          PKCS #12: Personal Information Exchange Syntax v1.1

Abstract

   PKCS #12 v1.1 describes a transfer syntax for personal identity
   information, including private keys, certificates, miscellaneous
   secrets, and extensions. Machines, applications, browsers, Internet
   kiosks, and so on, that support this standard will allow a user to
   import, export, and exercise a single set of personal identity
   information. This standard supports direct transfer of personal
   information under several privacy and integrity modes.

   This document represents a republication of PKCS #12 v1.1 from RSA
   Laboratories' Public Key Cryptography Standard (PKCS) series. By
   publishing this RFC, change control is transferred to the IETF.

IESG Note

   The IESG thanks RSA Laboratories for transferring change control to
   the IETF. Enhancements to this specification that preserve backward
   compatibility are expected in an upcoming IETF Standards Track
   document.

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 5741.

   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/rfc7292.

Copyright Notice

Table of Contents

1. Introduction

   This document represents a republication of PKCS #12 v1.1 from RSA
   Laboratories' Public Key Cryptography Standard (PKCS) series. By
   publishing this RFC, change control is transferred to the IETF. RSA
   and its parent company EMC reserve the right to continue publishing
   and distributing PKCS #12 v1.1 and its predecessors.

   The body of this document, except for the Security Considerations
   section, is taken directly from the PKCS #12 v1.1 specification. The
   list of references and the in-line cites have been updated or added
   where appropriate to cite the most current documents in addition to
   those current at the original publication of PKCS #12 v1.1.

   This standard describes a transfer syntax for personal identity
   information, including private keys, certificates, miscellaneous
   secrets, and extensions. Machines, applications, browsers, Internet
   kiosks, and so on, that support this standard will allow a user to
   import, export, and exercise a single set of personal identity
   information.

   This standard supports direct transfer of personal information under
   several privacy and integrity modes. The most secure of the privacy
   and integrity modes require the source and destination platforms to
   have trusted public/private key pairs usable for digital signatures
   and encryption, respectively. The standard also supports lower-
   security, password-based privacy and integrity modes for those cases
   where trusted public/private key pairs are not available.

   This standard should be amenable to both software and hardware
   implementations. Hardware implementations offer physical security in
   tamper-resistant tokens such as smart cards and Personal Computer
   Memory Card International Association (PCMCIA) devices.

   This standard can be viewed as building on PKCS #8 [15] [24] by
   including essential but ancillary identity information along with
   private keys and by instituting higher security through public-key
   privacy and integrity modes.

1.1. Changes from PKCS #12 Version 1

   This document transfers PKCS #12 [16] into the IETF and includes some
   minor changes from the authors for this submission.

   o Addition of hash algorithms.

   o Incorporation of Technical Corrigendum #1, which makes some minor
      corrections to the ASN.1 syntax.

   o Removed (from the ASN.1 syntax) 1024 as an example of the
      iteration count.

   o Addition of a recommendation that the technique in Appendix B no
      longer be used for a specific mode (password privacy mode) and
      that techniques from PKCS#5 v2.1 be used instead.

   o Addition of comments and minor corrections to the ASN.1 module in
      Appendix C.

   o Removal of the export regulations discussion in the former
      Appendix D.

   o Replacement of RSA with EMC in the "Intellectual Property
      Considerations".

   o Many changes and additions to the references.

   o A reference was added to NIST SP 800-132 for its recommendations
      on selection of the iteration count value for password integrity
      (part of dictionary-attack resistance).

   o Comment included on acronym expansion of PFX: The acronym is
      sometimes expanded as Personal Information Exchange.

   o In Appendix B, the phrase "no longer recommended" was changed to
      "not recommended" in the following sentence to address a question
      and make it clear the method was not recommended: "Note that this
      method for password privacy mode is no longer recommended."

2. Definitions and Notation

   AlgorithmIdentifier: An ASN.1 type that identifies an algorithm (by
      an object identifier) and any associated parameters. This type is
      defined in [8].

   ASN.1: Abstract Syntax Notation One, as defined in [2], [3], [4],
      and [5].

   Attribute: An ASN.1 type that identifies an attribute type (by an
      object identifier) and an associated attribute value. The ASN.1
      type Attribute is defined in [7].

   Certificate: A digitally signed data unit binding a public key to
      identity information. A specific format for identity certificates
      is defined in [8]. Another format is described in [17].

Certificate Revocation List (CRL): A digitally signed list of
   certificates that should no longer be honored, having been revoked
   by the issuers or a higher authority. One format for CRLs is
   defined in [8].

ContentInfo: An ASN.1 type used to hold data that may have been
   cryptographically protected. This type is defined in [21] and
   [14].

DER: Distinguished Encoding Rules, as defined in [6].

Destination platform: The ultimate, final target platform for the
   personal information originating from the source platform. Even
   though certain information may be transported from the destination
   platform to the source platform, the ultimate target for personal
   information is always called the destination platform.

DigestInfo: An ASN.1 type used to hold a message digest. This type
   is defined in [21] and [14].

Encryption Key Pair (DestEncK): A public/private key pair used for
   the public-key privacy mode of this standard. The public half is
   called PDestEncK (TPDestEncK when emphasizing that the public key
   is "trusted"), and the private half is called VDestEncK.

Export time: The time that a user reads personal information from a
   source platform and transforms the information into an
   interoperable, secure Protocol Data Unit (PDU).

Import time: The time that a user writes personal information from a
   Safe PDU to a destination platform.

Message Authentication Code (MAC): A type of collision-resistant,
   "unpredictable" function of a message and a secret key. MACs are
   used for data authentication and are akin to secret-key digital
   signatures in many respects.

