[MS-OFFCRYPTO]:

Office Document Cryptography Structure

Tools. The Open Specifications documentation does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments, you are free to take advantage of them. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Support. For questions and support, please contact dochelp@microsoft.com.

Revision Summary

Date	Revision History	Revision Class	Comments
4/4/2008	0.1	New	Initial Availability
6/27/2008	1.0	Major	Revised and edited the technical content
10/6/2008	1.01	Editorial	Revised and edited the technical content
12/12/2008	1.02	Editorial	Revised and edited the technical content
3/18/2009	1.03	Editorial	Revised and edited the technical content
7/13/2009	1.04	Major	Revised and edited the technical content
8/28/2009	1.05	Major	Updated and revised the technical content
11/6/2009	1.06	Editorial	Revised and edited the technical content
2/19/2010	2.0	Editorial	Revised and edited the technical content
3/31/2010	2.01	Editorial	Revised and edited the technical content
4/30/2010	2.02	Editorial	Revised and edited the technical content
6/7/2010	2.03	Editorial	Revised and edited the technical content
6/29/2010	2.04	Editorial	Changed language and formatting in the technical content.
7/23/2010	2.05	Minor	Clarified the meaning of the technical content.
9/27/2010	2.05	None	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	2.05	None	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	2.05	None	No changes to the meaning, language, or formatting of the technical content.
3/18/2011	2.05	None	No changes to the meaning, language, or formatting of the technical content.
6/10/2011	2.05	None	No changes to the meaning, language, or formatting of the technical content.
1/20/2012	2.6	Minor	Clarified the meaning of the technical content.
4/11/2012	2.6	None	No changes to the meaning, language, or formatting of the technical content.
7/16/2012	2.7	Minor	Clarified the meaning of the technical content.
10/8/2012	2.8	Minor	Clarified the meaning of the technical content.
2/11/2013	2.8	None	No changes to the meaning, language, or formatting of the technical content.
7/30/2013	2.8	None	No changes to the meaning, language, or formatting of the technical content.
11/18/2013	2.8	None	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
2/10/2014	2.8	None	No changes to the meaning, language, or formatting of the technical content.
4/30/2014	3.0	Major	Significantly changed the technical content.
7/31/2014	3.0	None	No changes to the meaning, language, or formatting of the technical content.
10/30/2014	3.0	None	No changes to the meaning, language, or formatting of the technical content.
3/16/2015	4.0	Major	Significantly changed the technical content.
9/4/2015	4.0	None	No changes to the meaning, language, or formatting of the technical content.
7/15/2016	4.0	None	No changes to the meaning, language, or formatting of the technical content.
9/14/2016	4.0	None	No changes to the meaning, language, or formatting of the technical content.
6/20/2017	4.0	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

The Office Document Cryptography Structure is relevant to documents that have Information Rights Management (IRM) policies, document encryption, or signing and write protection applied. More specifically, this file format describes the following:

- A structure that acts as a generic mechanism for storing data that has been transformed along with information about that data.
- A structure for storing rights management policies that have been applied to a particular document.
- Encryption, signing, and write protection structures.

Sections 1.7 and 2 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

Advanced Encryption Standard (AES): A block cipher that supersedes the **Data Encryption Standard (DES)**. AES can be used to protect electronic data. The AES algorithm can be used to encrypt (encipher) and decrypt (decipher) information. Encryption converts data to an unintelligible form called ciphertext; decrypting the ciphertext converts the data back into its original form, called plaintext. AES is used in symmetric-key cryptography, meaning that the same key is used for the encryption and decryption operations. It is also a block cipher, meaning that it operates on fixed-size blocks of plaintext and ciphertext, and requires the size of the plaintext as well as the ciphertext to be an exact multiple of this block size. AES is also known as the Rijndael symmetric encryption algorithm [FIPS197].

ASCII: The American Standard Code for Information Interchange (ASCII) is an 8-bit character-encoding scheme based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. ASCII refers to a single 8-bit ASCII character or an array of 8-bit ASCII characters with the high bit of each character set to zero.

base64 encoding: A binary-to-text encoding scheme whereby an arbitrary sequence of bytes is converted to a sequence of printable **ASCII** characters, as described in [RFC4648].

block cipher: A cryptographic algorithm that transforms a group of plaintext bits, referred to as a block, into a fixed-size block of cipher text. When the process is reversed, a fixed-size block of cipher text is transformed into a block of plaintext bits. See also stream cipher.

certificate: A certificate is a collection of attributes and extensions that can be stored persistently. The set of attributes in a certificate can vary depending on the intended usage of the certificate. A certificate securely binds a public key to the entity that holds the corresponding private key. A certificate is commonly used for authentication and secure exchange of information on open networks, such as the Internet, extranets, and intranets. Certificates are digitally signed by the issuing certification authority (CA) and can be issued for a user, a computer, or a service. The most widely accepted format for certificates is defined by the ITU-T X.509 version 3 international standards. For more information about attributes and extensions, see [RFC3280] and [X509] sections 7 and 8.

certificate chain: A sequence of **certificates**, where each certificate in the sequence is signed by the subsequent certificate. The last certificate in the chain is normally a self-signed certificate.

cipher block chaining (CBC): A method of encrypting multiple blocks of plaintext with a block cipher such that each ciphertext block is dependent on all previously processed plaintext blocks.

In the **CBC** mode of operation, the first block of plaintext is XOR'd with an Initialization Vector (IV). Each subsequent block of plaintext is XOR'd with the previously generated ciphertext block before encryption with the underlying block cipher. To prevent certain attacks, the IV must be unpredictable, and no IV should be used more than once with the same key. **CBC** is specified in [SP800-38A] section 6.2.

Component Object Model (COM): An object-oriented programming model that defines how objects interact within a single process or between processes. In **COM**, clients have access to an object through interfaces implemented on the object. For more information, see [MS-DCOM].

Coordinated Universal Time (UTC): A high-precision atomic time standard that approximately tracks Universal Time (UT). It is the basis for legal, civil time all over the Earth. Time zones around the world are expressed as positive and negative offsets from UTC. In this role, it is also referred to as Zulu time (Z) and Greenwich Mean Time (GMT). In these specifications, all references to UTC refer to the time at UTC-0 (or GMT).

Cryptographic Application Programming Interface (CAPI) or CryptoAPI: The Microsoft cryptographic application programming interface (API). An API that enables application developers to add authentication, encoding, and encryption to Windows-based applications.

cryptographic service provider (CSP): A software module that implements cryptographic functions for calling applications that generates digital signatures. Multiple **CSPs** may be installed. A **CSP** is identified by a name represented by a NULL-terminated Unicode string.

Data Encryption Standard (DES): A specification for encryption of computer data that uses a 56-bit key developed by IBM and adopted by the U.S. government as a standard in 1976. For more information see [FIPS46-3].

data space: A series of transforms that operate on original document content in a specific order. The first transform in a data space takes untransformed data as input and passes the transformed output to the next transform. The last transform in the data space produces data that is stored in the compound file. When the process is reversed, each transform in the data space is applied in reverse order to return the data to its original state.

data space reader: A software component that extracts **protected content** to perform an operation on the content or to display the content to users. A data space reader does not modify or create data spaces.

data space updater: A software component that can read and update **protected content**. A data space updater cannot change data space definitions.

data space writer: A software component that can read, update, or create a data space definition or **protected content**.

Distinguished Encoding Rules (DER): A method for encoding a data object based on Basic Encoding Rules (BER) encoding but with additional constraints. DER is used to encode X.509 certificates that need to be digitally signed or to have their signatures verified.

electronic codebook (ECB): A **block cipher** mode that does not use feedback and encrypts each block individually. Blocks of identical plaintext, either in the same message or in a different message that is encrypted with the same key, are transformed into identical ciphertext blocks. Initialization vectors cannot be used.

encryption key: One of the input parameters to an encryption algorithm. Generally speaking, an encryption algorithm takes as input a clear-text message and a key, and results in a cipher-text message. The corresponding decryption algorithm takes a cipher-text message, and the key, and results in the original clear-text message.

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the **GUID**. See also universally unique identifier (UUID).

Hash-based Message Authentication Code (HMAC): A mechanism for message authentication using cryptographic hash functions. HMAC can be used with any iterative cryptographic hash function (for example, **MD5** and **SHA-1**) in combination with a secret shared key. The cryptographic strength of HMAC depends on the properties of the underlying hash function.

Information Rights Management (IRM): A technology that provides persistent protection to digital data by using encryption, **certificates**, and authentication. Authorized recipients or users acquire a license to gain access to the protected files according to the rights or business rules that are set by the content owner.

language code identifier (LCID): A 32-bit number that identifies the user interface human language dialect or variation that is supported by an application or a client computer.

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

MD5: A one-way, 128-bit hashing scheme that was developed by RSA Data Security, Inc., as described in [RFC1321].

OLE compound file: A form of structured storage, as described in [MS-CFB]. A compound file allows independent storages and streams to exist within a single file.

protected content: Any content or information, such as a file, Internet message, or other object type, to which a rights-management usage policy is assigned and is encrypted according to that policy. See also **Information Rights Management (IRM)**.

RC4: A variable key-length symmetric encryption algorithm. For more information, see [SCHNEIER] section 17.1.

salt: An additional random quantity, specified as input to an encryption function that is used to increase the strength of the encryption.

SHA-1: An algorithm that generates a 160-bit hash value from an arbitrary amount of input data, as described in [RFC3174]. SHA-1 is used with the Digital Signature Algorithm (DSA) in the Digital Signature Standard (DSS), in addition to other algorithms and standards.

storage: An element of a compound file that is a unit of containment for one or more storages and streams, analogous to directories in a file system, as described in [MS-CFB].

stream: An element of a compound file, as described in [MS-CFB]. A stream contains a sequence of bytes that can be read from or written to by an application, and they can exist only in storages.

transform: An operation that is performed on data to change it from one form to another. Two examples of transforms are compression and encryption.

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The **Unicode** standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

Uniform Resource Identifier (URI): A string that identifies a resource. The URI is an addressing mechanism defined in Internet Engineering Task Force (IETF) Uniform Resource Identifier (URI): Generic Syntax [RFC3986].

Uniform Resource Locator (URL): A string of characters in a standardized format that identifies a document or resource on the World Wide Web. The format is as specified in [RFC1738].

UTF-8: A byte-oriented standard for encoding Unicode characters, defined in the Unicode standard. Unless specified otherwise, this term refers to the UTF-8 encoding form specified in [UNICODE5.0.0/2007] section 3.9.

X.509: An ITU-T standard for public key infrastructure subsequently adapted by the IETF, as specified in [RFC3280].

XOR obfuscation: A type of file encryption that helps protect private data by using an exclusive or bitwise operation. This is done by adding a mathematical expression that prevents a simple reverse-engineering process.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[BCMO800-38A] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation: Methods and Techniques", NIST Special Publication 800-38A, December 2001, http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[Can-XML-1.0] Boyer, J., "Canonical XML Version 1.0", W3C Recommendation, March 2001, http://www.w3.org/TR/2001/REC-xml-c14n-20010315

[DRAFT-DESX] Simpson, W.A. and Baldwin R., "The ESP DES-XEX3-CBC Transform", July 1997, http://tools.ietf.org/html/draft-ietf-ipsec-ciph-desx-00

[ECMA-376] ECMA International, "Office Open XML File Formats", 1st Edition, ECMA-376, December 2006, http://www.ecma-international.org/publications/standards/Ecma-376.htm

[ISO/IEC 10118] International Organization for Standardization, "Hash-functions -- Part 3: Dedicated hash-functions", March 2004, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39876

[ITUX680-1994] ITU-T, "Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation", ITU-T Recommendation X.680, July 1994, http://www.itu.int/rec/T-REC-X.680-199407-S/en

[MS-CFB] Microsoft Corporation, "Compound File Binary File Format".

[MS-DOC] Microsoft Corporation, "Word (.doc) Binary File Format".

[MS-DTYP] Microsoft Corporation, "Windows Data Types".

[MS-OSHARED] Microsoft Corporation, "Office Common Data Types and Objects Structures".

[MS-PPT] Microsoft Corporation, "PowerPoint (.ppt) Binary File Format".

[MS-RMPR] Microsoft Corporation, "Rights Management Services (RMS): Client-to-Server Protocol".

[MS-UCODEREF] Microsoft Corporation, "Windows Protocols Unicode Reference".

[MS-XLSB] Microsoft Corporation, "Excel (.xlsb) Binary File Format".

[MS-XLS] Microsoft Corporation, "Excel Binary File Format (.xls) Structure".

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

[RFC1320] Rivest, R., "The MD4 Message-Digest Algorithm", RFC 1320, April 1992, http://www.ietf.org/rfc/rfc1320.txt

[RFC1851] Karn, P., Metzger, P., and Simpson, W., "The ESP Triple DES Transform", RFC 1851, September 1995, http://www.rfc-editor.org/rfc/rfc1851.txt

[RFC2104] Krawczyk, H., Bellare, M., and Canetti, R., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997, http://www.ietf.org/rfc/rfc2104.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

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

[RFC2822] Resnick, P., Ed., "Internet Message Format", RFC 2822, April 2001, http://www.ietf.org/rfc/rfc2822.txt

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

[RFC3280] Housley, R., Polk, W., Ford, W., and Solo, D., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002, http://www.ietf.org/rfc/rfc3280.txt

[RFC3447] Jonsson, J. and Kaliski, B., "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003, http://www.ietf.org/rfc/rfc3447.txt

[RFC4634] Eastlake III, D. and Hansen, T., "US Secure Hash Algorithms (SHA and HMAC-SHA)", RFC 4634, July 2006, http://www.ietf.org/rfc/rfc4634.txt

[W3C-XSD] World Wide Web Consortium, "XML Schema Part 2: Datatypes Second Edition", October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028

[XAdES] ETSI, "XML Advanced Electronic Signatures (XAdES)", ETSI TS 101 903 V1.3.2, http://uri.etsi.org/01903/v1.3.2/

[XMLDSig] Bartel, M., Boyer, J., Fox, B., et al., "XML-Signature Syntax and Processing", W3C Recommendation, February 2002, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

1.2.2 Informative References

[ISO/IEC29500-1:2011] ISO/IEC, "Information Technology -- Document description and processing languages -- Office Open XML File Formats -- Part 1: Fundamentals and Markup Language Reference", ISO/IEC 29500-1:2011, 2011, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=59575

[MSDN-CAB] Microsoft Corporation, "Microsoft Cabinet Format", March 1997, http://msdn.microsoft.com/en-us/library/bb417343.aspx

1.3 Overview

1.3.1 Data Spaces

The data spaces structure describes a consistent method of storing content in **OLE compound files** that has been transformed in some way. The structure stores both the **protected content** and information about the **transforms** that have been applied to the content. By storing all of this information inside an OLE compound file, client software has all of the information required to read, write, or manipulate the content. A standard structure of **streams** and **storages** allows various software components to interact with the data in a consistent manner.

The data spaces structure allows client applications to describe one or more arbitrary transforms. Each transform represents a single arbitrary operation to be performed on a set of storages or streams in the original document content. One or more transforms can then be composited into a **data space** definition. Data space definitions can then be applied to arbitrary storages or streams in the original document content in the data space map (section 2.1).

Because of the layers of indirection between transforms and document content, different transforms can be applied to different parts of the document content, and transforms can be composited in any order.

The following figure illustrates the relationships between the **DataSpaceMap** stream, the **DataSpaceInfo** storage, the **TransformInfo** storages, and the protected content. Note that other streams and storages exist in this file format; this figure describes only the relationships between these storages and streams.

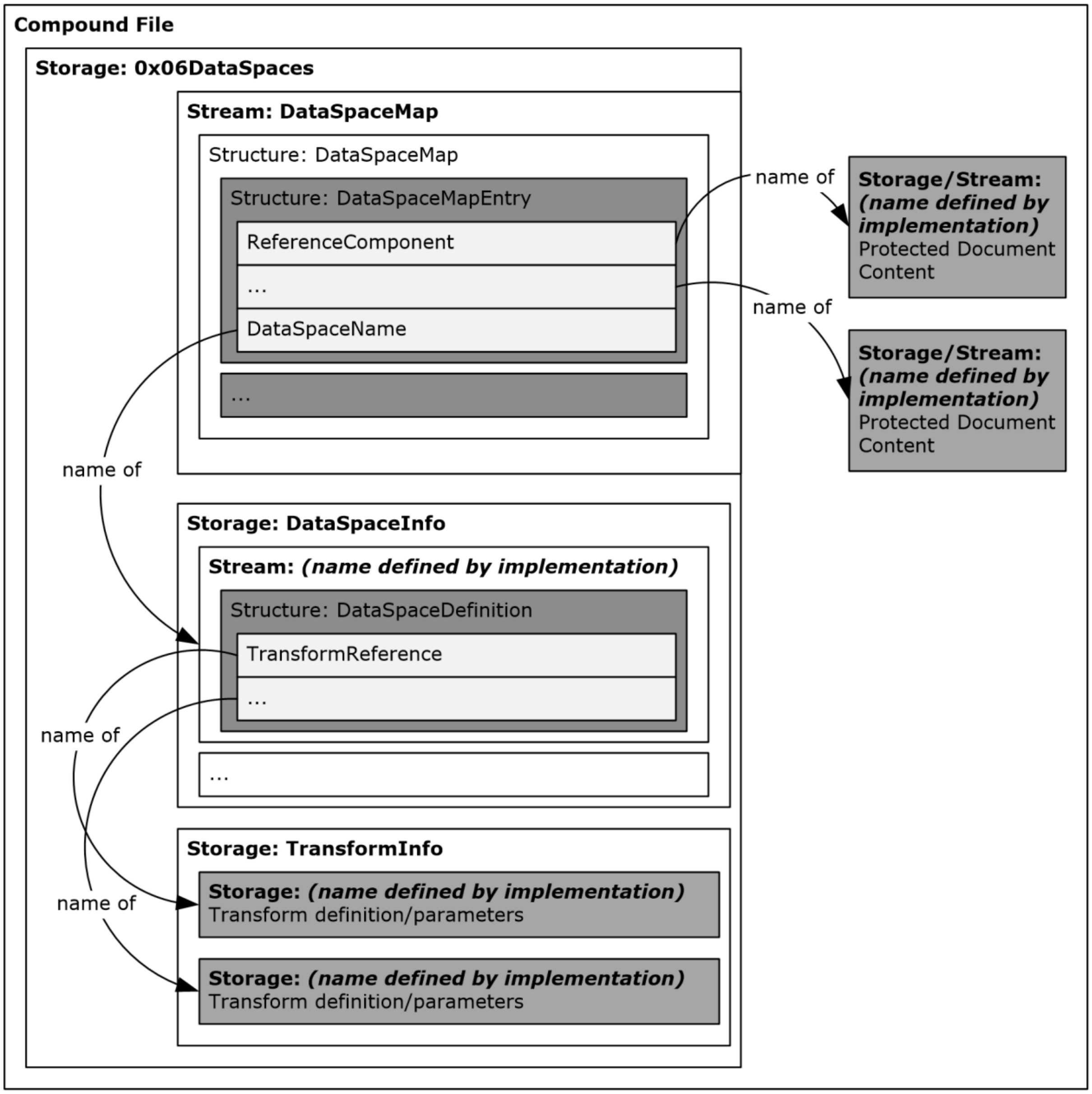

Figure 1: Relationships among the DataSpaceMap stream, the DataSpaceInfo storage, the TransformInfo storages, and the protected content

1.3.2 Information Rights Management Data Space

The Information Rights Management Data Space (IRMDS) structure is used to enforce a rights management policy applied to a document. The structure defines a transform that is used to encrypt document content, and it defines a second transform that can be used for certain document types to compress document content.

The original document content is transformed through encryption and placed in a storage not normally accessed by the application. When needed, the application uses the transforms defined in the document to decrypt the protected content.

This structure is an implementation of the data spaces structure. Therefore, implementing the structure implies storing document content in an OLE compound file.

Applications that implement this structure will typically store a second document in the OLE compound file called the *placeholder document*. The placeholder document is place into the **streams** or storages normally identified by the application as containing document content, such that an application that does not detect the IRMDS structure will instead open the placeholder document.

Applications that implement this structure will typically try to follow the licensing limitations placed on a document. Typical licensing limitations include the right to view, print, edit, forward, or view rights data, as described in [MS-RMPR].

The following figure shows the specific storages, streams, structures, and relationships among them that are created when the IRMDS structure is used in an ECMA-376 document [ECMA-376].

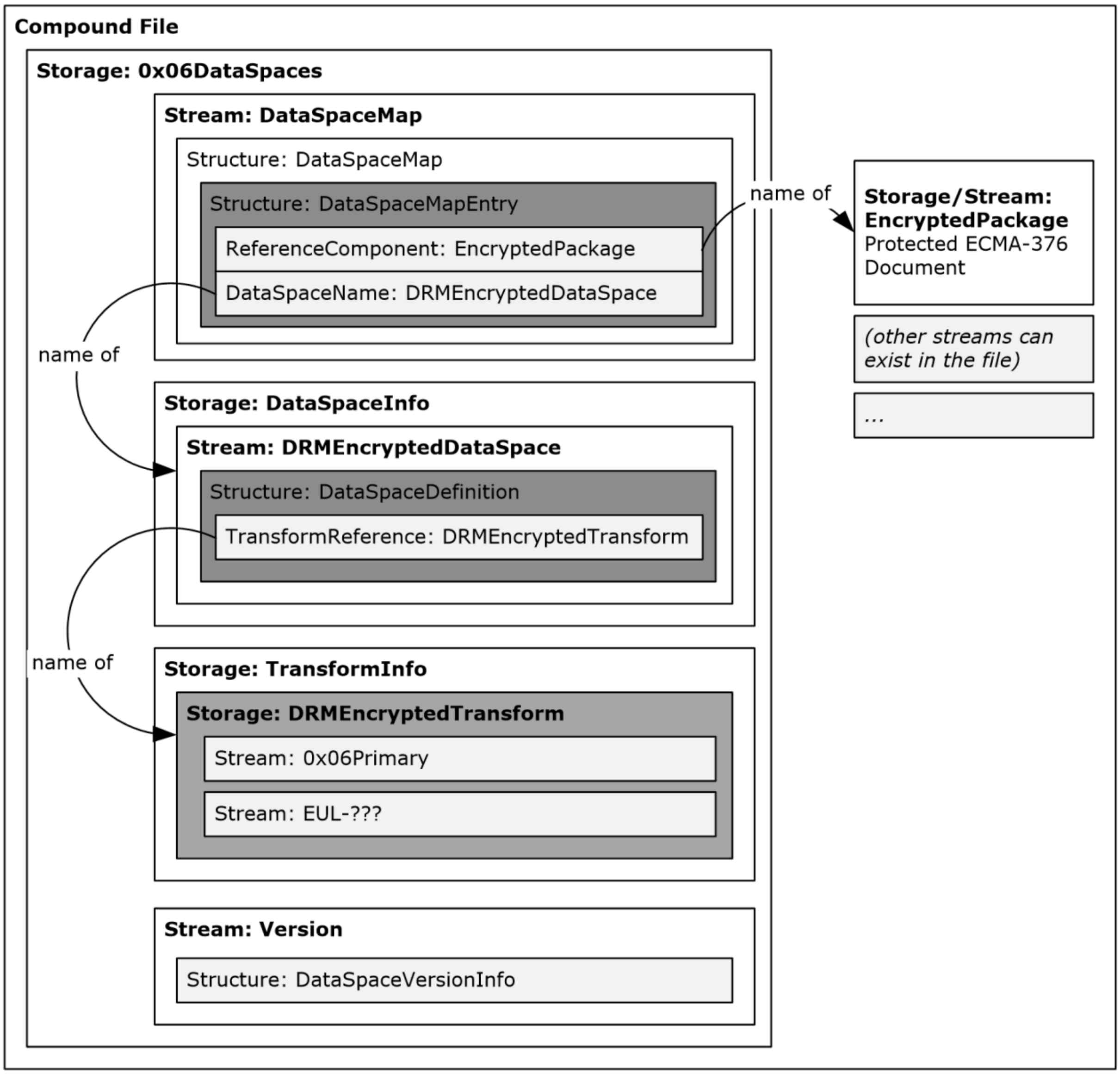

Figure 2: An ECMA-376 word processing document with the IRMDS structure applied

1.3.3 Encryption

Password-protected documents can be created by using one of four mechanisms:

- **XOR obfuscation**.
- 40-bit **RC4** encryption.
- **Cryptographic Application Programming Interface (CAPI) or CryptoAPI** encryption.
- ECMA-376 document encryption [ECMA-376], which can include one of three approaches:
  - **Standard encryption:** This approach uses a binary **EncryptionInfo** structure. It uses **Advanced Encryption Standard (AES)** as an encryption algorithm and **SHA-1** as a hashing algorithm.
  - **Agile encryption:** This approach uses an XML **EncryptionInfo** structure. The encryption and hashing algorithms are specified in the structure and can be for any encryption supported on the host computer.
  - **Extensible encryption:** This approach uses an extensible mechanism to allow arbitrary cryptographic modules to be used.

1.3.3.1 XOR Obfuscation

XOR obfuscation is performed on portions of Office binary documents. The normal **streams** contained within the document are modified in place. For more information about how an application can determine whether XOR obfuscation is being used and the placement of the password verifier see [MS-XLS] and [MS-DOC].

There are two methods for performing XOR obfuscation, known as Method 1 and Method 2. Method 1 specifies structures and procedures used by the Excel Binary File Format (.xls) Structure [MS-XLS], and Method 2 specifies structures and procedures used by the Word Binary File Format (.doc) Structure [MS-DOC].

1.3.3.2 40-bit RC4 Encryption

40-bit RC4 encryption is performed on portions of Office binary documents. For more information about how to determine whether 40-bit RC4 encryption is being used and the placement of the password verifier, see [MS-XLS] and [MS-DOC]. The same mechanisms for generating the password verifier, deriving the **encryption key**, and encrypting data are used for all file formats supporting 40-bit RC4 encryption.

1.3.3.3 CryptoAPI RC4 Encryption

CryptoAPI RC4 encryption is performed on portions of Office binary documents. The documents will contain a new **stream** to contain encrypted information but can also encrypt other streams in place. For more information about how to determine whether CryptoAPI RC4 encryption is being used and the placement of the password verifier, see [MS-XLS], [MS-DOC], and [MS-PPT]. The same mechanisms for generating the password verifier, storing data specifying the cryptography, deriving the encryption key, and encrypting data are used for all file formats supporting CryptoAPI RC4 encryption.

1.3.3.4 ECMA-376 Document Encryption

Encrypted ECMA-376 documents [ECMA-376] use the data spaces functionality (section 1.3.1) to contain the entire document as a single **stream** in an OLE compound file. All ECMA-376 documents [ECMA-376] adhere to the approaches specified in this document and do not require knowledge of application-specific behavior to perform encryption operations. The overall approach is very similar to that used by IRMDS (section 1.3.2).

1.3.4 Write Protection

The application of password-based write protection for Office binary documents is specified in section 2.4.2. Write-protected binary documents vary according to the file format. A summary of each type follows:

- The Excel Binary File Format (.xls) [MS-XLS]**:** The password is converted to a 16-bit password verifier, stored in the document as described in [MS-XLS], and the document is then encrypted as described in [MS-XLS] and in this specification. If the user does not supply an encryption password, a fixed password is used.
- The Word (.doc) Binary File Format [MS-DOC]**:** The password is stored in the clear, as described in [MS-DOC], and the document is not encrypted.
- The PowerPoint (.ppt) Binary File Format [MS-PPT]**:** The password is stored in the clear, as described in [MS-PPT], and the document can then be encrypted as described in [MS-PPT] and in this specification. If encryption is used and the user does not supply an encryption password, a fixed password is used.

1.3.5 Digital Signatures

Office binary documents can be signed by using one of the following methods:

- A binary format stored in a **_signatures** storage. This approach is described in section 2.5.1.
- A format that uses XML-Signature Syntax and Processing, as described in [XMLDSig], stored in an **_xmlsignatures** storage. This approach is described in sections 2.5.2 and 2.5.3.

