RSA Laboratories. A Division of RSA Data Security

PKCS 12 v1.0: Personal Information Exchange Syntax

RSA Laboratories

June 24, 1999

Table of Contents

003-903077-100-001-000

1 Introduction

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 PCMCIA devices.

This standard can be viewed as building on PKCS #8 [17] by including essential but ancillary identity information along with private keys and by instituting higher security through public-key privacy and integrity modes.

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 [10].

ASN.1: Abstract Syntax Notation One, as defined in [2], [3], [4], [5], [6] and [7].

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 [9].

Certificate: A digitally signed data unit binding a public key to identity information. A specific format for identity certificates is defined in [10]. Another format is described in [13].

Certificate Revocation List (CRL): A digitally signed list of certificates that should no longer be honored, having been revoked by the issuers or a higher authority.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 [10].

ContentInfo: An ASN.1 type used to hold data that may have been cryptographically protected. This type is defined in [16].

DER: Distinguished Encoding Rules, as defined in [8].

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 [16].

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.

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.

In this document, ASN.1 types, values and object sets are written in **bold Helvetica**.

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:

- *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.

- *Password privacy mode:* Personal information is encrypted with a symmetric key derived from a user name and a privacy password, as in [15]. 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:

- *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).

- *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. Unfortunately, weak symmetric encryption may be required for products exported from certain countries under applicable export regulations (see Appendix D).

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 Message Authentication Code 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 which 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'être* 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 [16], 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]. 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.

```
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. A higher
        -- value, like 1024, is recommended.
}
```

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 **SafeBag**s. 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
        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 safe bags 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 PKCS-8ShroudedKeyBag 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 {certTypes 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 at their own discretion to this set.

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 **KeyBag**s, **PKCS8ShroudedKeyBag**s, **CertBag**s, **CRLBag**s and **SecretBag**s within the top-level **SafeContents**.

5 Using PFX PDUs

This section describes creation and usage of **PFX** PDUs.

5.1 Creating PFX PDUs

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 SC_I, 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 SHA-1 HMAC 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, which 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.

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], which can be constructed from any of a variety of hash functions. Note that the specification in [11] differs somewhat from the specification in [12]. 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 should be SHA-1. As indicated in Section B.4, this implies that SHA-1 is also used to derive the MAC key itself in password integrity mode, and that the MAC key has 160 bits.

When password integrity mode is used to secure a **PFX** PDU, an SHA-1 HMAC 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).

B. Deriving keys and IVs from passwords and salt

We present here a general method for using a hash function to produce various types of pseudo-random 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[1].

B.1 Password formatting

The underlying password-based encryption methods in PKCS #5 v1.5 viewed 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 Version 1.5 says: "How passwords are entered by an entity (a user) is outside the scope of this standard. However, it is recommended in the interest of interoperability that when

---

[1] **Note**: In general, procedures and algorithms defined in [15] should be used, this appendix is present only to describe password-based key derivation using algorithms defined in this document.

messages encrypted according to this standard are to be transferred from one computer system to another, the password should consist of printable ASCII characters (values 32 to 126 inclusive). This recommendation may require that password-entry software support optional conversion from a local character set to ASCII."

In this specification however, all passwords are created from **BMPString**s 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 two 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: $\mathbf{Z}_2^u \times \mathbf{Z}_2^v \rightarrow \mathbf{Z}_2^u$ (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). For MD2 and MD5, u=128 and v=512; for SHA-1, u=160 and v=512. 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 pseudo-random bits required. In addition, u and v are of course non-zero.

The following procedure can be used to produce pseudo-random bits for a particular "purpose" which is identified by a byte, *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 $v \cdot \lceil s/v \rceil$ 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 $v \cdot \lceil p/v \rceil$ 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 $c=\lceil n/u \rceil$.

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

   a) Set $A_i=H^r(D\|I)$. (i.e. the r^{th} hash of $D\|I$, $H(H(H(\ldots H(D||I))))$)

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

   c) Treating I as a concatenation $I_0, I_1, \ldots, I_{k-1}$ of v-bit blocks, where $k=\lceil s/v \rceil+\lceil p/v \rceil$, modify I by setting $I_j=(I_j+B+1) \bmod 2^v$ for each j.