Object Identifier: A sequence of integers that uniquely identifies
   an associated data object in a global name space administrated by
   a hierarchy of naming authorities. This is a primitive data type
   in ASN.1.

PFX: The top-level exchange PDU defined in this standard. The
   acronym is sometimes expanded as Personal Information Exchange.

   Platform: A combination of machine, operating system, and
      applications software within which the user exercises personal
      identity. An application, in this context, is software that uses
      personal information. Two platforms differ if their machine types
      differ or if their applications software differs. There is at
      least one platform per user in multi-user systems.

   Protocol Data Unit (PDU): A sequence of bits in machine-independent
      format constituting a message in a protocol.

   Shrouding: Encryption as applied to private keys, possibly in
      concert with a policy that prevents the plaintext of the key from
      ever being visible beyond a certain, well-defined interface.

   Signature Key Pair (SrcSigK): A platform-specific signature key pair
      used for the public-key integrity mode of this standard. The
      public half is called PSrcSigK (TPSrcSigK when emphasizing that
      the public key is "trusted"), and the private half is called
      VSrcSigK.

   Source platform: The origin platform of the personal information
      ultimately intended for the destination platform. Even though
      certain information may be transported from the destination
      platform to the source platform, the platform that is the origin
      of personal information is always called the source platform.

3. Overview

3.1. Exchange Modes

   There are four combinations of privacy modes and integrity modes.
   The privacy modes use encryption to protect personal information from
   exposure, and the integrity modes protect personal information from
   tampering. Without protection from tampering, an adversary could
   conceivably substitute invalid information for the user's personal
   information without the user being aware of the substitution.

   The following are the privacy modes:

   o Public-key privacy mode: Personal information is enveloped on the
      source platform using a trusted encryption public key of a known
      destination platform (see Section 3.3). The envelope is opened
      with the corresponding private key.

o Password privacy mode: Personal information is encrypted with a
   symmetric key derived from a user name and a privacy password, as
   in [22] and [13]. If password integrity mode is used as well, the
   privacy password and the integrity password may or may not be the
   same.

The following are the integrity modes:

o Public-key integrity mode: Integrity is guaranteed through a
   digital signature on the contents of the PFX PDU, which is
   produced using the source platform's private signature key. The
   signature is verified on the destination platform by using the
   corresponding public key (see Section 3.4).

o Password integrity mode: Integrity is guaranteed through a Message
   Authentication Code (MAC) derived from a secret integrity
   password. If password privacy mode is used as well, the privacy
   password and the integrity password may or may not be the same.

3.2. Mode Choice Policies

All combinations of the privacy and integrity modes are permitted in
this standard. Of course, good security policy suggests that certain
practices be avoided, e.g., it can be unwise to transport private
keys without physical protection when using password privacy mode or
when using public-key privacy mode with weak symmetric encryption.

In general, the public-key modes for both privacy and integrity are
preferable to the password modes (from a security viewpoint).
However, it is not always possible to use the public-key modes. For
example, it may not be known at export time what the destination
platform is; if this is the case, then the use of the public-key
privacy mode is precluded.

3.3. Trusted Public Keys

Asymmetric key pairs may be used in this standard in two ways:
public-key privacy mode and public-key integrity mode. For public-
key privacy mode, an encryption key pair is required; for public-key
integrity mode, a signature key pair is required.

It may be appropriate for the keys discussed in this section to be
platform-specific keys dedicated solely for the purpose of
transporting a user's personal information. Whether or not that is
the case, though, the keys discussed here should not be confused with
the user's personal keys that the user wishes to transport from one
platform to another. (These latter keys are stored within the PDU.)

For public-key privacy mode, the private key from the encryption key
pair is kept on the destination platform, where it is ultimately used
to open a private envelope. The corresponding trusted public key is
called TPDestEncK.

For public-key integrity mode, the private key from the signature
pair is kept on the source platform, where it is used to sign
personal information. The corresponding trusted public key is called
TPSrcSigK.

For both uses of public/private key pairs, the public key from the
key pair must be transported to the other platform such that it is
trusted to have originated at the correct platform. Judging whether
or not a public key is trusted in this sense must ultimately be left
to the user. There are a variety of methods for ensuring that a
public key is trusted.

The processes of imbuing keys with trust and of verifying
trustworthiness of keys are not discussed further in this document.
Whenever asymmetric keys are discussed in what follows, the public
keys are assumed to be trusted.

3.4. The AuthenticatedSafe

Each compliant platform shall be able to import and export
AuthenticatedSafe PDUs wrapped in PFX PDUs.

For integrity, the AuthenticatedSafe is either signed (if public-key
integrity mode is used) or MACed (if password integrity mode is used)
to produce a PFX PDU. If the AuthenticatedSafe is signed, then it is
accompanied by a digital signature, which was produced on the source
platform with a private signature key, VSrcSigK, corresponding to a
trusted public signature key, TPSrcSigK. TPSrcSigK must accompany
the PFX to the destination platform, where the user can verify the
trust in the key and can verify the signature on the
AuthenticatedSafe. If the AuthenticatedSafe is MACed, then it is
accompanied by a MAC computed from a secret integrity password, salt
bits, an iteration count, and the contents of the AuthenticatedSafe.

The AuthenticatedSafe itself consists of a sequence of ContentInfo
values, some of which may consist of plaintext (data), and others
that may either be enveloped (if public-key privacy mode is used) or
encrypted (if password privacy mode is used). If the contents are
enveloped, then they are encrypted with a symmetric cipher under a
freshly generated key, which is in turn encrypted with RSA asymmetric
encryption. The RSA public key used to encrypt the symmetric key is
called TPDestEncK and corresponds to an RSA private key, VDestEncK,
on the destination platform. TPDestEncK needs to be trusted by the

user when it is used at export time. If the contents are encrypted,
then they are encrypted with a symmetric cipher under a key derived
from a secret privacy password, salt bits, and an iteration counter.

