps impossible) to define at this time. If sufficient demand emerges then this may be addressed in a future version of this document, for example by adding an optional identification field of an appropriate form.

## 2 Supported Algorithms

This section lists the algorithms that must be implemented. Additional algorithms that should be implemented are also included.

## 2.1 Key Derivation Algorithms

These algorithms are used to convert the password into a KEK. The key derivation algorithms are:

`KeyDerivationAlgorithmIdentifier ::= AlgorithmIdentifier`

Conforming implementations MUST include PBKDF2 [RFC2898]. Appendix B contains a more precise definition of the allowed algorithm type than is possible using 1988 ASN.1.

## 2.2 Key Encryption Algorithms

These algorithms are used to encrypt the CEK using the derived KEK. The key encryption algorithms are:

`KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier`

The PasswordRecipientInfo key encryption algorithm identifier is:

`id-alg-PWRI-KEK OBJECT IDENTIFIER ::= { iso(1) member-body(2)  
us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) alg(3) 9 }`

The AlgorithmIdentifier parameters field for this algorithm contains the KEK encryption algorithm used with the the key wrap algorithm specified in section 2.3.

There is no requirement that the CEK algorithm match the KEK encryption algorithm, although care should be taken to ensure that, if different algorithms are used, they offer an equivalent level of security (for example wrapping a Triple-DES key with an RC2/40 key leads to a severe impedance mismatch in encryption strength).

Conforming implementations MUST implement the id-alg-PWRI-KEK key wrap algorithm. For the KEK encryption algorithms used by id-alg-PWRI-KEK, conforming implementations MUST include Triple-DES in CBC mode and MAY include other algorithms such as AES, CAST-128, RC5, IDEA, Skipjack, Blowfish, and encryption modes as required. Implementations SHOULD NOT include any KSG (keystream generator) ciphers such as RC4 or a block cipher in OFB mode, and SHOULD NOT include a block cipher in ECB mode.

### 2.2.1 Rationale

The use of a level of indirection in specifying the KeyEncryptionAlgorithmIdentifier allows alternative wrapping algorithms to be used in the future. If the KEK algorithm were specified directly in this field then any use of an alternative

wrapping algorithm would require a change to the PasswordRecipientInfo structure rather than simply a change to the key encryption algorithm identifier.

The parameter field for this algorithm identifier could be specified to default to triple-DES, however due to the confusion over NULL vs absent parameters in algorithm identifiers it's left explicit with no default value.

### 2.3.1 Key Wrap

The key wrap algorithm encrypts a CEK with a KEK in a manner which ensures that every bit of plaintext effects every bit of ciphertext. This makes it equivalent in function to the package transform [PACKAGE] without requiring additional mechanisms or resources such as hash functions or cryptographically strong random numbers. The key wrap algorithm is performed in two phases, a first phase which formats the CEK into a form suitable for encryption by the KEK, and a second phase which wraps the formatted CEK using the KEK.

Key formatting: Create a formatted CEK block consisting of the following:

1. A one-byte count of the number of bytes in the CEK.
2. A check value containing the bitwise complement of the first three bytes of the CEK.
3. The CEK.
4. Enough random padding data to make the CEK data block a multiple of the KEK block length and at least two KEK cipher blocks long (the fact that 32 bits of count+check value are used means that even with a 40-bit CEK, the resulting data size will always be at least two (64-bit) cipher blocks long). The padding data does not have to be cryptographically strong, although unpredictability helps. Note that PKCS #5 padding is not used, since the length of the data is already known.

The formatted CEK block then looks as follows:

CEK byte count || check value || CEK || padding (if required)

Key wrapping:

1. Encrypt the padded key using the KEK.

2. Without resetting the IV (that is, using the last ciphertext block as the IV), encrypt the encrypted padded key a second time.

The resulting double-encrypted data is the EncryptedKey.

### 2.3.2 Key Unwrap

Key unwrapping:

1. Using the n-1'th ciphertext block as the IV, decrypt the n'th ciphertext block.
2. Using the decrypted n'th ciphertext block as the IV, decrypt the 1st ... n-1'th ciphertext blocks. This strips the outer layer of encryption.
3. Decrypt the inner layer of encryption using the KEK.