7. Concatenate $A_1, A_2, \ldots, A_c$ together to form a pseudo-random 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 pseudo-random bits being produced are to be used as key material for performing encryption or decryption.

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

3. If *ID*=3, then the pseudo-random bits being produced are to be used as an integrity key for MAC-ing.

B.4 Keys and IVs for password privacy mode

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 also does not prescribe a length for the salt. 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 [15] for more information).

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 Section B.2 and Section 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 which 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.

B.5 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 was given in Section B.4 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 is used for performing all MACing, and so all MAC keys are 160 bits. See Appendix A for more information on MACing.

C. 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)}

-- 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 is 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-CBCOBJECT IDENTIFIER ::= {pkcs-12PbeIds 3}
pbeWithSHAAnd2-KeyTripleDES-CBCOBJECT 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. A higher
        -- value, like 1024 is recommended.
}

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}}
pkcs-8ShroudedKeyBag 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 {certTypes 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
```

D. A note on export regulations

Several national governments, including the US government, restrict the export of products containing encryption capabilities under international trade regulations. Implementers are responsible for their own adherence to these regulations. No person or organization connected with this standard accepts any responsibility or liability with respect to implementers' adherence to applicable regulations.

In pursuing export permission for their products, implementers should argue that PFX is not a generic encryption facility; that it is intended for exchange of authentication information only; that their products cannot be used to exchange arbitrary information (as evidenced by their source code); and that, therefore, they should be allowed to export products with strong encryption in this context and this context only.

E. Intellectual property considerations

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

RC2 and RC4 are trademarks of RSA Data Security.

License to copy this document is granted provided that it is identified as "RSA Data Security, Inc. Public-Key Cryptography Standards (PKCS)" in all material mentioning or referencing this document.

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

F. References

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

[2] *ISO/IEC 8824-1:1995: Information technology — Abstract Syntax Notation One (ASN.1) — Specification of basic notation.* 1995.

[3] *ISO/IEC 8824-1:1995/Amd.1: 1995: Information technology — Abstract Syntax Notation One (ASN.1) — Specification of basic notation — Amendment 1 — Rules of extensibility.* 1995.

[4] *ISO/IEC 8824-2:1995: Information technology — Abstract Syntax Notation One (ASN.1) — Information object specification.* 1995.

[5] *ISO/IEC 8824-2:1995/Amd.1: 1995: Information technology — Abstract Syntax Notation One (ASN.1) — Information object specification — Amendment 1 — Rules of extensibility.* 1995.

[6] *ISO/IEC 8824-3:1995: Information technology — Abstract Syntax Notation One (ASN.1) — Constraint specification.* 1995.

[7] *ISO/IEC 8824-4:1995: Information technology — Abstract Syntax Notation One (ASN.1) — Parameterization of ASN.1 specifications.* 1995.

[8] *ISO/IEC 8825-1:1995: Information Technology – ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules. 1995.*

[9] *ISO/IEC 9594-2:1997. Information technology — Open Systems Interconnection — The Directory: Models. 1997.*

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

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

[12] Microsoft Corporation. *PFX: Personal Exchange Syntax and Protocol Standard*. Version 0.020, January 1997.

[13] R. Rivest and B. Lampson. *Simple Distributed Security Infrastructure*, http://theory.lcs.mit.edu/~rivest/sdsi.ps, 1996.

[14] RSA Laboratories. *PKCS #1: RSA Encryption Standard.* Version 2.0, October 1998.

[15] RSA Laboratories. *PKCS #5: Password-Based Encryption Standard*. Version 2.0, March 1999.

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

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

G. 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.

H. About PKCS

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

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

PKCS Editor
RSA Laboratories
20 Crosby Drive
Bedford, MA 01730 USA
pkcs-editor@rsa.com
http://www.rsa.com/rsalabs/pubs/PKCS