Each ContentInfo contains an arbitrary collection of private keys,
PKCS #8-shrouded private keys, certificates, CRLs, or opaque data
objects, at the user's discretion, stored in values of type
SafeContents.

The raison d'etre for the unencrypted option is that some governments
restrict certain uses of cryptography. Having several parts in an
AuthenticatedSafe keeps implementers' options open. For example, it
may be the case that strong cryptography can be used to make PKCS
#8-shrouded keys, but then these shrouded keys should not be further
encrypted, because super-encryption can limit a product's
exportability. The multi-part AuthenticatedSafe design permits this
possibility.

Around the AuthenticatedSafe is the integrity-mode wrapper, which
protects the entire contents of the AuthenticatedSafe (including
unencrypted parts, if they are present). This is the reverse of the
wrapping order in many protocols, in which privacy is the outermost
protection. This latter, more-common wrapping order avoids
signatures on encrypted data, which are undesirable under certain
circumstances; however, these circumstances do not apply to this
document, and it is therefore preferable to protect the integrity of
as much information as possible.

4. PFX PDU Syntax

This format corresponds to the data model presented above, with
wrappers for privacy and integrity. This section makes free
reference to PKCS #7 [14] [21] and assumes the reader is familiar
with terms defined in that document.

All modes of direct exchange use the same PDU format. ASN.1 and BER-
encoding ensure platform independence.

This standard has one ASN.1 export: PFX. This is the outer integrity
wrapper. Instances of PFX contain:

1. A version indicator. The version shall be v3 for this version of
    this document.

2. A PKCS #7 ContentInfo, whose contentType is signedData in public-
    key integrity mode and data in password integrity mode.

3. An optional instance of MacData, present only in password
    integrity. This object, if present, contains a PKCS #7
    DigestInfo, which holds the MAC value, a macSalt, and an
    iterationCount. As described in Appendix B, the MAC key is
    derived from the password, the macSalt, and the iterationCount;
    as described in Section 5, the MAC is computed from the authSafe
    value and the MAC key via HMAC [11] [20]. The password and the
    MAC key are not actually present anywhere in the PFX. The salt
    and (to a certain extent) the iteration count thwarts dictionary
    attacks against the integrity password. See NIST Special
    Publication 800-132 [12] about how to choose a reasonable value
    for the iteration count.

```
PFX ::= SEQUENCE {
    version     INTEGER {v3(3)}(v3,...),
    authSafe    ContentInfo,
    macData     MacData OPTIONAL
}

MacData ::= SEQUENCE {
    mac         DigestInfo,
    macSalt     OCTET STRING,
    iterations  INTEGER DEFAULT 1
    -- Note: The default is for historical reasons and its
    --       use is deprecated.
}
```

4.1. The AuthenticatedSafe Type

   As mentioned, the contentType field of authSafe shall be of type data
   or signedData. The content field of the authSafe shall, either
   directly (data case) or indirectly (signedData case), contain a BER-
   encoded value of type AuthenticatedSafe.

```
AuthenticatedSafe ::= SEQUENCE OF ContentInfo
    -- Data if unencrypted
    -- EncryptedData if password-encrypted
    -- EnvelopedData if public key-encrypted
```

   An AuthenticatedSafe contains a sequence of ContentInfo values. The
   content field of these ContentInfo values contains either plaintext,
   encrypted, or enveloped data. In the case of encrypted or enveloped
   data, the plaintext of the data holds the BER-encoding of an instance
   of SafeContents. Section 5.1 of this document describes the
   construction of values of type AuthenticatedSafe in more detail.

4.2. The SafeBag Type

   The SafeContents type is made up of SafeBags. Each SafeBag holds one
   piece of information -- a key, a certificate, etc. -- which is
   identified by an object identifier.

 SafeContents ::= SEQUENCE OF SafeBag

 SafeBag ::= SEQUENCE {
     bagId   BAG-TYPE.&id ({PKCS12BagSet})
     bagValue   [0] EXPLICIT BAG-TYPE.&Type({PKCS12BagSet}{@bagId}),
     bagAttributes SET OF PKCS12Attribute OPTIONAL
 }

 PKCS12Attribute ::= SEQUENCE {
     attrId   ATTRIBUTE.&id ({PKCS12AttrSet}),
     attrValues SET OF ATTRIBUTE.&Type ({PKCS12AttrSet}{@attrId})
 } -- This type is compatible with the X.500 type 'Attribute'

 PKCS12AttrSet ATTRIBUTE ::= {
     friendlyName | -- from PKCS #9 [23]
     localKeyId,   -- from PKCS #9
     ... -- Other attributes are allowed
 }

   The optional bagAttributes field allows users to assign nicknames and
   identifiers to keys, etc., and permits visual tools to display
   meaningful strings of some sort to the user.