Key format verification:

- 1a. If the CEK byte count is less than the minimum allowed key size (usually 5 bytes for 40-bit keys) or greater than the wrapped CEK length or not valid for the CEK algorithm (eg not 16 or 24 bytes for triple DES), the KEK was invalid.
- 1b. If the bitwise complement of the key check value doesn't match the first three bytes of the key, the KEK was invalid.

### 2.3.3 Example

Given a content-encryption algorithm of Skipjack and a KEK algorithm of Triple-DES, the wrap steps are as follows:

1. Set the first 4 bytes of the CEK block to the Skipjack key size (10 bytes) and the bitwise complement of the first three bytes of the CEK.
2. Append the 80-bit (10-byte) Skipjack CEK and pad the total to 16 bytes (two triple-DES blocks) using 2 bytes of random data.
2. Using the IV given in the KeyEncryptionAlgorithmIdentifier, encrypted the padded Skipjack key.
3. Without resetting the IV, encrypt the encrypted padded key a second time.

The unwrap steps are as follows:

1. Using the first 8 bytes of the double-encrypted key as the IV, decrypt the second 8 bytes.
2. Without resetting the IV, decrypt the first 8 bytes.
3. Decrypt the inner layer of encryption using the the IV given in the KeyEncryptionAlgorithmIdentifier to recover the padded Skipjack key.
4. If the length byte isn't equal to the Skipjack key size (80 bits or 10 bytes) or the bitwise complement of the check bytes doesn't match the first three bytes of the CEK, the KEK was invalid.

#### 2.3.4 Rationale for the Double Wrapping

If many CEKs are encrypted in a standard way with the same KEK and the KEK has a 64-bit block size then after about  $2^{32}$  encryptions there is a high probability of a collision between different blocks of encrypted CEKs. If an opponent manages to obtain a CEK, they may be able to solve for other CEKs. The double-encryption wrapping process, which makes every bit of ciphertext dependent on every bit of the CEK, eliminates this collision problem (as well as preventing other potential problems such as bit-flipping attacks). Since the IV is applied to the inner layer of encryption, even wrapping the same CEK with the same KEK will result in a completely different wrapped key each time.

An additional feature of the double wrapping is that it doesn't require the use of any extra algorithms such as hash algorithms in addition to the wrapping algorithm itself, allowing it to be implemented in devices which only support one type of encryption algorithm. A typical example of such a device is a crypto token such as a smart card which often only supports a single block cipher and a single public-key algorithm, making it impossible to wrap keys if the use of an additional algorithm were required.

### 3. Test Vectors

This section contains two sets of test vectors, a very basic set for DES which can be used to verify correctness and which uses an algorithm which is freely exportable from the US, and a stress-test version which uses very long passphrase and key sizes and a mixture of algorithms which can be used to verify the behaviour in extreme cases.

The basic test contains two subtests, a known-answer test for the key derivation stage and a full test of the key wrapping. Both tests use a DES-CBC key derived from the password "password" with salt { 12 34 56 78 78 56 34 12 } using 5 iterations of PBKDF2. In the known answer test the IV is set to all zeroes (equivalent to using ECB) and used to encrypt an all-zero data block.

The following values are obtained for the known-answer test:

PKCS #5v2 values:

```
input          70 61 73 73 77 6f 72 64
passphrase:    "password"
input salt:    12 34 56 78 78 56 34 12
iterations:    5
```

```
output key:    D1 DA A7 86 15 F2 87 E6
known answer: 9B BD 78 FC 11 A3 A9 08
```

The following values are obtained when wrapping a 64-bit (parity-adjusted) DES-ECB key:

PKCS #5v2 values:

```
input          70 61 73 73 77 6f 72 64
passphrase:    "password"
input salt:    12 34 56 78 78 56 34 12
iterations:    5
```

```
output key:    D1 DA A7 86 15 F2 87 E6
```

CEK formatting phase:

```
length byte:   08
key check:     73 9D 83
CEK:           8C 62 7C 89 73 23 A2 F8
padding:       C4 36 F5 41
```

```
complete      08 73 9D 83 8C 62 7C 89 73 23 A2 F8 C4 36 F5 41
CEK block:
```

