Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation (“this documentation”) for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.

- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.

- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.

- **Patents.** Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft [Open Specifications Promise](https://aka.ms/osp) or the [Microsoft Community Promise](https://aka.ms/mcpp). If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.

- **License Programs.** To see all of the protocols in scope under a specific license program and the associated patents, visit the [Patent Map](https://aka.ms/patentmap).

- **Trademarks.** The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit [www.microsoft.com/trademarks](https://aka.ms/trademarks).

- **Fictitious Names.** The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

**Reservation of Rights.** All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

**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

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision History</th>
<th>Revision Class</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/4/2008</td>
<td>0.1</td>
<td>New</td>
<td>Initial Availability</td>
</tr>
<tr>
<td>6/27/2008</td>
<td>1.0</td>
<td>Major</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>10/6/2008</td>
<td>1.01</td>
<td>Editorial</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>12/12/2008</td>
<td>1.02</td>
<td>Editorial</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>3/18/2009</td>
<td>1.03</td>
<td>Editorial</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>7/13/2009</td>
<td>1.04</td>
<td>Major</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>8/28/2009</td>
<td>1.05</td>
<td>Major</td>
<td>Updated and revised the technical content</td>
</tr>
<tr>
<td>11/6/2009</td>
<td>1.06</td>
<td>Editorial</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>2/19/2010</td>
<td>2.0</td>
<td>Editorial</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>3/31/2010</td>
<td>2.01</td>
<td>Editorial</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>4/30/2010</td>
<td>2.02</td>
<td>Editorial</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>6/7/2010</td>
<td>2.03</td>
<td>Editorial</td>
<td>Revised and edited the technical content</td>
</tr>
<tr>
<td>6/29/2010</td>
<td>2.04</td>
<td>Editorial</td>
<td>Changed language and formatting in the technical content.</td>
</tr>
<tr>
<td>7/23/2010</td>
<td>2.05</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>9/27/2010</td>
<td>2.05</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>11/15/2010</td>
<td>2.05</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>12/17/2010</td>
<td>2.05</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>3/18/2011</td>
<td>2.05</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>6/10/2011</td>
<td>2.05</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>1/20/2012</td>
<td>2.6</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>4/11/2012</td>
<td>2.6</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>7/16/2012</td>
<td>2.7</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>10/8/2012</td>
<td>2.8</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>2/11/2013</td>
<td>2.8</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>7/30/2013</td>
<td>2.8</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>11/18/2013</td>
<td>2.8</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>Date</td>
<td>Revision History</td>
<td>Revision Class</td>
<td>Comments</td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td>----------------</td>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>2/10/2014</td>
<td>2.8</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>4/30/2014</td>
<td>3.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>7/31/2014</td>
<td>3.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>10/30/2014</td>
<td>3.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>3/16/2015</td>
<td>4.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>9/4/2015</td>
<td>4.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>7/15/2016</td>
<td>4.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>9/14/2016</td>
<td>4.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>6/20/2017</td>
<td>4.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
<tr>
<td>12/12/2017</td>
<td>4.1</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>4/27/2018</td>
<td>5.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>8/28/2018</td>
<td>6.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>12/11/2018</td>
<td>6.1</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>11/17/2020</td>
<td>7.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>2/16/2021</td>
<td>7.1</td>
<td>Minor</td>
<td>Clarified the meaning of the technical content.</td>
</tr>
<tr>
<td>4/22/2021</td>
<td>8.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>7/20/2021</td>
<td>9.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>8/17/2021</td>
<td>10.0</td>
<td>Major</td>
<td>Significantly changed the technical content.</td>
</tr>
<tr>
<td>10/5/2021</td>
<td>10.0</td>
<td>None</td>
<td>No changes to the meaning, language, or formatting of the technical content.</td>
</tr>
</tbody>
</table>
# Table of Contents

1 Introduction ........................................................................................................... 8
   1.1 Glossary .............................................................................................................. 8
   1.2 References ......................................................................................................... 12
      1.2.1 Normative References .................................................................................. 12
      1.2.2 Informative References ................................................................................ 14
   1.3 Overview ............................................................................................................ 14
      1.3.1 Data Spaces .................................................................................................. 14
      1.3.2 Information Rights Management Data Space ................................................. 15
      1.3.3 Encryption ................................................................................................... 17
         1.3.3.1 XOR Obfuscation ................................................................................. 17
         1.3.3.2 40-bit RC4 Encryption ......................................................................... 17
         1.3.3.3 CryptoAPI RC4 Encryption ................................................................. 17
         1.3.3.4 ECMA-376 Document Encryption ..................................................... 17
      1.3.4 Write Protection ........................................................................................... 18
      1.3.5 Digital Signatures ......................................................................................... 18
      1.3.6 Byte Ordering ............................................................................................. 18
      1.3.7 String Encoding .......................................................................................... 18
      1.3.8 OLE Compound File Path Encoding ............................................................. 18
      1.3.9 Pseudocode Standard Objects .................................................................... 18
         1.3.9.1 Array ..................................................................................................... 19
         1.3.9.2 String .................................................................................................... 19
         1.3.9.3 Storage ................................................................................................. 19
         1.3.9.4 Stream ................................................................................................. 19
   1.4 Relationship to Protocols and Other Structures .................................................. 19
   1.5 Applicability Statement ....................................................................................... 19
      1.5.1 Data Spaces .................................................................................................. 19
      1.5.2 Information Rights Management Data Space ................................................. 20
      1.5.3 Encryption ................................................................................................... 20
   1.6 Versioning and Localization ................................................................................ 20
   1.7 Vendor-Extensible Fields ..................................................................................... 20