   Six types of SafeBags are defined in this version of this document:

   bagtypes OBJECT IDENTIFIER ::= {pkcs-12 10 1}

   BAG-TYPE ::= TYPE-IDENTIFIER

   keyBag BAG-TYPE ::=
       {KeyBag IDENTIFIED BY {bagtypes 1}}
   pkcs8ShroudedKeyBag BAG-TYPE ::=
       {PKCS8ShroudedKeyBag IDENTIFIED BY {bagtypes 2}}
   certBag BAG-TYPE ::=
       {CertBag IDENTIFIED BY {bagtypes 3}}
   crlBag BAG-TYPE ::=
       {CRLBag IDENTIFIED BY {bagtypes 4}}
   secretBag BAG-TYPE ::=
       {SecretBag IDENTIFIED BY {bagtypes 5}}
   safeContentsBag BAG-TYPE ::=
       {SafeContents IDENTIFIED BY {bagtypes 6}}

```
   PKCS12BagSet BAG-TYPE ::= {
       keyBag |
       pkcs8ShroudedKeyBag |
       certBag |
       crlBag |
       secretBag |
       safeContentsBag,
       ... -- For future extensions
   }
```

   As new bag types become recognized in future versions of this
   standard, the PKCS12BagSet may be extended.

4.2.1. The KeyBag Type

   A KeyBag is a PKCS #8 PrivateKeyInfo. Note that a KeyBag contains
   only one private key.

```
   KeyBag ::= PrivateKeyInfo
```

4.2.2. The PKCS8ShroudedKeyBag Type

   A PKCS8ShroudedKeyBag holds a private key, which has been shrouded in
   accordance with PKCS #8. Note that a PKCS8ShroudedKeyBag holds only
   one shrouded private key.

```
   PKCS8ShroudedKeyBag ::= EncryptedPrivateKeyInfo
```

4.2.3. The CertBag Type

   A CertBag contains a certificate of a certain type. Object
   identifiers are used to distinguish between different certificate
   types.

```
   CertBag ::= SEQUENCE {
       certId      BAG-TYPE.&id   ({CertTypes}),
       certValue   [0] EXPLICIT BAG-TYPE.&Type ({CertTypes}{@certId})
   }

   x509Certificate BAG-TYPE ::=
       {OCTET STRING IDENTIFIED BY {certTypes 1}}
       -- DER-encoded X.509 certificate stored in OCTET STRING
   sdsiCertificate BAG-TYPE ::=
       {IA5String IDENTIFIED BY {certTypes 2}}
       -- Base64-encoded SDSI certificate stored in IA5String

   CertTypes BAG-TYPE ::= {
       x509Certificate |
```

```
        sdsiCertificate,
        ... -- For future extensions
    }
```

4.2.4. The CRLBag Type

   A CRLBag contains a Certificate Revocation List (CRL) of a certain
   type. Object identifiers are used to distinguish between different
   CRL types.

```
    CRLBag ::= SEQUENCE {
        crlId      BAG-TYPE.&id  ({CRLTypes}),
        crlValue  [0] EXPLICIT BAG-TYPE.&Type ({CRLTypes}{@crlId})
    }

    x509CRL BAG-TYPE ::=
        {OCTET STRING IDENTIFIED BY {crlTypes 1}}
        -- DER-encoded X.509 CRL stored in OCTET STRING

    CRLTypes BAG-TYPE ::= {
        x509CRL,
        ... -- For future extensions
    }
```

4.2.5. The SecretBag Type

   Each of the user's miscellaneous personal secrets is contained in an
   instance of SecretBag, which holds an object identifier-dependent
   value. Note that a SecretBag contains only one secret.

```
    SecretBag ::= SEQUENCE {
        secretTypeId   BAG-TYPE.&id ({SecretTypes}),
        secretValue    [0] EXPLICIT BAG-TYPE.&Type ({SecretTypes}
                           {@secretTypeId})
    }

    SecretTypes BAG-TYPE ::= {
        ... -- For future extensions
    }
```

   Implementers can add values to this set at their own discretion.

4.2.6. The SafeContents Type

   The sixth type of bag that can be held in a SafeBag is a
   SafeContents. This recursive structure allows for arbitrary nesting
   of multiple KeyBags, PKCS8ShroudedKeyBags, CertBags, CRLBags, and
   SecretBags within the top-level SafeContents.

5. Using PFX PDUs

   This section describes the creation and usage of PFX PDUs.

5.1. Creating PFX PDUs

   The steps for creating PFX PDUs are as follows.

   1. It is somewhat clear from the ASN.1 how to make a number of
       instances of SafeContents, each containing a number of (possibly
       nested) instances of SafeBag. Let us assume, therefore, a number
       of instances SC_1, SC_2,..., SC_n of SafeContents. Note that
       there can be a more or less arbitrary number of instances of
       SafeContents in a PFX PDU. As will be seen in step 2, each
       instance can be encrypted (or not) separately.

   2. For each SCI, depending on the chosen encryption option,

       A. If SC_i is not to be encrypted, make a ContentInfo CI_i
           holding content type Data. The contents of the Data OCTET
           STRING shall be a BER-encoding of SC_i (including tag,
           length, and value octets).

       B. If SC_i is to be encrypted with a password, make a
           ContentInfo CI_i of type EncryptedData. The
           encryptedContentInfo field of CI_i has its contentType field
           set to data and its encryptedContent field set to the
           encryption of the BER-encoding of SC_i (note that the tag and
           length octets shall be present).

       C. If SC_i is to be encrypted with a public key, make a
           ContentInfo CI_i of type EnvelopedData in essentially the
           same fashion as the EncryptedData ContentInfo was made in B.

   3. Make an instance of AuthenticatedSafe by stringing together the
       CI_i's in a SEQUENCE.

   4. Make a ContentInfo T holding content type Data. The contents of
       the Data OCTET STRING shall be a BER-encoding of the
       AuthenticatedSafe value (including tag, length, and value
       octets).

   5. For integrity protection,

       A. If the PFX PDU is to be authenticated with a digital
           signature, make a ContentInfo C of type SignedData. The
           contentInfo field of the SignedData in C has T in it. C is
           the ContentInfo in the top-level PFX structure.