Key wrap phase (wrap CEK block using DES key):

IV: EF E5 98 EF 21 B3 3D 6D

first encr. 06 A0 43 86 1E 82 88 E4 8B 59 9E B9 76 10 00 D4

pass output:

second encr. B8 1B 25 65 EE 37 3C A6 DE DC A2 6A 17 8B 0C 10

pass output:

ASN.1 encoded PasswordRecipientInfo:

```

0 A3 68: [3] {
2 02 1:  INTEGER 0
5 A0 26:  [0] {
7 06 9:  OBJECT IDENTIFIER id-PBKDF2 (1 2 840 113549 1 5 12)
18 30 13: SEQUENCE {
20 04 8:  OCTET STRING
      :    12 34 56 78 78 56 34 12
30 02 1:  INTEGER 5
      :    }
      :  }
34 30 32: SEQUENCE {
36 06 11: OBJECT IDENTIFIER id-alg-PWRI-KEK
      :    (1 2 840 113549 1 9 16 3 9)
33 30 17: SEQUENCE {
35 06 5:  OBJECT IDENTIFIER des-CBC (1 3 14 3 2 7)
42 04 8:  OCTET STRING
      :    EF E5 98 EF 21 B3 3D 6D
      :    }
      :  }
68 04 16: OCTET STRING
      :    B8 1B 25 65 EE 37 3C A6 DE DC A2 6A 17 8B 0C 10
      :  }

```

The following values are obtained when wrapping a 256-bit key (for example one for AES or Blowfish) using a triple DES-CBC key derived from the passphrase "All n-entities must communicate with other n-entities via n-1 entiteeheehees" with salt { 12 34 56 78 78 56 34 12 } using 500 iterations of PBKDF2.

PKCS #5v2 values:

```

input          41 6C 6C 20 6E 2D 65 6E 74 69 74 69 65 73 20 6D
passphrase:    75 73 74 20 63 6F 6D 6D 75 6E 69 63 61 74 65 20
               77 69 74 68 20 6F 74 68 65 72 20 6E 2d 65 6E 74
               69 74 69 65 73 20 76 69 61 20 6E 2D 31 20 65 6E
               74 69 74 65 65 68 65 65 68 65 65 73
               "All n-entities must communicate with other "
               "n-entities via n-1 entiteeheehees"

input
salt:          12 34 56 78 78 56 34 12
iterations:    500

output         6A 89 70 BF 68 C9 2C AE A8 4A 8D F2 85 10 85 86
3DES key:      07 12 63 80 CC 47 AB 2D

```

CEK formatting phase:

```

length byte:   20
key check:     73 9C 82
CEK:           8C 63 7D 88 72 23 A2 F9 65 B5 66 EB 01 4B 0F A5
               D5 23 00 A3 F7 EA 40 FF FC 57 72 03 C7 1B AF 3B
padding:       FA 06 0A 45

complete      20 73 9C 82 8C 63 7D 88 72 23 A2 F9 65 B5 66 EB
CEK block:    01 4B 0F A5 D5 23 00 A3 F7 EA 40 FF FC 57 72 03
               C7 1B AF 3B FA 06 0A 45

```

Key wrap phase (wrap CEK block using 3DES key):

```

IV:           BA F1 CA 79 31 21 3C 4E

first encr.   F8 3F 9E 16 78 51 41 10 64 27 65 A9 F5 D8 71 CD
pass output:  27 DB AA 41 E7 BD 80 48 A9 08 20 FF 40 82 A2 80
               96 9E 65 27 9E 12 6A EB

second encr.  C0 3C 51 4A BD B9 E2 C5 AA C0 38 57 2B 5E 24 55
pass output:  38 76 B3 77 AA FB 82 EC A5 A9 D7 3F 8A B1 43 D9
               EC 74 E6 CA D7 DB 26 0C