1.3.6 Byte Ordering

All data and structures in this file format are assumed to be in **little-endian** format.

1.3.7 String Encoding

In this file format, several storages and **stream** names include the strings "0x01", "0x05", "0x06", and "0x09".These strings are not literally included in the name. Instead, they represent the **ASCII** characters with hexadecimal values 0x01, 0x05, 0x06, and 0x09 respectively.

1.3.8 OLE Compound File Path Encoding

Paths to specific storages and **streams** in an OLE compound file are separated by the backslash (\). The backslash is a delimiter between parts of the path and, therefore, is not part of the name of any specific storage or stream. Paths that begin with a backslash signify the root storage of the OLE compound file.

1.3.9 Pseudocode Standard Objects

The pseudocode in this document refers to several objects with associated properties. Accessing a property of an object is denoted with the following syntax: `Object.Property`. This section describes the properties of each object as it is used in this document.

1.3.9.1 Array

An array is a collection of zero or more child objects of uniform type, where each child is addressable by using an unsigned integer index. Referencing a child object of an array is denoted by using the following syntax: `array[index]`.

Indexes are zero-based and monotonically increase by 1. Therefore, Index 0 references the first element in an array, and Index 1 references the second child in the array.

Arrays have the following property:

- **Length:** The number of child objects in the array.

1.3.9.2 String

A string is an array of ASCII characters. As in arrays, individual characters in the string are addressable by using a zero-based index.

1.3.9.3 Storage

A storage is an OLE storage as described by [MS-CFB]. Storages have the following properties:

- **Name:** A unique identifier for the storage within its parent, as described in [MS-CFB].
- **GUID:** A 16-byte identifier associated with the storage, as described in [MS-CFB].
- **Children:** Zero or more child storages or **streams**. Each child is addressable by its name.

1.3.9.4 Stream

A **stream** is an OLE storage as described in [MS-CFB]. Streams have the following properties:

- **Name:** A unique identifier for the stream within its parent, as described in [MS-CFB].
- **Data:** An array of zero or more unsigned 8-bit integers containing the data in the stream.

1.4 Relationship to Protocols and Other Structures

This file format builds on the file format as described in [MS-CFB].

Some structures in this specification reference structures described in [MS-RMPR]. In addition, the protocols described in [MS-RMPR] are necessary for obtaining the information required to understand the transformed data in a document with a rights management policy applied.

For encryption operations, this specification also requires an understanding of the file formats as described in [MS-XLS], [MS-PPT], or [MS-DOC].

1.5 Applicability Statement

1.5.1 Data Spaces

The data spaces structure specifies a set of storages and **streams** within an OLE compound file, the structures contained in them, and relationships among them. OLE compound files that conform to the data spaces structure can also have other storages or streams in them that are not specified by this file format.

1.5.2 Information Rights Management Data Space

The IRMDS structure is required when reading, modifying, or creating documents with rights management policies applied.

1.5.3 Encryption

The ECMA-376 [ECMA-376] encryption structure, **streams**, and storages are required when encrypting ECMA-376 documents. When binary file types are encrypted, either CryptoAPI RC4 encryption, RC4 encryption, or XOR obfuscation is required.

1.6 Versioning and Localization

None.

1.7 Vendor-Extensible Fields

The data spaces structure allows vendors to implement arbitrary transforms, data space definitions, and data space maps. In this way, the structure can be used to represent any arbitrary transformation to any arbitrary data.

The IRMDS structure does not contain any vendor-extensible fields.

ECMA-376 document encryption [ECMA-376] MAY be extended if either additional CryptoAPI providers are installed or extensible encryption is used.

2 Structures

2.1 Data Spaces

The data spaces structure consists of a set of interrelated storages and **streams** in an OLE compound file as specified in [MS-CFB].

Software components that interact with data spaces MUST check the **DataSpaceVersionInfo** structure (section 2.1.5) contained in the **\0x06DataSpaces\Version** stream for the version numbers and respect the following rules.

Data space readers:

- **Data space readers** MUST read the protected content when the reader version is less than or equal to the highest data spaces structure version understood by the software component.
- Readers MUST NOT read the protected content when the reader version is greater than the highest data spaces structure version understood by the software component.

Data space updaters:

- **Data space updaters** MUST preserve the format of the protected content when the updater version is less than or equal to the highest data spaces structure version understood by the software component.
- Updaters MUST NOT change the protected content when the updater version is greater than the highest data spaces structure version understood by the software component.

Data space writers:

- **Data space writers** MUST set the writer version to "1.0".
- Writers MUST set the updater version to "1.0".
- Writers MUST set the reader version to "1.0".

2.1.1 File

Every document that conforms to the data spaces structure (section 2.1) MUST be an OLE compound **File** structure as specified in [MS-CFB]. The **File** structure MUST contain the following storages and **streams**:

- **\0x06DataSpaces storage:** A storage that contains all of the necessary information to understand the transforms applied to original document content in a given OLE compound file.
- **\0x06DataSpaces\Version stream:** A stream containing the **DataSpaceVersionInfo** structure, as specified in section 2.1.5. This stream specifies the version of the data spaces structure used in the file.
- **\0x06DataSpaces\DataSpaceMap stream:** A stream containing a **DataSpaceMap** structure as specified in section 2.1.6. This stream associates protected content with the data space definition used to transform it.
- **\0x06DataSpaces\DataSpaceInfo storage:** A storage containing the data space definitions used in the file. This storage MUST contain one or more streams, each of which contains a **DataSpaceDefinition** structure as specified in section 2.1.7. The storage MUST contain exactly one stream for each **DataSpaceMapEntry** structure (section 2.1.6.1) in the

\0x06DataSpaces\DataSpaceMap stream (section 2.2.1). The name of each stream MUST be equal to the **DataSpaceName** field of exactly one **DataSpaceMapEntry** structure contained in the **\0x06DataSpaces\DataSpaceMap** stream.

- **Transformed content streams and storages:** One or more storages or streams containing protected content. The transformed content is associated with a data space definition by an entry in the **\0x06DataSpaces\DataSpaceMap** stream.

- **\0x06DataSpaces\TransformInfo storage:** A storage containing definitions for the transforms used in the data space definitions stored in the **\0x06DataSpaces\DataSpaceInfo** storage as specified in section 2.2.2. The stream contains zero or more definitions for the possible transforms that can be applied to the data in content streams.

Every transform referenced from a data space MUST be defined in a child storage of the **\0x06DataSpaces\TransformInfo** storage (section 2.2.3), each of which is called a *transform storage*. Transform storages MUST have a valid storage name.

Each transform storage identifies an algorithm used to transform data and any parameters needed by that algorithm. Transform storages do not contain actual implementations of transform algorithms but merely definitions and parameters. It is presumed that all software components that interact with the protected content have access to an existing implementation of the transform algorithm.

Every transform storage MUST contain a stream named "0x06Primary". The 0x06Primary stream MUST begin with a **TransformInfoHeader** structure (section 2.1.8). Transform storages can contain other streams or storages if needed by a particular transform.

2.1.2 Length-Prefixed Padded Unicode String (UNICODE-LP-P4)

The Length-Prefixed Padded Unicode String structure (**UNICODE-LP-P4**) contains a length-prefixed **Unicode** string, which MUST be padded so it is a multiple of 4 bytes.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Length																															
Data (variable)																															
...																															
Padding (variable)																															
...																															

Length (4 bytes): An unsigned integer that specifies the size, in bytes, of the **Data** field. It MUST be a multiple of 2 bytes.

Data (variable): A Unicode string containing the value of the **UNICODE-LP-P4** structure. It MUST NOT be null-terminated.

Padding (variable): A set of bytes that MUST be of the correct size such that the size of the **UNICODE-LP-P4** structure is a multiple of 4 bytes. If **Padding** is present, it MUST be exactly 2 bytes long, and each byte MUST be 0x00.

2.1.3 Length-Prefixed UTF-8 String (UTF-8-LP-P4)

The Length-Prefixed UTF-8 String structure (**UTF-8-LP-P4**) contains a length-prefixed **UTF-8** string, padded to always use a multiple of 4 bytes.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Length																															
Data (variable)																															
...																															
Padding (variable)																															
...																															

Length (4 bytes): An unsigned integer that specifies the size, in bytes, of the **Data** field.

Data (variable): A UTF-8 string that specifies the value of the **UTF-8-LP-P4** structure. It MUST NOT be null-terminated.

Padding (variable): A set of bytes that MUST be of correct size such that the size of the **UTF-8-LP-P4** structure is a multiple of 4 bytes. If **Padding** is present, each byte MUST be 0x00. If the value of the **Length** field is exactly 0x00000000, the **Data** field specifies a null string, and the entire structure uses exactly 4 bytes. If the value of the **Length** field is exactly 0x00000004, the **Data** field specifies an empty string, and the entire structure also uses exactly 4 bytes.

2.1.4 Version

The **Version** structure specifies the version of a product or feature. It contains a major and a minor version number. When comparing version numbers, **vMajor** MUST be considered the most significant component and **vMinor** MUST be considered the least significant component.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
vMajor																vMinor															

vMajor (2 bytes): An unsigned integer that specifies the major version number.

vMinor (2 bytes): An unsigned integer that specifies the minor version number.

2.1.5 DataSpaceVersionInfo

The **DataSpaceVersionInfo** structure indicates the version of the data spaces structure used in a given file.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
FeatureIdentifier (variable)																															

...
ReaderVersion
UpdaterVersion
WriterVersion

FeatureIdentifier (variable): A **UNICODE-LP-P4** structure (section 2.1.2) that specifies the functionality for which the **DataSpaceVersionInfo** structure specifies version information. It MUST be "Microsoft.Container.DataSpaces".

ReaderVersion (4 bytes): A **Version** structure (section 2.1.4) that specifies the reader version of the data spaces structure (section 2.1). **ReaderVersion.vMajor** MUST be 1. **ReaderVersion.vMinor** MUST be 0.

UpdaterVersion (4 bytes): A **Version** structure that specifies the updater version of the data spaces structure. **UpdaterVersion.vMajor** MUST be 1. **UpdaterVersion.vMinor** MUST be 0.

WriterVersion (4 bytes): A **Version** structure that specifies the writer version of the data spaces structure. **WriterVersion.vMajor** MUST be 1. **WriterVersion.vMinor** MUST be 0.

2.1.6 DataSpaceMap

The **DataSpaceMap** structure associates protected content with data space definitions. The data space definition, in turn, describes the series of transforms that MUST be applied to that protected content to restore it to its original form.

By using a map to associate data space definitions with content, a single data space definition can be used to define the transforms applied to more than one piece of protected content. However, a given piece of protected content can be referenced only by a single data space definition.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
HeaderLength																															
EntryCount																															
MapEntries (variable)																															
...																															

HeaderLength (4 bytes): An unsigned integer that specifies the number of bytes in the **DataSpaceMap** structure before the first entry in the **MapEntries** array. It MUST be equal to 0x00000008.

EntryCount (4 bytes): An unsigned integer that specifies the number of **DataSpaceMapEntry** items (section 2.1.6.1) in the **MapEntries** array.

MapEntries (variable): An array of one or more **DataSpaceMapEntry** structures.

2.1.6.1 DataSpaceMapEntry Structure

The **DataSpaceMapEntry** structure associates protected content with a specific data space definition. It is contained within the **DataSpaceMap** structure (section 2.1.6).

Reference components MUST be listed from the most general—that is, storages—to the most specific—that is, **streams**. For example, a stream titled "Chapter 1" in a substorage called "Book" off the root storage of an OLE compound file would have two reference components: "Book" and "Chapter 1", in that order. The simplest content stream reference is one with a single reference component indicating the name of a stream in the root storage of the OLE compound file.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Length																															
ReferenceComponentCount																															
ReferenceComponents (variable)																															
...																															
DataSpaceName (variable)																															
...																															

Length (4 bytes): An unsigned integer that specifies the size, in bytes, of the **DataSpaceMapEntry** structure.

ReferenceComponentCount (4 bytes): An unsigned integer that specifies the number of **DataSpaceReferenceComponent** items (section 2.1.6.2) in the **ReferenceComponents** array.

ReferenceComponents (variable): An array of one or more **DataSpaceReferenceComponent** structures. Each **DataSpaceReferenceComponent** structure specifies the name of a storage or stream containing protected content that is associated with the data space definition named in the **DataSpaceName** field.

DataSpaceName (variable): A **UNICODE-LP-P4** structure (section 2.1.2) that specifies the name of the data space definition associated with the protected content specified in the **ReferenceComponents** field. It MUST be equal to the name of a stream in the **\0x06DataSpaces\DataSpaceInfo** storage as specified in section 2.2.2.

2.1.6.2 DataSpaceReferenceComponent Structure

The **DataSpaceReferenceComponent** structure stores the name of a specific storage or **stream** containing protected content. It is contained within the **DataSpaceMapEntry** structure (section 2.1.6.1).

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
ReferenceComponentType																															

ReferenceComponent (variable)
...

ReferenceComponentType (4 bytes): An unsigned integer that specifies whether the referenced component is a stream or storage. It MUST be 0x00000000 for a stream or 0x00000001 for a storage.

ReferenceComponent (variable): A **UNICODE-LP-P4** structure (section 2.1.2) that specifies the name of the stream or storage containing the protected content to be transformed. If **ReferenceComponentType** is 0x00000000, then **ReferenceComponent** MUST be equal to the name of a stream contained in the root storage of the OLE compound file. If **ReferenceComponentType** is 0x00000001, then **ReferenceComponent** MUST be equal to the name of a storage contained in the root storage of the OLE compound file.

2.1.7 DataSpaceDefinition

Each **DataSpaceDefinition** structure stores a data space definition. A document can contain more than one data space definition—for example, if one content **stream** is both compressed and encrypted while a second stream is merely encrypted.

Each **DataSpaceDefinition** structure MUST be stored in a stream in the **\0x06DataSpaces\DataSpaceInfo** storage (section 2.2.2). The name of the stream MUST be referenced by a **DataSpaceReferenceComponent** structure (section 2.1.6.2) within a **DataSpaceMapEntry** structure (section 2.1.6.1) stored in the **\0x06DataSpaces\DataSpaceMap** stream (section 2.2.1).

TransformReferences MUST be stored in the reverse order in which they have been applied to the protected content. When reversing the transformation, a software component will apply the transforms in the order specified in the **TransformReferences** array.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
HeaderLength																															
TransformReferenceCount																															
TransformReferences (variable)																															
...																															

HeaderLength (4 bytes): An unsigned integer that specifies the number of bytes in the **DataSpaceDefinition** structure before the **TransformReferences** field. It MUST be 0x00000008.

TransformReferenceCount (4 bytes): An unsigned integer that specifies the number of items in the **TransformReferences** array.

TransformReferences (variable): An array of one or more **UNICODE-LP-P4** structures (section 2.1.2) that specify the transforms associated with this data space definition. Each transform MUST be equal to the name of a storage contained in the **\0x06DataSpaces\TransformInfo** storage (section 2.2.3 and 2.2.4).

2.1.8 TransformInfoHeader

The **TransformInfoHeader** structure specifies the identity of a transform. Additional data or structures can follow this header in a **stream**. See section 2.2.6 for an example of the usage of additional data.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
TransformLength																															
TransformType																															
TransformID (variable)																															
...																															
TransformName (variable)																															
...																															
ReaderVersion																															
UpdaterVersion																															
WriterVersion																															

TransformLength (4 bytes): An unsigned integer that specifies the number of bytes in this structure before the **TransformName** field.

TransformType (4 bytes): An unsigned integer that specifies the type of transform to be applied. It MUST be 0x00000001.

TransformID (variable): A **UNICODE-LP-P4** structure (section 2.1.2) that specifies an identifier associated with a specific transform.

TransformName (variable): A **UNICODE-LP-P4** structure that specifies the friendly name of the transform.

ReaderVersion (4 bytes): A **Version** structure (section 2.1.4) that specifies the reader version.

UpdaterVersion (4 bytes): A **Version** structure that specifies the updater version.

WriterVersion (4 bytes): A **Version** structure that specifies the writer version.

2.1.9 EncryptionTransformInfo

The **EncryptionTransformInfo** structure specifies the encryption used for ECMA-376 document encryption [ECMA-376].

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
EncryptionName (variable)																															
...																															
EncryptionBlockSize																															
CipherMode																															
Reserved																															

EncryptionName (variable): A **UTF-8-LP-P4** structure (section 2.1.3) that specifies the name of the encryption algorithm. The name MUST be the name of an encryption algorithm, such as "AES 128", "AES 192", or "AES 256". When used with extensible encryption, this value is specified by the extensible encryption module.

EncryptionBlockSize (4 bytes): An unsigned integer that specifies the block size for the encryption algorithm specified by **EncryptionName**. It MUST be 0x00000010 as specified by the **Advanced Encryption Standard (AES)**. When used with extensible encryption, this value is specified by the extensible encryption module.

CipherMode (4 bytes): A value that MUST be 0x00000000, except when used with extensible encryption. When used with extensible encryption, this value is specified by the extensible encryption module.

Reserved (4 bytes): A value that MUST be 0x00000004.

2.2 Information Rights Management Data Space

IRMDS defines several data space definitions used to enforce rights management policies that have been applied to a document. This structure is an extension of the data spaces structure specified in section 2.1.

IRMDS can be applied to the following types of documents:

- Office binary documents
- ECMA-376 documents [ECMA-376]

In each case, the protected content contains the original document transformed as specified by the IRMDS structure.<1>

2.2.1 \0x06DataSpaces\DataSpaceMap Stream

If the original document content is an Office binary document:

- The **\0x06DataSpaces\DataSpaceMap stream** MUST contain a **DataSpaceMap** structure (section 2.1.6) containing at least one **DataSpaceMapEntry** structure (section 2.1.6.1). The **DataSpaceMapEntry** structure:
  - MUST have a **DataSpaceName** equal to "0x09DRMDataSpace".

  - MUST have exactly one **ReferenceComponents** entry with the name "0x09DRMContent" and the type 0x00000000, which signifies a stream.

- The **\0x06DataSpaces\DataSpaceMap** stream MAY<2> contain a second **DataSpaceMapEntry** structure in the **DataSpaceMap** structure. The second **DataSpaceMapEntry** structure:

  - MUST have a **DataSpaceName** equal to "0x09LZXDRMDataSpace".

  - MUST have exactly one **ReferenceComponents** entry with the name "0x09DRMViewerContent" and the type 0x00000000, which signifies a stream.

If the original document content is an ECMA-376 document [ECMA-376]:

- The **\0x06DataSpaces\DataSpaceMap** stream MUST contain a **DataSpaceMap** structure containing exactly one **DataSpaceMapEntry** structure.

- The **DataSpaceMapEntry** substructure:

  - MUST have a **DataSpaceName** equal to "DRMEncryptedDataSpace".

  - MUST have exactly one **ReferenceComponents** entry with the name "EncryptedPackage" and the type 0x00000000, which signifies a stream.

2.2.2 \0x06DataSpaces\DataSpaceInfo Storage

If the original document content is an Office binary document:

- The **\0x06DataSpaces\DataSpaceInfo** storage MUST contain a **stream** named "0x09DRMDataSpace", which MUST contain a **DataSpaceDefinition** structure (section 2.1.7):

  - The **DataSpaceDefinition** structure MUST have exactly one **TransformReferences** entry, which MUST be "0x09DRMTransform".

- The **\0x06DataSpaces\DataSpaceInfo** storage MAY<3> contain a stream named "0x09LZXDRMDataSpace". If this stream exists, it MUST contain a **DataSpaceDefinition** structure:

  - The **DataSpaceDefinition** structure MUST have exactly two **TransformReferences** entries.

  - The first **TransformReferences** entry MUST be "0x09DRMTransform".

  - The second **TransformReferences** entry MUST be "0x09LZXTransform".

If the original document content is an ECMA-376 document [ECMA-376]:

- The **\0x06DataSpaces\DataSpaceInfo** storage MUST contain a stream named "DRMEncryptedDataSpace", which MUST contain a **DataSpaceDefinition** structure.

- The **DataSpaceDefinition** structure MUST have exactly one **TransformReferences** entry, which MUST be "DRMEncryptedTransform".

2.2.3 \0x06DataSpaces\TransformInfo Storage for Office Binary Documents

If the original document content is an Office binary document, the **\0x06DataSpaces\TransformInfo** storage MUST contain one storage named "0x09DRMTransform". The "0x09DRMTransform" storage MUST contain a **stream** named "0x06Primary". The "0x06Primary" stream MUST contain an **IRMDSTransformInfo** structure (section 2.2.6). Within the **IRMDSTransformInfo** structure, the following values MUST be set:

- **TransformInfoHeader.TransformType** MUST be 0x00000001.
- **TransformInfoHeader.TransformID** MUST be "{C73DFACD-061F-43B0-8B64-0C620D2A8B50}".
- **TransformInfoHeader.TransformName** MUST be "Microsoft.Metadata.DRMTransform".
- **TransformInfoHeader.ReaderVersion** MUST be "1.0".
- **TransformInfoHeader.UpdaterVersion** MUST be "1.0".
- **TransformInfoHeader.WriterVersion** MUST be "1.0".

The 0x09DRMTransform storage MUST also contain one or more end-user license streams as specified in section 2.2.7.

The **\0x06DataSpaces\TransformInfo** storage MAY<4> contain a substorage named "0x09LZXTransform". If the 0x09LZXTransform storage exists, it MUST contain a stream named "0x06Primary". The **0x06Primary** stream MUST contain a **TransformInfoHeader** structure (section 2.1.8). Within the **TransformInfoHeader** structure, the following values MUST be set:

- **TransformType** MUST be 0x00000001.
- **TransformID** MUST be "{86DE7F2B-DDCE-486d-B016-405BBE82B8BC}".
- **TransformName** MUST be "Microsoft.Metadata.CompressionTransform".
- **ReaderVersion** MUST be "1.0".
- **UpdaterVersion** MUST be "1.0".
- **WriterVersion** MUST be "1.0".

2.2.4 \0x06DataSpaces\TransformInfo Storage for ECMA-376 Documents

If the original document is an ECMA-376 document [ECMA-376] conforming to the IRMDS structure, the **\0x06DataSpaces\TransformInfo** storage MUST contain one storage named "DRMEncryptedTransform". The "DRMEncryptedTransform" storage MUST contain a **stream** named "0x06Primary". The "0x06Primary" stream MUST contain an **IRMDSTransformInfo** structure (section 2.2.6). Within the **IRMDSTransformInfo** structure, the following values MUST be set:

- **TransformInfoHeader.TransformType** MUST be 0x00000001.
- **TransformInfoHeader.TransformID** MUST be "{C73DFACD-061F-43B0-8B64-0C620D2A8B50}".
- **TransformInfoHeader.TransformName** MUST be "Microsoft.Metadata.DRMTransform".
- **TransformInfoHeader.ReaderVersion** MUST be 1.0.
- **TransformInfoHeader.UpdaterVersion** MUST be 1.0.
- **TransformInfoHeader.WriterVersion** MUST be 1.0.

The DRMEncryptedTransform storage MUST also contain one or more end-user license streams as specified in section 2.2.7.

2.2.5 ExtensibilityHeader

The **ExtensibilityHeader** structure provides a facility to allow an updated header with more information to be inserted into a larger structure in the future. This structure consists of a single element.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
Length																															

Length (4 bytes): An unsigned integer that specifies the size of the **ExtensibilityHeader** structure. It MUST be 0x00000004.

2.2.6 IRMDSTransformInfo

The **IRMDSTransformInfo** structure specifies a specific transform that has been applied to protected content to enforce rights management policies applied to the document.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
TransformInfoHeader (variable)																															
...																															
ExtensibilityHeader																															
XrMLLicense (variable)																															
...																															

TransformInfoHeader (variable): A **TransformInfoHeader** structure (section 2.1.8) that specifies the identity of the transform applied.

ExtensibilityHeader (4 bytes): An **ExtensibilityHeader** structure (section 2.2.5).

XrMLLicense (variable): A **UTF-8-LP-P4** structure (section 2.1.3) containing a valid XrML signed issuance license as specified in [MS-RMPR] section 2.2.9.9. The signed issuance license MAY<5> contain the application-specific name-value attribute pairs **name** and **id**, as specified in [MS-RMPR] section 2.2.9.7.6, as part of the **AUTHENTICATEDDATA** element.

2.2.7 End-User License Stream

The end-user license **stream** contains cached use licenses.

The end-user license stream name MUST be prefixed with "EUL-", with a base-32-encoded **GUID** as the remainder of the stream name.

The license stream MUST consist of an **EndUserLicenseHeader** structure (section 2.2.9), followed by a **UTF-8-LP-P4** string (section 2.1.3) containing XML specifying a **certificate chain**. The certificate chain MUST include a use license with an **enablingbits** element containing the symmetric content key encrypted with the user's RAC public key, as specified in [MS-RMPR] section 2.2.9.1.13. The XML in this string is derived from a **certificatechain** element as specified in [MS-RMPR] section 2.2.3.2. Each XrML **certificate** or license from a **certificate** element as specified in [MS-RMPR] section 2.2.3.1 is encoded as a **base64**-encoded Unicode string.

The certificate chain has been transformed in the following manner:

1. For each **certificate** element in the certificate chain:
    1. The XrML content of the **certificate** element is encoded as Unicode.
    2. Each resulting string is subsequently base64-encoded.
    3. Each resulting string is then placed in a **certificate** element.
2. The resulting collection of new **certificate** elements is accumulated in a **certificatechain** element.
3. The XML header `<?xml version="1.0"?>` is prefixed to the resulting **certificatechain** element.
4. The resulting XML is stored in the stream as a **UTF-8-LP-P4** string.

2.2.8 LicenseID

A **LicenseID** specifies the identity of a user as a Unicode string. The string MUST be of the form "Windows:<*emailaddr*>" or "Passport:<*emailaddr*>", where *emailaddr* represents a valid email address as specified in [RFC2822].

2.2.9 EndUserLicenseHeader

The **EndUserLicenseHeader** structure is a container for a **LicenseID** (section 2.2.8) as specified in [MS-RMPR].

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Length																															
ID_String (variable)																															
...																															

Length (4 bytes): An unsigned integer that specifies the size of the **EndUserLicenseHeader** structure.

ID_String (variable): A **UTF-8-LP-P4** structure (section 2.1.3) that contains a base64-encoded Unicode **LicenseID**.

2.2.10 Protected Content Stream

The protected content **stream** MUST be contained within the root storage. If the original document content is an ECMA-376 document [ECMA-376], the stream MUST be named "EncryptedPackage". For all other original document content types, it MUST be named "\0x09DRMContent".

The protected content stream has the following structure.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">Length</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">Contents (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

Length (8 bytes): An unsigned 64-bit integer that specifies the size, in bytes, of the plaintext data that is stored encrypted in the **Contents** field.

Contents (variable): Specifies the protected content. The protected content MUST be encrypted or decrypted with the content symmetric key encrypted for the user in the end-user license as specified in [MS-RMPR]. Protected content MUST be encrypted or decrypted using AES-128, a 16-byte block size, **electronic codebook (ECB)** mode, and an initialization vector of all zeros.

2.2.11 Viewer Content Stream

The viewer content **stream** MAY<6> be present. The purpose of the viewer content stream is to provide a MIME Encapsulation of Aggregate HTML Documents (MHTML) representation of the document to enable an application that cannot parse the protected content stream (section 2.2.10) to present a read-only representation of the document to the user. If the viewer content stream is present, the stream MUST be named "\0x09DRMViewerContent".

The viewer content stream has the following structure.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">Length</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">Contents (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

Length (8 bytes): An unsigned 64-bit integer that specifies the size, in bytes, of the compressed plaintext data stored encrypted in the **Contents** field.