2 Structures .............................................................................................................. 21
   2.1 Data Spaces ....................................................................................................... 21
      2.1.1 File ............................................................................................................. 21
      2.1.2 Length-Prefixed Padded Unicode String (UNICODE-LP-P4) .................... 22
      2.1.3 Length-Prefixed UTF-8 String (UTF-8-LP-P4) ........................................ 23
      2.1.4 Version ....................................................................................................... 23
      2.1.5 DataSpaceVersionInfo .............................................................................. 23
      2.1.6 DataSpaceMap ............................................................................................ 24
         2.1.6.1 DataSpaceMapEntry Structure ........................................................... 25
         2.1.6.2 DataSpaceReferenceComponent Structure ......................................... 25
      2.1.7 DataSpaceDefinition ................................................................................... 26
      2.1.8 TransformInfoHeader ................................................................................ 27
      2.1.9 EncryptionTransformInfo .......................................................................... 27
   2.2 Information Rights Management Data Space ..................................................... 28
      2.2.1 \\0x06DataSpaces\DataSpaceMap Stream ....................................................... 28
      2.2.2 \\0x06DataSpaces\DataSpaceInfo Storage ....................................................... 29
      2.2.3 \\0x06DataSpaces\TransformInfo Storage for Office Binary Documents .......... 29
      2.2.4 \\0x06DataSpaces\TransformInfo Storage for ECMA-376 Documents .......... 30
      2.2.5 ExtensibilityHeader ..................................................................................... 31
      2.2.6 IRMDSTransformInfo ................................................................................ 31
      2.2.7 End-User License Stream .......................................................................... 31
      2.2.8 LicenseID ................................................................................................... 32
      2.2.9 EndUserLicenseHeader .............................................................................. 32
      2.2.10 Protected Content Stream ....................................................................... 32
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.

**Azure Active Directory (Azure AD):** The identity service in Microsoft Azure that provides identity management and access control capabilities through a REST-based API, an Azure portal, or a PowerShell command window.

**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.

**Child**: An object that is immediately below the current object in a hierarchy.

**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].

**Compound file**: (1) A structure for storing a file system, similar to a simplified FAT file system inside a single file, by dividing the single file into sectors.

(2) A file that is created as defined in [MS-CFB] and that is capable of storing data that is structured as storage and streams.

**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.

**Cyclic redundancy check (CRC)**: An algorithm used to produce a checksum (a small, fixed number of bits) against a block of data, such as a packet of network traffic or a block of a computer file. The CRC is a broad class of functions used to detect errors after transmission or storage. A CRC is designed to catch random errors, as opposed to intentional errors. If errors might be introduced by a motivated and intelligent adversary, a cryptographic hash function should be used instead.

**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.

decryption: In cryptography, the process of transforming encrypted information to its original clear text form.

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.

document: An object in a content database such as a file, folder, list, or site. Each object is identified by a URI.

document library: A type of list that is a container for documents and folders.

document property: A name/value pair that serves as metadata for a document.

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: In cryptography, the process of obscuring information to make it unreadable without special knowledge.

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.

OPC package: A .ZIP file archive [PKZIP] that follows the Open Packaging Conventions (OPC).
**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).*

**publishing license:** An XrML 1.2 license that defines the usage policy for protected content and contains the content key with which that content is encrypted. The usage policy identifies all authorized users and the actions that they are authorized to take with the content, in addition to any usage conditions. The publishing license tells a server which usage policies apply to a specific piece of content and grants a server the right to issue use licenses (ULs) based on that policy. The publishing license is created when content is protected. Also referred to as "Issuance License (IL)."

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

**root storage object:** A storage object in a compound file that must be accessed before any other storage objects and stream objects are referenced. It is the uppermost parent object in the storage object and stream object hierarchy.

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

**sensitivity label:** An identifier that correlates content with associated data classifications and their related relative sensitivity. It is defined by administrative policy as integrated via the Microsoft Information Protection SDK.

**sensitivity label metadata:** Information specific to a particular instance of one or more sensitivity labels as applied to content.

**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.

**site identifier:** A GUID that is used to identify a site in a site collection.

**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:** (1) 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.

(2) A sequence of bytes written to a file on the target file system. Every file stored on a volume that uses the file system contains at least one stream, which is normally used to store the primary contents of the file. Additional streams within the file can be used to store file attributes, application parameters, or other information specific to that file. Every file has a default data stream, which is unnamed by default. That data stream, and any other data stream associated with a file, can optionally be named.

**stream schema:** A numeric selector that specifies the format of a file's stream binary pieces.

**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.


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


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


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 (1) 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 (1) 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 (1) 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 (1), the DataSpaceInfo storage, the TransformInfo storages, and the protected content. Note that other streams (1) and storages exist in this file format; this figure describes only the relationships between these storages and streams (1).
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 placed into the streams (1) 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 (1), 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 (1) 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 (1) to contain encrypted information but can also encrypt other streams (1) 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 (1) 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 (1) 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 (1) 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 (1). 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 (1). Each child is addressable by its name.

1.3.9.4 Stream

A stream (1) is an OLE storage as described in [MS-CFB]. Streams (1) have the following properties:

- **Name**: A unique identifier for the stream (1) 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).

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 (1) 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 (1) 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 (1), 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] can 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 (1) 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 (1) 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 (1):

- \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 (1) containing the DataSpaceVersionInfo structure, as specified in section 2.1.5. This stream (1) specifies the version of the data spaces structure used in the file.
- \0x06DataSpaces\DataSpaceMap stream: A stream (1) containing a DataSpaceMap structure as specified in section 2.1.6. This stream (1) 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 (1), each of which contains a DataSpaceDefinition structure as specified in section 2.1.7. The storage MUST contain exactly one stream (1) for each DataSpaceMapEntry structure (section 2.1.6.1) in the
The name of each stream (1) MUST be equal to the `DataSpaceName` field of exactly one `DataSpaceMapEntry` structure contained in the `\0x06DataSpaces\DataSpaceMap` stream (1).

- **Transformed content streams and storages:** One or more storages or streams (1) containing protected content. The transformed content is associated with a data space definition by an entry in the `\0x06DataSpaces\DataSpaceMap` stream (1).

- **\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 (1) contains zero or more definitions for the possible transforms that can be applied to the data in content streams (1).

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 (1) named "0x06Primary". The 0x06Primary stream (1) MUST begin with a `TransformInfoHeader` structure (section 2.1.8). Transform storages can contain other streams (1) 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 | 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 (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 use a multiple of 4 bytes.

| 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 (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 | 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 |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

**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 | 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 |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
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.