        B. If the PFX PDU is to be authenticated with HMAC, then an HMAC
            with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224,
            or SHA-512/256 is computed on the contents of the Data in T
            (i.e., excluding the OCTET STRING tag and length bytes).
            This is exactly what would be initially digested in step 5A
            if public-key authentication were being used.

5.2. Importing Keys, etc., from a PFX PDU

   Importation from a PFX is accomplished essentially by reversing the
   procedure for creating a PFX. In general, when an application
   imports keys, etc., from a PFX, it should ignore any object
   identifiers that it is not familiar with. At times, it may be
   appropriate to alert the user to the presence of such object
   identifiers.

   Special care may be taken by the application when importing an item
   in the PFX would require overwriting an item that already exists
   locally. The behavior of the application when such an item is
   encountered may depend on what the item is (i.e., it may be that a
   PKCS #8-shrouded private key and a CRL should be treated differently
   here). Appropriate behavior may be to ask the user what action
   should be taken for this item.

6. Security Considerations

   When using passwords in privacy or integrity mode, it needs to be
   considered that password-based cryptography is generally limited in
   the security that it can provide, particularly for methods such as
   those defined in this document where off-line password search is
   possible. While the use of salt and iteration count can increase the
   complexity of attack, it is essential that passwords are selected
   well and that relevant guidelines (e.g., NIST SP 800-61-1) are taken
   into account. It is also important that passwords be protected well
   if stored.

   When choosing a salt value in password privacy or integrity mode, the
   recommendations in Section 4 of PKCS #5 2.1 [13] [22] should be taken
   into account. Ideally, the salt is as long as the output of the hash
   function being used and consists of randomly generated data.

7. Normative References

   [1]   Dobbertin, H., "The status of MD5 after a recent attack.",
           CryptoBytes Vol. 2, #2, 1996.

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

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

   [4]   ISO/IEC, "Information technology -- Abstract Syntax Notation
           One (ASN.1) -- Constraint specification", ISO/IEC 88247-3:2008,
           2008.

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

   [6]   ISO/IEC, "Information Technology - ASN.1 Encoding Rules:
           Specification of Basic Encoding Rules (BER), Canonical Encoding
           Rules (CER), and Distinguished Encoding Rules", ISO/IEC
           8825-1:2008, 2008.

   [7]   ISO/IEC, "Information technology -- Open Systems
           Interconnection -- The Directory: Models", ISO/IEC 9594-2:1997,
           1997.

   [8]   ISO/IEC, "Information technology -- Open Systems
           Interconnection -- The Directory: Authentication Framework",
           ISO/IEC 9594-8:1997, 1997.

   [9]   Microsoft, "PFX: Personal Exchange Syntax and Protocol
           Standard", ISO/IEC Version 0.020, January 1997.

   [10]   National Institute of Standards and Technology (NIST), "Secure
           Hash Standard", FIPS Publication 180-4, March 2012.

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

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

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

   [14] RSA Laboratories, "PKCS #7: Cryptographic Message Syntax
         Standard", PKCS Version 1.5, November 1993.

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

   [16] RSA Laboratories, "PKCS #12: Personal Information Exchange
         Syntax", PKCS Version 1.1, December 2012.

   [17] Rivest, R. and B. Lampson, "SDSI - A Simple Distributed
         Security Infrastructure", 1996,
         <http://people.csail.mit.edu/rivest/sdsi10.html>.

   [18] Turner, S. and L. Chen, "MD2 to Historic Status", RFC 6149,
         March 2011.

   [19] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April
         1992.

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

   [21] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version
         1.5", RFC 2315, March 1998.

   [22] Kaliski, B., "PKCS #5: Password-Based Cryptography
         Specification Version 2.0", RFC 2898, September 2000.

   [23] Nystrom, M. and B. Kaliski, "PKCS #9: Selected Object Classes
         and Attribute Types Version 2.0", RFC 2985, November 2000.

   [24] Turner, S., "Asymmetric Key Packages", RFC 5958, August 2010.

   [25] Turner, S. and L. Chen, "Updated Security Considerations for
         the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151,
         March 2011.

Appendix A. Message Authentication Codes (MACs)

   A MAC is a special type of function of a message (data bits) and an
   integrity key. It can be computed or checked only by someone
   possessing both the message and the integrity key. Its security
   follows from the secrecy of the integrity key. In this standard,
   MACing is used in password integrity mode.

   This document uses a particular type of MAC called HMAC [11] [20],
   which can be constructed from any of a variety of hash functions.
   Note that the specifications in [20] and [11] differ somewhat from
   the specification in [9]. The hash function HMAC is based on is
   identified in the MacData, which holds the MAC; for this version of
   this standard, the hash function can be one of the following: SHA-1,
   SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, or SHA-512/256 [10].
   As indicated in Appendix B.4, this structure implies that the same
   hash algorithm must be used to derive the MAC key itself in password
   integrity mode and that the MAC key has either 160, 224, 256, 384, or
   512 bits.

   When password integrity mode is used to secure a PFX PDU, an HMAC
   with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, or
   SHA-512/256 is computed on the BER-encoding of the contents of the
   content field of the authSafe field in the PFX PDU (see Section 5.1).

Appendix B. Deriving Keys and IVs from Passwords and Salt

   Note that this method for password privacy mode is not recommended
   and is deprecated for new usage. The procedures and algorithms
   defined in PKCS #5 v2.1 [13] [22] should be used instead.
   Specifically, PBES2 should be used as encryption scheme, with PBKDF2
   as the key derivation function.

   The method presented here is still used to generate the key in
   password integrity mode.