Contents (variable): The MHTML representation of the protected content. The protected content MUST be encrypted or decrypted as specified in [MS-RMPR]. Once decrypted, the plaintext MUST be decompressed with the LZX compression algorithm, as described in [MSDN-CAB].

2.3 Encryption

This section specifies encryption and obfuscation. The four different techniques are:

- ECMA-376 encryption [ECMA-376], which leverages the data spaces storages specified in section 2.1.

- CryptoAPI RC4 encryption.
- RC4 encryption.
- XOR obfuscation.

ECMA-376 encryption [ECMA-376] also includes encryption using a third-party cryptography extension, which will be called *extensible encryption* in the remainder of this document.

2.3.1 EncryptionHeaderFlags

The **EncryptionHeaderFlags** structure specifies properties of the encryption algorithm used. It MUST be contained within an **EncryptionHeader** structure (section 2.3.2).

If the **fCryptoAPI** bit is set and the **fAES** bit is not set, RC4 encryption MUST be used. If the **fAES** encryption bit is set, a **block cipher** that supports **ECB** mode MUST be used. For compatibility with current implementations, **AES** encryption with a key length of 128, 192, or 256 bits SHOULD<7> be used.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
A	B	C	D	E	F	Unused																									

A – Reserved1 (1 bit): A value that MUST be 0 and MUST be ignored.

B – Reserved2 (1 bit): A value that MUST be 0 and MUST be ignored.

C – fCryptoAPI (1 bit): A flag that specifies whether CryptoAPI RC4 or ECMA-376 encryption [ECMA-376] is used. It MUST be 1 unless **fExternal** is 1. If **fExternal** is 1, it MUST be 0.

D – fDocProps (1 bit): A value that MUST be 0 if document properties are encrypted. The encryption of document properties is specified in section 2.3.5.4.

E – fExternal (1 bit): A value that MUST be 1 if extensible encryption is used. If this value is 1, the value of every other field in this structure MUST be 0.

F – fAES (1 bit): A value that MUST be 1 if the protected content is an ECMA-376 document [ECMA-376]; otherwise, it MUST be 0. If the **fAES** bit is 1, the **fCryptoAPI** bit MUST also be 1.

Unused (26 bits): A value that is undefined and MUST be ignored.

2.3.2 EncryptionHeader

The **EncryptionHeader** structure is used by ECMA-376 document encryption [ECMA-376] and Office binary document RC4 CryptoAPI encryption, as defined in section 2.3.5, to specify encryption properties for an encrypted **stream**.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Flags																															
SizeExtra																															
AlgID																															

AlgIDHash
KeySize
ProviderType
Reserved1
Reserved2
CSPName
...

Flags (4 bytes): An **EncryptionHeaderFlags** structure, as specified in section 2.3.1, that specifies properties of the encryption algorithm used.

SizeExtra (4 bytes): A field that is reserved and for which the value MUST be 0x00000000.

AlgID (4 bytes): A signed integer that specifies the encryption algorithm. It MUST be one of the values described in the following table.

Value	Algorithm
0x00000000	Determined by **Flags**
0x00006801	RC4
0x0000660E	128-bit AES
0x0000660F	192-bit AES
0x00006610	256-bit AES

The **Flags** field and **AlgID** field contain related values and MUST be set to one of the combinations in the following table.

Flags.fCryptoAPI	Flags.fAES	Flags.fExternal	AlgID	Algorithm
0	0	1	0x00000000	Determined by the application
1	0	0	0x00000000	RC4
1	0	0	0x00006801	RC4
1	1	0	0x00000000	128-bit AES
1	1	0	0x0000660E	128-bit AES
1	1	0	0x0000660F	192-bit AES
1	1	0	0x00006610	256-bit AES

AlgIDHash (4 bytes): A signed integer that specifies the hashing algorithm together with the **Flags.fExternal** bit. It MUST be one of the combinations in the following table.

AlgIDHash	Flags.fExternal	Algorithm
0x00000000	1	Determined by the application

AlgIDHash	Flags.fExternal	Algorithm
0x00000000	0	SHA-1
0x00008004	0	SHA-1

KeySize (4 bytes): An unsigned integer that specifies the number of bits in the encryption key. It MUST be a multiple of 8 and MUST be one of the values in the following table.

Algorithm	Value	Comment
Any	0x00000000	Determined by **Flags**
RC4	0x00000028 – 0x00000080 (inclusive)	8-bit increments
AES	0x00000080, 0x000000C0, 0x00000100	128-bit, 192-bit, or 256-bit

If the **Flags** field does not have the **fCryptoAPI** bit set, the **KeySize** field MUST be 0x00000000. If RC4 is used, the value MUST be compatible with the chosen **cryptographic service provider (CSP)**.

ProviderType (4 bytes): An implementation-specific value that corresponds to constants accepted by the specified CSP. It MUST be compatible with the chosen CSP. It SHOULD<8> be one of the following values.

Algorithm	Value	Comment
Any	0x00000000	Determined by **Flags**
RC4	0x00000001	
AES	0x00000018	

If the **Flags** field does not have the **fCryptoAPI** bit set, the **ProviderType** field MUST be 0x00000000.

Reserved1 (4 bytes): A value that is undefined and MUST be ignored.

Reserved2 (4 bytes): A value that MUST be 0x00000000 and MUST be ignored.

CSPName (variable): A null-terminated Unicode string that specifies the CSP name.

2.3.3 EncryptionVerifier

The **EncryptionVerifier** structure is used by Office Binary Document RC4 CryptoAPI Encryption (section 2.3.5) and ECMA-376 Document Encryption (section 2.3.4). Every usage of this structure MUST specify the hashing algorithm and encryption algorithm used in the **EncryptionVerifier** structure.

Verifier can be 16 bytes of data randomly generated each time the structure is created. **Verifier** is not stored in this structure directly.

The **EncryptionVerifier** structure MUST be set by using the following process:

1. Generate random data and write it into the **Salt** field.
2. Derive the encryption key from the password and **salt**, as specified in either section 2.3.4.7 or section 2.3.5.2, with block number 0.

3. Generate 16 bytes of additional random data as the **Verifier**.
4. Encrypt the result of step 3 and write it into the **EncryptedVerifier** field.
5. For the chosen hashing algorithm, obtain the size of the hash data and write this value into the **VerifierHashSize** field.
6. Obtain the hashing algorithm output by using as input the data generated in step 3.
7. Encrypt the hashing algorithm output from step 6 by using the chosen encryption algorithm, and write the output into the **EncryptedVerifierHash** field.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">SaltSize</td></tr>
<tr><td colspan="32">Salt (16 bytes)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">EncryptedVerifier (16 bytes)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">VerifierHashSize</td></tr>
<tr><td colspan="32">EncryptedVerifierHash (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

SaltSize (4 bytes): An unsigned integer that specifies the size of the **Salt** field. It MUST be 0x00000010.

Salt (16 bytes): An array of bytes that specifies the salt value used during password hash generation. It MUST NOT be the same data used for the verifier stored encrypted in the **EncryptedVerifier** field.

EncryptedVerifier (16 bytes): A value that MUST be the randomly generated **Verifier** value encrypted using the algorithm chosen by the implementation.

VerifierHashSize (4 bytes): An unsigned integer that specifies the number of bytes needed to contain the hash of the data used to generate the **EncryptedVerifier** field.

EncryptedVerifierHash (variable): An array of bytes that contains the encrypted form of the hash of the randomly generated **Verifier** value. The length of the array MUST be the size of the encryption block size multiplied by the number of blocks needed to encrypt the hash of the **Verifier**. If the encryption algorithm is RC4, the length MUST be 20 bytes. If the encryption algorithm is **AES**, the length MUST be 32 bytes. After decrypting the **EncryptedVerifierHash** field, only the first **VerifierHashSize** bytes MUST be used.

2.3.4 ECMA-376 Document Encryption

When an ECMA-376 document [ECMA-376] is encrypted as specified in [ECMA-376] Part 2 Annex C Table C-5 BIT 0, a structured storage utilizing the data spaces construct as specified in section 2.1 MUST be used. Unless exceptions are noted in the following subsections, **streams** and storages contained within the **\0x06DataSpaces** storage MUST be present as specified in section 2.1.1.

2.3.4.1 \0x06DataSpaces\DataSpaceMap Stream

The data space map MUST contain the following structure:

- The **\0x06DataSpaces\DataSpaceMap stream** MUST contain a **DataSpaceMap** structure (section 2.1.6) containing exactly one **DataSpaceMapEntry** structure (section 2.1.6.1).
- The **DataSpaceMapEntry** structure:
- MUST have a **DataSpaceName** equal to "StrongEncryptionDataSpace".
- MUST have exactly one **ReferenceComponents** entry with the name "EncryptedPackage" and the type 0x00000000, which signifies a stream.

2.3.4.2 \0x06DataSpaces\DataSpaceInfo Storage

The **DataSpaceInfo** storage MUST contain a **stream** that is defined as follows:

- The **\0x06DataSpaces\DataSpaceInfo** storage MUST contain a stream named "StrongEncryptionDataSpace", which MUST contain a **DataSpaceDefinition** structure (section 2.1.7).
- The **DataSpaceDefinition** structure MUST have exactly one **TransformReferences** entry, which MUST be "StrongEncryptionTransform".

2.3.4.3 \0x06DataSpaces\TransformInfo Storage

The **\0x06DataSpaces\TransformInfo** storage MUST contain one storage named "StrongEncryptionTransform". The "StrongEncryptionTransform" storage MUST contain a **stream** named "0x06Primary". The "0x06Primary" stream MUST contain an **IRMDSTransformInfo** structure (section 2.2.6). Within the **IRMDSTransformInfo** structure, the following values MUST be set:

- **TransformInfoHeader.TransformType** MUST be 0x00000001.
- **TransformInfoHeader.TransformID** MUST be "{FF9A3F03-56EF-4613-BDD5-5A41C1D07246}".
- **TransformInfoHeader.TransformName** MUST be "Microsoft.Container.EncryptionTransform".
- **TransformInfoHeader.ReaderVersion** MUST be "1.0".
- **TransformInfoHeader.UpdaterVersion** MUST be "1.0".
- **TransformInfoHeader.WriterVersion** MUST be "1.0".

Following the **IRMDSTransformInfo** structure, an **EncryptionTransformInfo** structure (section 2.1.9) MUST exist that specifies the encryption algorithms to be used. However, if the algorithms specified in the **EncryptionTransformInfo** structure differ from the algorithms specified in the **EncryptionInfo** stream (as specified in section 2.3.4.5, section 2.3.4.6, and section 2.3.4.10), the **EncryptionInfo** stream MUST be considered authoritative. If the agile encryption method is used, the **EncryptionName** field of the **EncryptionTransformInfo** structure MUST be a null string (0x00000000).

2.3.4.4 \EncryptedPackage Stream

The **\EncryptedPackage** stream is an encrypted **stream** of bytes containing the entire ECMA-376 source file [ECMA-376] in compressed form.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
StreamSize																															
...																															
EncryptedData (variable)																															
...																															

StreamSize (8 bytes): An unsigned integer that specifies the number of bytes used by data encrypted within the **EncryptedData** field, not including the size of the **StreamSize** field. Note that the actual size of the **\EncryptedPackage** stream can be larger than this value, depending on the block size of the chosen encryption algorithm

EncryptedData (variable): A block of data that is encrypted by using the algorithm specified within the **\EncryptionInfo** stream (section 2.3.4.5).

2.3.4.5 \EncryptionInfo Stream (Standard Encryption)

The **\EncryptionInfo stream** contains detailed information that is used to initialize the cryptography used to encrypt the **\EncryptedPackage** stream, as specified in section 2.3.4.4, when standard encryption is used.

If an external encryption provider is used, see section 2.3.4.6.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
EncryptionVersionInfo																															
EncryptionHeader.Flags																															
EncryptionHeaderSize																															
EncryptionHeader (variable)																															
...																															
EncryptionVerifier (variable)																															
...																															

EncryptionVersionInfo (4 bytes): A **Version** structure (section 2.1.4) where **Version.vMajor** MUST be 0x0002, 0x0003 or 0x0004<9>, and **Version.vMinor** MUST be 0x0002.

EncryptionHeader.Flags (4 bytes): A copy of the **Flags** stored in the **EncryptionHeader** field of this structure.

EncryptionHeaderSize (4 bytes): An unsigned integer that specifies the size, in bytes, of the **EncryptionHeader** field of this structure.

EncryptionHeader (variable): An **EncryptionHeader** structure (section 2.3.2) that specifies parameters used to encrypt data. The values MUST be set as specified in the following table.

Field	Value
Flags	The **fCryptoAPI** and **fAES** bits MUST be set. The **fDocProps** bit MUST be 0.
SizeExtra	This value MUST be 0x00000000.
AlgID	This value MUST be 0x0000660E (AES-128), 0x0000660F (AES-192), or 0x00006610 (AES-256).
AlgIDHash	This value MUST be 0x00008004 (SHA-1).
KeySize	This value MUST be 0x00000080 (AES-128), 0x000000C0 (AES-192), or 0x00000100 (AES-256).
ProviderType	This value SHOULD<10> be 0x00000018 (AES).
Reserved1	This value is undefined and MUST be ignored.
Reserved2	This value MUST be 0x00000000 and MUST be ignored.
CSPName	This value SHOULD<11> be set to either "Microsoft Enhanced RSA and AES Cryptographic Provider" or "Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)" as a null-terminated Unicode string.

EncryptionVerifier (variable): An **EncryptionVerifier** structure, as specified in section 2.3.3, that is generated as specified in section 2.3.4.8.

2.3.4.6 \EncryptionInfo Stream (Extensible Encryption)

ECMA-376 documents [ECMA-376] can optionally use user-provided custom (extensible) encryption modules. When extensible encryption is used, the **\EncryptionInfo stream** MUST contain the structure described in the following table.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">EncryptionVersionInfo</td></tr>
<tr><td colspan="32">EncryptionHeader.Flags</td></tr>
<tr><td colspan="32">EncryptionHeaderSize</td></tr>
<tr><td colspan="32">EncryptionHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">EncryptionInfo(variable)</td></tr>
</table>

...
EncryptionVerifier (variable)
...

EncryptionVersionInfo (4 bytes): A **Version** structure (section 2.1.4) where **Version.vMajor** MUST be 0x0003 or 0x0004 and **Version.vMinor** MUST be 0x0003.

EncryptionHeader.Flags (4 bytes): A copy of the **Flags** stored in the **EncryptionHeader** field of this structure as specified in section 2.3.1. It MUST have the **fExternal** bit set to 1. All other bits in this field MUST be set to 0.

EncryptionHeaderSize (4 bytes): An unsigned integer that specifies the size, in bytes, of the **EncryptionHeader** field of this structure, including the **GUID** specifying the extensible encryption module.

EncryptionHeader (variable): An **EncryptionHeader** structure (section 2.3.2) used to encrypt the structure. The values MUST be set as described in the following table.

Field	Value
Flags	A value that MUST have the **fExternal** bit set to 1. All other bits MUST be set to 0.
SizeExtra	A value that MUST be 0x00000000.
AlgID	A value that MUST be 0x00000000.
AlgIDHash	A value that MUST be 0x00000000.
KeySize	A value that MUST be 0x00000000.
ProviderType	A value that MUST be 0x00000000.
Reserved1	A value that is undefined and MUST be ignored.
Reserved2	A value that MUST be 0x00000000 and MUST be ignored.
CSPName	A unique identifier of an encryption module.<12>

EncryptionInfo (variable): A Unicode string that specifies an **EncryptionData** element. The first Unicode code point MUST be 0xFEFF.

The **EncryptionData** XML element MUST conform to the following XMLSchema namespace as specified by [W3C-XSD].

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="urn:schemas-microsoft-com:office:office"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">
  <xs:element name="EncryptionData">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="EncryptionProvider">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="EncryptionProviderData">
                <xs:simpleType>
```

```
                    <xs:restriction base="xs:base64Binary"/>
                  </xs:simpleType>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="Id" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:pattern value="\{[0-9A-Fa-f]{8}\-[0-9A-Fa-f]{4}\-
                     [0-9A-Fa-f]{4}\-[0-9A-Fa-f]{4}\-[0-9A-Fa-f]{12}\}"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="Url" type="xs:anyURI" use="required"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Element	Parent	Attribute	Value
EncryptionData			
EncryptionProvider	**EncryptionData**		
		Id	The GUID of the extensible encryption module, expressed as a string.
		Url	A **URL** where the extensible encryption module can be obtained.
EncryptionProviderData	**EncryptionProvider**		Base64-encoded data used by the extensible module.

EncryptionVerifier (variable): An **EncryptionVerifier** structure, as specified in section 2.3.3, that is generated as specified in section 2.3.4.8.

2.3.4.7 ECMA-376 Document Encryption Key Generation (Standard Encryption)

The encryption key for ECMA-376 document encryption [ECMA-376] MUST be generated by using the following method, which is derived from PKCS #5: Password-Based Cryptography Version 2.0 [RFC2898].

Let H() be a hashing algorithm as determined by the **EncryptionHeader.AlgIDHash** field, H_n be the hash data of the n^{th} iteration, and a plus sign (+) represent concatenation. This hashing algorithm MUST be **SHA-1**. The password MUST be provided as an array of Unicode characters. Limitations on the length of the password and the characters used by the password are implementation-dependent. The initial password hash is generated as follows:

- H_0 = H(salt + password)

The salt used MUST be generated randomly and MUST be 16 bytes in size. The salt MUST be stored in the **EncryptionVerifier.Salt** field contained within the **\EncryptionInfo stream** as specified in section 2.3.4.5. The hash is then iterated by using the following approach:

- H_n = H(iterator + H_{n-1})

where **iterator** is an unsigned 32-bit value that is initially set to 0x00000000 and then incremented monotonically on each iteration until 50,000 iterations have been performed. The value of **iterator** on the last iteration MUST be 49,999.

After the final hash data has been obtained, the encryption key MUST be generated by using the final hash data, and the block number MUST be 0x00000000. The encryption algorithm MUST be specified in the **EncryptionHeader.AlgID** field. The encryption algorithm MUST use **ECB** mode. The method used to generate the hash data that is the input into the key derivation algorithm is as follows:

- $H_{final} = H(H_n + block)$

The encryption key derivation method is specified by the following steps:

1. Let **cbRequiredKeyLength** be equal to the size, in bytes, of the required key length for the relevant encryption algorithm as specified by the **EncryptionHeader** structure. Note that **cbRequiredKeyLength** MUST be less than or equal to 40.

2. Let **cbHash** be the number of bytes output by the hashing algorithm H.

3. Form a 64-byte buffer by repeating the constant 0x36 64 times. XOR H_{final} into the first **cbHash** bytes of this buffer, and compute a hash of the resulting 64-byte buffer by using hashing algorithm H. This will yield a hash value of length **cbHash**. Let the resulting value be called **X1**.

4. Form another 64-byte buffer by repeating the constant 0x5C 64 times. XOR H_{final} into the first **cbHash** bytes of this buffer, and compute a hash of the resulting 64-byte buffer by using hash algorithm H. This yields a hash value of length **cbHash**. Let the resulting value be called **X2**.

5. Concatenate **X1** with **X2** to form **X3**, which will yield a value twice the length of **cbHash**.

6. Let **keyDerived** be equal to the first **cbRequiredKeyLength** bytes of **X3**.

2.3.4.8 Password Verifier Generation (Standard Encryption)

The password verifier uses an **EncryptionVerifier** structure as specified in section 2.3.3. The password verifier **Salt** field MUST be equal to the salt created during password key generation, as specified in section 2.3.4.7. A randomly generated verifier is then hashed using the SHA-1 hashing algorithm specified in the **EncryptionHeader** structure, and encrypted using the key generated as specified in section 2.3.4.7, with a block number of 0x00000000.

2.3.4.9 Password Verification (Standard Encryption)

Passwords MUST be verified by using the following steps:

1. Generate an encryption key as specified in section 2.3.4.7.

2. Decrypt the **EncryptedVerifier** field of the **EncryptionVerifier** structure as specified in section 2.3.3, and generated as specified in section 2.3.4.8, to obtain the **Verifier** value. The resulting **Verifier** value MUST be an array of 16 bytes.

3. Decrypt the **EncryptedVerifierHash** field of the **EncryptionVerifier** structure to obtain the hash of the **Verifier** value. The number of bytes used by the encrypted **Verifier** hash MUST be 32. The number of bytes used by the decrypted **Verifier** hash is given by the **VerifierHashSize** field, which MUST be 20.

4. Calculate the **SHA-1** hash value of the **Verifier** value calculated in step 2.

5. Compare the results of step 3 and step 4. If the two hash values do not match, the password is incorrect.

2.3.4.10 \EncryptionInfo Stream (Agile Encryption)

The **\EncryptionInfo stream** contains detailed information about the cryptography used to encrypt the **\EncryptedPackage** stream (section 2.3.4.4) when agile encryption is used.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">EncryptionVersionInfo</td></tr>
<tr><td colspan="32">Reserved</td></tr>
<tr><td colspan="32">XmlEncryptionDescriptor (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

EncryptionVersionInfo (4 bytes): A **Version** structure (section 2.1.4), where **Version.vMajor** MUST be 0x0004 and **Version.vMinor** MUST be 0x0004.

Reserved (4 bytes): A value that MUST be 0x00000040.

XmlEncryptionDescriptor (variable): An XML element that MUST conform to the following XML schema namespace, as specified in [W3C-XSD]:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
   targetNamespace="http://schemas.microsoft.com/office/2006/encryption"
   xmlns="http://schemas.microsoft.com/office/2006/encryption"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="ST_SaltSize">
    <xs:restriction base="xs:unsignedInt">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="65536" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ST_BlockSize">
    <xs:restriction base="xs:unsignedInt">
      <xs:minInclusive value="2" />
      <xs:maxInclusive value="4096" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ST_KeyBits">
    <xs:restriction base="xs:unsignedInt">
      <xs:minInclusive value="8" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ST_HashSize">
    <xs:restriction base="xs:unsignedInt">
      <xs:minInclusive value="1" />
      <xs:maxInclusive value="65536" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ST_SpinCount">
    <xs:restriction base="xs:unsignedInt">
      <xs:minInclusive value="0" />
      <xs:maxInclusive value="10000000" />
    </xs:restriction>
```

```
  </xs:simpleType>

  <xs:simpleType name="ST_CipherAlgorithm">
    <xs:restriction base="xs:string">
      <xs:minLength value="1" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ST_CipherChaining">
    <xs:restriction base="xs:string">
      <xs:minLength value="1" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ST_HashAlgorithm">
    <xs:restriction base="xs:string">
      <xs:minLength value="1" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="CT_KeyData">
    <xs:attribute name="saltSize" type="ST_SaltSize" use="required" />
    <xs:attribute name="blockSize" type="ST_BlockSize" use="required" />
    <xs:attribute name="keyBits" type="ST_KeyBits" use="required" />
    <xs:attribute name="hashSize" type="ST_HashSize" use="required" />
    <xs:attribute name="cipherAlgorithm" type="ST_CipherAlgorithm" use="required" />
    <xs:attribute name="cipherChaining" type="ST_CipherChaining" use="required" />
    <xs:attribute name="hashAlgorithm" type="ST_HashAlgorithm" use="required" />
    <xs:attribute name="saltValue" type="xs:base64Binary" use="required" />
  </xs:complexType>

  <xs:complexType name="CT_DataIntegrity">
    <xs:attribute name="encryptedHmacKey" type="xs:base64Binary" use="required" />
    <xs:attribute name="encryptedHmacValue" type="xs:base64Binary" use="required" />
  </xs:complexType>

  <xs:complexType name="CT_KeyEncryptor">
    <xs:sequence>
      <xs:any processContents="lax" />
    </xs:sequence>
    <xs:attribute name="uri" type="xs:token" />
  </xs:complexType>

  <xs:complexType name="CT_KeyEncryptors">
    <xs:sequence>
      <xs:element name="keyEncryptor" type="CT_KeyEncryptor" minOccurs="1"
maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="CT_Encryption">
    <xs:sequence>
      <xs:element name="keyData" type="CT_KeyData" minOccurs="1" maxOccurs="1" />
      <xs:element name="dataIntegrity" type="CT_DataIntegrity" minOccurs="0" maxOccurs="1" />
      <xs:element name="keyEncryptors" type="CT_KeyEncryptors" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>

  <xs:element name="encryption" type="CT_Encryption" />
</xs:schema>
```

SaltSize: An unsigned integer that specifies the number of bytes used by a salt. It MUST be at least 1 and no greater than 65,536.

BlockSize: An unsigned integer that specifies the number of bytes used to encrypt one block of data. It MUST be at least 2, no greater than 4096, and a multiple of 2.

KeyBits: An unsigned integer that specifies the number of bits used by an encryption algorithm. It MUST be at least 8 and a multiple of 8.

HashSize: An unsigned integer that specifies the number of bytes used by a hash value. It MUST be at least 1, no greater than 65,536, and the same number of bytes as the hash algorithm emits.

SpinCount: An unsigned integer that specifies the number of times to iterate on a hash of a password. It MUST NOT be greater than 10,000,000.

CipherAlgorithm: A string that specifies the cipher algorithm. The values in the following table are defined.

Value	Cipher algorithm
AES	MUST conform to the **AES** algorithm.
RC2	MUST conform to the algorithm as specified in [RFC2268].<13>
RC4	MUST NOT be used.
DES	MUST conform to the **DES** algorithm.<14>
DESX	MUST conform to the algorithm as specified in [DRAFT-DESX].<15>
3DES	MUST conform to the algorithm as specified in [RFC1851].<16>
3DES_112	MUST conform to the algorithm as specified in [RFC1851].<17>

Values that are not defined MAY<18> be used, and a compliant implementation is not required to support all defined values. The string MUST be at least 1 character.

CipherChaining: A string that specifies the chaining mode used by **CipherAlgorithm**. For more details about chaining modes, see [BCMO800-38A]. It MUST be one of the values described in the following table.

Value	Chaining mode
ChainingModeCBC	**Cipher block chaining (CBC)**
ChainingModeCFB	Cipher feedback chaining (CFB), with an 8-bit window

HashAlgorithm: A string specifying a hashing algorithm. The values described in the following table are defined.

Value	Hash algorithm
SHA-1	MUST conform to the algorithm as specified in [RFC4634].
SHA256	MUST conform to the algorithm as specified in [RFC4634].
SHA384	MUST conform to the algorithm as specified in [RFC4634].
SHA512	MUST conform to the algorithm as specified in

Value	Hash algorithm
	[RFC4634].
MD5	MUST conform to **MD5**.
MD4	MUST conform to the algorithm as specified in [RFC1320].
MD2	MUST conform to the algorithm as specified in [RFC1319].
RIPEMD-128	MUST conform to the hash functions specified in [ISO/IEC 10118].
RIPEMD-160	MUST conform to the hash functions specified in [ISO/IEC 10118].
WHIRLPOOL	MUST conform to the hash functions specified in [ISO/IEC 10118].

Values that are not defined MAY<19> be used, and a compliant implementation is not required to support all defined values. The string MUST be at least 1 character. For more information, see section 4.

KeyData: A complex type that specifies the encryption used within this element. The **saltValue** attribute is a base64-encoded binary value that is randomly generated. The number of bytes required to decode the **saltValue** attribute MUST be equal to the value of the **saltSize** attribute.

DataIntegrity: A complex type that specifies data used to verify whether the encrypted data passes an integrity check. It MUST be generated using the method specified in section 2.3.4.14. This type is composed of the following simple types:

- **encryptedHmacKey:** A base64-encoded value that specifies an encrypted key used in calculating the **encryptedHmacValue**.
- **encryptedHmacValue:** A base64-encoded value that specifies an **HMAC** derived from **encryptedHmacKey** and the encrypted data.

KeyEncryptor: A complex type that specifies the parameters used to encrypt an intermediate key, which is used to perform the final encryption of the document. To ensure extensibility, arbitrary elements can be defined to encrypt the intermediate key. The intermediate key MUST be the same for all **KeyEncryptor** elements. **PasswordKeyEncryptor** and **CertificateKeyEncryptor** are defined later in this section.

KeyEncryptors: A sequence of **KeyEncryptor** elements. Exactly one **KeyEncryptors** element MUST be present, and the **KeyEncryptors** element MUST contain at least one **KeyEncryptor**.

Encryption: A complex type composed of the following elements that specify the encryption properties:

- **keyData:** One **KeyData** element MUST be present.
- **dataIntegrity:** One **DataIntegrity** element MUST<20> be present.
- **keyEncryptors:** One **KeyEncryptors** sequence MUST be present.

The **KeyEncryptor** element, which MUST be used when encrypting password-protected agile encryption documents, is either a **PasswordKeyEncryptor** or a **CertificateKeyEncryptor**. Exactly one **PasswordKeyEncryptor** MUST be present. Zero or more **CertificateKeyEncryptor** elements are contained within the **KeyEncryptors** element. The **PasswordKeyEncryptor** is specified by the following schema:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
   targetNamespace="http://schemas.microsoft.com/office/2006/keyEncryptor/password"
   xmlns="http://schemas.microsoft.com/office/2006/keyEncryptor/password"
   xmlns:e="http://schemas.microsoft.com/office/2006/encryption"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:import namespace="http://schemas.microsoft.com/office/2006/encryption"
schemaLocation="encryptionInfo.xsd" />

  <xs:simpleType name="ST_PasswordKeyEncryptorUri">
    <xs:restriction base="xs:token">
      <xs:enumeration value="http://schemas.microsoft.com/office/2006/keyEncryptor/password"
/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="CT_PasswordKeyEncryptor">
    <xs:attribute name="saltSize" type="e:ST_SaltSize" use="required" />
    <xs:attribute name="blockSize" type="e:ST_BlockSize" use="required" />
    <xs:attribute name="keyBits" type="e:ST_KeyBits" use="required" />
    <xs:attribute name="hashSize" type="e:ST_HashSize" use="required" />
    <xs:attribute name="cipherAlgorithm" type="e:ST_CipherAlgorithm" use="required" />
    <xs:attribute name="cipherChaining" type="e:ST_CipherChaining" use="required" />
    <xs:attribute name="hashAlgorithm" type="e:ST_HashAlgorithm" use="required" />
    <xs:attribute name="saltValue" type="xs:base64Binary" use="required" />
    <xs:attribute name="spinCount" type="e:ST_SpinCount" use="required" />
    <xs:attribute name="encryptedVerifierHashInput" type="xs:base64Binary" use="required" />
    <xs:attribute name="encryptedVerifierHashValue" type="xs:base64Binary" use="required" />
    <xs:attribute name="encryptedKeyValue" type="xs:base64Binary" use="required" />
  </xs:complexType>

  <xs:element name="encryptedKey" type="CT_PasswordKeyEncryptor" />
</xs:schema>
```

saltSize: A **SaltSize** that specifies the size of the salt for a **PasswordKeyEncryptor**.