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 | 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)

...
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 (1). For example, a stream (1) 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 (1) reference is one with a single reference component indicating the name of a stream (1) in the root storage of the OLE compound file.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 3 | 0 | 1 |
|   |   |   |   | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 3 |
| 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 (1) 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 (1) 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 (1) 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 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 3 | 0 | 1 |
|   |   |   |   | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 3 |
| ReferenceComponentType |

[MS-OFFCRYPTO] - v20211005
Office Document Cryptography Structure
Copyright © 2021 Microsoft Corporation
Release: October 5, 2021
ReferenceComponent (variable)

...  

ReferenceComponentType (4 bytes): An unsigned integer that specifies whether the referenced component is a stream (1) or storage. It MUST be 0x00000000 for a stream (1) or 0x00000001 for a storage.

ReferenceComponent (variable): A UNICODE-LP-P4 structure (section 2.1.2) that specifies the name of the stream (1) or storage containing the protected content to be transformed. If ReferenceComponentType is 0x00000000, then ReferenceComponent MUST be equal to the name of a stream (1) 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 (1) is both compressed and encrypted while a second stream (1) is merely encrypted.

Each DataSpaceDefinition structure MUST be stored in a stream (1) in the \0x06DataSpaces\DataSpaceInfo storage (section 2.2.2). The name of the stream (1) MUST be referenced by a DataSpaceReferenceComponent structure (section 2.1.6.1) within a DataSpaceMapEntry structure (section 2.1.6.1) stored in the \0x06DataSpaces\DataSpaceMap stream (1) (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 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 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 (1). See section 2.2.6 for an example of the usage of additional data.

| 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 |
|   |   |   |   |   |   |   |   |   |   | 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].
### 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 (1) 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 (1).

- The `\0x06DataSpaces\DataSpaceMap` stream (1) **MAY**<sup>2</sup> 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 (1).

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

- The `\0x06DataSpaces\DataSpaceMap` stream (1) **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 (1).

### 2.2.2 `\0x06DataSpaces\DataSpaceInfo` Storage

If the original document content is an Office binary document:

- The `\0x06DataSpaces\DataSpaceInfo` storage **MUST** contain a stream (1) 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**<sup>3</sup> contain a stream (1) named "0x09LZXDRMDataSpace". If this stream (1) 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 (1) 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 (1) named "0x06Primary". The "0x06Primary" stream (1) **MUST** contain an **IRMDDSTransformInfo** structure (section 2.2.6). Within the **IRMDDSTransformInfo** 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 (1) 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 (1) named "0x06Primary". The **0x06Primary** stream (1) 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 (1)** named "0x06Primary". The "0x06Primary" stream (1) 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 (1) 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.

<table>
<thead>
<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>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
</table>
| 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.

<table>
<thead>
<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>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
</table>
| 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 (1) contains cached use licenses.

The end-user license stream (1) name MUST be prefixed with "EUL-", with a base-32-encoded GUID as the remainder of the stream (1) name.

The license stream (1) 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 (1) 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 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 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 (1) MUST be contained within the root storage. If the original document content is an ECMA-376 document [ECMA-376], the stream (1) MUST be named "EncryptedPackage". For all other original document content types, it MUST be named "\0x09DRMContent".

The protected content stream (1) has the following structure.
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 (1) MAY<6> be present. The purpose of the viewer content stream (1) 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 (1) (section 2.2.10) to present a read-only representation of the document to the user. If the viewer content stream (1) is present, the stream (1) MUST be named "\0x09DRMViewerContent".

The viewer content stream (1) has the following structure.

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 be used.

```
<p>| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |</p>
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th></th>
<th></th>
</tr>
</thead>
</table>
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 (1).
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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>Determined by Flags</td>
</tr>
<tr>
<td>0x00006801</td>
<td>RC4</td>
</tr>
<tr>
<td>0x0000660E</td>
<td>128-bit AES</td>
</tr>
<tr>
<td>0x0000660F</td>
<td>192-bit AES</td>
</tr>
<tr>
<td>0x00006610</td>
<td>256-bit AES</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Flags.fCryptoAPI</th>
<th>Flags.fAES</th>
<th>Flags.fExternal</th>
<th>AlgID</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0x00000000</td>
<td>Determined by the application</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0x00000000</td>
<td>RC4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0x00006801</td>
<td>RC4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0x00000000</td>
<td>128-bit AES</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0x0000660E</td>
<td>128-bit AES</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0x0000660F</td>
<td>192-bit AES</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0x00006610</td>
<td>256-bit AES</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>AlgIDHash</th>
<th>Flags.fExternal</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>1</td>
<td>Determined by the application</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>0x00000000</td>
<td>Determined by Flags</td>
</tr>
<tr>
<td>RC4</td>
<td>0x00000028 – 0x00000080 (inclusive)</td>
<td>8-bit increments</td>
</tr>
<tr>
<td>AES</td>
<td>0x00000080, 0x000000C0, 0x00000100</td>
<td>128-bit, 192-bit, or 256-bit</td>
</tr>
</tbody>
</table>

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 be one of the following values.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>0x00000000</td>
<td>Determined by Flags</td>
</tr>
<tr>
<td>RC4</td>
<td>0x00000001</td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>0x00000018</td>
<td></td>
</tr>
</tbody>
</table>

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.

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

**SaltSize**

**Salt (16 bytes)**

...  

...  

**EncryptedVerifier (16 bytes)**

...  

...  

**VerifierHashSize**

**EncryptedVerifierHash (variable)**

...  

**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 (1) 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 (1) 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 (1).

2.3.4.2 \0x06DataSpaces\DataSpaceInfo Storage

The DataSpaceInfo storage MUST contain a stream (1) that is defined as follows:

- The \0x06DataSpaces\DataSpaceInfo storage MUST contain a stream (1) 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 (1) named "0x06Primary". The "0x06Primary" stream (1) 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 (1) (as specified in section 2.3.4.5, section 2.3.4.6, and section 2.3.4.10), the EncryptionInfo stream (1) 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 (1) of bytes containing the entire ECMA-376 source file [ECMA-376] in compressed form.

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9</th>
<th>0 1 2 3 4 5 6 7 8 9</th>
<th>0 1 2 3 4 5 6 7 8 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreamSize</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>EncryptedData (variable)</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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 (1) 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 (1) (section 2.3.4.5).

2.3.4.5 \EncryptionInfo Stream (Standard Encryption)

The \EncryptionInfo stream (1) contains detailed information that is used to initialize the cryptography used to encrypt the \EncryptedPackage stream (1), 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.
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.

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

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 (1) MUST contain the structure described in the following table.
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flags</td>
<td>A value that MUST have the fExternal bit set to 1. All other bits MUST be set to 0.</td>
</tr>
<tr>
<td>SizeExtra</td>
<td>A value that MUST be 0x00000000.</td>
</tr>
<tr>
<td>AlgID</td>
<td>A value that MUST be 0x00000000.</td>
</tr>
<tr>
<td>AlgIDHash</td>
<td>A value that MUST be 0x00000000.</td>
</tr>
<tr>
<td>KeySize</td>
<td>A value that MUST be 0x00000000.</td>
</tr>
<tr>
<td>ProviderType</td>
<td>A value that MUST be 0x00000000.</td>
</tr>
<tr>
<td>Reserved1</td>
<td>A value that is undefined and MUST be ignored.</td>
</tr>
<tr>
<td>Reserved2</td>
<td>A value that MUST be 0x00000000 and MUST be ignored.</td>
</tr>
<tr>
<td>CSPName</td>
<td>A unique identifier of an encryption module. &lt;12&gt;</td>
</tr>
</tbody>
</table>

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
<?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="EncryptionProvider">
                <xs:complexType>
```
Element | Parent | Attribute | Value
--- | --- | --- | ---
EncryptionData |  |  | 
EncryptionProvider | EncryptionData | Id | The GUID of the extensible encryption module, expressed as a string.
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, Hn be the hash data of the nth 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:

- H0 = 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 (1) as specified in section 2.3.4.5. The hash is then iterated by using the following approach:

- Hn = H(iterator + Hn-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:

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

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 \( H_{\text{final}} \) 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 \( X_1 \).

4. Form another 64-byte buffer by repeating the constant 0x5C 64 times. XOR \( H_{\text{final}} \) 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 \( X_2 \).

5. Concatenate \( X_1 \) with \( X_2 \) to form \( X_3 \), which will yield a value twice the length of `cbHash`.

6. Let `keyDerived` be equal to the first `cbRequiredKeyLength` bytes of \( X_3 \).

### 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.
### 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.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |
| EncryptionVersionInfo |
| Reserved |
| XmlEncryptionDescriptor (variable) |
| ... |

**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
<?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: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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Cipher algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>MUST conform to the <strong>AES</strong> algorithm.</td>
</tr>
<tr>
<td>RC2</td>
<td>MUST conform to the algorithm as specified in [RFC2268].&lt;13&gt;</td>
</tr>
<tr>
<td>RC4</td>
<td>MUST NOT be used.</td>
</tr>
<tr>
<td>DES</td>
<td>MUST conform to the <strong>DES</strong> algorithm.&lt;14&gt;</td>
</tr>
<tr>
<td>DESX</td>
<td>MUST conform to the algorithm as specified in [DRAFT-DESX].&lt;15&gt;</td>
</tr>
<tr>
<td>3DES</td>
<td>MUST conform to the algorithm as specified in [RFC1851].&lt;16&gt;</td>
</tr>
<tr>
<td>3DES_112</td>
<td>MUST conform to the algorithm as specified in [RFC1851].&lt;17&gt;</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Chaining mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChainingModeCBC</td>
<td><strong>Cipher block chaining (CBC)</strong></td>
</tr>
<tr>
<td>ChainingModeCFB</td>
<td>Cipher feedback chaining (CFB), with an 8-bit window</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Hash algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>MUST conform to the algorithm as specified in [RFC4634].</td>
</tr>
<tr>
<td>SHA256</td>
<td>MUST conform to the algorithm as specified in [RFC4634].</td>
</tr>
<tr>
<td>SHA384</td>
<td>MUST conform to the algorithm as specified in [RFC4634].</td>
</tr>
<tr>
<td>SHA512</td>
<td>MUST conform to the algorithm as specified in</td>
</tr>
</tbody>
</table>
### Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Hash algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>MUST conform to <strong>MD5</strong>.</td>
</tr>
<tr>
<td>MD4</td>
<td>MUST conform to the algorithm as specified in [RFC1320].</td>
</tr>
<tr>
<td>MD2</td>
<td>MUST conform to the algorithm as specified in [RFC1319].</td>
</tr>
<tr>
<td>RIPEMD-128</td>
<td>MUST conform to the hash functions specified in [ISO/IEC 10118].</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>MUST conform to the hash functions specified in [ISO/IEC 10118].</td>
</tr>
<tr>
<td>WHIRLPOOL</td>
<td>MUST conform to the hash functions specified in [ISO/IEC 10118].</td>
</tr>
</tbody>
</table>

Values that are not defined **MAY** 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 **SHOULD** 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:
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
<?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:xsi="http://www.w3.org/2001/XMLSchema"
  schemaLocation="encryptionInfo.xsd" />
<xs:simpleType name="ST_PasswordKeyEncryptorUri">
  <xs:restriction base="xs:token">
  </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 (1) 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 PasswordKeyEncryptor.spinCount iterations have been performed. The value of iterator on the last iteration MUST be one less than PasswordKeyEncryptor.spinCount.

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

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

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 \( H_{\text{final}} \) 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 \texttt{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 \texttt{blockSize} bytes, pad the array with 0x00 to an integral multiple of \texttt{blockSize} bytes.

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

\subsection*{2.3.4.14 DataIntegrity Generation (Agile Encryption)}

The \texttt{DataIntegrity} element contained within an \texttt{Encryption} element MUST be generated by using the following steps:

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

2. Generate a random array of bytes, known as \texttt{Salt}, of the same length as the value of the \texttt{KeyData.saltSize} attribute.

3. Encrypt the random array of bytes generated in step 2 by using the binary form of the \texttt{KeyData.saltValue} attribute and a \texttt{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 \texttt{blockSize} bytes, pad the array with 0x00 to the next integral multiple of \texttt{blockSize} bytes.