   We present here a general method for using a hash function to produce
   various types of pseudorandom bits from a password and a string of
   salt bits. This method is used for password privacy mode and
   password integrity mode in the present standard.

B.1. Password Formatting

   The underlying password-based encryption methods in PKCS #5 v2.1 view
   passwords (and salt) as being simple byte strings. The underlying
   password-based encryption methods and the underlying password-based
   authentication methods in this version of this document are similar.

What's left unspecified in the above paragraph is precisely where the
byte string representing a password comes from. (This is not an
issue with salt strings, since they are supplied as a password-based
encryption (or authentication) parameter.) PKCS #5 v2.1 says: "[...]
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
are two possibilities."

In this specification, however, all passwords are created from
BMPStrings with a NULL terminator. This means that each character in
the original BMPString is encoded in 2 bytes in big-endian format
(most-significant byte first). There are no Unicode byte order
marks. The 2 bytes produced from the last character in the BMPString
are followed by 2 additional bytes with the value 0x00.

To illustrate with a simple example, if a user enters the 6-character
password "Beavis", the string that PKCS #12 implementations should
treat as the password is the following string of 14 bytes:

0x00 0x42 0x00 0x65 0x00 0x61 0x00 0x76 0x00 0x69 0x00 0x73 0x00 0x00

B.2. General Method

Let H be a hash function built around a compression function f:

   $Z_2^u \times Z_2^v \rightarrow Z_2^u$

(that is, H has a chaining variable and output of length u bits, and
the message input to the compression function of H is v bits). The
values for u and v are as follows:

```
        HASH FUNCTION       VALUE u          VALUE v
         MD2, MD5            128              512
          SHA-1             160              512
          SHA-224           224              512
          SHA-256           256              512
          SHA-384           384              1024
          SHA-512           512              1024
        SHA-512/224         224              1024
        SHA-512/256         256              1024
```

Furthermore, let r be the iteration count.

We assume here that u and v are both multiples of 8, as are the
lengths of the password and salt strings (which we denote by p and s,
respectively) and the number n of pseudorandom bits required. In
addition, u and v are of course non-zero.

For information on security considerations for MD5 [19], see [25] and
[1], and on those for MD2, see [18].

The following procedure can be used to produce pseudorandom bits for
a particular "purpose" that is identified by a byte called "ID". The
meaning of this ID byte will be discussed later.

1. Construct a string, D (the "diversifier"), by concatenating v/8
    copies of ID.

2. Concatenate copies of the salt together to create a string S of
    length v(ceiling(s/v)) bits (the final copy of the salt may be
    truncated to create S). Note that if the salt is the empty
    string, then so is S.

3. Concatenate copies of the password together to create a string P
    of length v(ceiling(p/v)) bits (the final copy of the password
    may be truncated to create P). Note that if the password is the
    empty string, then so is P.

4. Set I=S||P to be the concatenation of S and P.

5. Set c=ceiling(n/u).

6. For i=1, 2, ..., c, do the following:

    A. Set A2=H^r(D||I). (i.e., the r-th hash of D||1,
        H(H(H(... H(D||I))))

    B. Concatenate copies of Ai to create a string B of length v
        bits (the final copy of Ai may be truncated to create B).

    C. Treating I as a concatenation I_0, I_1, ..., I_(k-1) of v-bit
        blocks, where k=ceiling(s/v)+ceiling(p/v), modify I by
        setting I_j=(I_j+B+1) mod 2^v for each j.

7. Concatenate A_1, A_2, ..., A_c together to form a pseudorandom
    bit string, A.

8. Use the first n bits of A as the output of this entire process.

If the above process is being used to generate a DES key, the process
should be used to create 64 random bits, and the key's parity bits
should be set after the 64 bits have been produced. Similar concerns
hold for 2-key and 3-key triple-DES keys, for CDMF keys, and for any
similar keys with parity bits "built into them".

B.3. More on the ID Byte

   This standard specifies 3 different values for the ID byte mentioned
   above:

   1. If ID=1, then the pseudorandom bits being produced are to be used
       as key material for performing encryption or decryption.

   2. If ID=2, then the pseudorandom bits being produced are to be used
       as an IV (Initial Value) for encryption or decryption.

   3. If ID=3, then the pseudorandom bits being produced are to be used
       as an integrity key for MACing.

B.4. Keys for Password Integrity Mode

   When password integrity mode is used to protect a PFX PDU, a password
   and salt are used to derive a MAC key. As with password privacy
   mode, the password is a Unicode string, and the salt is a byte
   string. No particular lengths are prescribed in this standard for
   either the password or the salt, but the general advice about
   passwords and salt that is given in Appendix C applies here, as well.

   The hash function used to derive MAC keys is whatever hash function
   is going to be used for MACing. The MAC keys that are derived have
   the same length as the hash function's output. In this version of
   this standard, SHA-1, SHA-224, SHA-256, SHA384, SHA-512, SHA-512/224,
   or SHA/512/256 can be used to perform MACing, and so the MAC keys can
   be 160, 224, 256, 384, or 512 bits. See Appendix A for more
   information on MACing.

Appendix C. Keys and IVs for Password Privacy Mode

   As stated at the start of Appendix B, use of this method for password
   privacy mode is not recommended; this specification of keys and IVs
   for password privacy mode is retained for backwards compatibility
   with PKCS #12 v1.0 only.

   When password privacy mode is used to encrypt a PFX PDU, a password
   (typically entered by the user), a salt and an iteration parameter
   are used to derive a key (and an IV, if necessary). The password is

a Unicode string, and as such, each character in it is represented by
2 bytes. The salt is a byte string and so can be represented
directly as a sequence of bytes.