blockSize: A **BlockSize** that specifies the block size for a **PasswordKeyEncryptor**.

keyBits: A **KeyBits** that specifies the number of bits for a **PasswordKeyEncryptor**.

hashSize: A **HashSize** that specifies the size of the binary form of the hash for a **PasswordKeyEncryptor**.

cipherAlgorithm: A **CipherAlgorithm** that specifies the cipher algorithm for a **PasswordKeyEncryptor**. The cipher algorithm specified MUST be the same as the cipher algorithm specified for the **Encryption.keyData** element.

cipherChaining: A **CipherChaining** that specifies the cipher chaining mode for a **PasswordKeyEncryptor**.

hashAlgorithm: A **HashAlgorithm** that specifies the hashing algorithm for a **PasswordKeyEncryptor**. The hashing algorithm specified MUST be the same as the hashing algorithm specified for the **Encryption.keyData** element.

saltValue: A base64-encoded binary byte array that specifies the salt value for a **PasswordKeyEncryptor**. The number of bytes required by the decoded form of this element MUST be **saltSize**.

spinCount: A **SpinCount** that specifies the spin count for a **PasswordKeyEncryptor**.

encryptedVerifierHashInput: A base64-encoded value that specifies the encrypted verifier hash input for a **PasswordKeyEncryptor** used in password verification.

encryptedVerifierHashValue: A base64-encoded value that specifies the encrypted verifier hash value for a **PasswordKeyEncryptor** used in password verification.

encryptedKeyValue: A base64-encoded value that specifies the encrypted form of the intermediate key.

The **CertificateKeyEncryptor** is specified by the following schema:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://schemas.microsoft.com/office/2006/keyEncryptor/certificate"
xmlns="http://schemas.microsoft.com/office/2006/keyEncryptor/certificate"
xmlns:e="http://schemas.microsoft.com/office/2006/encryption"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:import namespace="http://schemas.microsoft.com/office/2006/encryption"
schemaLocation="encryptionInfo.xsd" />
  <xs:simpleType name="ST_PasswordKeyEncryptorUri">
    <xs:restriction base="xs:token">
      <xs:enumeration
value="http://schemas.microsoft.com/office/2006/keyEncryptor/certificate" />
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="CT_CertificateKeyEncryptor">
    <xs:attribute name="encryptedKeyValue" type="xs:base64Binary" use="required" />
    <xs:attribute name="X509Certificate" type="xs:base64Binary" use="required" />
    <xs:attribute name="certVerifier" type="xs:base64Binary" use="required" />
  </xs:complexType>
  <xs:element name="encryptedKey" type="CT_CertificateKeyEncryptor" />
</xs:schema>
```

encryptedKeyValue: A base64-encoded value that specifies the encrypted form of the intermediate key, which is encrypted with the public key contained within the **X509Certificate** attribute.

X509Certificate: A base64-encoded value that specifies a **DER**-encoded **X.509 certificate** used to encrypt the intermediate key. The certificate MUST contain only the public portion of the public-private key pair.

certVerifier: A base64-encoded value that specifies the HMAC of the binary data obtained by base64-decoding the **X509Certificate** attribute. The hashing algorithm used to derive the HMAC MUST be the hashing algorithm specified for the **Encryption.keyData** element. The secret key used to derive the HMAC MUST be the intermediate key.

If the intermediate key is reset, any **CertificateKeyEncryptor** elements are also reset to contain the new intermediate key, except that the **certVerifier** attribute MUST match the value calculated using the current intermediate key, to verify that the **CertificateKeyEncryptor** element actually encrypted the current intermediate key. If a **CertificateKeyEncryptor** element does not have a correct **certVerifier** attribute, it MUST be discarded.

2.3.4.11 Encryption Key Generation (Agile Encryption)

The encryption key for ECMA-376 document encryption [ECMA-376] using agile encryption MUST be generated by using the following method, which is derived from PKCS #5: Password-Based Cryptography Version 2.0 [RFC2898].

Let H() be a hashing algorithm as determined by the **PasswordKeyEncryptor.hashAlgorithm** element, H_n be the hash data of the n^{th} iteration, and a plus sign (+) represent concatenation. The password MUST be provided as an array of Unicode characters. Limitations on the length of the password and the characters used by the password are implementation-dependent. The initial password hash is generated as follows:

- H_0 = H(salt + password)

The salt used MUST be generated randomly. The salt MUST be stored in the **PasswordKeyEncryptor.saltValue** element contained within the **\EncryptionInfo stream** as specified in section 2.3.4.10. The hash is then iterated by using the following approach:

- $H_n = H(\text{iterator} + H_{n-1})$

where **iterator** is an unsigned 32-bit value that is initially set to 0x00000000 and then incremented monotonically on each iteration until **PasswordKey.spinCount** iterations have been performed. The value of **iterator** on the last iteration MUST be one less than **PasswordKey.spinCount**.

The final hash data that is used for an encryption key is then generated by using the following method:

- $H_{final} = H(H_n + \text{blockKey})$

where **blockKey** represents an array of bytes used to prevent two different blocks from encrypting to the same cipher text.

If the size of the resulting H_{final} is smaller than that of **PasswordKeyEncryptor.keyBits**, the key MUST be padded by appending bytes with a value of 0x36. If the hash value is larger in size than **PasswordKeyEncryptor.keyBits**, the key is obtained by truncating the hash value.

2.3.4.12 Initialization Vector Generation (Agile Encryption)

Initialization vectors are used in all cases for agile encryption. An initialization vector MUST be generated by using the following method, where H() is a hash function that MUST be the same as specified in section 2.3.4.11 and a plus sign (+) represents concatenation:

1. If a **blockKey** is provided, let **IV** be a hash of the **KeySalt** and the following value:
    1. blockKey:IV = H(KeySalt + blockKey)
2. If a **blockKey** is not provided, let **IV** be equal to the following value:
    1. KeySalt:IV = KeySalt.
3. If the number of bytes in the value of **IV** is less than the value of the **blockSize** attribute corresponding to the **cipherAlgorithm** attribute, pad the array of bytes by appending 0x36 until the array is **blockSize** bytes. If the array of bytes is larger than **blockSize** bytes, truncate the array to **blockSize** bytes.

2.3.4.13 PasswordKeyEncryptor Generation (Agile Encryption)

For agile encryption, the password key encryptor XML element specified in section 2.3.4.10 MUST be created as follows:

saltSize: Set this attribute to the number of bytes used by the binary form of the **saltValue** attribute. It MUST conform to a **SaltSize** type.

blockSize: Set this attribute to the number of bytes needed to contain an encrypted block of data, as defined by the **cipherAlgorithm** used. It MUST conform to a **BlockSize** type.

keyBits: Set this attribute to the number of bits needed to contain an encryption key, as defined by the **cipherAlgorithm** used. It MUST conform to a **KeyBits** type.

hashSize: Set this attribute to the number of bytes needed to contain the output of the hashing algorithm defined by the **hashAlgorithm** element. It MUST conform to a **HashSize** type.

cipherAlgorithm: Set this attribute to a string containing the cipher algorithm used to encrypt the **encryptedVerifierHashInput**, **encryptedVerifierHashValue**, and **encryptedKeyValue**. It MUST conform to a **CipherAlgorithm** type.

cipherChaining: Set this attribute to the cipher chaining mode used to encrypt **encryptedVerifierHashInput**, **encryptedVerifierHashValue**, and **encryptedKeyValue**. It MUST conform to a **CipherChaining** type.

hashAlgorithm: Set this attribute to the hashing algorithm used to derive the encryption key from the password and that is also used to obtain the **encryptedVerifierHashValue**. It MUST conform to a **HashAlgorithm** type.

saltValue: Set this attribute to a base64-encoded, randomly generated array of bytes. It MUST conform to a **SaltValue** type. The number of bytes required by the decoded form of this element MUST be **saltSize**.

spinCount: Set this attribute to the number of times to iterate the password hash when creating the key used to encrypt the **encryptedVerifierHashInput**, **encryptedVerifierHashValue**, and **encryptedKeyValue**. It MUST conform to a **SpinCount** type.

encryptedVerifierHashInput: This attribute MUST be generated by using the following steps:

1. Generate a random array of bytes with the number of bytes used specified by the **saltSize** attribute.
2. Generate an encryption key as specified in section 2.3.4.11 by using the user-supplied password, the binary byte array used to create the **saltValue** attribute, and a **blockKey** byte array consisting of the following bytes: 0xfe, 0xa7, 0xd2, 0x76, 0x3b, 0x4b, 0x9e, and 0x79.
3. Encrypt the random array of bytes generated in step 1 by using the binary form of the **saltValue** attribute as an initialization vector as specified in section 2.3.4.12. If the array of bytes is not an integral multiple of **blockSize** bytes, pad the array with 0x00 to the next integral multiple of **blockSize** bytes.
4. Use base64 to encode the result of step 3.

encryptedVerifierHashValue: This attribute MUST be generated by using the following steps:

1. Obtain the hash value of the random array of bytes generated in step 1 of the steps for **encryptedVerifierHashInput**.
2. Generate an encryption key as specified in section 2.3.4.11 by using the user-supplied password, the binary byte array used to create the **saltValue** attribute, and a **blockKey** byte array consisting of the following bytes: 0xd7, 0xaa, 0x0f, 0x6d, 0x30, 0x61, 0x34, and 0x4e.
3. Encrypt the hash value obtained in step 1 by using the binary form of the **saltValue** attribute as an initialization vector as specified in section 2.3.4.12. If **hashSize** is not an integral multiple of **blockSize** bytes, pad the hash value with 0x00 to an integral multiple of **blockSize** bytes.
4. Use base64 to encode the result of step 3.

encryptedKeyValue: This attribute MUST be generated by using the following steps:

1. Generate a random array of bytes that is the same size as specified by the **Encryptor.KeyData.keyBits** attribute of the parent element.
2. Generate an encryption key as specified in section 2.3.4.11, using the user-supplied password, the binary byte array used to create the **saltValue** attribute, and a **blockKey** byte array consisting of the following bytes: 0x14, 0x6e, 0x0b, 0xe7, 0xab, 0xac, 0xd0, and 0xd6.

3. Encrypt the random array of bytes generated in step 1 by using the binary form of the **saltValue** attribute as an initialization vector as specified in section 2.3.4.12. If the array of bytes is not an integral multiple of **blockSize** bytes, pad the array with 0x00 to an integral multiple of **blockSize** bytes.

4. Use base64 to encode the result of step 3.

2.3.4.14 DataIntegrity Generation (Agile Encryption)

The **DataIntegrity** element contained within an **Encryption** element MUST be generated by using the following steps:

1. Obtain the intermediate key by decrypting the **encryptedKeyValue** from a **KeyEncryptor** contained within the **KeyEncryptors** sequence. Use this key for encryption operations in the remaining steps of this section.

2. Generate a random array of bytes, known as **Salt**, of the same length as the value of the **KeyData.saltSize** attribute.

3. Encrypt the random array of bytes generated in step 2 by using the binary form of the **KeyData.saltValue** attribute and a **blockKey** byte array consisting of the following bytes: 0x5f, 0xb2, 0xad, 0x01, 0x0c, 0xb9, 0xe1, and 0xf6 used to form an initialization vector as specified in section 2.3.4.12. If the array of bytes is not an integral multiple of **blockSize** bytes, pad the array with 0x00 to the next integral multiple of **blockSize** bytes.

4. Assign the **encryptedHmacKey** attribute to the base64-encoded form of the result of step 3.

5. Generate an **HMAC**, as specified in [RFC2104], of the encrypted form of the data (message), which the **DataIntegrity** element will verify by using the **Salt** generated in step 2 as the key. Note that the entire **EncryptedPackage** **stream**, including the **StreamSize** field, MUST be used as the message.

6. Encrypt the HMAC as in step 3 by using a **blockKey** byte array consisting of the following bytes: 0xa0, 0x67, 0x7f, 0x02, 0xb2, 0x2c, 0x84, and 0x33.

7. Assign the **encryptedHmacValue** attribute to the base64-encoded form of the result of step 6.

2.3.4.15 Data Encryption (Agile Encryption)

The **EncryptedPackage** **stream** MUST be encrypted in 4096-byte segments to facilitate nearly random access while allowing **CBC** modes to be used in the encryption process.

The initialization vector for the encryption process MUST be obtained by using the zero-based segment number as a **blockKey** and the binary form of the **KeyData.saltValue** as specified in section 2.3.4.12. The block number MUST be represented as a 32-bit unsigned integer.

Data blocks MUST then be encrypted by using the initialization vector and the intermediate key obtained by decrypting the **encryptedKeyValue** from a **KeyEncryptor** contained within the **KeyEncryptors** sequence as specified in section 2.3.4.10. The final data block MUST be padded to the next integral multiple of the **KeyData.blockSize** value. Any padding bytes can be used. Note that the **StreamSize** field of the **EncryptedPackage** stream specifies the number of bytes of unencrypted data as specified in section 2.3.4.4.

2.3.5 Office Binary Document RC4 CryptoAPI Encryption

The storages and **streams** encrypted by Office binary document RC4 CryptoAPI encryption are specified in the documentation for the relevant application; for more information see [MS-DOC], [MS-

[XLS], and [MS-PPT]. The following subsections specify the structures and key generation methods used by the application.

2.3.5.1 RC4 CryptoAPI Encryption Header

The encryption header structure used for RC4 CryptoAPI encryption is specified as shown in the following diagram.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">EncryptionVersionInfo</td></tr>
<tr><td colspan="32">EncryptionHeader.Flags</td></tr>
<tr><td colspan="32">EncryptionHeaderSize</td></tr>
<tr><td colspan="32">EncryptionHeader (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">EncryptionVerifier (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

EncryptionVersionInfo (4 bytes): A **Version** structure (section 2.1.4) that specifies the encryption version used to create the document and the encryption version required to open the document. **Version.vMajor** MUST be 0x0002, 0x0003, or 0x0004<21> and **Version.vMinor** MUST be 0x0002.

EncryptionHeader.Flags (4 bytes): A copy of the **Flags** stored in the **EncryptionHeader** structure (section 2.3.2) that is stored in this **stream**.

EncryptionHeaderSize (4 bytes): An unsigned integer that specifies the size, in bytes, of the **EncryptionHeader** structure.

EncryptionHeader (variable): An **EncryptionHeader** structure (section 2.3.2) used to encrypt the structure. The values MUST be set as described in the following table.

Field	Value
Flags	The **fCryptoAPI** bit MUST be set. The **fDocProps** bit MUST be set if the document properties are not encrypted.
SizeExtra	MUST be 0x00000000.
AlgID	MUST be 0x00006801 (RC4 encryption).
AlgIDHash	MUST be 0x00008004 (SHA-1).
KeySize	MUST be greater than or equal to 0x00000028 bits and less than or equal to 0x00000080 bits, in increments of 8 bits. If set to 0x00000000, it MUST be interpreted as 0x00000028 bits. It MUST be compatible with the chosen cryptographic service provider (CSP).

Field	Value
ProviderType	MUST be 0x00000001.
Reserved1	Undefined and MUST be ignored.
Reserved2	MUST be 0x00000000 and MUST be ignored.
CSPName	MUST be set to a recognized CSP name that supports RC4 and SHA-1 algorithms with a key length compatible with the **KeySize** field value.<22>

EncryptionVerifier (variable): An **EncryptionVerifier** structure as specified in section 2.3.3 that is generated as specified in section 2.3.5.5.

2.3.5.2 RC4 CryptoAPI Encryption Key Generation

The encryption key for RC4 CryptoAPI binary document encryption MUST be generated by using the following approach.

Let H() be a hashing algorithm as determined by the **EncryptionHeader.AlgIDHash** field, and a plus sign (+) represents concatenation. The password MUST be provided as an array of Unicode characters.

Limitations on the length of the password and the characters used by the password are implementation-dependent. For details about behavior variations, see [MS-DOC], [MS-XLS], and [MS-PPT]. Unless otherwise specified, the maximum password length MUST be 255 Unicode characters.

The password hash is generated as follows:

- H_0 = H(salt + password)

The **salt** used MUST be generated randomly and MUST be 16 bytes in size. The **salt** MUST be stored in the **EncryptionVerifier.Salt** field as specified in section 2.3.4.5. Note that the hash MUST NOT be iterated. See section 4 for additional notes.

After the hash has been obtained, the encryption key MUST be generated by using the hash data and a block number that is provided by the application. The encryption algorithm MUST be specified in the **EncryptionHeader.AlgID** field.

The method used to generate the hash data that is the input into the key derivation algorithm is as follows:

- H_{final} = H(H_0 + block)

The block number MUST be a 32-bit unsigned value provided by the application.

Let **keyLength** be the key length, in bits, as specified by the RC4 CryptoAPI Encryption Header **KeySize** field.

The first **keyLength** bits of H_{final} MUST be considered the derived encryption key, unless **keyLength** is exactly 40 bits long. An SHA-1 hash is 160 bits long, and the maximum RC4 key length is 128 bits; therefore, **keyLength** MUST be less than or equal to 128 bits. If **keyLength** is exactly 40 bits, the encryption key MUST be composed of the first 40 bits of H_{final} and 88 bits set to zero, creating a 128-bit key.

2.3.5.3 RC4 CryptoAPI EncryptedStreamDescriptor Structure

The RC4 CryptoAPI **EncryptedStreamDescriptor** structure specifies information about encrypted **streams** and storages contained within an RC4 CryptoAPI Encrypted Summary stream as specified in section 2.3.5.4. It is specified as shown in the following diagram.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">StreamOffset</td></tr>
<tr><td colspan="32">StreamSize</td></tr>
<tr><td colspan="16">Block</td><td colspan="8">NameSize</td><td>A</td><td>B</td><td colspan="6">Unused</td></tr>
<tr><td colspan="32">Reserved2</td></tr>
<tr><td colspan="32">StreamName (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

StreamOffset (4 bytes): An unsigned integer that specifies the offset, in bytes, within the summary stream where the encrypted stream is written.

StreamSize (4 bytes): An unsigned integer that specifies the size, in bytes, of the encrypted stream.

Block (2 bytes): An unsigned integer that specifies the block number used to derive the encryption key for this encrypted stream.

NameSize (1 byte): An unsigned integer that specifies the number of characters used by the **StreamName** field, not including the terminating NULL character.

A – fStream (1 bit): A value that MUST be 1 if the encrypted data is a stream. It MUST be 0 if the encrypted data is a storage.

B – Reserved1 (1 bit): A value that MUST be 0 and MUST be ignored.

Unused (6 bits): A value that MUST be ignored.

Reserved2 (4 bytes): A value that MUST be ignored.

StreamName (variable): A null-terminated Unicode string specifying the name of the stream (or storage) stored within the encrypted summary stream.

2.3.5.4 RC4 CryptoAPI Encrypted Summary Stream

When RC4 CryptoAPI encryption is used, an encrypted summary **stream** MAY<23> be created. The name of the stream MUST be specified by the application. If the encrypted summary stream is present, the **\0x05DocumentSummaryInformation** stream MUST be present, MUST conform to the details as specified in [MS-OSHARED] section 2.3.3.2, and MUST contain no properties. The **\0x05SummaryInformation** stream MUST NOT be present.

For details about the contents of the **\0x05SummaryInformation** and **\0x05DocumentSummaryInformation** streams, see [MS-OSHARED] section 2.3.3.2.1 and [MS-OSHARED] section 2.3.3.2.2.

For brevity, this section refers to the RC4 CryptoAPI Encrypted Summary stream as the *encrypted summary* stream.

The stream MUST have the format that is shown in the following diagram.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">StreamDescriptorArrayOffset</td></tr>
<tr><td colspan="32">StreamDescriptorArraySize</td></tr>
<tr><td colspan="32">EncryptedStreamData (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">EncryptedStreamDescriptorCount</td></tr>
<tr><td colspan="32">EncryptedStreamDescriptorArray (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

StreamDescriptorArrayOffset (4 bytes): An unsigned integer that specifies the offset within the encrypted summary stream where the **EncryptedStreamDescriptorCount** structure is found.

StreamDescriptorArraySize (4 bytes): An unsigned integer that specifies the number of bytes used by the **EncryptedStreamDescriptorArray** structure.

EncryptedStreamData (variable): One or more encrypted streams stored within the encrypted summary stream.

EncryptedStreamDescriptorCount (4 bytes): An encrypted unsigned integer specifying the count of **EncryptedStreamDescriptor** structures (section 2.3.5.3).

EncryptedStreamDescriptorArray (variable): One or more **EncryptedStreamDescriptor** structures that specify the offsets and names of the encrypted streams and storages contained within the encrypted summary stream.

The encrypted summary stream MUST be written as specified in the following steps:

1. Seek forward from the start of the encrypted summary stream by 8 bytes to provide space for the **StreamDescriptorArrayOffset** and **StreamDescriptorArraySize** fields, which will be written in step 8. Let **BlockNumber** initially be 0x00000000.

2. If additional streams or storages are provided by the application, for each stream or storage the following steps MUST be performed:

    1. If the data is contained within a stream, retrieve the contents of the stream. Initialize the encryption key as specified in section 2.3.5.2, using a block number of 0x00000000, and encrypt the stream data. Write the encrypted bytes into the encrypted summary stream.

    2. If the data is contained within a storage, convert the storage into a file as specified in [MS-CFB]. Initialize the encryption key as specified in section 2.3.5.2, using a block number of **BlockNumber**, and encrypt the storage data as a stream of bytes. Write the encrypted bytes into the encrypted summary stream.

    3. Set the fields within the associated **EncryptedStreamDescriptor** for the stream or storage. Do not write it to the encrypted summary stream yet.

    4. Increment **BlockNumber**.

3. Generate or retrieve the entire contents of the **\0x05SummaryInformation** stream. Initialize the encryption key as specified in section 2.3.5.2, using a block number of **BlockNumber**, and encrypt the **\0x05SummaryInformationStream** data. Write the encrypted bytes into the encrypted summary stream. Increment **BlockNumber**.

4. Set the fields within the associated **EncryptedStreamDescriptor** for the **\0x05SummaryInformation** stream. Do not write it to the encrypted summary stream yet.

5. Generate or retrieve data contained within the **\0x05DocumentSummaryInformation** stream. Initialize the encryption key as specified in section 2.3.5.2, using a block number of **BlockNumber**, and encrypt the **\0x05DocumentSummaryInformationStream** data. Write the encrypted bytes into the encrypted summary stream immediately following the data written in step 2.

6. Set the fields within the associated **EncryptedStreamDescriptor** for the **\0x05DocumentSummaryInformation** stream. Do not write it to the encrypted summary stream yet.

7. Write the **EncryptedStreamDescriptorCount** and **EncryptedStreamDescriptorArray** by initializing the encryption key as specified in section 2.3.5.2, using a block number of 0x00000000. Concatenate and encrypt the **EncryptedStreamDescriptorCount** and the **EncryptedStreamDescriptor**. Write the encrypted bytes into the encrypted summary stream.

8. Initialize the **StreamDescriptorArrayOffset** and **StreamDescriptorArraySize** fields to specify the encrypted location of the **EncryptedStreamDescriptorCount** and size of the **EncryptedStreamDescriptorCount** and **EncryptedStreamDescriptorArray** within the encrypted summary stream. Initialize the encryption key as specified in section 2.3.5.2, using a block number of 0x00000000.

2.3.5.5 Password Verifier Generation

The password verifier uses an **EncryptionVerifier** structure, as specified in section 2.3.3. The password verifier **Salt** field MUST be populated with the salt created during password key generation, as specified in section 2.3.5.2. An additional 16-byte verifier is then hashed using the SHA-1 hashing algorithm specified in the encryption header structure, and encrypted using the key generated in section 2.3.5.2, with a block number of 0x00000000.

2.3.5.6 Password Verification

The password verification process is specified by the following steps:

1. Generate an encryption key as specified in section 2.3.3, using a block number of 0x00000000.

2. Decrypt the **EncryptedVerifier** field of the **EncryptionVerifier** structure to obtain the **Verifier** value. The resulting **Verifier** value MUST be an array of 16 bytes.

3. Decrypt the **EncryptedVerifierHash** field of the **EncryptionVerifier** structure to obtain the hash of the **Verifier** value. The number of bytes used by the encrypted **Verifier** hash MUST be 20.

4. Calculate the SHA-1 hash value of the **Verifier** value calculated in step 2.

5. Compare the results of step 3 and step 4. If the two hash values do not match, the password is incorrect.

The RC4 decryption **stream** MUST NOT be reset between the two decryption operations specified in steps 2 and 3.

2.3.6 Office Binary Document RC4 Encryption

Office binary document RC4 encryption does not alter the storages and **streams** used. If a stream is encrypted, it is encrypted in place. The following subsections specify the structures and key generation methods used by the application.

2.3.6.1 RC4 Encryption Header

The encryption header used for RC4 encryption is specified as shown in the following diagram.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
EncryptionVersionInfo																															
Salt (16 bytes)																															
...																															
...																															
EncryptedVerifier (16 bytes)																															
...																															
...																															
EncryptedVerifierHash (16 bytes)																															
...																															
...																															

EncryptionVersionInfo (4 bytes): A **Version** structure (section 2.1.4), where **Version.vMajor** MUST be 0x0001 and **Version.vMinor** MUST be 0x0001.

Salt (16 bytes): A randomly generated array of bytes that specifies the salt value used during password hash generation. It MUST NOT be the same data used for the verifier stored encrypted in the **EncryptedVerifier** field.

EncryptedVerifier (16 bytes): An additional 16-byte verifier encrypted using a 40-bit RC4 cipher initialized as specified in section 2.3.6.2, with a block number of 0x00000000.

EncryptedVerifierHash (16 bytes): A 40-bit RC4 encrypted **MD5** hash of the verifier used to generate the **EncryptedVerifier** field.

2.3.6.2 Encryption Key Derivation

The encryption key for Office binary document RC4 encryption is generated by using the following method: Let H() be the MD5 hashing algorithm, H_n be the hash data of the n^{th} iteration, and a plus sign (+) represent concatenation. The password MUST be provided as an array of Unicode characters.

Limitations on the length of the password and the characters used by the password are implementation-dependent. For details about behavior variations, see [MS-DOC] and [MS-XLS]. Unless otherwise specified, the maximum password length MUST be 255 Unicode characters.

The initial password hash is generated as follows.

- H_0 = H(password)

The salt used MUST be generated randomly and MUST be 16 bytes in size. The salt MUST be stored in the **Salt** field of the **RC4 Encryption Header** structure (section 2.3.6.1). The hash is then computed by using the following approach:

1. Let **TruncatedHash** be the first 5 bytes of H_0.
2. Let **IntermediateBuffer** be a 336-byte buffer.
3. Form a 21-byte buffer by concatenating **TruncatedHash** plus the salt. Initialize **IntermediateBuffer** by copying the 21-byte buffer into **IntermediateBuffer** a total of 16 times.
4. Use the following: H_1 = H(IntermediateBuffer).

After the final hash has been obtained, the encryption key MUST be generated by using the first 5 bytes of the final hash data and a block number that is provided by the application. The encryption algorithm MUST be RC4. The method used to generate the hash data that is the input into the key derivation algorithm is the following:

- Let **TruncatedHash** be the first 5 bytes of H_1.
- Use the following: H_{final} equals H(TruncatedHash + block).

The block number MUST be a 32-bit unsigned value provided by the application.

The first 128 bits of H_{final} MUST then be used as the derived encryption key.

2.3.6.3 Password Verifier Generation

The password verifier uses a **BinaryRC4EncryptionHeader** structure, as specified in section 2.3.6.1. The password verifier **Salt** field MUST be populated with the salt created during password key generation, as specified in section 2.3.6.2. An additional 16-byte verifier is then hashed by using the MD5 hashing algorithm and encrypted by using the key generated in section 2.3.6.2, with a block number of 0x00000000.

The RC4 decryption **stream** MUST NOT be reset between decrypting **EncryptedVerifier** and **EncryptedVerifierHash**.

2.3.6.4 Password Verification

The password verification process is specified by the following steps:

1. Generate an encryption key as specified in section 2.3.6.2, using a block number of 0x00000000.
2. Decrypt the **EncryptedVerifier** field of the RC4 Encryption Header structure to obtain the **Verifier** value. The resulting **Verifier** value MUST be an array of 16 bytes.

3. Decrypt the **EncryptedVerifierHash** field of the RC4 Encryption Header structure to obtain the hash of the **Verifier** value. The number of bytes used by the encrypted **Verifier** hash MUST be 16.
4. Calculate the MD5 hash value of the results of step 2.
5. Compare the results of step 3 and step 4. If the two hash values do not match, the password is incorrect.

The RC4 decryption **stream** MUST NOT be reset between decrypting **EncryptedVerifier** and **EncryptedVerifierHash**.

2.3.7 XOR Obfuscation

XOR obfuscation is supported for backward compatibility with older file formats.

2.3.7.1 Binary Document Password Verifier Derivation Method 1

The **CreatePasswordVerifier_Method1** procedure specifies how a 16-bit password verifier is obtained from an ASCII password string. The password verifier is used in XOR obfuscation as well as for document write protection.

The **CreatePasswordVerifier_Method1** procedure takes the following parameter:

- **Password:** An ASCII string that specifies the password to be used when generating the verifier.