4. Assign the \texttt{encryptedHmacKey} attribute to the base64-encoded form of the result of step 3.

5. Generate an \texttt{HMAC}, as specified in \cite{RFC2104}, of the encrypted form of the data (message), which the \texttt{DataIntegrity} element will verify by using the \texttt{Salt} generated in step 2 as the key. Note that the entire \texttt{EncryptedPackage stream (1)}, including the \texttt{StreamSize} field, MUST be used as the message.

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

7. Assign the \texttt{encryptedHmacValue} attribute to the base64-encoded form of the result of step 6.

\subsection*{2.3.4.15 Data Encryption (Agile Encryption)}

The \texttt{EncryptedPackage stream (1)} MUST be encrypted in 4096-byte segments to facilitate nearly random access while allowing \texttt{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 \texttt{blockKey} and the binary form of the \texttt{KeyData.saltSize} 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 \texttt{encryptedKeyValue} from a \texttt{KeyEncryptor} contained within the \texttt{KeyEncryptors} sequence as specified in section 2.3.4.10. The final data block MUST be padded to the next integral multiple of the \texttt{KeyData.blockSize} value. Any padding bytes can be used. Note that the \texttt{StreamSize} field of the \texttt{EncryptedPackage} stream (1) specifies the number of bytes of unencrypted data as specified in section 2.3.4.4.

\section*{2.3.5 Office Binary Document RC4 CryptoAPI Encryption}

The storages and \texttt{streams (1)} encrypted by Office binary document RC4 CryptoAPI encryption are specified in the documentation for the relevant application; for more information see \cite{MS-DOC}, \cite{MS-
The encryption header structure used for RC4 CryptoAPI encryption is specified as shown in the following diagram.

**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<sup>21</sup> 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 (1).

**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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flags</td>
<td>The fCryptoAPI bit MUST be set. The fDocProps bit MUST be set if the document properties are not encrypted.</td>
</tr>
<tr>
<td>SizeExtra</td>
<td>MUST be 0x00000000.</td>
</tr>
<tr>
<td>AlgID</td>
<td>MUST be 0x00006801 (RC4 encryption).</td>
</tr>
<tr>
<td>AlgIDHash</td>
<td>MUST be 0x00080004 (SHA-1).</td>
</tr>
<tr>
<td>KeySize</td>
<td>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 0x000000028 bits. It MUST be compatible with the chosen cryptographic service provider (CSP).</td>
</tr>
<tr>
<td>Field</td>
<td>Value</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>ProviderType</td>
<td>MUST be 0x00000001.</td>
</tr>
<tr>
<td>Reserved1</td>
<td>Undefined and MUST be ignored.</td>
</tr>
<tr>
<td>Reserved2</td>
<td>MUST be 0x00000000 and MUST be ignored.</td>
</tr>
<tr>
<td>CSPName</td>
<td>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].</td>
</tr>
</tbody>
</table>

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:

- \( H_0 = H(\text{salt} + \text{password}) \)

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:

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

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 \( H_{\text{final}} \) 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 \( H_{\text{final}} \) 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 (1) and storages contained within an RC4 CryptoAPI Encrypted Summary stream (1) as specified in section 2.3.5.4. It is specified as shown in the following diagram.

<table>
<thead>
<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>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreamOffset</td>
<td>StreamSize</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block</td>
<td>NameSize</td>
<td>A</td>
<td>B</td>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reserved2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StreamName (variable)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

StreamOffset (4 bytes): An unsigned integer that specifies the offset, in bytes, within the summary stream (1) where the encrypted stream (1) is written.

StreamSize (4 bytes): An unsigned integer that specifies the size, in bytes, of the encrypted stream (1).

Block (2 bytes): An unsigned integer that specifies the block number used to derive the encryption key for this encrypted stream (1).

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 (1). 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 (1) (or storage) stored within the encrypted summary stream (1).

2.3.5.4 RC4 CryptoAPI Encrypted Summary Stream

When RC4 CryptoAPI encryption is used, an encrypted summary stream (1) MAY be created. The name of the stream (1) MUST be specified by the application. If the encrypted summary stream (1) is present, the \0x05DocumentSummaryInformation stream (1) 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 (1) MUST NOT be present.

For details about the contents of the \0x05SummaryInformation and \0x05DocumentSummaryInformation streams (1), see [MS-OSHARED] section 2.3.3.2.1 and [MS-OSHARED] section 2.3.3.2.
For brevity, this section refers to the RC4 CryptoAPI Encrypted Summary stream (1) as the encrypted summary stream (1).

The stream (1) MUST have the format that is shown in the following diagram.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 3 | 0 | 1 |
| StreamDescriptorArrayOffset |
| StreamDescriptorArraySize |
| EncryptedStreamData (variable) |
| ... |
| EncryptedStreamDescriptorCount |
| EncryptedStreamDescriptorArray (variable) |
| ... |

StreamDescriptorArrayOffset (4 bytes): An unsigned integer that specifies the offset within the encrypted summary stream (1) 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 (1) stored within the encrypted summary stream (1).

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 (1) and storages contained within the encrypted summary stream (1).

The encrypted summary stream (1) MUST be written as specified in the following steps:

1. Seek forward from the start of the encrypted summary stream (1) 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 (1) or storages are provided by the application, for each stream (1) or storage the following steps MUST be performed:
   1. If the data is contained within a stream (1), retrieve the contents of the stream (1). Initialize the encryption key as specified in section 2.3.5.2, using a block number of 0x00000000, and encrypt the stream (1) data. Write the encrypted bytes into the encrypted summary stream (1).
   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 (1) of bytes. Write the encrypted bytes into the encrypted summary stream (1).

3. Set the fields within the associated EncryptedStreamDescriptor for the stream (1) or storage. Do not write it to the encrypted summary stream (1) yet.

4. Increment BlockNumber.

3. Generate or retrieve the entire contents of the \0x05SummaryInformation stream (1). 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 (1). Increment BlockNumber.

4. Set the fields within the associated EncryptedStreamDescriptor for the \0x05SummaryInformation stream (1). Do not write it to the encrypted summary stream (1) yet.

5. Generate or retrieve data contained within the \0x05DocumentSummaryInformation stream (1). 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 (1) immediately following the data written in step 2.