This standard does not prescribe a length for the password. As
usual, however, too short a password might compromise privacy. A
particular application might well require a user-entered privacy
password for creating a PFX PDU to have a password exceeding some
specific length.

This standard does not prescribe a length for the salt either.
Ideally, the salt is as long as the output of the hash function being
used and consists of completely random bits.

The iteration count is recommended to be 1024 or more. (See [22] and
[13] for more information.)

The PBES1 encryption scheme defined in PKCS #5 provides a number of
algorithm identifiers for deriving keys and IVs; here, we specify a
few more, all of which use the procedure detailed in Appendices B.2
and B.3 to construct keys (and IVs, where needed). As is implied by
their names, all of the object identifiers below use the hash
function SHA-1.

```
pkcs-12PbeIds                      OBJECT IDENTIFIER ::= {pkcs-12 1}
pbeWithSHAAnd128BitRC4             OBJECT IDENTIFIER ::= {pkcs-12PbeIds 1}
pbeWithSHAAnd40BitRC4              OBJECT IDENTIFIER ::= {pkcs-12PbeIds 2}
pbeWithSHAAnd3-KeyTripleDES-CBC    OBJECT IDENTIFIER ::= {pkcs-12PbeIds 3}
pbeWithSHAAnd2-KeyTripleDES-CBC    OBJECT IDENTIFIER ::= {pkcs-12PbeIds 4}
pbeWithSHAAnd128BitRC2-CBC         OBJECT IDENTIFIER ::= {pkcs-12PbeIds 5}
pbewithSHAAnd40BitRC2-CBC          OBJECT IDENTIFIER ::= {pkcs-12PbeIds 6}
```

Each of the six PBE object identifiers above has the following ASN.1
type for parameters:

```
pkcs-12PbeParams ::= SEQUENCE {
    salt        OCTET STRING,
    iterations  INTEGER
}
```

The pkcs-12PbeParams holds the salt that is used to generate the key
(and IV, if necessary) and the number of iterations to carry out.

Note that the first two algorithm identifiers above (the algorithm
identifiers for RC4) only derive keys; it is unnecessary to derive an
IV for RC4.

   This section is here for two reasons: first, to enable backwards
   compatibility as described in the first paragraph of this section;
   second, because it is still used in password integrity mode. In
   order to not use it in password integrity mode, the ASN.1 definitions
   require updates. This document recommends that future definitions of
   the PFX structure replace the existing MacData object, optionally
   present in password integrity mode, with a new object definition that
   holds a MAC based on PKCS#5 [13] [22] PBMAC1 message authentication
   scheme. This change would simplify the requirements for key
   derivation functions used across all parts of the PFX structure.

Appendix D. ASN.1 Module

   This appendix documents all ASN.1 types, values, and object sets
   defined in this specification. It does so by providing an ASN.1
   module called PKCS-12.

 PKCS-12 {
     iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-12(12)
     modules(0) pkcs-12(1)}

 -- PKCS #12 v1.1 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

 DEFINITIONS IMPLICIT TAGS ::=

 BEGIN

 -- EXPORTS ALL
 -- All types and values defined in this module are exported for use
 -- in other ASN.1 modules.

 IMPORTS

 informationFramework
     FROM UsefulDefinitions {joint-iso-itu-t(2) ds(5) module(1)
                             usefulDefinitions(0) 3}