```
FUNCTION CreatePasswordVerifier_Method1
    PARAMETERS Password
    RETURNS 16-bit unsigned integer

    DECLARE Verifier AS 16-bit unsigned integer
    DECLARE PasswordArray AS array of 8-bit unsigned integers

    SET Verifier TO 0x0000
    SET PasswordArray TO (empty array of bytes)
    SET PasswordArray[0] TO Password.Length
    APPEND Password TO PasswordArray

    FOR EACH PasswordByte IN PasswordArray IN REVERSE ORDER
       IF (Verifier BITWISE AND 0x4000) is 0x0000
          SET Intermediate1 TO 0
       ELSE
          SET Intermediate1 TO 1
       ENDIF

       SET Intermediate2 TO Verifier MULTIPLED BY 2
       SET most significant bit of Intermediate2 TO 0

       SET Intermediate3 TO Intermediate1 BITWISE OR Intermediate2
       SET Verifier TO Intermediate3 BITWISE XOR PasswordByte
    ENDFOR

    RETURN Verifier BITWISE XOR 0xCE4B
END FUNCTION
```

For more information, see section 4.

2.3.7.2 Binary Document XOR Array Initialization Method 1

The **CreateXorArray_Method1** procedure specifies how a 16-byte XOR obfuscation array is initialized. The procedure takes the following parameter:

- **Password:** An ASCII string that specifies the password to be used to encrypt the data. *Password* MUST NOT be longer than 15 characters.

```
SET PadArray TO ( 0xBB, 0xFF, 0xFF, 0xBA, 0xFF, 0xFF, 0xB9, 0x80,
                  0x00, 0xBE, 0x0F, 0x00, 0xBF, 0x0F, 0x00 )

SET InitialCode TO ( 0xE1F0, 0x1D0F, 0xCC9C, 0x84C0, 0x110C,
                     0x0E10, 0xF1CE, 0x313E, 0x1872, 0xE139,
                     0xD40F, 0x84F9, 0x280C, 0xA96A, 0x4EC3 )

SET XorMatrix TO ( 0xAEFC, 0x4DD9, 0x9BB2, 0x2745, 0x4E8A, 0x9D14, 0x2A09,
                   0x7B61, 0xF6C2, 0xFDA5, 0xEB6B, 0xC6F7, 0x9DCF, 0x2BBF,
                   0x4563, 0x8AC6, 0x05AD, 0x0B5A, 0x16B4, 0x2D68, 0x5AD0,
                   0x0375, 0x06EA, 0x0DD4, 0x1BA8, 0x3750, 0x6EA0, 0xDD40,
                   0xD849, 0xA0B3, 0x5147, 0xA28E, 0x553D, 0xAA7A, 0x44D5,
                   0x6F45, 0xDE8A, 0xAD35, 0x4A4B, 0x9496, 0x390D, 0x721A,
                   0xEB23, 0xC667, 0x9CEF, 0x29FF, 0x53FE, 0xA7FC, 0x5FD9,
                   0x47D3, 0x8FA6, 0x0F6D, 0x1EDA, 0x3DB4, 0x7B68, 0xF6D0,
                   0xB861, 0x60E3, 0xC1C6, 0x93AD, 0x377B, 0x6EF6, 0xDDEC,
                   0x45A0, 0x8B40, 0x06A1, 0x0D42, 0x1A84, 0x3508, 0x6A10,
                   0xAA51, 0x4483, 0x8906, 0x022D, 0x045A, 0x08B4, 0x1168,
                   0x76B4, 0xED68, 0xCAF1, 0x85C3, 0x1BA7, 0x374E, 0x6E9C,
                   0x3730, 0x6E60, 0xDCC0, 0xA9A1, 0x4363, 0x86C6, 0x1DAD,
                   0x3331, 0x6662, 0xCCC4, 0x89A9, 0x0373, 0x06E6, 0x0DCC,
                   0x1021, 0x2042, 0x4084, 0x8108, 0x1231, 0x2462, 0x48C4 )


FUNCTION CreateXorArray_Method1
    PARAMETERS Password
    RETURNS array of 8-bit unsigned integers

    DECLARE XorKey AS 16-bit unsigned integer
    DECLARE ObfuscationArray AS array of 8-bit unsigned integers

    SET XorKey TO CreateXorKey_Method1(Password)

    SET Index TO Password.Length
    SET ObfuscationArray TO (0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)

    IF Index MODULO 2 IS 1
        SET Temp TO most significant byte of XorKey
        SET ObfuscationArray[Index] TO XorRor(PadArray[0], Temp)

        DECREMENT Index

        SET Temp TO least significant byte of XorKey
        SET PasswordLastChar TO Password[Password.Length MINUS 1]
        SET ObfuscationArray[Index] TO XorRor(PasswordLastChar, Temp)
    END IF

    WHILE Index IS GREATER THAN to 0
        DECREMENT Index
        SET Temp TO most significant byte of XorKey
        SET ObfuscationArray[Index] TO XorRor(Password[Index], Temp)

        DECREMENT Index
        SET Temp TO least significant byte of XorKey
        SET ObfuscationArray[Index] TO XorRor(Password[Index], Temp)
    END WHILE

    SET Index TO 15
    SET PadIndex TO 15 MINUS Password.Length
    WHILE PadIndex IS greater than 0

        SET Temp TO most significant byte of XorKey
        SET ObfuscationArray[Index] TO XorRor(PadArray[PadIndex], Temp)
        DECREMENT Index
        DECREMENT PadIndex
```

```
        SET Temp TO least significant byte of XorKey
        SET ObfuscationArray[Index] TO XorRor(PadArray[PadIndex], Temp)
        DECREMENT Index
        DECREMENT PadIndex
    END WHILE

    RETURN ObfuscationArray
END FUNCTION


FUNCTION CreateXorKey_Method1
    PARAMETERS Password
    RETURNS 16-bit unsigned integer

    DECLARE XorKey AS 16-bit unsigned integer

    SET XorKey TO InitialCode[Password.Length MINUS 1]

    SET CurrentElement TO 0x00000068

    FOR EACH Char IN Password IN REVERSE ORDER
        FOR 7 iterations
            IF (Char BITWISE AND 0x40) IS NOT 0
                SET XorKey TO XorKey BITWISE XOR XorMatrix[CurrentElement]
            END IF
            SET Char TO Char MULTIPLIED BY 2
            DECREMENT CurrentElement
        END FOR
    END FOR

    RETURN XorKey
END FUNCTION


FUNCTION XorRor
    PARAMETERS byte1, byte2
    RETURNS 8-bit unsigned integer

    RETURN Ror(byte1 XOR byte2)
END FUNCTION


FUNCTION Ror
    PARAMETERS byte
    RETURNS 8-bit unsigned integer

    SET temp1 TO byte DIVIDED BY 2
    SET temp2 TO byte MULTIPLIED BY 128
    SET temp3 TO temp1 BITWISE OR temp2
    RETURN temp3 MODULO 0x100
END FUNCTION
```

2.3.7.3 Binary Document XOR Data Transformation Method 1

Data transformed by Binary Document XOR Data Transformation Method 1 for encryption MUST be as specified in the **EncryptData_Method1** procedure. This procedure takes the following parameters:

- **Password:** An ASCII string that specifies the password to be used to encrypt the data.
- **Data:** An array of unsigned 8-bit integers that specifies the data to be encrypted.
- **XorArrayIndex:** An unsigned integer that specifies the initial index into the XOR obfuscation array to be used.

```
FUNCTION EncryptData_Method1
    PARAMETERS Password, Data, XorArrayIndex
    DECLARE XorArray as array of 8-bit unsigned integers

    SET XorArray TO CreateXorArray_Method1(Password)

    FOR Index FROM 0 TO Data.Length
        SET Value TO Data[Index]
        SET Value TO (Value rotate left 5 bits)
        SET Value TO Value BITWISE XOR XorArray[XorArrayIndex]
        SET DATA[Index] TO Value

        INCREMENT XorArrayIndex
        SET XorArrayIndex TO XorArrayIndex MODULO 16
    END FOR
END FUNCTION
```

Data transformed by the Binary Document XOR Data Transformation Method 1 for decryption MUST be as specified in the **DecryptData_Method1** procedure. This procedure takes the following parameters:

- **Password:** An ASCII string that specifies the password to be used to decrypt the data.
- **Data:** An array of unsigned 8-bit integers that specifies the data to be decrypted.
- **XorArrayIndex:** An unsigned integer that specifies the initial index into the XOR obfuscation array to be used.

```
FUNCTION DecryptData_Method1
    PARAMETERS Password, Data, XorArrayIndex
    DECLARE XorArray as array of 8-bit unsigned integers

    SET XorArray TO CreateXorArray_Method1(Password)

    FOR Index FROM 0 to Data.Length
        SET Value TO Data[Index]
        SET Value TO Value BITWISE XOR XorArray[XorArrayIndex]
        SET Value TO (Value rotate right 5 bits)
        SET Data[Index] TO Value

        INCREMENT XorArrayIndex
        SET XorArrayIndex TO XorArrayIndex MODULO 16
    END FOR
END FUNCTION
```

2.3.7.4 Binary Document Password Verifier Derivation Method 2

The **CreatePasswordVerifier_Method2** procedure specifies how a 32-bit password verifier is obtained from a string of single-byte characters that has been transformed from a Unicode string. The password verifier is used in XOR obfuscation.

Two different approaches exist for preprocessing the password string to convert it from Unicode to single-byte characters:

- Using the current **language code identifier (LCID)**, convert Unicode input into an ANSI string, as specified in [MS-UCODEREF]. Truncate the resulting string to 15 single-byte characters.
- For each input Unicode character, copy the least significant byte into the single-byte string, unless the least significant byte is 0x00. If the least significant byte is 0x00, copy the most significant byte. Truncate the resulting string to 15 characters.

When writing files, the second approach MUST be used. When reading files, both methods MUST be tried, and the password MUST be considered correct if either approach results in a match.

The **CreatePasswordVerifier_Method2** procedure takes the following parameter:

- **Password:** A string of single-byte characters that specifies the password to be used to encrypt the data. *Password* MUST NOT be longer than 15 characters. *Password* MUST be transformed from Unicode to single-byte characters by using the method specified in this section.

```
FUNCTION CreatePasswordVerifier_Method2
    PARAMETERS Password
    RETURNS 32-bit unsigned integer

    DECLARE Verifier as 32-bit unsigned integer
    DECLARE KeyHigh as 16-bit unsigned integer
    DECLARE KeyLow as 16-bit unsigned integer

    SET KeyHigh TO CreateXorKey_Method1(Password)
    SET KeyLow TO CreatePasswordVerifier_Method1(Password)

    SET most significant 16 bits of Verifier TO KeyHigh
    SET least significant 16 bits of Verifier TO KeyLow

    RETURN Verifier
END FUNCTION
```

2.3.7.5 Binary Document XOR Array Initialization Method 2

The **CreateXorArray_Method2** procedure specifies how a 16-byte XOR obfuscation array is initialized. The procedure takes the following parameter:

- **Password:** A string of single-byte characters that specifies the password to be used to encrypt the data. *Password* MUST NOT be longer than 15 characters. *Password* MUST be transformed from Unicode to single-byte characters by using the method specified in section 2.3.7.4, which results in the password verifier matching.

```
FUNCTION CreateXorArray_Method2
    PARAMETERS Password
    RETURNS array of 8-bit unsigned integers

    DECLARE Verifier as 32-bit unsigned integer
    DECLARE VerifierHighWord as 16-bit unsigned integer
    DECLARE KeyHigh as 8-bit unsigned integer
    DECLARE KeyLow as 8-bit unsigned integer

    SET Verifier TO CreatePasswordVerifier_Method2(Password)
    SET VerifierHighWord TO 16 most significant bits of Verifier
    SET KeyHigh TO 8 most significant bits of VerifierHighWord
    SET KeyLow TO 8 least significant bits of VerifierHighWord

    SET PadArray TO (0xBB, 0xFF, 0xFF, 0xBA, 0xFF, 0xFF, 0xB9, 0x80,
                     0x00, 0xBE, 0x0F, 0x00, 0xBF, 0x0F, 0x00)
    SET ObfuscationArray TO (0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)

    SET Index TO 0
    WHILE Index IS LESS THAN Password.Length
        SET ObfuscationArray[Index] TO Password[Index]
        INCREMENT Index
    END WHILE
    WHILE Index IS LESS THAN 16
        SET ObfuscationArray[Index] TO PadArray[Index MINUS Password.Length]
        INCREMENT Index
    END WHILE
```

```
    SET Index TO 0
    WHILE Index IS LESS THAN 16
        SET Temp1 TO ObfuscationArray[Index] BITWISE XOR KeyLow
        SET ObfuscationArray[Index] TO Ror(Temp1)

        INCREMENT Index

        SET Temp1 TO ObfuscationArray[Index] BITWISE XOR KeyHigh
        SET ObfuscationArray[Index] TO Ror(Temp1)

        INCREMENT Index
    END WHILE

    RETURN ObfuscationArray
END FUNCTION
```

2.3.7.6 Binary Document XOR Data Transformation Method 2

Data transformed by Binary Document XOR data transformation method 2 takes the result of an XOR operation on each byte of input in sequence and the 16-byte XOR obfuscation array that is initialized as specified in section 2.3.7.2, except when the byte of input is 0x00 or the binary XOR of the input and the obfuscation array element is 0x00, in which case the byte of input is not modified. When the end of the XOR obfuscation array is reached, start again at the beginning.

2.3.7.7 Password Verification

Calculate the password verifier for the applicable password verifier derivation method, as specified in section 2.3.7.1 or section 2.3.7.4, depending on the document type. Compare the derived password verifier with the password verifier stored in the file. If the two do not match, the password is incorrect.

2.4 Document Write Protection

Document write protection is meant to discourage tampering with the file or sections of the file by users. See section 4.1.4 for more information.

Limitations on the length of the password and the characters used by the password are implementation-dependent. For more details about behavior variations, see [MS-DOC] and [MS-XLS]. Unless otherwise specified, the maximum password length MUST be 255 Unicode characters.

2.4.1 ECMA-376 Document Write Protection

ECMA-376 document write protection [ECMA-376] is as specified in [ECMA-376] Part 4 Sections 2.15.1.28, 2.15.1.94, 3.2.12, and 4.3.1.17.<24>

2.4.2 Binary Document Write Protection

2.4.2.1 Binary Document Write Protection Method 1

Binary documents that conform to the file format as specified in [MS-DOC] MUST store the write protection password in the file in plaintext as specified in [MS-DOC] section 2.9.276.

2.4.2.2 Binary Document Write Protection Method 2

Binary documents that conform to the file format as specified in [MS-XLS] MUST store the write protection password verifier in the file, as specified in [MS-XLS] section 2.2.9 and created by using the method specified in section 2.3.7.1. When a binary document using write protection method 2 is write protected, the document can also be encrypted by using one of the methods specified in section 2.3.<25>

2.4.2.3 Binary Document Write Protection Method 3

Binary documents that conform to the file format as specified in [MS-PPT] MUST store the write protection password in the file in plaintext, as specified in [MS-PPT] section 2.4.7. When a binary document using write protection method 3 is write protected, it SHOULD NOT<26> also be encrypted by using one of the methods specified in section 2.3.

If the user has not supplied an encryption password and the document is encrypted, the default encryption choice using the techniques specified in section 2.3 MUST be the following password: "\x2f\x30\x31\x48\x61\x6e\x6e\x65\x73\x20\x52\x75\x65\x73\x63\x68\x65\x72\x2f\x30\x31".

2.4.2.4 ISO Write Protection Method

Cases where binary documents use the following hashing algorithm, intended to be compatible with ISO/IEC 29500 (for more information, see [ISO/IEC29500-1:2011]), are specified in [MS-XLSB]. The ISO password hashing algorithm takes the following parameters:

- **Password:** An array of Unicode characters specifying the write protection password. The password MUST be a minimum of 1 and a maximum of 255 Unicode characters.

AlgorithmName: A Unicode string specifying the name of the cryptographic hash algorithm used to compute the password hash value. The values in the following table are reserved. (Values that are not defined MAY<27> be used, and a compliant implementation is not required to support all defined values. The string MUST be at least 1 character. See section 4 for additional information.)

Value	Hash algorithm
SHA-1	MUST conform to the details as specified in [RFC4634].
SHA-256	MUST conform to the details as specified in [RFC4634].
SHA-384	MUST conform to the details as specified in [RFC4634].
SHA-512	MUST conform to the details as specified in [RFC4634].
MD5	MUST conform to **MD5**.
MD4	MUST conform to the details as specified in [RFC1320].
MD2	MUST conform to the details as specified in [RFC1319].
RIPEMD-128	MUST conform to the details as specified in [ISO/IEC 10118].
RIPEMD-160	MUST conform to the details as specified in [ISO/IEC 10118].
WHIRLPOOL	MUST conform to the details as specified in [ISO/IEC 10118].

- **Salt:** An array of bytes that specifies the **salt** value used during password hash generation. When computing hashes for new passwords, this MUST be generated using an arbitrary pseudorandom function. When verifying a password, the salt value retrieved from the document MUST be used. The salt MUST NOT be larger than 65,536 bytes.

- **SpinCount:** A 32-bit unsigned integer that specifies the number of times to iterate on a hash of a password. It MUST NOT be greater than 10,000,000.

Let H() be an implementation of the hashing algorithm specified by **AlgorithmName**, iterator be an unsigned 32-bit integer, H_n be the hash data of the n^{th} iteration, and a plus sign (+) represent concatenation. The initial password hash is generated as follows.

- H_0 = H(salt + password)

The hash is then iterated using the following approach.

- $H_n = H(H_{n-1} + \text{iterator})$

where **iterator** is initially set to 0 and is incremented monotonically on each iteration until **SpinCount** iterations have been performed. The value of **iterator** on the last iteration MUST be one less than **SpinCount**. The final hash is then $H_{final} = H_{SpinCount-1}$.

2.5 Binary Document Digital Signatures

This section specifies the process used to create and store digital signatures within Office binary documents, and it specifies XML Advanced Electronic Signatures [XAdES] support for all documents using xmldsig digital signatures. There are two digital signature formats. The first is referred to as a CryptoAPI digital signature, and the second is referred to as an xmldsig digital signature.

The process used by ECMA-376 documents [ECMA-376] for xmldsig digital signatures is very similar to the process used by xmldsig digital signatures when applied to Office binary documents, as specified in [ECMA-376] Part 2 Section 12. Both document types use an XML signature format as specified in [XMLDSig]. For details about a schema reference, see [ECMA-376] Part 2 Section 12.2.4.

2.5.1 CryptoAPI Digital Signature Structures and Streams

2.5.1.1 TimeEncoding Structure

The **TimeEncoding** structure specifies a date and time in **Coordinated Universal Time (UTC)**, with the most significant 32 bits and the least significant 32 bits of the structure swapped. To be processed as a valid UTC time, **HighDateTime** and **LowDateTime** MUST be assigned to a **FILETIME** structure as specified in [MS-DTYP]. Because of the reverse ordering, the **HighDateTime** field MUST be assigned to the **dwHighDateTime** field of the **FILETIME** structure, and the **LowDateTime** field MUST be assigned to the **dwLowDateTime** field of the **FILETIME** structure. After the **HighDateTime** and **LowDateTime** fields are correctly assigned to a **FILETIME** structure, the UTC time can be obtained.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">HighDateTime</td></tr>
<tr><td colspan="32">LowDateTime</td></tr>
</table>

HighDateTime (4 bytes): An unsigned integer specifying the high order 32 bits of a **UTCTime**.

LowDateTime (4 bytes): An unsigned integer specifying the low order 32 bits of a **UTCTime**.

2.5.1.2 CryptoAPI Digital Signature CertificateInfo Structure

The **CertificateInfo** structure has the format that is shown in the following diagram.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">CertificateInfoSize</td></tr>
<tr><td colspan="32">SignerLength</td></tr>
<tr><td colspan="32">IssuerLength</td></tr>
<tr><td colspan="32">ExpireTime</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">SignTime</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">AlgIDHash</td></tr>
<tr><td colspan="32">SignatureSize</td></tr>
<tr><td colspan="32">EncodedCertificateSize</td></tr>
<tr><td colspan="32">Version</td></tr>
<tr><td colspan="32">SerialNumberSize</td></tr>
<tr><td colspan="32">IssuerBlobSize</td></tr>
<tr><td colspan="32">Reserved</td></tr>
<tr><td colspan="32">SignerName (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">IssuerName (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">Signature (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">EncodedCertificate (variable)</td></tr>
</table>

...
SerialNumber (variable)
...
IssuerBlob (variable)
...

CertificateInfoSize (4 bytes): An unsigned integer specifying the number of bytes used by the remainder of this structure, not including **CertificateInfoSize**.

SignerLength (4 bytes): An unsigned integer specifying the number of characters needed to store the **SignerName** field, not including the terminating null character.

IssuerLength (4 bytes): An unsigned integer specifying the number of characters needed to store the **IssuerName** field, not including the terminating null character.

ExpireTime (8 bytes): A **TimeEncoding** structure (section 2.5.1.1) specifying the expiration time of this signature.

SignTime (8 bytes): A **TimeEncoding** structure specifying the time this signature was created.

AlgIDHash (4 bytes): A signed integer specifying the algorithm identifier. It MUST be 0x00008003 (MD5).

SignatureSize (4 bytes): An unsigned integer specifying the number of bytes used by the **Signature** field.

EncodedCertificateSize (4 bytes): An unsigned integer specifying the number of bytes used by the **EncodedCertificate** field.

Version (4 bytes): A value that MUST be 0x00000000.

SerialNumberSize (4 bytes): An unsigned integer specifying the number of bytes used by the **SerialNumber** field.

IssuerBlobSize (4 bytes): An unsigned integer specifying the number of bytes used by the **IssuerBlob** field.

Reserved (4 bytes): A value that MUST be 0x00000000.

SignerName (variable): A null-terminated Unicode string specifying the name of the signer.

IssuerName (variable): A null-terminated Unicode string specifying the name of the issuer.

Signature (variable): A binary representation of the signature, generated as specified in [RFC3280], except stored in little-endian form.

EncodedCertificate (variable): An encoded representation of the **certificate**. MUST contain the ASN.1 [ITUX680-1994] DER encoding of an X.509 certificate. For more details, see [RFC3280].

SerialNumber (variable): An array of bytes specifying the serial number of the certificate as specified in [RFC3280], with the least significant byte first. Any leading 0x00 bytes MUST be truncated.

IssuerBlob (variable): An ASN.1 structure as specified in IETF [RFC3280] section 4.1.2.4.

2.5.1.3 CryptoAPI Digital Signature Structure

A CryptoAPI digital signature structure MUST contain exactly one **IntermediateCertificatesStore** and MUST contain at least one CryptoAPI Digital Signature **CertificateInfo** structure (section 2.5.1.2).

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
CertificateSize																															
IntermediateCertificatesStore (variable)																															
...																															
Reserved																															
CertificateInfoArray (variable)																															
...																															
EndMarker																															

CertificateSize (4 bytes): An unsigned integer specifying the number of bytes in the **IntermediateCertificatesStore** field.

IntermediateCertificatesStore (variable): A binary representation of the certificates in the certificate chains of the certificates used to sign the document, excluding the self-signed root CA certificates and end-entity certificates. This store is generated as specified in [MS-OSHARED] section 2.3.9.1.

Reserved (4 bytes): A value that MUST be 0x00000000.

CertificateInfoArray (variable): An array that MUST contain a single **CertificateInfo** structure for every signature included in this **stream**.

EndMarker (4 bytes): A value that MUST be 0x00000000.

2.5.1.4 _signatures Stream

A binary document containing a CryptoAPI digital signature MUST have a **stream** named "_signatures" in the root storage. The contents of the **_signatures** stream MUST contain exactly one CryptoAPI Digital Signature structure (section 2.5.1.3).

2.5.1.5 CryptoAPI Digital Signature Generation

The hash used to generate a document signature is created by recursively traversing the OLE compound file **streams** and storages. Certain streams and storages MUST NOT be used, as specified later in this section. A document MAY have more than one signature, each of which MUST be generated by using the **GenerateSignature** function. Each individual **certificate** MUST be stored in the **CertificateInfoArray** of the CryptoAPI Digital Signature structure.

Let H() be a hashing function, which MUST be **MD5**, and a plus sign (+) represent concatenation. Let **HashObject** be an object that can be initialized, that can append data in blocks into the object, and that can finalize to extract the resultant hash value H_{final}.

Let **ClsID** be the GUID identifier for an OLE compound file storage as specified in [MS-CFB].

Let **TimeStamp** be a **FILETIME** structure as specified in [MS-DTYP], containing the current system time, expressed in Coordinated Universal Time (UTC). **TimeStamp** MUST be stored in the CryptoAPI Digital Signature Structure **SignTime** field, as specified in section 2.5.1.3.

Let **ExcludedStorages** be defined as follows:

- 0x06DataSpaces
- 0x05Bagaaqy23kudbhchAaq5u2chNd

Let **ExcludedStreams** be defined as follows:

- _signatures
- 0x09DRMContent

```
FUNCTION GenerateSignature
   PARAMETERS Storage, Certificate
   RETURNS Signature

   CALL HashObject.Initialize
   CALL GenerateSignatureHash(Storage, HashObject, IsFiltered, AppFilter)
   SET Hdata TO HashObject.Finalize
   SET Hfinal TO H(Hdata + TimeStamp)
   SET Signature TO RFC3447(Hfinal, Certificate)
   RETURN Signature
END FUNCTION
```

In the **GenerateSignatureHash** function, **IsFiltered** MUST be **true** if the document conforms to the details as specified in [MS-XLS] and the stream name is "Workbook" or if the document conforms to the details as specified in [MS-PPT] and the stream name is "Current User". It MUST be **false** for all other document types and streams.

For documents that conform to the details as specified in [MS-XLS], let **AppFilter** be defined as the process specified in [MS-XLS] section 2.1.7.15, which appends data to **HashObject**, excluding a portion of the stream from being used in the hashing operation.

For documents that conform to the details as specified in [MS-PPT], let **AppFilter** be defined as a process that returns without appending data to **HashObject**. The result is that the name of the **CurrentUser** stream MUST be appended to the **HashObject**, but the data contained within the **CurrentUser** stream MUST NOT be appended to the **HashObject**.

When stream or storage names are appended to a **HashObject**, the terminating Unicode null character MUST NOT be included.

Let **SORT** be a string sorting method that is case sensitive and ascending and that skips any nonprintable characters, such that if two streams named "Data" and "0x05DocumentSummaryInformation" are input, the stream named "Data" is ordered first.