6. Set the fields within the associated EncryptedStreamDescriptor for the \0x05DocumentSummaryInformation stream (1). Do not write it to the encrypted summary stream (1) 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 (1).

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 (1). 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 (1)** 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 (1)** used. If a stream (1) 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 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 3 | 0 | 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 $\text{TruncatedHash}$ be the first 5 bytes of $H_0$.
2. Let $\text{IntermediateBuffer}$ be a 336-byte buffer.
3. Form a 21-byte buffer by concatenating $\text{TruncatedHash}$ plus the salt. Initialize $\text{IntermediateBuffer}$ by copying the 21-byte buffer into $\text{IntermediateBuffer}$ a total of 16 times.
4. Use the following: $H_1 = H(\text{IntermediateBuffer})$.

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

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

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

2.3.6.3 Password Verifier Generation

The password verifier uses a RC4 Encryption Header 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 (1) 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 (1) 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.

```plaintext
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 X 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, 0x89, 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, 0xFD25, 0xEB6B, 0xC6F7, 0x9DCF, 0x2BBF,
0x4563, 0x8AC6, 0x06EA, 0x0DD4, 0x1BA8, 0x3750, 0x6EA0, 0xDD40,
0xDB49, 0x0A08, 0x5147, 0xA28E, 0x553D, 0xAA7A, 0x44D5,
0x6F45, 0x0DE9, 0x2DB3, 0x9C67, 0x29FF, 0x53FE, 0xA7FC, 0x5FD9,
0x47D3, 0x9FA6, 0x0FED, 0x1EDE, 0xA96A, 0x6E9C,
0xBB61, 0x60E3, 0xC1C6, 0x93AD, 0x377B, 0x6EF6, 0x6DCC,
0x45A0, 0x8B40, 0x06A1, 0x0D42, 0x1A84, 0x3508, 0x6A10,
0xAA51, 0x4483, 0x8906, 0x022D, 0x045A, 0x80B4, 0x1168,
0x76B4, 0x0EDE, 0xCAFI, 0x85C3, 0x1BA7, 0x374E, 0x6E9C,
0x3730, 0x6B60, 0xDDCC, 0xA9A1, 0x4363, 0x86C6, 0x1DAD,
0x3331, 0x6662, 0xCCC4, 0x8909, 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)
DECIMENT 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
DECIMENT Index
SET Temp TO most significant byte of XorKey
SET ObfuscationArray[Index] TO XorRor(Password[Index], Temp)

DECIMENT 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)
DECIMENT Index
DECIMENT 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** 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.