 ATTRIBUTE
     FROM InformationFramework informationFramework

 ContentInfo, DigestInfo
     FROM PKCS-7 {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
                  pkcs-7(7) modules(0) pkcs-7(1)}

```
PrivateKeyInfo, EncryptedPrivateKeyInfo
    FROM PKCS-8 {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
                 pkcs-8(8) modules(1) pkcs-8(1)}

pkcs-9, friendlyName, localKeyId, certTypes, crlTypes
    FROM PKCS-9 {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
                 pkcs-9(9) modules(0) pkcs-9(1)};

-- ===========================
-- Object identifiers
-- ===========================


rsadsi  OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840)
                               rsadsi(113549)}
pkcs    OBJECT IDENTIFIER ::= {rsadsi pkcs(1)}
pkcs-12 OBJECT IDENTIFIER ::= {pkcs 12}
pkcs-12PbeIds OBJECT IDENTIFIER ::= {pkcs-12 1}
pbeWithSHAAnd128BitRC4           OBJECT IDENTIFIER ::= {pkcs-12PbeIds 1}
pbeWithSHAAnd40BitRC4            OBJECT IDENTIFIER ::= {pkcs-12PbeIds 2}
pbeWithSHAAnd3-KeyTripleDES-CBC OBJECT IDENTIFIER ::= {pkcs-12PbeIds 3}
pbeWithSHAAnd2-KeyTripleDES-CBC OBJECT IDENTIFIER ::= {pkcs-12PbeIds 4}
pbeWithSHAAnd128BitRC2-CBC       OBJECT IDENTIFIER ::= {pkcs-12PbeIds 5}
pbewithSHAAnd40BitRC2-CBC        OBJECT IDENTIFIER ::= {pkcs-12PbeIds 6}

bagtypes OBJECT IDENTIFIER ::= {pkcs-12 10 1}

-- ===========================
-- The PFX PDU
-- ===========================

PFX ::= SEQUENCE {
    version    INTEGER {v3(3)}(v3,...),
    authSafe   ContentInfo,
    macData    MacData OPTIONAL
}

MacData ::= SEQUENCE {
    mac        DigestInfo,
    macSalt    OCTET STRING,
    iterations INTEGER DEFAULT 1
    -- Note: The default is for historical reasons and its use is
    -- deprecated.
}
```

```
AuthenticatedSafe ::= SEQUENCE OF ContentInfo
    -- Data if unencrypted
    -- EncryptedData if password-encrypted
    -- EnvelopedData if public key-encrypted

SafeContents ::= SEQUENCE OF SafeBag

SafeBag ::= SEQUENCE {
    bagId          BAG-TYPE.&id ({PKCS12BagSet}),
    bagValue       [0] EXPLICIT BAG-TYPE.&Type({PKCS12BagSet}{@bagId}),
    bagAttributes SET OF PKCS12Attribute OPTIONAL
}

-- ===========================
-- Bag types
-- ===========================

keyBag BAG-TYPE ::=
    {KeyBag              IDENTIFIED BY {bagtypes 1}}
pkcs8ShroudedKeyBag BAG-TYPE ::=
    {PKCS8ShroudedKeyBag IDENTIFIED BY {bagtypes 2}}
certBag BAG-TYPE ::=
    {CertBag             IDENTIFIED BY {bagtypes 3}}
crlBag BAG-TYPE ::=
    {CRLBag              IDENTIFIED BY {bagtypes 4}}
secretBag BAG-TYPE ::=
    {SecretBag           IDENTIFIED BY {bagtypes 5}}
safeContentsBag BAG-TYPE ::=
    {SafeContents        IDENTIFIED BY {bagtypes 6}}

PKCS12BagSet BAG-TYPE ::= {
    keyBag |
    pkcs8ShroudedKeyBag |
    certBag |
    crlBag |
    secretBag |
    safeContentsBag,
    ... -- For future extensions
}

BAG-TYPE ::= TYPE-IDENTIFIER

-- KeyBag
KeyBag ::= PrivateKeyInfo

-- Shrouded KeyBag
PKCS8ShroudedKeyBag ::= EncryptedPrivateKeyInfo
```

```
-- CertBag
CertBag ::= SEQUENCE {
    certId    BAG-TYPE.&id   ({CertTypes}),
    certValue [0] EXPLICIT BAG-TYPE.&Type ({CertTypes}{@certId})
}

x509Certificate BAG-TYPE ::=
    {OCTET STRING IDENTIFIED BY {certTypes 1}}
    -- DER-encoded X.509 certificate stored in OCTET STRING
sdsiCertificate BAG-TYPE ::=
    {IA5String IDENTIFIED BY {certTypes 2}}
    -- Base64-encoded SDSI certificate stored in IA5String

CertTypes BAG-TYPE ::= {
    x509Certificate |
    sdsiCertificate,
    ... -- For future extensions
}

-- CRLBag
CRLBag ::= SEQUENCE {
    crlId     BAG-TYPE.&id ({CRLTypes}),
    crltValue [0] EXPLICIT BAG-TYPE.&Type ({CRLTypes}{@crlId})
}

x509CRL BAG-TYPE ::=
    {OCTET STRING IDENTIFIED BY {crlTypes 1}}
    -- DER-encoded X.509 CRL stored in OCTET STRING

CRLTypes BAG-TYPE ::= {
    x509CRL,
    ... -- For future extensions
}

-- Secret Bag
SecretBag ::= SEQUENCE {
    secretTypeId  BAG-TYPE.&id ({SecretTypes}),
    secretValue   [0] EXPLICIT BAG-TYPE.&Type ({SecretTypes}
                                              {@secretTypeId})
}

SecretTypes BAG-TYPE ::= {
    ... -- For future extensions
}

-- ===========================
-- Attributes
-- ===========================
```

```
PKCS12Attribute ::= SEQUENCE {
    attrId      ATTRIBUTE.&id ({PKCS12AttrSet}),
    attrValues  SET OF ATTRIBUTE.&Type ({PKCS12AttrSet}{@attrId})
} -- This type is compatible with the X.500 type 'Attribute'

PKCS12AttrSet ATTRIBUTE ::= {
    friendlyName |
    localKeyId,
    ... -- Other attributes are allowed
}
```

```
END
```

Appendix E. Intellectual Property Considerations

   EMC Corporation makes no patent claims on the general constructions
   described in this document, although specific underlying techniques
   may be covered.

   RC2 and RC4 are trademarks of EMC Corporation.

   EMC Corporation makes no representations regarding intellectual
   property claims by other parties. Such determination is the
   responsibility of the user.

Appendix F. Acknowledgments

   Many thanks to Dan Simon of Microsoft Corporation and Jim Spring of
   Netscape Communications Corporation for their assistance in preparing
   early drafts of this document. Especial thanks to Brian Beckman of
   Microsoft Corporation for writing the specification that this
   document is based on.

Appendix G. 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 the IETF. Suggestions for
   improvement are welcome.

Authors' Addresses

   Kathleen M. Moriarty (editor)
   EMC Corporation
   176 South Street
   Hopkinton, MA
   United States

   EMail: Kathleen.Moriarty@emc.com

   Magnus Nystrom
   Microsoft Corporation
   1 Microsoft Way
   Redmond, WA 98052
   United States

   EMail: mnystrom@microsoft.com

   Sean Parkinson
   RSA Security Inc.
   345 Queen Street
   Brisbane, QLD, 4000
   Australia

   EMail: Sean.Parkinson@rsa.com

   Andreas Rusch
   RSA Security Inc.
   345 Queen Street
   Brisbane, QLD, 4000
   Australia

   EMail: Andreas.Rusch@rsa.com

   Michael Scott
   RSA Security Inc.
   345 Queen Street
   Brisbane, QLD, 4000
   Australia

   EMail: Michael2.Scott@rsa.com