```
FUNCTION GenerateSignatureHash
    PARAMETERS Storage, HashObject, IsFiltered, AppFilter
    RETURNS VOID

    DECLARE StorageNameArray as (empty array of Unicode strings)
    DECLARE StreamNameArray as (empty array of Unicode strings)

    SET ClsID TO Storage.GUID
    CALL HashObject.AppendData(ClsID)
    FOR EACH Child IN Storage.Children
        IF Child IS a storage AND Child.Name NOT IN ExcludedStorages
```

```
            APPEND Child.Name to StorageNameArray
        END IF
        IF Child IS a stream AND Child.Name NOT IN ExcludedStreams
            APPEND Child.Name to StreamNameArray
        END IF
    END FOR

    SORT StorageNameArray     SORT StreamNameArray

    FOR EACH StreamName IN StreamNameArray

        CALL HashObject.AppendData(StreamName)

        SET ChildStream TO Storage.Children[StreamName]
        IF IsFiltered IS true
            CALL AppFilter(ChildStream, HashObject)
        ELSE
            CALL HashObject.AppendData(ChildStream.Data)
        ENDIF
    ENDFOR

    FOR EACH StorageName IN StorageNameArray

       CALL HashObject.AppendData(StorageName)

    SET ChildStorage TO Storage.Children[StorageName]
       CALL GenerateSignatureHash(ChildStorage, HashObject,
IsFiltered, AppFilter)
    END FOR

END FUNCTION
```

When signing H_{final}, the certificate MUST be an RSA certificate as specified in [RFC3447], and the signing operation MUST be performed as specified in [RFC3447] section 9.2.

If a document is protected as specified in section 2.2, the hash MUST be created by first appending the unencrypted form of the storage that is decrypted from the **0x09DRMContent** stream, followed by the entire original encrypted file storage with the **0x09DRMContent** stream excluded as noted previously.

2.5.2 Xmldsig Digital Signature Elements

A binary document digital signature is specified as containing the elements that are specified in the following subsections. If not explicitly stated in each subsection, the content of an element MUST be generated as specified in [XMLDSig].

2.5.2.1 SignedInfo Element

The **SignedInfo** element MUST contain the following elements:

- **CanonicalizationMethod**, where the algorithm MUST be as specified in [Can-XML-1.0].
- **Reference**, where the **URI** attribute MUST be "#idPackageObject", and **DigestMethod** is provided by the application.<28>
- **Reference**, where the URI attribute MUST be "#idOfficeObject", and **DigestMethod** is provided by the application.<29>

2.5.2.2 SignatureValue Element

The **SignatureValue** element contains the value of the signature, as specified in [XMLDSig].

2.5.2.3 KeyInfo Element

The **KeyInfo** element contains the key information, as specified in [XMLDSig].

2.5.2.4 idPackageObject Object Element

The **idPackageObject** element contains the following:

- A **Manifest** element as specified in [XMLDSig], which contains **Reference** elements corresponding to each **stream** that is signed. Except for streams and storages enumerated later in this section, all streams and storages MUST be included in the **Manifest** element. **DigestMethod** is provided by the application.<30>

- A **SignatureProperties** element containing a **SignatureProperty** element with a time stamp, as specified in [ECMA-376] Part 2 Section 12.2.4.20.

When constructing the **Manifest** element, the following storages and any storages or streams contained within listed storages MUST be excluded:

- 0x05Bagaaqy23kudbhchAaq5u2chNd
- 0x06DataSpaces
- Xmlsignatures
- MsoDataStore

The following streams MUST also be excluded:

- 0x09DRMContent
- _signatures
- 0x05SummaryInformation
- 0x05DocumentSummaryInformation

If the document conforms to the details as specified in [MS-XLS], and the name of the stream is Workbook, the stream MUST be filtered as specified in [MS-XLS] section 2.1.7.21.

If the document conforms to the details as specified in [MS-PPT], the hash of the **CurrentUser** stream MUST be calculated when verifying the signature as if the stream were empty, which would be the result of hashing 0 bytes.

2.5.2.5 idOfficeObject Object Element

The **idOfficeObject** element contains the following:

- A **SignatureProperties** element containing a **SignatureProperty** element, which MUST contain a **SignatureInfoV1** element that specifies the details of a digital signature in a document. The following XML Schema specifies the contents of the **SignatureProperty** element:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema targetNamespace="http://schemas.microsoft.com/office/2006/digsig"
elementFormDefault="qualified" xmlns="http://schemas.microsoft.com/office/2006/digsig"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="ST_PositiveInteger">
    <xsd:restriction base="xsd:int">
      <xsd:minExclusive value="0" />
    </xsd:restriction>
  </xsd:simpleType>
```

```
  <xsd:simpleType name="ST_SignatureComments">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="255" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ST_SignatureProviderUrl">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="2083" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ST_SignatureText">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="100" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ST_SignatureType">
    <xsd:restriction base="xsd:int">
      <xsd:enumeration value="1"></xsd:enumeration>
      <xsd:enumeration value="2"></xsd:enumeration>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ST_Version">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="64" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ST_UniqueIdentifierWithBraces">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\{[0-9a-fA-F]{8}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{4}\-
[0-9a-fA-F]{12}\}|" />
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:group name="EG_RequiredChildren">
    <xsd:sequence>
      <xsd:element name="SetupID" type="ST_UniqueIdentifierWithBraces"></xsd:element>
      <xsd:element name="SignatureText" type="ST_SignatureText"></xsd:element>
      <xsd:element name="SignatureImage" type="xsd:base64Binary"></xsd:element>
      <xsd:element name="SignatureComments" type="ST_SignatureComments"></xsd:element>
      <xsd:element name="WindowsVersion" type="ST_Version"></xsd:element>
      <xsd:element name="OfficeVersion" type="ST_Version"></xsd:element>
      <xsd:element name="ApplicationVersion" type="ST_Version"></xsd:element>
      <xsd:element name="Monitors" type="ST_PositiveInteger"></xsd:element>
      <xsd:element name="HorizontalResolution" type="ST_PositiveInteger"></xsd:element>
      <xsd:element name="VerticalResolution" type="ST_PositiveInteger"></xsd:element>
      <xsd:element name="ColorDepth" type="ST_PositiveInteger"></xsd:element>
      <xsd:element name="SignatureProviderId"
type="ST_UniqueIdentifierWithBraces"></xsd:element>
      <xsd:element name="SignatureProviderUrl" type="ST_SignatureProviderUrl"></xsd:element>
      <xsd:element name="SignatureProviderDetails" type="xsd:int"></xsd:element>
      <xsd:element name="SignatureType" type="ST_SignatureType"></xsd:element>
    </xsd:sequence>
  </xsd:group>
  <xsd:group name="EG_OptionalChildren">
    <xsd:sequence>
      <xsd:element name="DelegateSuggestedSigner" type="xsd:string"></xsd:element>
      <xsd:element name="DelegateSuggestedSigner2" type="xsd:string"></xsd:element>
      <xsd:element name="DelegateSuggestedSignerEmail" type="xsd:string"></xsd:element>
      <xsd:element name="ManifestHashAlgorithm" type="xsd:anyURI"
minOccurs="0"></xsd:element>
    </xsd:sequence>
  </xsd:group>
  <xsd:group name="EG_OptionalChildrenV2">
    <xsd:sequence>
      <xsd:element name="Address1" type="xsd:string"></xsd:element>
      <xsd:element name="Address2" type="xsd:string"></xsd:element>
    </xsd:sequence>
  </xsd:group>
  <xsd:complexType name="CT_SignatureInfoV1">
    <xsd:sequence>
```

```
      <xsd:group ref="EG_RequiredChildren" />
      <xsd:group ref="EG_OptionalChildren" minOccurs="0" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="CT_SignatureInfoV2">
    <xsd:sequence>      <xsd:group ref="EG_OptionalChildrenV2" minOccurs="0" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="SignatureInfoV1" type="CT_SignatureInfoV1"></xsd:element>
  <xsd:element name="SignatureInfoV2" type="CT_SignatureInfoV2"></xsd:element>
</xsd:schema>
```

The child elements of the **SignatureInfoV1** element are further specified as follows:

ApplicationVersion: The version of the application that created the digital signature.

ColorDepth: The color depth of the primary monitor of the computer on which the digital signature was created.

HorizontalResolution: The horizontal resolution of the primary monitor of the computer on which the digital signature was created.

ManifestHashAlgorithm: An optional element containing a URI that identifies the particular hash algorithm for the signature. The value of this element MUST be ignored.

Monitors: The count of monitors on the computer where the digital signature was created.

OfficeVersion: The version of the application suite that created the digital signature.

SetupID: A GUID that can be cross-referenced with the identifier of the signature line stored in the document content.

SignatureComments: The comments on the digital signature.

SignatureImage: An image for the digital signature.

SignatureProviderDetails: The details of the signature provider. The value MUST be an integer computed from a bitmask of the flags that are described in the following table.

Value	Description
0x00000000	Specifies that there are no restrictions on the provider's usage.
0x00000001	Specifies that the provider MUST only be used for the user interface (UI).
0x00000002	Specifies that the provider MUST only be used for invisible signatures.
0x00000004	Specifies that the provider MUST only be used for visible signatures.
0x00000008	Specifies that the application UI MUST be used for the provider.
0x00000010	Specifies that the application stamp UI MUST be used for the provider.

SignatureProviderId: The class identifier of the signature provider.<31>

SignatureProviderUrl: The URL of the software used to generate the digital signature.

SignatureText: The text of actual signature in the digital signature.

SignatureType: The type of the digital signature. Its value MUST be one of those in the following table.

Value	Description
1	The digital signature MUST NOT be printed.
2	The digital signature MUST be printed.

If set to 2, there MUST be two additional objects in the signature with the following identifier values:

- **idValidSigLnImg:** The image of a valid signature.
- **idInvalidSigLnImg:** The image of an invalid signature.

VerticalResolution: The vertical resolution of the primary monitor of the computer on which the digital signature was created.

WindowsVersion: The version of the operating system on which the digital signature was created.

DelegateSuggestedSigner: The name of a person to whom the signature has been delegated.

DelegateSuggestedSigner2: The title of a person to whom the signature has been delegated.

DelegateSuggestedSignerEmail: The email address of a person to whom the signature has been delegated.

The child elements of the **SignatureInfoV2** element are specified as follows:

Address1: The location at which the signature was created.

Address2: The location at which the signature was created.

The optional **SignatureInfoV2** element is used to provide additional information to the **SignatureProductionPlace** element, which is specified in [XAdES] section 7.2.7.

2.5.2.6 XAdES Elements

XML Advanced Electronic Signatures [XAdES] extensions to xmldsig signatures MAY<32> be present in either binary or ECMA-376 documents [ECMA-376] when using xmldsig signatures. XAdES-EPES through XAdES-X-L extensions are specified within a signature. Unless otherwise specified, any optional elements as specified in [XAdES] are ignored.

The **Object** element containing the information as specified in [XAdES] has a number of optional elements, and many of the elements have more than one method specified. A document compliant with this file format uses the following options:

- The **SignedSignatureProperties** element MUST contain a **SigningCertificate** property as specified in [XAdES] section 7.2.2.
- A **SigningTime** element MUST be present as specified in [XAdES] section 7.2.1.
- A **SignaturePolicyIdentifier** element MUST be present as specified in [XAdES] section 7.2.3.
- If the information as specified in [XAdES] contains a time stamp as specified by the requirements for XAdES-T, the time stamp information MUST be specified as an **EncapsulatedTimeStamp** element containing DER encoded ASN.1. data.
- If the information as specified in [XAdES] contains references to validation data, the certificates used in the certificate chain, except for the signing **certificate**, MUST be contained within the **CompleteCertificateRefs** element as specified in [XAdES] section 7.4.1. In addition, for the signature to be considered a well-formed XAdES-C signature, a **CompleteRevocationRefs** element MUST be present, as specified in [XAdES] section 7.4.2.

- If the information as specified in [XAdES] contains time stamps on references to validation data, the **SigAndRefsTimestamp** element as specified in [XAdES] section 7.5.1 and [XAdES] section 7.5.1.1 MUST be used. The **SigAndRefsTimestamp** element MUST specify the time stamp information as an **EncapsulatedTimeStamp** element containing DER encoded ASN.1. data.
- If the information as specified in [XAdES] contains properties for data validation values, the **CertificateValues** and **RevocationValues** elements MUST be constructed as specified in [XAdES] section 7.6.1 and [XAdES] section 7.6.2. Except for the signing certificate, all certificates used in the validation chain MUST be entered into the **CertificateValues** element.

There MUST be a **Reference** element specifying the digest of the **SignedProperties** element, as specified in [XAdES], section 6.2.1. A **Reference** element is placed in one of two parent elements, as specified in [XMLDSig]:

- The **SignedInfo** element of the top-level Signature XML.
- A **Manifest** element contained within an **Object** element.

A document compliant with this file format SHOULD<33> place the **Reference** element specifying the digest of the **SignedProperties** element within the **SignedInfo** element. If the **Reference** element is instead placed in a **Manifest** element, the containing **Object** element MUST have an **id** attribute set to "idXAdESReferenceObject".

2.5.3 _xmlsignatures Storage

Digital signatures MUST be stored as **streams** contained in a storage named "_xmlsignatures", based on the root of the compound document. Streams containing a signature MUST be named using a base-10 string representation of a random number. The name of the stream MUST NOT be the same as an existing signature contained within the storage. A single signature is stored directly into each stream, as UTF-8 characters, with no leading header. The content of each stream MUST be a valid signature as specified in [XMLDSig] and generated as specified in section 2.5.2. More than one signature can be present in the "_xmlsignatures" storage.

3 Structure Examples

This section provides examples of the following structures:

- An ECMA-376 document [ECMA-376] conforming to the **IRMDS** structure.
- Office binary data file structures with corresponding hexadecimal and graphical representation.

The example for the ECMA-376 document [ECMA-376] contains the following **streams** and storages:

- **0x06DataSpaces** storage:
  - **Version** stream containing a **DataSpaceVersionInfo** structure as specified in section 3.1.
  - **DataSpaceMap** stream containing a **DataSpaceMap** structure as specified in section 3.2.
  - **DataSpaceInfo** storage:
    - **DRMEncryptedDataSpace** stream containing a **DataSpaceDefinition** structure as described in section 3.3.
  - **TransformInfo** storage:
    - **0x06Primary** stream containing an **IRMDSTransformInfo** structure as described in section 3.4.
    - **EUL-ETRHA1143ZLUDD412YTI3M5CTZ** stream containing an **EndUserLicenseHeader** structure and a certificate chain as described in section 3.5.
- **EncryptedPackage** stream.
- **0x05SummaryInformation** stream.
- **0x05DocumentSummaryInformation** stream.

Note that not all of the streams and storages in the file, including the **0x05SummaryInformation** stream and **0x05DocumentSummaryInformation** stream, are specified in the **IRMDS** structure, and examples are not provided for those streams in this section. OLE compound files conforming to this structure frequently contain other storages and streams.

3.1 Version Stream

This section provides an example of a **Version stream** that contains a **DataSpaceVersionInfo** structure (section 2.1.5).

```
00000000:  3C 00 00 00 4D 00 69 00  63 00 72 00 6F 00 73 00
00000010:  6F 00 66 00 74 00 2E 00  43 00 6F 00 6E 00 74 00
00000020:  61 00 69 00 6E 00 65 00  72 00 2E 00 44 00 61 00
00000030:  74 00 61 00 53 00 70 00  61 00 63 00 65 00 73 00
00000040:  01 00 00 00 01 00 00 00  01 00 00 00
```

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">FeatureIdentifier (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

ReaderVersion.vMajor	ReaderVersion.vMinor
UpdaterVersion.vMajor	UpdaterVersion.vMinor
WriterVersion.vMajor	WriterVersion.vMinor

FeatureIdentifier (variable): "Microsoft.Container.DataSpaces" specifies the functionality for which this version information applies. This string is contained in a **UNICODE-LP-P4** structure (section 2.1.2); therefore, the first 4 bytes of the structure contain 0x0000003C, which specifies the length, in bytes, of the string. The string is not null-terminated.

ReaderVersion.vMajor (2 bytes): 0x0001 specifies the major component of the reader version of the software component that created this structure.

ReaderVersion.vMinor (2 bytes): 0x0000 specifies the minor component of the reader version of the software component that created this structure.

UpdaterVersion.vMajor (2 bytes): 0x0001 specifies the major component of the updater version of the software component that created this structure.

UpdaterVersion.vMinor (2 bytes): 0x0000 specifies the minor component of the updater version of the software component that created this structure.

WriterVersion.vMajor (2 bytes): 0x0001 specifies the major component of the writer version of the software component that created this structure.

WriterVersion.vMinor (2 bytes): 0x0000 specifies the minor component of the writer version of the software component that created this structure.

3.2 DataSpaceMap Stream

This section provides an example of a **DataSpaceMap stream** that contains a **DataSpaceMap** structure (section 2.1.6). The **DataSpaceMap** structure, in turn, contains a **DataSpaceMapEntry** structure (section 2.1.6.1).

```
00000000:  08 00 00 00 01 00 00 00  60 00 00 00 01 00 00 00
00000010:  00 00 00 00 20 00 00 00  45 00 6E 00 63 00 72 00
00000020:  79 00 70 00 74 00 65 00  64 00 50 00 61 00 63 00
00000030:  6B 00 61 00 67 00 65 00  2A 00 00 00 44 00 52 00
00000040:  4D 00 45 00 6E 00 63 00  72 00 79 00 70 00 74 00
00000050:  65 00 64 00 44 00 61 00  74 00 61 00 53 00 70 00
00000060:  61 00 63 00 65 00 00 00
```

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
HeaderLength																															
EntryCount																															
MapEntries (variable)																															
...																															

HeaderLength (4 bytes): 0x00000008 specifies the number of bytes in the **DataSpaceMap** structure before the first **MapEntry**.

EntryCount (4 bytes): 0x00000001 specifies the number of **DataSpaceMapEntry** items in the **MapEntries** array.

MapEntries (variable): The contents of the **MapEntries** array. For more information, see section 3.2.1.

3.2.1 DataSpaceMapEntry Structure

This section provides an example of a **DataSpaceMapEntry** structure (section 2.1.6.1).

```
00000000:                            60 00 00 00 01 00 00 00
00000010:  00 00 00 00 20 00 00 00   45 00 6E 00 63 00 72 00
00000020:  79 00 70 00 74 00 65 00   64 00 50 00 61 00 63 00
00000030:  6B 00 61 00 67 00 65 00   2A 00 00 00 44 00 52 00
00000040:  4D 00 45 00 6E 00 63 00   72 00 79 00 70 00 74 00
00000050:  65 00 64 00 44 00 61 00   74 00 61 00 53 00 70 00
00000060:  61 00 63 00 65 00 00 00
```

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
Length																															
ReferenceComponentCount																															
ReferenceComponent.ReferenceComponentType																															
ReferenceComponent.ReferenceComponent																															
...																															
DataSpaceName																															
...																															

Length (4 bytes): 0x00000060 specifies the size, in bytes, of the **DataSpaceMapEntry** structure.

ReferenceComponentCount (4 bytes): 0x00000001 specifies the number of **DataSpaceReferenceComponent** items (section 2.1.6.2) in the **ReferenceComponents** array.

ReferenceComponent.ReferenceComponentType (4 bytes): 0x00000000 specifies that the referenced component is a **stream**.

ReferenceComponent.ReferenceComponent (variable): "EncryptedPackage" specifies the functionality for which this version information applies. This string is contained in a **UNICODE-LP-P4** structure (section 2.1.2); therefore, the first 4 bytes of the structure contain 0x00000020, which specifies the length, in bytes, of the string. The string is not null-terminated. "EncryptedPackage" matches the name of the stream in the OLE compound file that contains the protected contents.

DataSpaceName (variable): "DRMEncryptedDataSpace" specifies the functionality that this version information applies to. This string is contained in a **UNICODE-LP-P4** structure; therefore, the first

4 bytes of the structure contain 0x0000002A, which specifies the length, in bytes, of the string. The string is not null-terminated; however, the structure is padded with 2 bytes to ensure that its length is a multiple of 4 bytes.

3.3 DRMEncryptedDataSpace Stream

This section provides an example of a **stream** in the **\0x06DataSpaces\DataSpaceInfo** storage (section 2.2.2) that contains a **DataSpaceDefinition** structure (section 2.1.7).

```
00000000:  08 00 00 00 01 00 00 00  2A 00 00 00 44 00 52 00
00000010:  4D 00 45 00 6E 00 63 00  72 00 79 00 70 00 74 00
00000020:  65 00 64 00 54 00 72 00  61 00 63 00 73 00 66 00
00000030:  6F 00 72 00 6D 00 00 00
```

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
HeaderLength																															
TransformReferenceCount																															
TransformReferences																															
...																															

HeaderLength (4 bytes): 0x00000008 specifies the number of bytes in the **DataSpaceDefinition** before the **TransformReferences** field.

TransformReferenceCount (4 bytes): 0x00000001 specifies the number of items in the **TransformReferences** array.

TransformReferences (variable): "DRMEncryptedTransform" specifies the transform associated with this **DataSpaceDefinition** structure. This string is contained in a **UNICODE-LP-P4** structure (section 2.1.2); therefore, the first 4 bytes of the structure contain 0x0000002A, which specifies the length, in bytes, of the string. The string is not null-terminated; however, the structure is padded with 2 bytes to ensure that its length is a multiple of 4 bytes. "DRMEncryptedTransform" matches the name of the transform storage contained in the **\0x06DataSpaces\TransformInfo** storage (section 2.2.3).

3.4 0x06Primary Stream

This section provides an example of a **0x06Primary** **stream** that contains an **IRMDSTransformInfo** structure (section 2.2.6). Note that the first portion of this structure consists of a **TransformInfoHeader** structure (section 2.1.8).

```
00000000:  58 00 00 00 01 00 00 00  4C 00 00 00 7B 00 43 00
00000010:  37 00 33 00 44 00 46 00  41 00 43 00 44 00 2D 00
00000020:  30 00 36 00 31 00 46 00  2D 00 34 00 33 00 42 00
00000030:  30 00 2D 00 38 00 42 00  36 00 34 00 2D 00 30 00
00000040:  43 00 36 00 32 00 30 00  44 00 32 00 41 00 38 00
00000050:  42 00 35 00 30 00 7D 00  3E 00 00 00 4D 00 69 00
00000060:  63 00 72 00 69 00 73 00  6F 00 66 00 74 00 2E 00
00000070:  4D 00 65 00 74 00 61 00  64 00 61 00 74 00 61 00
00000080:  2E 00 44 00 52 00 4D 00  54 00 72 00 61 00 6E 00
00000090:  73 00 66 00 6F 00 72 00  6D 00 00 00 01 00 00 00
000000A0:  01 00 00 00 01 00 00 00  04 00 00 00 26 2F 00 00
```

```
000000B0:  3C 3F 78 6D 6C 20 76 65  72 73 69 6F 6E 3D 22 31
```

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
TransformInfoHeader.TransformLength																															
TransformInfoHeader.TransformType																															
TransformInfoHeader.TransformID (variable)																															
...																															
TransformInfoHeader.TransformName (variable)																															
...																															
TransformInfoHeader.ReaderVersion.vMajor																TransformInfoHeader.ReaderVersion.vMinor															
TransformInfoHeader.UpdaterVersion.vMajor																TransformInfoHeader.UpdaterVersion.vMinor															
TransformInfoHeader.WriterVersion.vMajor																TransformInfoHeader.WriterVersion.vMinor															
ExtensibilityHeader																															
XrMLLicense (variable)																															
...																															

TransformInfoHeader.TransformLength (4 bytes): 0x00000058 specifies the number of bytes in this structure before **TransformInfoHeader.TransformName**.

TransformInfoHeader.TransformType (4 bytes): 0x00000001 specifies the type of transform to be applied.

TransformInfoHeader.TransformID (variable): "{C73DFACD-061F-43B0-8B64-0C620D2A8B50}" specifies a unique, invariant identifier associated with this transform. This string is contained in a **UNICODE-LP-P4** structure (section 2.1.2); therefore, the first 4 bytes of the structure contain 0x0000004C, which specifies the length, in bytes, of the string. The string is not null-terminated.

TransformInfoHeader.TransformName (variable): "Microsoft.Metadata.DRMTransform" specifies the logical name of the transform. This string is contained in a **UNICODE-LP-P4** structure; therefore, the first 4 bytes of the structure contain 0x0000003E, which specifies the length, in bytes, of the string. The string is not null-terminated; however, the structure is padded with 2 bytes to ensure that its length is a multiple of 4 bytes.

TransformInfoHeader.ReaderVersion.vMajor (2 bytes): 0x0001 specifies the major component of the reader version of the software component that created this structure.

TransformInfoHeader.ReaderVersion.vMinor (2 bytes): 0x0000 specifies the minor component of the reader version of the software component that created this structure.

TransformInfoHeader.UpdaterVersion.vMajor (2 bytes): 0x0001 specifies the major component of the updater version of the software component that created this structure.

TransformInfoHeader.UpdaterVersion.vMinor (2 bytes): 0x0000 specifies the minor component of the updater version of the software component that created this structure.