```

ASN.1 encoded PasswordRecipientInfo:

```

0 A3 96: [3] {
2 02 1:  INTEGER 0
5 A0 27:  [0] {
7 06 9:  OBJECT IDENTIFIER id-PBKDF2 (1 2 840 113549 1 5 12)
18 30 14:  SEQUENCE {
20 04 8:  OCTET STRING
      :    12 34 56 78 78 56 34 12
30 02 2:  INTEGER 500
      :    }
      :  }
34 30 35:  SEQUENCE {
36 06 11:  OBJECT IDENTIFIER id-alg-PWRI-KEK
      :    (1 2 840 113549 1 9 16 3 9)
34 30 20:  SEQUENCE {
36 06 8:  OBJECT IDENTIFIER des-EDE3-CBC (1 2 840 113549 3 7)
46 04 8:  OCTET STRING
      :    BA F1 CA 79 31 21 3C 4E
      :    }
      :  }
71 04 40:  OCTET STRING
      :    C0 3C 51 4A BD B9 E2 C5 AA C0 38 57 2B 5E 24 55
      :    38 76 B3 77 AA FB 82 EC A5 A9 D7 3F 8A B1 43 D9
      :    EC 74 E6 CA D7 DB 26 0C
      :  }

```

#### 4. Security Considerations

The security of this recipient information type rests on the security of the underlying mechanisms employed, for which further information can be found in RFC 2630 and PKCS5v2. More importantly, however, when used with a password the security of this information type rests on the entropy of the user-selected password, which is typically quite low. Pass phrases (as opposed to simple passwords) are **STRONGLY RECOMMENDED**, although it should be recognized that even with pass phrases it will be difficult to use this recipient information type to derive a KEK with sufficient entropy to properly protect a 128-bit (or higher) CEK.

## 5. IANA Considerations

The PasswordRecipientInfo key encryption algorithms are identified by object identifiers (OIDs). OIDs were assigned from an arc contributed to the S/MIME Working Group by the RSA Security. Should additional encryption algorithms be introduced, the advocates for such algorithms are expected to assign the necessary OIDs from their own arcs. No action by the IANA is necessary for this document or any anticipated updates.

## Acknowledgments

The author would like to thank Jim Schaad, Phil Griffin, and the members of the S/MIME Working Group for their comments and feedback on this document.

## Author Address

Peter Gutmann  
University of Auckland  
Private Bag 92019  
Auckland, New Zealand

EMail: pgut001@cs.auckland.ac.nz

## References

- [ASN1] CCITT Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1), 1988.
- [RFC2119] Bradner, S., "Key Words for Use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2630] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.
- [RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification, Version 2.0", RFC 2898, September 2000.
- [PACKAGE] All-or-Nothing Encryption and the Package Transform, R. Rivest, Proceedings of Fast Software Encryption '97, Haifa, Israel, January 1997.

## Appendix A: ASN.1:1988 Module

PasswordRecipientInfo-88

```
{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
  smime(16) modules(0) pwri(17) }
```

```
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
```

IMPORTS

AlgorithmIdentifier

```
FROM AuthenticationFramework { joint-iso-itu-t ds(5) module(1)
  authenticationFramework(7) 3 }
```