```plaintext
FUNCTION CreatePasswordVerifier
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 CreateXorKey1(Password)
SET KeyLow TO CreatePasswordVerifier1(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** 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.

```plaintext
FUNCTION CreateXorArray
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(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, 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\x20\x72\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.)

<table>
<thead>
<tr>
<th>Value</th>
<th>Hash algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>MUST conform to the details as specified in [RFC4634].</td>
</tr>
<tr>
<td>SHA-256</td>
<td>MUST conform to the details as specified in [RFC4634].</td>
</tr>
<tr>
<td>SHA-384</td>
<td>MUST conform to the details as specified in [RFC4634].</td>
</tr>
<tr>
<td>SHA-512</td>
<td>MUST conform to the details as specified in [RFC4634].</td>
</tr>
<tr>
<td>MD5</td>
<td>MUST conform to MD5.</td>
</tr>
<tr>
<td>MD4</td>
<td>MUST conform to the details as specified in [RFC1320].</td>
</tr>
<tr>
<td>MD2</td>
<td>MUST conform to the details as specified in [RFC1319].</td>
</tr>
<tr>
<td>RIPEMD-128</td>
<td>MUST conform to the details as specified in [ISO/IEC 10118].</td>
</tr>
<tr>
<td>RIPEMD-160</td>
<td>MUST conform to the details as specified in [ISO/IEC 10118].</td>
</tr>
<tr>
<td>WHIRLPOOL</td>
<td>MUST conform to the details as specified in [ISO/IEC 10118].</td>
</tr>
</tbody>
</table>
- **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 \( \text{AlgorithmName} \), \( \text{iterator} \) 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.

\[
H_0 = H(\text{salt} + \text{password})
\]

The hash is then iterated using the following approach.

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

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

### 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 xmlsig 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.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |
| HighDateTime |
| LowDateTime |

**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.

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

CertificateInfoSize

SignerLength

IssuerLength

ExpireTime

...

SignTime

...

AlgIDHash

SignatureSize

EncodedCertificateSize

Version

SerialNumberSize

IssuerBlobSize

Reserved

SignerName (variable)

...

IssuerName (variable)

...

Signature (variable)

...

EncodedCertificate (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 | 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 (1).

**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 (1) named `_signatures` in the root storage. The contents of the `_signatures` stream (1) 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 (1) and storages. Certain streams (1) and storages MUST NOT be used, as specified later in this section. A document can 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 \( H_{\text{final}} \).
Let \texttt{ClsID} be the GUID identifier for an OLE compound file storage as specified in [MS-CFB].

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

Let \texttt{ExcludedStorages} be defined as follows:

- \texttt{0x06DataSpaces}
- \texttt{0x05Bagaaqy23kudbhchAaq5u2chNd}

Let \texttt{ExcludedStreams} be defined as follows:

- \texttt{_signatures}
- \texttt{0x09DRMContent}

\begin{verbatim}
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)
    RETU RN Signature
END FUNCTION
\end{verbatim}

In the \texttt{GenerateSignatureHash} function, \texttt{IsFiltered} MUST be \texttt{true} if the document conforms to the details as specified in [MS-XLS] and the stream (1) name is "Workbook" or if the document conforms to the details as specified in [MS-PPT] and the stream (1) name is "Current User". It MUST be \texttt{false} for all other document types and streams (1).

For documents that conform to the details as specified in [MS-XLS], let \texttt{AppFilter} be defined as the process specified in [MS-XLS] section 2.1.7.15, which appends data to \texttt{HashObject}, excluding a portion of the stream (1) from being used in the hashing operation.

For documents that conform to the details as specified in [MS-PPT], let \texttt{AppFilter} be defined as a process that returns without appending data to \texttt{HashObject}. The result is that the name of the \texttt{CurrentUser} stream (1) MUST be appended to the \texttt{HashObject}, but the data contained within the \texttt{CurrentUser} stream (1) MUST NOT be appended to the \texttt{HashObject}.

When stream (1) or storage names are appended to a \texttt{HashObject}, the terminating Unicode null character MUST NOT be included.

Let \texttt{SORT} be a string sorting method that is case sensitive and ascending and that skips any nonprintable characters, such that if two streams (1) named "Data" and "0x05DocumentSummaryInformation" are input, the stream (1) named "Data" is ordered first.

\begin{verbatim}
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
            CALL HashObject.AppendData(Child.Name)
            CALL HashObject.AppendData(\texttt{SORT}(Child.Name))
        END IF
    END FOR
    CALL HashObject.AppendData(TimeStamp)
    CALL HashObject.AppendData(SignTime)
    CALL HashObject.Finalize
END FUNCTION
\end{verbatim}
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 (1), followed by the entire original encrypted file storage with the 0x09DRMContent stream (1) 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 (1)** that is signed. Except for streams (1) and storages enumerated later in this section, all streams (1) 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 (1) contained within listed storages MUST be excluded:

- 0x05Bagaaqy23kudbhchAaq5u2chNd
- 0x06DataSpaces
- Xmlsignatures
- MsoDataStore

The following streams (1) 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 (1) is Workbook, the stream (1) 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 (1) MUST be calculated when verifying the signature as if the stream (1) 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
<?xml version="1.0" encoding="utf-8"?>
  <xsd:simpleType name="ST_PositiveInteger">
    <xsd:restriction base="xsd:int">
      <xsd:minExclusive value="0" />
    </xsd:restriction>
  </xsd:simpleType>
</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. The version can be appended with a '/' followed by a signing version represented by an unsigned integer. The signing version can be used to ensure that future application versions do not attempt to verify relationships that did not exist in the signing version. Not including the '/' results in all signed relationships being verified which is consistent with previous behavior.

**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.

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

**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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The digital signature MUST NOT be printed.</td>
</tr>
<tr>
<td>2</td>
<td>The digital signature MUST be printed.</td>
</tr>
</tbody>
</table>

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 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 (1) contained in a storage named ".xmlsignatures", based on the root of the compound document. Streams (1) containing a signature MUST be named using a base-10 string representation of a random number. The name of the stream (1) MUST NOT be the same as an existing signature contained within the storage. A single signature is stored directly into each stream (1), as UTF-8 characters, with no leading header. The content of each stream (1) 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.

2.6 Sensitivity Labels

This section covers details about how and where sensitivity label metadata is stored.

2.6.1 Sensitivity Label Metadata

When a sensitivity label is applied to content, certain sensitivity label metadata is generated at the time the label is associated with the content. The sensitivity label metadata applied to a particular document shall be stored within the same document content as specified in section 2.6.2 and section 2.6.3.

Note that in all cases if sensitivity label metadata conflicts with the label in the publishing license that the label defined in the publishing license shall be construed to be the label applied to the content.
2.6.2 LabelInfo Stream Locations

The LabelInfo stream (1) comprised of XML described in section 2.6.4 shall be written in the \0x06DataSpaces\TransformInfo storage when Information Rights Management (IRM) is applied to the document and the publishing license is present in the file and there is at least one qualifying sensitivity label per section 2.6.3 to have corresponding sensitivity label metadata written to the LabelInfo stream (2).

The LabelInfo stream (2) comprised of XML described in section 2.6.4 shall be written to the OPC package in the Sensitivity Label Information part ([MS-OI29500] section 3.4.1.5) when Information Rights Management (IRM) is not applied to the document or the publishing license is not present in the file and there is at least one qualifying sensitivity label per section 2.6.3 to have corresponding sensitivity label metadata written to the LabelInfo stream (2).

For either location an implementation might write an empty labelList element per section 2.6.4.3 if there are no qualifying sensitivity labels.

Files ought not to contain sensitivity labels in both places, and implementations reading shall ignore sensitivity label metadata written to the encrypted OPC package if the \0x06DataSpaces\TransformInfo storage contains a valid LabelInfo stream (1).

2.6.3 LabelInfo versus Custom Document Properties

When reading sensitivity label metadata from a persisted document, it shall exist in custom document properties ([MS-OI29500] section 2.1.31) or a LabelInfo stream (2) location specified in section 2.6.2. Implementations shall use both sensitivity label policy and actual location of the sensitivity label metadata to determine where to read and write sensitivity label metadata as follows.

When reading sensitivity label metadata, for each sensitivity label implementations have these cases to consider:

1. If the sensitivity label policy opts in to the LabelInfo stream (2) then all applicable sensitivity label metadata shall be first read from the LabelInfo location (section 2.6.2), and subsequently metadata shall only be read for custom document properties where there is no label element (section 2.6.4.4). This preserves the sensitivity label metadata so the sensitivity label policy can change from opted out to opted in without losing applicable sensitivity label metadata for content created and persisted prior to the policy change.

2. If the sensitivity label policy is known and does not opt in to the LabelInfo stream (2) then all applicable sensitivity label metadata shall only be read from custom document properties.

3. If the sensitivity label policy is not known, then it shall be inferred to be opted in to the LabelInfo stream (2) or not by the presence or absence of sensitivity label metadata in the LabelInfo stream (2) per Azure AD tenant as given by the siteId attribute value of the corresponding label element (section 2.6.4.4).

When writing, for each sensitivity label implementations have these cases to consider:

1. If the sensitivity label policy opts in to the LabelInfo stream (2) OR is unknown but the sensitivity label metadata originally was present in the LabelInfo stream (2) then the sensitivity label metadata shall be written to the LabelInfo stream (2) and any sensitivity label metadata associated with the same Azure AD tenant that was present in custom document properties shall be preserved as-is in the custom document properties even if the sensitivity label was removed or changed.

2. If the sensitivity label policy is known and does not opt in to the LabelInfo stream (2) OR is unknown and the sensitivity label metadata originally was not present in the LabelInfo stream (2) then the sensitivity label metadata shall be written to the custom document properties.
Note that for the case where sensitivity label policy opts in to the LabelInfo stream (2) but there is sensitivity label metadata present only in custom document properties, on read this custom document property sensitivity label metadata shall be read and on write it shall be written as-is to the custom document property stream (2) AND it shall be written to the LabelInfo stream (2). Any metadata that existed in custom document properties and for which there is no LabelInfo stream (2) schema (section 2.6.4.4) for (such as parent labels, Application, Owner, Name, SetDate, and others) shall not be written to the LabelInfo stream (2) and any sensitivity label metadata formats shall be converted to be compatible. For example, Enabled value in custom document properties being True shall be written as enabled="1" or Method of Auto shall be written as method="Standard" to conform to the LabelInfo schema detailed in section 2.6.4, especially section 2.6.4.4. The same conversion shall be done regardless of the source of the sensitivity label metadata when writing to the LabelInfo stream (2).

Implementations shall not read the sensitivity label metadata from the LabelInfo stream (2) if policy is known for the sensitivity labels from that Azure AD tenant to not opt in to the LabelInfo stream (2). The presence of a label element in the LabelInfo stream (2) and absence of corresponding sensitivity label metadata for the same Azure AD tenant in the custom document properties shall not result in the transfer of sensitivity label metadata from the LabelInfo stream (2) to the custom document properties for any cases, since it would render older or unaware implementations or implementations which chose to ignore the policy unable to remove sensitivity label metadata.

All implementations which read sensitivity label metadata from the LabelInfo stream (2) and subsequently remove the sensitivity label and corresponding sensitivity label metadata shall write a label element (section 2.6.4.4) with removed="1" on write. The presence of the label element (section 2.6.4.4) with removed="1" shall indicate on subsequent read that corresponding sensitivity label metadata shall not be read from custom document properties for that Azure AD tenant given by the siteId attribute (section 2.6.4.4). In the absence of label policy the SiteId sensitivity label metadata value included in custom document property ([MS-OI29500] section 2.1.31) might be used, if present.

2.6.4 LabelInfo Stream Schema

The following describes the schema for the XML in the LabelInfo stream (2), where labelList (see section 2.6.4.3) is the root element.