TransformInfoHeader.WriterVersion.vMajor (2 bytes): 0x0001 specifies the major component of the writer version of the software component that created this structure.

TransformInfoHeader.WriterVersion.vMinor (2 bytes): 0x0000 specifies the minor component of the writer version of the software component that created this structure.

ExtensibilityHeader (4 bytes): 0x00000004 specifies that no further information exists in the **ExtensibilityHeader** structure (section 2.2.5).

XrMLLicense (variable): An XrML license as described in [MS-RMPR]. This string is contained in a **UTF-8-LP-P4** structure (section 2.1.3); therefore, the first 4 bytes of the structure contain 0x00002F26, which specifies the length, in bytes, of the string. The string is not null-terminated; however, the structure is padded with 2 bytes to ensure that its length is a multiple of 4 bytes.

3.5 EUL-ETRHA1143ZLUDD412YTI3M5CTZ Stream

This section provides an example of an end-user license **stream** (section 2.2.7), which contains an **EndUserLicenseHeader** structure (section 2.2.9) followed by a certificate chain containing one use license.


```
00000000:  48 00 00 00 40 00 00 00  56 77 42 70 41 47 34 41
00000010:  5A 41 42 76 41 48 63 41  63 77 41 36 41 48 55 41
00000020:  63 77 42 6C 41 48 49 41  51 41 42 6A 41 47 38 41
00000030:  62 67 42 30 41 47 38 41  63 77 42 76 41 43 34 41
00000040:  59 77 42 76 41 47 30 41  94 BE 00 00 3C 3F 78 6D
00000050:  6C 20 76 65 72 73 69 6F  6E 3D 22 31 2E 30 22 3F
00000060:  3E 3C 43 45 52 54 49 46  49 43 41 54 45 43 48 41
00000070:  49 4E 3E 3C 43 45 52 54  49 46 49 43 41 54 45 3E
00000080:  50 41 42 59 41 48 49 41  54 51 42 4D 41 43 41 41
00000090:  64 67 42 6C 41 48 49 41  63 77 42 70 41 47 38 41
000000a0:  62 67 41 39 41 43 49 41  4D 51 41 75 41 44 49 41
000000b0:  49 67 41 67 41 48 67 41  62 51 42 73 41 47 34 41
000000c0:  63 77 41 39 41 43 49 41  49 67 41 67 41 48 41 41
000000d0:  64 51 42 79 41 48 41 41  62 77 42 7A 41 47 55 41
000000e0:  50 51 41 69 41 45 4D 41  62 77 42 75 41 48 51 41
000000f0:  5A 51 42 75 41 48 51 41  4C 51 42 4D 41 47 6B 41
00000100:  59 77 42 6C 41 47 34 41  63 77 42 6C 41 43 49 41
00000110:  50 67 41 38 41 45 49 41  54 77 42 45 41 46 6B 41
00000120:  49 41 42 30 41 48 6B 41  63 41 42 6C 41 44 30 41
00000130:  49 67 42 4D 41 45 6B 41  51 77 42 46 41 45 34 41
00000140:  55 77 42 46 41 43 49 41  49 41 42 32 41 47 55 41
00000150:  63 67 42 7A 41 47 6B 41  62 77 42 75 41 44 30 41
00000160:  49 67 41 7A 41 43 34 41  4D 41 41 69 41 44 34 41
00000170:  50 41 42 4A 41 46 4D 41  55 77 42 56 41 45 55 41
00000180:  52 41 42 55 41 45 6B 41  54 51 42 46
```


Bytes 0x00000000 through 0x000000047 specify an **EndUserLicenseHeader** structure (section 2.2.9). The contents of this section are illustrated in section 3.5.1.

Byte 0x00000048 through the end of this stream specify a certificate chain stored in a **UTF-8-LP-P4** structure (section 2.1.3). The contents of this section are illustrated in section 3.5.2.

3.5.1 EndUserLicenseHeader Structure

This section provides an example of an **EndUserLicenseHeader** structure (section 2.2.9) containing one **LicenseId** (section 2.2.8).

```
00000000:  48 00 00 00 40 00 00 00  56 77 42 70 41 47 34 41
00000010:  5A 41 42 76 41 48 63 41  63 77 41 36 41 48 55 41
00000020:  63 77 42 6C 41 48 49 41  51 41 42 6A 41 47 38 41
00000030:  62 67 42 30 41 47 38 41  63 77 42 76 41 43 34 41
00000040:  59 77 42 76 41 47 30 41
```

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">Length</td></tr>
<tr><td colspan="32">ID_String.Length (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">ID_String.Data (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

Length (4 bytes): 0x00000048 specifies the size of the **EndUserLicenseHeader** structure.

ID_String.Length (variable): 0x00000040 specifies the size of the ASCII string that follows. Note that **ID_String.Size** and **ID_String.Data** together form a **UTF-8-LP-P4** structure (section 2.1.3).

ID_String.Data (variable): "VwBpAG4AZABvAHcAcwA6AHUAcwBlAHIAQABjAG8AbgB0AG8AcwBvAC4AYwBvAG0A" specifies a base64-encoded **LicenseId** that has the value "Windows:user@contoso.com".

3.5.2 Certificate Chain

This section provides an example of a certificate chain contained in an end-user license **stream** (section 2.2.7).

```
00000040:                           94 BE 00 00 3C 3F 78 6D
00000050:  6C 20 76 65 72 73 69 6F  6E 3D 22 31 2E 30 22 3F
00000060:  3E 3C 43 45 52 54 49 46  49 43 41 54 45 43 48 41
00000070:  49 4E 3E 3C 43 45 52 54  49 46 49 43 41 54 45 3E
00000080:  50 41 42 59 41 48 49 41  54 51 42 4D 41 43 41 41
00000090:  64 67 42 6C 41 48 49 41  63 77 42 70 41 47 38 41
000000a0:  62 67 41 39 41 43 49 41  4D 51 41 75 41 44 49 41
000000b0:  49 67 41 67 41 48 67 41  62 51 42 73 41 47 34 41
000000c0:  63 77 41 39 41 43 49 41  49 67 41 67 41 48 41 41
000000d0:  64 51 42 79 41 48 41 41  62 77 42 7A 41 47 55 41
000000e0:  50 51 41 69 41 45 4D 41  62 77 42 75 41 48 51 41
000000f0:  5A 51 42 75 41 48 51 41  4C 51 42 4D 41 47 6B 41
00000100:  59 77 42 6C 41 47 34 41  63 77 42 6C 41 43 49 41
00000110:  50 67 41 38 41 45 49 41  54 77 42 45 41 46 6B 41
00000120:  49 41 42 30 41 48 6B 41  63 41 42 6C 41 44 30 41
00000130:  49 67 42 4D 41 45 6B 41  51 77 42 46 41 45 34 41
00000140:  55 77 42 46 41 43 49 41  49 41 42 32 41 47 55 41
00000150:  63 67 42 7A 41 47 6B 41  62 77 42 75 41 44 30 41
```

```
00000160:  49 67 41 7A 41 43 34 41  4D 41 41 69 41 44 34 41
00000170:  50 41 42 4A 41 46 4D 41  55 77 42 56 41 45 55 41
00000180:  52 41 42 55 41 45 6B 41  54 51 42 46
```


0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
Length																															
Data																															
...																															

Length (4 bytes): 0x0000BE94 specifies the size of the ASCII string that follows. Note that **Length** and **Data** together form a **UTF-8-LP-P4** structure (section 2.1.3).

Data (variable): `<?xml version="1.0"?><CERTIFICATECHAIN><CERTIFICATE>PABYAH IATQBMACAAdgBlAHIAcwBp…` specifies an encoded certificate chain.

The **Data** field has been transformed from the form of certificate chain, as described in [MS-RMPR], in the following way:

1. The original SOAP response contained the following certificate chain:


```
<CertificateChain><Certificate><XrML version="1.2" xmlns="" purpose="Content-License"><BODY
type="LICENSE" version="3.0"><ISSUEDTIME>…
```


2. The body of the **Certificate** element was then base64-encoded to yield the following:


```
PABYAHIATQBMACAAdgBlAHIAcwBpAG8AbgA9ACIAMQAuADIAIgAgAHgAbQBsAG4AcwA9ACIAIgAgAHAAdQByAHAAbwBzA
GUAPQAiAEMAbwBuAHQAZQBuAHQALQBMAGkAYwBlAG4AcwBlACIAPgA8AEIATwBEAFkAIAB0AHkAcABlAD0AIgBMAEkAQw
BFAE4AUwBFACIAIAB2AGUAcgBzAGkAbwBuAD0AIgAzAC4AMAAiAD4APABJAFMAUwBVAEUARABUAEkATQBF…
```


3. The base64-encoded string was then placed in a **Certificate** element, again in a **CertificateChain** element, and finally prefixed with "<?xml version="1.0"?>".

4. The final value of **Data** is thus as follows:


```
<?xml version="1.0"?><CERTIFICATECHAIN><CERTIFICATE>PABYAHIATQBMACAAdgBlAH
IAcwBpAG8AbgA9ACIAMQAuADIAIgAgAHgAbQBsAG4AcwA9ACIAIgAgAHAAdQByAHAAbwBzAGUAPQAiAEMAbwBuAHQAZQB
uAHQALQBMAGkAYwBlAG4AcwBlACIAPgA8AEIATwBEAFkAIAB0AHkAcABlAD0AIgBMAEkAQwBFAE4AUwBFACIAIAB2AGUA
cgBzAGkAbwBuAD0AIgAzAC4AMAAiAD4APABJAFMAUwBVAEUARABUAEkATQBF…
```


3.6 EncryptionHeader Structure

This section provides an example of an **EncryptionHeader** structure (section 2.3.2) used by Office Binary Document RC4 CryptoAPI Encryption (section 2.3.5) to specify the encryption properties for an encrypted **stream**.


```
00001400:                                        04 00 00 00
00001410:  00 00 00 00 01 68 00 00  04 80 00 00 28 00 00 00
00001420:  01 00 00 00 B0 0A 86 02  00 00 00 00 4D 00 69 00
00001430:  63 00 72 00 6F 00 73 00  6F 00 66 00 74 00 20 00
```

```
00001440:  42 00 61 00 73 00 65 00  20 00 43 00 72 00 79 00
00001450:  70 00 74 00 6F 00 67 00  72 00 61 00 70 00 68 00
00001460:  69 00 63 00 20 00 50 00  72 00 6F 00 76 00 69 00
00001470:  64 00 65 00 72 00 20 00  76 00 31 00 2E 00 30 00
```

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
Flags																															
SizeExtra																															
AlgID																															
AlgIDHash																															
KeySize																															
ProviderType																															
Reserved1																															
Reserved2																															
CSPName																															
...																															

Flags (4 bytes): 0x00000004 specifies that the encryption algorithm uses CryptoAPI encryption.

SizeExtra (4 bytes): 0x00000000 is the value in a reserved field.

AlgID (4 bytes): 0x00006801 specifies that the encryption algorithm used is RC4.

AlgIDHash (4 bytes): 0x00008004 specifies that SHA-1 is the hashing algorithm that is used.

KeySize (4 bytes): 0x00000028 specifies that the key is 40 bits long.

ProviderType (4 bytes): 0x00000001 specifies that RC4 is the provider type.

Reserved1 (4 bytes): 0x02860AB0 is the value in a reserved field.

Reserved2 (4 bytes): 0x00000000 is the value in a reserved field.

CSPName (variable): "Microsoft Base Cryptographic Provider v1.0" specifies the name of the cryptographic provider supplying the RC4 implementation that was used to encrypt the file.

3.7 EncryptionVerifier Structure

This section provides an example of an **EncryptionVerifier** structure (section 2.3.3) using **AES** encryption.

```
000018B0:  10 00 00 00 92 25 50 F6  B6 4F FE 5B D3 96 DF 5E
```

```
000018C0:  E9 17 DA 3A BF 86 E1 8F  64 9D 17 D0 A5 41 D9 45
000018D0:  CE FD 96 0C 14 00 00 00  12 FF DC 88 A1 BD 26 23
000018E0:  59 32 27 1F 73 0B 8F 79  4E 45 DA B3 AB 08 04 F4
000018F0:  0B B9 50 46 D3 91 41 84
```

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">SaltSize</td></tr>
<tr><td colspan="32">Salt (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">EncryptedVerifier (16 bytes)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">VerifierHashSize</td></tr>
<tr><td colspan="32">EncryptedVerifierHash (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

SaltSize (4 bytes): 0x00000010 specifies the number of bytes used by the **Salt** field and the number of bytes used by **EncryptedVerifier** field.

Salt (variable): "92 25 50 F6 B6 4F FE 5B D3 96 DF 5E E9 17 DA 3A" specifies a randomly generated value used when generating the encryption key.

EncryptedVerifier (16 bytes): An encrypted form of a randomly generated, 16-byte verifier value, which is the randomly generated **Verifier** value encrypted using the algorithm chosen by the implementation—for example, "BF 86 E1 8F 64 9D 17 D0 A5 41 D9 45 CE FD 96 0C".

VerifierHashSize (4 bytes): 0x00000014 specifies the number of bytes used by the hash of the randomly generated **Verifier**.

EncryptedVerifierHash (variable): An array of bytes that contains the encrypted form of the hash of the randomly generated **Verifier** value—for example, "12 FF DC 88 A1 BD 26 23 59 32 27 1F 73 0B 8F 79 4E 45 DA B3 AB 08 04 F4 0B B9 50 46 D3 91 41 84".

3.8 \EncryptionInfo Stream

This section provides an example of an **\EncryptionInfo** **stream** containing detailed information used to initialize the cryptography that is used to encrypt the **\EncryptedPackage** stream.

```
00001800: 03 00 02 00 24 00 00 00  A4 00 00 00 24 00 00 00
00001810: 00 00 00 00 0E 66 00 00  04 80 00 00 80 00 00 00
00001820: 18 00 00 00 E0 BC 3B 07  00 00 00 00 4D 00 69 00
00001830: 63 00 72 00 6F 00 73 00  6F 00 66 00 74 00 20 00
00001840: 45 00 6E 00 68 00 61 00  6E 00 63 00 65 00 64 00
```

```
00001850: 20 00 52 00 53 00 41 00  20 00 61 00 6E 00 64 00
00001860: 20 00 41 00 45 00 53 00  20 00 43 00 72 00 79 00
00001870: 70 00 74 00 6F 00 67 00  72 00 61 00 70 00 68 00
00001880: 69 00 63 00 20 00 50 00  72 00 6F 00 76 00 69 00
00001890: 64 00 65 00 72 00 20 00  28 00 50 00 72 00 6F 00
000018A0: 74 00 6F 00 74 00 79 00  70 00 65 00 29 00 00 00
000018B0: 10 00 00 00 92 25 50 F6  B6 4F FE 5B D3 96 DF 5E
000018C0: E9 17 DA 3A BF 86 E1 8F  64 9D 17 D0 A5 41 D9 45
000018D0: CE FD 96 0C 14 00 00 00  12 FF DC 88 A1 BD 26 23
000018E0: 59 32 27 1F 73 0B 8F 79  4E 45 DA B3 AB 08 04 F4
000018F0: 0B B9 50 46 D3 91 41 84
```

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="16">EncryptionVersionInfo.vMajor</td><td colspan="16">EncryptionVersionInfo.vMinor</td></tr>
<tr><td colspan="32">EncryptionHeader.Flags</td></tr>
<tr><td colspan="32">EncryptionHeaderSize</td></tr>
<tr><td colspan="32">EncryptionHeader</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">EncryptionVerifier</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

EncryptionVersionInfo.vMajor (2 bytes): 0x0003 specifies the major version.

EncryptionVersionInfo.vMinor (2 bytes): 0x0002 specifies the minor version.

EncryptionHeader.Flags (4 bytes): 0x00000024 specifies that the CryptoAPI implementation (0x0000004) of the ECMA-376 AES (0x00000020) algorithm [ECMA-376] was used to encrypt the file.

EncryptionHeaderSize (4 bytes): 0x000000A4 specifies the number of bytes used by the **EncryptionHeader** structure (section 2.3.2).

EncryptionHeader (variable): This field consists of the following:

- **Flags:** 0x00000024 is a bit flag that specifies that the CryptoAPI implementation (0x0000004) of the ECMA-376 AES (0x00000020) algorithm [ECMA-376] was used to encrypt the file.
- **SizeExtra:** 0x00000000 is unused.
- **AlgID:** 0x0000660E specifies that the file is encrypted using the AES-128 encryption algorithm.
- **AlgIDHash:** 0x00008004 specifies that the hashing algorithm used is SHA-1.
- **KeySize:** 0x00000080 specifies that the key size is 128 bits.
- **ProviderType:** 0x00000018 specifies that AES is the provider type.
- **Reserved1:** 0x073BBCE0 is a reserved value.

- **Reserved2:** 0x00000000 is a reserved value.
- **CSPName:** "Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)" specifies the name of the cryptographic provider.

Example

```
24 00 00 00 00 00 00 00 0E 66 00 00 04 80 00 00
80 00 00 00 18 00 00 00 E0 BC 3B 07 00 00 00 00
4D 00 69 00 63 00 72 00 6F 00 73 00 6F 00 66 00
74 00 20 00 45 00 6E 00 68 00 61 00 6E 00 63 00
65 00 64 00 20 00 52 00 53 00 41 00 20 00 61 00
6E 00 64 00 20 00 41 00 45 00 53 00 20 00 43 00
72 00 79 00 70 00 74 00 6F 00 67 00 72 00 61 00
70 00 68 00 69 00 63 00 20 00 50 00 72 00 6F 00
76 00 69 00 64 00 65 00 72 00 20 00 28 00 50 00
72 00 6F 00 74 00 6F 00 74 00 79 00 70 00 65 00
29 00 00 00
```

EncryptionVerifier (variable): This field consists of the following:

- **SaltSize:** 0x00000010 specifies the number of bytes that make up the **Salt** field.
- **Salt:** "92 25 50 F6 B6 4F FE 5B D3 96 DF 5E E9 17 DA 3A" specifies a randomly generated value used when generating the encryption key.
- **EncryptedVerifier:** "BF 86 E1 8F 64 9D 17 D0 A5 41 D9 45 CE FD 96 0C" specifies the encrypted form of the verifier.
- **VerifierHashSize**: 0x00000014 specifies the number of bytes needed to contain the hash of the verifier used to generate the **EncryptedVerifier** field.
- **EncryptedVerifierHash:** "12 FF DC 88 A1 BD 26 23 59 32 27 1F 73 0B 8F 79 4E 45 DA B3 AB 08 04 F4 0B B9 50 46 D3 91 41 84" specifies the encrypted hash of the verifier used to generate the **EncryptedVerifier** field.

Example

```
92 25 50 F6 B6 4F FE 5B D3 96 DF 5E E9 17 DA 3A
BF 86 E1 8F 64 9D 17 D0 A5 41 D9 45 CE FD 96 0C
14 00 00 00 12 FF DC 88 A1 BD 26 23 59 32 27 1F
73 0B 8F 79 4E 45 DA B3 AB 08 04 F4 0B B9 50 46
D3 91 41 84
```

3.9 \EncryptionInfo Stream (Third-Party Extensible Encryption)

This section provides an example of the XML structure for an **EncryptionInfo** field as specified in section 2.3.4.6.

```
<EncryptionData xmlns="urn:schemas-microsoft-com:office:office">
     <EncryptionProvider Id="{05F17A8A-189E-42CD-9B21-E8F6B730EC8A}"
                         Url="http://www.contoso.com/DownloadProvider/">
         <EncryptionProviderData>AAAAAA==</EncryptionProviderData>
    </EncryptionProvider>
</EncryptionData>
```

EncryptionData xmlns: "urn:schemas-microsoft-com:office:office" specifies the XML namespace for this XML fragment.

EncryptionProvider: Specifies the code module that contains the cryptographic functionality used in this document with the following attributes:

- **Id:** "{05F17A8A-189E-42CD-9B21-E8F6B730EC8A}" specifies a unique identifier for the encryption provider.
- **Url:** "http://www.contoso.com/DownloadProvider/" specifies the URL for the location of the **EncryptionProvider** code module.

EncryptionProviderData: Data for consumption by the extensible encryption module specified in the **EncryptionProvider** node.

3.10 Office Binary Document RC4 Encryption

3.10.1 Encryption Header

This section provides an example of an RC4 encryption header structure (section 2.3.6.1) used by Office Binary Document RC4 Encryption (section 2.3.6) to specify the encryption properties for an encrypted **stream**.

```
00001200: 01 00 01 00 C4 DC 85 69  91 13 EC 1C F1 E5 29 06
00001210: 0E 49 00 B3 F3 53 BB 80  36 63 CD E3 DD F2 D1 CB
00001220: 10 23 9B 5A 39 8F EA C2  43 EC F4 4B 9A 62 29 1B
00001230: 1A 4C 9D CD
```

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">EncryptionVersionInfo</td></tr>
<tr><td colspan="32">Salt (16 bytes)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">EncryptedVerifier (16 bytes)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">EncryptedVerifierHash (16 bytes)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

EncryptionVersionInfo (4 bytes): A value specifying that **Version.vMajor** is 0x0001 and **Version.vMinor** is 0x0001.

Salt (16 bytes): "C4 DC 85 69 91 13 EC 1C F1 E5 29 06 0E 49 00 B3" specifies a randomly generated value that is used when generating the encryption key.

EncryptedVerifier (16 bytes): "F3 53 BB 80 36 63 CD E3 DD F2 D1 CB 10 23 9B 5A" specifies that the verifier is encrypted using a 40-bit RC4 cipher initialized as specified in section 2.3.6.2, with a block number of 0x00000000.

EncryptedVerifierHash (16 bytes): "39 8F EA C2 43 EC F4 4B 9A 62 29 1B 1A 4C 9D CD" specifies an MD5 hash of the verifier used to create the **EncryptedVerifier** field.

3.11 PasswordKeyEncryptor (Agile Encryption)


```
00000000:  04 00 04 00 40 00 00 00  3C 3F 78 6D 6C 20 76 65
00000010:  72 73 69 6F 6E 3D 22 31  2E 30 22 20 65 6E 63 6F
00000020:  64 69 6E 67 3D 22 55 54  46 2D 38 22 20 73 74 61
00000030:  6E 64 61 6C 6F 6E 65 3D  22 79 65 73 22 3F 3E 0D
00000040:  0A 3C 65 6E 63 72 79 70  74 69 6F 6E 20 78 6D 6C
00000050:  6E 73 3D 22 68 74 74 70  3A 2F 2F 73 63 68 65 6D
00000060:  61 73 2E 6D 69 63 72 6F  73 6F 66 74 2E 63 6F 6D
00000070:  2F 6F 66 66 69 63 65 2F  32 30 30 36 2F 65 6E 63
00000080:  72 79 70 74 69 6F 6E 22  20 78 6D 6C 6E 73 3A 70
00000090:  3D 22 68 74 74 70 3A 2F  2F 73 63 68 65 6D 61 73
000000A0:  2E 6D 69 63 72 6F 73 6F  66 74 2E 63 6F 6D 2F 6F
000000B0:  66 66 69 63 65 2F 32 30  30 36 2F 6B 65 79 45 6E
000000C0:  63 72 79 70 74 6F 72 2F  70 61 73 73 77 6F 72 64
000000D0:  22 3E 3C 6B 65 79 44 61  74 61 20 73 61 6C 74 53
000000E0:  69 7A 65 3D 22 31 36 22  20 62 6C 6F 63 6B 53 69
000000F0:  7A 65 3D 22 31 36 22 20  6B 65 79 42 69 74 73 3D
00000100:  22 31 32 38 22 20 68 61  73 68 53 69 7A 65 3D 22
00000110:  32 30 22 20 63 69 70 68  65 72 41 6C 67 6F 72 69
00000120:  74 68 6D 3D 22 41 45 53  22 20 63 69 70 68 65 72
00000130:  43 68 61 69 6E 69 6E 67  3D 22 43 68 61 69 6E 69
00000140:  6E 67 4D 6F 64 65 43 42  43 22 20 68 61 73 68 41
00000150:  6C 67 6F 72 69 74 68 6D  3D 22 53 48 41 31 22 20
00000160:  73 61 6C 74 56 61 6C 75  65 3D 22 2F 61 34 69 57
00000170:  71 50 79 49 76 45 32 63  55 6F 6C 4A 4D 4B 72 49
00000180:  77 3D 3D 22 2F 3E 3C 64  61 74 61 49 6E 74 65 67
00000190:  72 69 74 79 20 65 6E 63  72 79 70 74 65 64 48 6D
000001A0:  61 63 4B 65 79 3D 22 75  77 70 41 45 46 57 31 68
000001B0:  51 79 44 32 4F 30 31 6B  7A 31 6C 68 6A 65 76 4E
000001C0:  77 30 45 43 79 41 41 30  75 32 4F 78 44 79 67 73
000001D0:  66 59 3D 22 20 65 6E 63  72 79 70 74 65 64 48 6D
000001E0:  61 63 56 61 6C 75 65 3D  22 75 66 36 48 62 4A 6A
000001F0:  74 72 79 4A 4F 6A 53 46  71 72 6B 71 6B 4E 51 59
00000200:  39 4E 6A 4E 51 55 50 49  2B 78 63 6B 38 51 38 79
00000210:  34 6D 6B 6F 3D 22 2F 3E  3C 6B 65 79 45 6E 63 72
00000220:  79 70 74 6F 72 73 3E 3C  6B 65 79 45 6E 63 72 79
00000230:  70 74 6F 72 20 75 72 69  3D 22 68 74 74 70 3A 2F
00000240:  2F 73 63 68 65 6D 61 73  2E 6D 69 63 72 6F 73 6F
00000250:  66 74 2E 63 6F 6D 2F 6F  66 66 69 63 65 2F 32 30
00000260:  30 36 2F 6B 65 79 45 6E  63 72 79 70 74 6F 72 2F
00000270:  70 61 73 73 77 6F 72 64  22 3E 3C 70 3A 65 6E 63
00000280:  72 79 70 74 65 64 4B 65  79 20 73 70 69 6E 43 6F
00000290:  75 6E 74 3D 22 31 30 30  30 30 30 22 20 73 61 6C
000002A0:  74 53 69 7A 65 3D 22 31  36 22 20 62 6C 6F 63 6B
000002B0:  53 69 7A 65 3D 22 31 36  22 20 6B 65 79 42 69 74
000002C0:  73 3D 22 31 32 38 22 20  68 61 73 68 53 69 7A 65
000002D0:  3D 22 32 30 22 20 63 69  70 68 65 72 41 6C 67 6F
000002E0:  72 69 74 68 6D 3D 22 41  45 53 22 20 63 69 70 68
000002F0:  65 72 43 68 61 69 6E 69  6E 67 3D 22 43 68 61 69
00000300:  6E 69 6E 67 4D 6F 64 65  43 42 43 22 20 68 61 73
00000310:  68 41 6C 67 6F 72 69 74  68 6D 3D 22 53 48 41 31
00000320:  22 20 73 61 6C 74 56 61  6C 75 65 3D 22 70 70 73
00000330:  36 42 31 62 6D 71 43 46  58 67 6F 70 73 6D 31 72
00000340:  57 6E 51 3D 3D 22 20 65  6E 63 72 79 70 74 65 64
00000350:  56 65 72 69 66 69 65 72  48 61 73 68 49 6E 70 75
00000360:  74 3D 22 4A 59 55 34 51  30 75 32 42 68 71 7A 51
```

```
00000370:  41 35 44 34 4A 2F 76 6F  41 3D 3D 22 20 65 6E 63
00000380:  72 79 70 74 65 64 56 65  72 69 66 69 65 72 48 61
00000390:  73 68 56 61 6C 75 65 3D  22 65 42 32 6A 58 35 6D
000003A0:  76 68 42 4A 2B 39 4F 37  66 66 43 2B 36 58 32 4D
000003B0:  79 64 7A 32 67 6C 48 4F  58 78 30 54 39 50 6E 36
000003C0:  6E 4B 2B 77 3D 22 20 65  6E 63 72 79 70 74 65 64
000003D0:  4B 65 79 56 61 6C 75 65  3D 22 32 46 38 36 48 47
000003E0:  2B 78 56 33 6E 47 61 32  37 44 45 6C 67 71 67 77
000003F0:  3D 3D 22 2F 3E 3C 2F 6B  65 79 45 6E 63 72 79 70
00000400:  74 6F 72 3E 3C 2F 6B 65  79 45 6E 63 72 79 70 74
00000410:  6F 72 73 3E 3C 2F 65 6E  63 72 79 70 74 69 6F 6E
00000420:  3E
```

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="16">EncryptionVersionInfo.vMajor</td><td colspan="16">EncryptionVersionInfo.vMinor</td></tr>
<tr><td colspan="32">Reserved</td></tr>
<tr><td colspan="32">XmlEncryptionDescriptor (variable)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

EncryptionVersionInfo.vMajor (2 bytes): 0x0004 specifies the major version.