CMSVersion, EncryptedKey

```
FROM CryptographicMessageSyntax { iso(1) member-body(2) us(840)
  rsadsi(113549) pkcs(1) pkcs-9(9)
  smime(16) modules(0) cms(1) };
```

```
-- The following PDU is defined in PKCS5 { iso(1) member-body(2)
-- us(840) rsadsi(113549) pkcs(1) pkcs-5(5) modules(16)
-- pkcs5v2-0(1) }, however it can't be imported because
-- it's specified in 1994/1997 ASN.1. Because of this it's copied
-- here from the source but rephrased as 1988 ASN.1. Further
-- details are given in [RFC 2898].
```

```
PBKDF2-params ::= SEQUENCE {
  salt OCTET STRING,
  iterationCount INTEGER (1..MAX),
  keyLength INTEGER (1..MAX) OPTIONAL,
  prf AlgorithmIdentifier
    DEFAULT { algorithm id-hmacWithSHA1, parameters NULL } }
```

```
-- The PRF algorithm is also defined in PKCS5 and can neither be
-- imported nor expressed in 1988 ASN.1, however it is encoded as
-- an AlgorithmIdentifier with the OID:
```

```
id-hmacWithSHA1 OBJECT IDENTIFIER ::= { iso(1) member-body(2)
  us(840) rsadsi(113549) digestAlgorithm(2) 7 }
```

```
-- and NULL parameters. Further details are given in [RFC 2898].
```

```
-- Implementation note: Because of the inability to precisely
-- specify the PBKDF2 PDU or its parameters in 1988 ASN.1, it is
-- likely that implementors will also encounter alternative
-- interpretations of these parameters, usually using an alternate
-- OID from the IPsec arc which is generally used for HMAC-SHA1:
```

```

--
-- hMAC-SHA1 OBJECT IDENTIFIER ::= { iso(1)
--     identified-organization(3) dod(6) internet(1) security(5)
--     mechanisms(5) 8 1 2 }
--
-- with absent (rather than NULL) parameters.

-- The PasswordRecipientInfo

id-alg-PWRI-KEK OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) alg(3) 9 }

PasswordRecipientInfo ::= SEQUENCE {
    version CMSVersion,          -- Always set to 0
    keyDerivationAlgorithm
        [0] KeyDerivationAlgorithmIdentifier OPTIONAL,
    keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
    encryptedKey EncryptedKey }

KeyDerivationAlgorithmIdentifier ::= AlgorithmIdentifier

KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

END -- PasswordRecipientInfo-88 --

Appendix B: ASN.1:1997 Module

This appendix contains the same information as Appendix A in a more
recent (and precise) ASN.1 notation, however Appendix A takes
precedence in case of conflict.

PasswordRecipientInfo-97
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) pwri(18) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

IMPORTS

    id-PBKDF2, PBKDF2-params,
    FROM PKCS5 { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
        pkcs-5(5) }

    CMSVersion, EncryptedKey, des-ede3-cbc, CBCParameter
    FROM CryptographicMessageSyntax { iso(1) member-body(2) us(840)
        rsadsi(113549) pkcs(1) pkcs-9(9)
        smime(16) modules(0) cms(1) };

```

```
id-alg-PWRI-KEK OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) alg(3) 9 }
```

```
PasswordRecipientInfo ::= SEQUENCE {
    version CMSVersion,          -- Always set to 0
    keyDerivationAlgorithm
        [0] KeyDerivationAlgorithmIdentifier OPTIONAL,
    keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
    encryptedKey            EncryptedKey }
```

```
KeyDerivationAlgorithmIdentifier ::=
    AlgorithmIdentifier {{ KeyDerivationAlgorithms }}
```

```
KeyDerivationAlgorithms ALGORITHM ::= {
    { OID id-PBKDF2 PARMS PBKDF2-params },
    ...
}
```

```
KeyEncryptionAlgorithmIdentifier ::=
    AlgorithmIdentifier {{ KeyEncryptionAlgorithms }}
```

```
KeyEncryptionAlgorithms ALGORITHM ::= {
    { OID id-alg-PWRI-KEK PARMS
        AlgorithmIdentifier {{ PWRIAlgorithms }} },
    ...
}
```

```
-- Algorithm identifiers for algorithms used with the
-- id-alg-PWRI-KEK key wrap algorithm.  Currently only 3DES is a
-- MUST, all others are optional
```

```
PWRIAlgorithms ALGORITHM ::= {
    { OID des-ede3-cbc PARMS CBCParameter },
    ...
}
```

```
-- Supporting definitions.  We could also pull in the
-- AlgorithmIdentifier from an appropriately recent X.500 module (or
-- wherever) but it's just as easy (and more convenient for readers)
-- to provide a definition here
```

```
AlgorithmIdentifier { ALGORITHM:IOSet } ::= SEQUENCE {
    algorithm          ALGORITHM.&id({IOSet}),
    parameters         ALGORITHM.&Type({IOSet}{@algorithm})  OPTIONAL
}
```

```
ALGORITHM ::= CLASS {
    &id                OBJECT IDENTIFIER  UNIQUE,
```

```
&Type          OPTIONAL
}
WITH SYNTAX { OID &id [PARMS &Type] }

END -- PasswordRecipientInfo-97 --
```

## Full Copyright Statement

Copyright (C) The Internet Society (2001). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