EncryptionVersionInfo.vMinor (2 bytes): 0x0004 specifies the minor version.

Reserved (4 bytes): 0x00000040 is a reserved value.

XmlEncryptionDescriptor (variable): An XML block that specifies the encryption algorithms used and that contains the following XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<encryption
  xmlns="http://schemas.microsoft.com/office/2006/encryption"
  xmlns:p="http://schemas.microsoft.com/office/2006/keyEncryptor/password">
  <keyData
    saltSize="16"
    blockSize="16"
    keyBits="128"
    hashSize="20"
    cipherAlgorithm="AES"
    cipherChaining="ChainingModeCBC"
    hashAlgorithm="SHA-1"
    saltValue="/a4iWqPyIvE2cUolJMKrIw=="/>

  <dataIntegrity
    encryptedHmacKey="uwpAEFW1hQyD2O01kz1lhjevNw0ECyAA0u2OxDygsfY="
    encryptedHmacValue="uf6HbJjtryJOjSFqrkqkNQY9NjNQUPI+xck8Q8y4mko="/>

  <keyEncryptors>
    <keyEncryptor uri="http://schemas.microsoft.com/office/2006/keyEncryptor/password">
      <p:encryptedKey
        spinCount="100000"
        saltSize="16"
        blockSize="16"
        keyBits="128"
        hashSize="20"
        cipherAlgorithm="AES"
        cipherChaining="ChainingModeCBC"
```

```
        hashAlgorithm="SHA-1"
        saltValue="pps6B1bmqCFXgopsm1rWnQ=="
        encryptedVerifierHashInput="JYU4Q0u2BhqzQA5D4J/voA=="
        encryptedVerifierHashValue="eB2jX5mvhBJ+9O7ffC+6X2Mydz2glHOXx0T9Pn6nK+w="
        encryptedKeyValue="2F86HG+xV3nGa27DElgqgw=="/>
    </keyEncryptor>
  </keyEncryptors>
</encryption>
```

keyData: The cryptographic attributes used to encrypt the data.

saltSize: 16 specifies that the salt value is 16 bytes in length.

blockSize: 16 specifies that 16 bytes were used to encrypt each block of data.

keyBits: 128 specifies that the key used to encrypt the data is 128 bits in length.

hashSize: 20 specifies that the hash size is 20 bytes in length.

cipherAlgorithm: "AES" specifies that the cipher algorithm used to encrypt the data is AES.

cipherChaining: "ChainingModeCBC" specifies that the chaining mode to encrypt the data is CBC.

hashAlgorithm: "SHA-1" specifies that the hashing algorithm used to hash the data is SHA-1.

SaltValue: "/a4iWqPyIvE2cUolJMKrIw==" specifies a randomly generated value used when generating the encryption key.

dataIntegrity: Specifies the encrypted copies of the salt and hash values used to help ensure that the integrity of the encrypted data has not been compromised.

encryptedHmacKey: "uwpAEFW1hQyD2O01kz1lhjevNw0ECyAA0u2OxDygsfY=" specifies the encrypted copy of the randomly generated value used when generating the encryption key.

encryptedHmacValue: "uf6HbJjtryJOjSFqrkqkNQY9NjNQUPI+xck8Q8y4mko=" specifies the encrypted copy of the hash value that is generated during the creation of the encryption key.

keyEncryptors: Specifies the key encryptors used to encrypt the data.

keyEncryptor: "http://schemas.microsoft.com/office/2006/keyEncryptor/password" specifies that the schema used by this encryptor is the schema specified in section 2.3.4.10 for password-based encryptors.

p:encryptedKey: The attributes used to generate the encrypting key.

spinCount: 100000 specifies that there are 100000 iterations on the hash of the password.

saltSize: 16 specifies that the salt value is 16 bytes long.

blockSize: 16 specifies that 16 bytes were used to encrypt each block of data.

keyBits: 128 specifies that the key is 128 bits in length.

hashSize: 20 specifies that the hash is 20 bytes in length.

cipherAlgorithm: "AES" specifies that the cipher used to encrypt the data is AES.

cipherChaining: "ChainingModeCBC" specifies that the chaining mode used for encrypting is CBC.

hashAlgorithm: "SHA-1" specifies that the hashing algorithm used is SHA-1.

saltValue: "pps6B1bmqCFXgopsm1rWnQ==" specifies the randomly generated value used for encrypting the data.

encryptedVerifierHashInput: "JYU4Q0u2BhqzQA5D4J/voA==" specifies the **VerifierHashInput** attribute encoded as specified in section 2.3.4.13.

encryptedVerifierHashValue: "eB2jX5mvhBJ+9O7ffC+6X2Mydz2glHOXx0T9Pn6nK+w=" specifies the **VerifierHashValue** encoded as specified in section 2.3.4.13.

encryptedKeyValue: "2F86HG+xV3nGa27DElgqgw==" specifies the **KeyValue** encoded as specified in section 2.3.4.13.

4 Security

4.1 Security Considerations for Implementers

4.1.1 Data Spaces

None.

4.1.2 Information Rights Management

It is recommended that software components that implement the **Information Rights Management (IRM)** Data Space make a best effort to respect the licensing limitations applied to the protected content in the document.

Security considerations concerning rights management are as described in [MS-RMPR].

4.1.3 Encryption

4.1.3.1 ECMA-376 Document Encryption

ECMA-376 document encryption [ECMA-376] using standard encryption does not support **CBC** and does not have a provision for detecting corruption, although a block cipher (specifically, AES) is not known to be subject to bit-flipping attacks. ECMA-376 documents using agile encryption are required to use CBC and corruption detection, and are not subject to the issues noted for standard encryption.

When setting algorithms for agile encryption, the SHA-2 series of hashing algorithms is preferred. MD2, MD4, and MD5 are not recommended. Older cipher algorithms, such as DES, are also not recommended.

Passwords are limited to 255 Unicode code points.

4.1.3.2 Office Binary Document RC4 CryptoAPI Encryption

The Office binary document RC4 CryptoAPI encryption method is not recommended and ought to be used only when backward compatibility is required.

Passwords are limited to 255 Unicode characters.

Office binary document RC4 CryptoAPI encryption has the following known cryptographic weaknesses:

- The key derivation algorithm described in section 2.3.5.2 is weak because of the lack of a repeated iteration mechanism, and the password might be subject to rapid brute-force attacks.
- Encryption begins with the first byte and does not throw away an initial range as is recommended to overcome a known weakness in the RC4 pseudorandom number generator.
- No provision is made for detecting corruption within the encryption **stream**, which exposes encrypted data to bit-flipping attacks.
- When used with small key lengths (such as 40-bit), brute-force attacks on the key without knowing the password are possible.
- Some streams are not encrypted.
- Key stream reuse can occur in document data streams, potentially with known plaintext, implying that certain portions of encrypted data can be either directly extracted or trivially retrieved.

- Key stream reuse occurs multiple times within the RC4 CryptoAPI Encrypted Summary stream.
- Document properties might not be encrypted, which could result in information leakage.

Because of the cryptographic weaknesses of the Office binary document RC4 CryptoAPI encryption, it is considered insecure, and therefore is not recommended when storing sensitive materials.

4.1.3.3 Office Binary Document RC4 Encryption

The Office binary document RC4 encryption method is not recommended, and ought to be used only when backward compatibility is required.

Passwords are limited to 255 Unicode characters.

Office binary document RC4 encryption has the following known cryptographic weaknesses:

- The key derivation algorithm is not an iterated hash, as described in [RFC2898], which allows brute-force attacks against the password to be performed rapidly.
- Encryption begins with the first byte, and does not throw away an initial range as is recommended to overcome a known weakness in the RC4 pseudorandom number generator.
- No provision is made for detecting corruption within the encryption **stream**, which exposes encrypted data to bit-flipping attacks.
- While the derived encryption key is actually 128 bits, the input used to derive the key is fixed at 40 bits, and current hardware enables brute-force attacks on the encryption key without knowing the password in a relatively short period of time so that even if the password cannot easily be recovered, the information could still be disclosed.
- Some streams might not be encrypted.
- Depending on the application, key stream reuse could occur, potentially with known plaintext, implying that certain portions of encrypted data could be either directly extracted or easily retrieved.
- Document properties might not be encrypted, which could result in information leakage.

Because of the cryptographic weaknesses of the Office Binary Document RC4 Encryption, it is considered easily reversible and therefore is not recommended when storing sensitive materials.

4.1.3.4 XOR Obfuscation

XOR obfuscation is not recommended. Document data can easily be extracted. The document password could be retrievable.

Passwords are truncated to 15 characters. It is possible for multiple passwords to map to the same key.

4.1.4 Document Write Protection

Document write protection methods 1 (section 2.4.2.1) and 3 (section 2.4.2.3) both embed the password in plaintext into the file. Although method 3 subsequently encrypts the file, the encryption is flawed, and the password is described in section 2.4.2.3. In both cases, the password can be extracted with little difficulty. Document write protection is not considered to be a security mechanism, and the write protection can easily be removed by using a binary editor. Document write protection is meant to protect against accidental modification only.

Some file formats, such as those described in [MS-DOC] and [MS-XLS], restrict password length to 15 characters. It is possible for multiple passwords to map to the same key when using document write protection method 2 (section 2.4.2.2).

4.1.5 Binary Document Digital Signatures

Certain **streams** and storages are not subject to signing. Tampering with these streams or storages does not invalidate the signature.

4.2 Index of Security Fields

None.

5 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs.

- Microsoft Office 97
- Microsoft Office 2000
- Microsoft Office XP
- Microsoft Office 2003
- The 2007 Microsoft Office system
- Microsoft Office 2010 suites
- Microsoft Office 2013
- Microsoft Office SharePoint Server 2007
- Microsoft SharePoint Server 2010
- Microsoft SharePoint Server 2013
- Microsoft Office 2016

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms "SHOULD" or "SHOULD NOT" implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term "MAY" implies that the product does not follow the prescription.

<1> Section 2.2: Applications in Office 2003, the 2007 Microsoft Office system, Microsoft Office 2010 suites and Office 2013 versions encrypt the Microsoft Office binary documents by persisting the entire document to a temporary OLE compound file and then transforming the physical representation of the OLE compound file as a single stream of bytes. Similarly, ECMA-376 documents [ECMA-376] are encrypted by adding the entire file package to a temporary file and then transforming the physical representation of the file as a single stream of bytes.

The following streams are also stored outside the protected content to preserve interoperability with applications that do not understand the IRMDS structure:

- **_signatures**
- **0x01CompObj**
- **Macros**
- **_VBA_PROJECT_CUR**
- **0x05SummaryInformation**
- **0x05DocumentSummaryInformation**
- **MsoDataStore**

Applications in Office 2003, the 2007 Office system, Office 2010 and Office 2013 also create the streams and storages necessary to create a default document within the OLE compound file. This default document contains a short message to the user indicating that the actual document contents are encrypted. This allows versions of Microsoft Office that do not understand the IRMDS structure to open the default document instead of rejecting the file.

<2> Section 2.2.1: Office 2003, the 2007 Office system, Office 2010 and Office 2013 offer the user the option of creating a transformed MHTML representation of the document when applying a rights management policy to a document. This option is on by default in Microsoft Office Excel 2003 and off by default in all other applications in Office 2003, and it is off by default in all applications in the 2007 Office system, Office 2010 and Office 2013. If the transformed MHTML representation is created, the **0x09LZXDRMDataSpace** data space definition is applied to it (which includes both LZX compression and encryption).

<3> Section 2.2.2: Office 2003, the 2007 Office system, Office 2010 and Office 2013 offer the user the option of creating a transformed MHTML representation of the document when applying a rights management policy to a document. This option is on by default in Office Excel 2003 and off by default in all other Office 2003 applications, and it is off by default in all applications in the 2007 Office system and newer versions. If the transformed MHTML representation is created, the **0x09LZXDRMDataSpace** data space definition is applied to it (which includes both LZX compression and encryption).

<4> Section 2.2.3: Office 2003, the 2007 Office system, Office 2010 and Office 2013 offer the user the option of creating a transformed MHTML representation of the document when applying a rights management policy to a document. This option is on by default in Office Excel 2003 and off by default in all other Office 2003 applications, and it is off by default in all applications in the 2007 Office system, Office 2010 and Office 2013. If the transformed MHTML representation is created, the **0x09LZXDRMDataSpace** data space definition is applied to it (which includes both LZX compression and encryption).

<5> Section 2.2.6: Office SharePoint Server 2007 uses the **AUTHENTICATEDDATA** element with the **name** set to "ListGUID" as the application-specific GUID that identifies the storage location for the document. This is stored encrypted within the element as follows.

```
<AUTHENTICATEDDATA id="Encrypted-Rights-Data">
```

Once decrypted, the XrML document contains an element named **AUTHENTICATEDDATA**, containing an attribute named **id** with a value of "APPSPECIFIC" and an attribute named **name** with a value of ListGUID with the contents of the ListGUID.

<6> Section 2.2.11: Office 2003, the 2007 Office system, Office 2010 and Office 2013 offer the user the option of creating a transformed MHTML representation of the document when applying a rights management policy to a document. This option is on by default in Office Excel 2003 and off by default in all other Office 2003 applications, and it is off by default in all applications in the 2007 Office system, Office 2010 and Office 2013. If the transformed MHTML representation is created, the **0x09LZXDRMDataSpace** data space definition is applied to it (which includes both LZX compression and encryption).

<7> Section 2.3.1: In the 2007 Office system, the 2007 Office system, Office 2010 and Office 2013, the default encryption algorithm for ECMA-376 standard encryption documents [ECMA-376] is 128-bit AES, and both 192-bit and 256-bit AES are also supported. It is possible to use alternate encryption algorithms, and for best results, a block cipher supporting **ECB** mode is recommended. Additionally, the algorithm ought to convert one block of plaintext to one block of encrypted data, where both blocks are the same size. This information is for guidance only, and it is possible that if alternate algorithms are used, the applications in the 2007 Office system, Office 2010 and Office 2013 might not open the document properly or that information leakage could occur.

<8> Section 2.3.2: Several of the cryptographic techniques specified in this document use the **Cryptographic Application Programming Interface (CAPI) or CryptoAPI** when implemented by Microsoft Office on the Microsoft Windows operating systems. While an implementation is not required to use CryptoAPI, if an implementation is required to interoperate with the 2007 Office system, the 2007 Office system, Office 2010 and Office 2013 on the Windows XP operating system, Windows Vista operating system, Windows 7 operating system, Windows 8 operating system and Windows 8.1 operating systems, the following are required:

Cryptographic service provider (CSP): A library containing implementations of cryptographic algorithms. Several CSPs that support the algorithms required in this specification are present by default on Windows XP, Windows Vista, Windows 7, Windows 8 and Windows 8.1 operating systems. Alternate CSPs can be used, if the CSP is installed on all systems consuming or producing a document.

AlgID: An integer representing an encryption algorithm in the CryptoAPI. Required **AlgID** values are specified in the remainder of this document. Alternate **AlgID** values can be used if the CSP supporting the alternate **AlgID** is installed on all systems consuming or producing a document.

AlgIDHash: An integer representing a hashing algorithm in the CryptoAPI. Required **AlgIDHash** values are specified in the remainder of this document. For encryption operations, the hashing algorithm is fixed and cannot vary from the algorithms specified.

The following cryptographic providers are recommended to facilitate interoperability across all supported versions of Windows:

- Microsoft Base Cryptographic Provider v1.0
- Microsoft Enhanced Cryptographic Provider v1.0
- Microsoft Enhanced RSA and AES Cryptographic Provider

Note that the following providers are equivalent:

- Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)
- Microsoft Enhanced RSA and AES Cryptographic Provider

The provider listed as "Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)" is found on Windows XP. An implementation needs to treat these providers as equivalent when attempting to resolve a CSP on a Windows system.

When using AES encryption for ECMA-376 documents [ECMA-376], the Microsoft Enhanced RSA and AES Cryptographic Provider is written into the header, unless AES encryption facilities are obtained from an alternate cryptographic provider as noted in the next paragraph. When using CryptoAPI RC4 encryption, be aware that the Microsoft Base Cryptographic Provider v1.0 is limited to 56-bit key lengths. The other providers listed support up to 128-bit key lengths.

Other cryptographic providers can be used, but documents specifying other providers will not open properly if the cryptographic provider is not present. On a non-Windows system, the cryptographic provider will be ignored when opening a file, and the algorithm and key length will be determined by the **EncryptionHeader.AlgID** and **EncryptionHeader.KeySize** fields. When writing a file from a non-Windows system, a correct cryptographic provider needs to be supplied for implementations on Windows systems to properly open the file.

Additionally, a **ProviderType** parameter is required for an **EncryptionHeader** structure that is compatible with the CSP and encryption algorithm chosen. To facilitate interoperability, the **ProviderTypes** listed in section 2.3.2 are recommended.

Additionally, see section 4.1.3 for additional information regarding the cryptography used.

<9> Section 2.3.4.5: Office 2003 applications set a **Version.vMajor** version value of 0x0002. Applications in the 2007 Office system and Microsoft Office 2007 Service Pack 1 (SP1) set a

Version.vMajor value of 0x0003. Versions Microsoft Office 2007 Service Pack 2 (SP2), Office 2010 and Office 2013 set a **Version.vMajor** value of 0x0004.

<10> Section 2.3.4.5: In the 2007 Office system, Office 2010 and Office 2013, the default encryption algorithm for ECMA-376 standard encryption documents [ECMA-376] is 128-bit AES, and both 192-bit and 256-bit AES are also supported. It is possible to use alternate encryption algorithms, and for best results, a block cipher supporting ECB mode is recommended. Additionally, the algorithm ought to convert one block of plaintext to one block of encrypted data, where both blocks are the same size. This information is for guidance only, and it is possible that if alternate algorithms are used, the applications in the 2007 Office system, Office 2010 and Office 2013 might not open the document properly or that information leakage could occur.

<11> Section 2.3.4.5: In the 2007 Office system, Office 2010 and Office 2013, the default encryption algorithm for ECMA-376 standard encryption documents [ECMA-376] is 128-bit AES, and both 192-bit and 256-bit AES are also supported. It is possible to use alternate encryption algorithms, and for best results, a block cipher supporting ECB mode is recommended. Additionally, the algorithm ought to convert one block of plaintext to one block of encrypted data, where both blocks are the same size. This information is for guidance only, and it is possible that if alternate algorithms are used, the applications in the 2007 Office system, Office 2010 and Office 2013 might not open the document properly or that information leakage could occur.

<12> Section 2.3.4.6: On Windows XP, Windows Vista, Windows 7, Windows 8 and Windows 8.1, **CSPName** specifies the GUID of the extensible encryption module used for this file format. This GUID specifies the CLSID of the **COM** module containing cryptographic functionality. The **CSPName** is required to be a null-terminated Unicode string.

<13> Section 2.3.4.10: The use of RC2 is not recommended. If RC2 is used with a key length of less than 128 bits, documents could interoperate incorrectly across different operating system versions.

<14> Section 2.3.4.10: The use of DES is not recommended. If DES is used, the key length specified in the **KeyBits** element is required to be set to 64 for 56-bit encryption, and the key decrypted from **encryptedKeyValue** of **KeyEncryptor** is required to include the DES parity bits.

<15> Section 2.3.4.10: The use of DESX is not recommended. If DESX is used, documents could interoperate incorrectly across different operating system versions.

<16> Section 2.3.4.10: If 3DES or 3DES_112 is used, the key length specified in the **KeyBits** element is required to be set to 192 for 168-bit encryption and 128 for 112-bit encryption, and the key decrypted from **encryptedKeyValue** of **KeyEncryptor** is required to include the DES parity bits.

<17> Section 2.3.4.10: If 3DES or 3DES_112 is used, the key length specified in the **KeyBits** element is required to be set to 192 for 168-bit encryption and 128 for 112-bit encryption, and the key decrypted from **encryptedKeyValue** of **KeyEncryptor** is required to include the DES parity bits.

<18> Section 2.3.4.10: Any algorithm that can be resolved by name by the underlying operating system can be used for hashing or encryption. Only block algorithms are supported for encryption. AES-128 is the default encryption algorithm, and SHA-1 is the default hashing algorithm if no other algorithms have been configured.

<19> Section 2.3.4.10: Any algorithm that can be resolved by name by the underlying operating system can be used for hashing or encryption. Only block algorithms are supported for encryption. AES-128 is the default encryption algorithm, and SHA-1 is the default hashing algorithm if no other algorithms have been configured.

<20> Section 2.3.4.10: All ECMA-376 documents [ECMA-376] encrypted by Microsoft Office using agile encryption will have a **DataIntegrity** element present. The schema allows for a **DataIntegrity** element to not be present because the encryption schema can be used by applications that do not create ECMA-376 documents [ECMA-376].

<21> Section 2.3.5.1: Office 2003 applications set a **Version.vMajor** version of 0x0002. Applications in the 2007 Office system and Office 2007 SP1 set a **Version.vMajor** value of 0x0003. Versions such as Office 2007 SP2, Office 2010 and Office 2013set a **Version.vMajor** value of 0x004.

<22> Section 2.3.5.1: Several of the cryptographic techniques specified in this document use the Cryptographic Application Programming Interface (CAPI) or CryptoAPI when implemented by Microsoft Office on the Windows operating systems. While an implementation is not required to use CryptoAPI, if an implementation is required to interoperate with Microsoft Office on the Windows operating systems, the following are required:

Cryptographic service provider (CSP): A CSP refers to a library containing implementations of cryptographic algorithms. Several CSPs that support the algorithms required in this specification are present by default on the latest versions of Windows. Alternate CSPs can be used, if the CSP is installed on all systems consuming or producing a document.

AlgID: An integer representing an encryption algorithm in the CryptoAPI. Required **AlgID** values are specified in the remainder of this document. Alternate **AlgIDs** can be used if the CSP supporting the alternate **AlgID** is installed on all systems consuming or producing a document.

AlgIDHash: An integer representing a hashing algorithm in the CryptoAPI. Required **AlgIDHash** values are specified in the remainder of this document. For encryption operations, the hashing algorithm is fixed and cannot vary from the algorithms specified.

The following cryptographic providers are recommended to facilitate interoperability across all supported versions of Windows:

- Microsoft Base Cryptographic Provider v1.0
- Microsoft Enhanced Cryptographic Provider v1.0
- Microsoft Enhanced RSA and AES Cryptographic Provider

Note that the following providers are equivalent:

- Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)
- Microsoft Enhanced RSA and AES Cryptographic Provider

The provider listed as "Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)" is found on Windows XP. An implementation needs to treat these providers as equivalent when attempting to resolve a CSP on a Windows system.

When using AES encryption for ECMA-376 documents [ECMA-376], the Microsoft Enhanced RSA and AES Cryptographic Provider is written into the header, unless AES encryption facilities are obtained from an alternate cryptographic provider as noted in the next paragraph. When using CryptoAPI RC4 encryption, be aware that the Microsoft Base Cryptographic Provider v1.0 is limited to 56-bit key lengths. The other providers listed support up to 128-bit key lengths.

Other cryptographic providers can be used, but documents specifying other providers might not open properly if the cryptographic provider is not present. On a non-Windows system, the cryptographic provider will be ignored when opening a file, and the algorithm and key length will be determined by the **EncryptionHeader.AlgID** and **EncryptionHeader.KeySize** fields. When writing a file from a non-Windows system, a correct cryptographic provider needs to be supplied for implementations on Windows systems to properly open the file.

Additionally, a **ProviderType** parameter is required for an **EncryptionHeader** structure that is compatible with the CSP and encryption algorithm chosen. To facilitate interoperability, the **ProviderTypes** listed in section 2.3.2 are recommended.

Additionally, see section 4.1.3 for additional information regarding the cryptography used.

<23> Section 2.3.5.4: Office 2003, the 2007 Office system, Office 2010 and Office 2013 allow the user to optionally encrypt the **\0x05SummaryInformation** and **\0x05DocumentSummaryInformation** streams. Additional streams and storages can also be encrypted within the RC4 CryptoAPI summary stream.

<24> Section 2.4.1: Documents generated by Microsoft Office Excel 2007, Microsoft Excel2010 and Microsoft Excel 2013 can be encrypted as specified in section 2.3 with the following password: "\x56\x65\x6C\x76\x65\x74\x53\x77\x65\x61\x74\x73\x68\x6F\x70". The conditions under which this password is used are described in [MS-XLS] and [MS-XLSB].

<25> Section 2.4.2.2: Documents generated by Office Excel 2007, Excel 2010 and Excel 2013 can be encrypted as specified in section 2.3 with the following password: "\x56\x65\x6C\x76\x65\x74\x53\x77\x65\x61\x74\x73\x68\x6F\x70". The conditions under which this password is used are described in [MS-XLS] and [MS-XLSB].

<26> Section 2.4.2.3: Documents created by Microsoft Office PowerPoint 2003, Microsoft Office PowerPoint 2007 and Microsoft Office PowerPoint 2007 Service Pack 1 use the default password. Microsoft Office PowerPoint 2007 Service Pack 2 does not use the default password. A document created without the default password can be opened in earlier versions. Due to security concerns, it is preferable not to use the default password.

<27> Section 2.4.2.4: Any algorithm that can be resolved by name by the underlying operating system can be used for hashing or encryption. Only block algorithms are supported for encryption. AES-128 is the default encryption algorithm, and SHA-1 is the default hashing algorithm if no other algorithms have been configured.

<28> Section 2.5.2.1: In the 2007 Office system, the SHA-1 hashing algorithm is required to be used for this purpose. Office 2010 and Office 2013 require only that the underlying operating system support the hashing algorithm.

<29> Section 2.5.2.1: In the 2007 Office system, the SHA-1 hashing algorithm is required to be used for this purpose. Office 2010 and Office 2013 require only that the underlying operating system support the hashing algorithm.

<30> Section 2.5.2.4: In the 2007 Office system, the SHA-1 hashing algorithm is required to be used for this purpose. Office 2010 and Office 2013 versions require only that the underlying operating system support the hashing algorithm.

<31> Section 2.5.2.5: Office 2010, Office 2013 and the 2007 Office system reserve the value of {00000000-0000-0000-0000-000000000000} for their default signature providers and {000CD6A4-0000-0000-C000-000000000046} for their East Asian signature providers.

<32> Section 2.5.2.6: Office 2010 and Office 2013 adds XML Advanced Electronic Signatures ([XAdES]) extensions to xmldsig signatures when configured to do so by the user. By default, XAdES-EPES signatures are used, as specified in [XAdES] section 4.4.2.

<33> Section 2.5.2.6: By default, Office 2010 and Office 2013 places the reference to the **SignedProperties** element within the **SignedInfo** element. the 2007 Office system needs an update to correctly validate a reference within the **SignedInfo** element that is not to a top-level **Object** element, and incorrectly rejects these signatures as invalid. To ensure compatibility with earlier versions of Office that have not been updated to validate the signature correctly, an implementation can place the **Reference** element within a manifest.

6 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

7 Index

E

K

L

N

O

P

R

S

T

U

V

W

X