```xml
<xsd:schema elementFormDefault="qualified"
  xmlns:clbl="http://schemas.microsoft.com/office/2020/mipLabelMetadata"
  xmlns:r="http://schemas.microsoft.com/office/2020/02/relationships"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="ST_ClassificationGuid">
    <xsd:restriction base="xsd:token">
      <xsd:pattern value="\{[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}\}"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="CT_ClassificationExtension">
    <xsd:sequence>
      <xsd:any/>
    </xsd:sequence>
    <xsd:attribute name="uri" type="xsd:token" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="CT_ClassificationExtensionList">
    <xsd:sequence>
      <xsd:element name="ext" type="CT_ClassificationExtension" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="CT_ClassificationLabel">
    <xsd:sequence>
      <xsd:element name="ext" type="CT_ClassificationExtension" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```
2.6.4.1 Namespaces

Content in the LabelInfo stream (2) shall be in namespace http://schemas.microsoft.com/office/2020/mipLabelMetadata unless otherwise specified.

2.6.4.2 xml Preprocessor Directive

The initial LabelInfo stream (2) bytes shall comprise of an xml preprocessor directive for example:

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

The value of the version attribute might be updated in the future to reflect files written with schema updates.

2.6.4.3 labelList Element

The labelList element shall comprise the one and only root element of the XML in the LabelInfo stream (2) (see section 2.6.4) and shall contain CT_ClassificationLabelList (see section 2.6.5.5).

```xml
<xsd:element name="labelList" type="CT_ClassificationLabelList" />
```

An empty labelList element example:

```xml
```

2.6.4.4 label Element

Each label element shall contain CT_ClassificationLabel (see section 2.6.5.4).

```xml
<xsd:element name="label" type="CT_ClassificationLabel" minOccurs="0" maxOccurs="unbounded" />
```
2.6.4.5 extLst Element

The extLst element, if present, shall contain CT_ClassificationExtensionList (see section 2.6.5.3).

The content in the extLst element shall be preserved for all XML that is not understood or currently documented in order to provide a mechanism whereby future file format changes might be introduced and existing clients, while not interpreting the content, shall preserve it.

```xml
<xsd:element name="extLst" type="CT_ClassificationExtensionList" minOccurs="0" maxOccurs="1"/>
```

2.6.4.6 ext Element

Each ext element shall contain CT_ClassificationExtension (see section 2.6.5.2).

```xml
<xsd:element name="ext" type="CT_ClassificationExtension" minOccurs="0" maxOccurs="unbounded"/>
```

2.6.5 LabelInfo Stream Structures

This section details structures which describe elements in section 2.6.4.

2.6.5.1 ST_ClassificationGuid

A 128 bit unsigned value which shall be expressed in the following format.

```xml
<xsd:restriction base="xsd:token">
    <xsd:pattern value="\{[0-9a-f]{8}\-[0-9a-f]{4}\-[0-9a-f]{4}\-[0-9a-f]{4}\-[0-9a-f]{12}\}" />
</xsd:restriction>
```

2.6.5.2 CT_ClassificationExtension

The CT_ClassificationExtension type describes contents of the ext element (see section 2.6.4.6). Additional specific contents of this element shall be defined and documented as needed in the future in section 2.6.6 and other sections as appropriate. All content present shall be preserved except to implement corresponding features as they are described in section 2.6.6. Implementations which understand some ext elements shall preserve all ext elements which are not implemented, including any not yet documented.

```xml
<xsd:complexType name="CT_ClassificationExtension">
    <xsd:sequence>
        <xsd:any/>
    </xsd:sequence>
    <xsd:attribute name="uri" type="xsd:token" use="required"/>
</xsd:complexType>
```

The uri attribute value shall be a value specified to indicate the CT_ClassificationExtension includes XML for a particular feature which was not defined when the LabelInfo stream (2) was first introduced, to allow for extensibility of the file format within the LabelInfo stream (2) on a feature by feature basis. As extensions are added to the schema, they will be listed in section 2.6.6 wherein the schema of the ext element along with its corresponding uri attribute value shall be specified. Each uri attribute value shall correspond to one of the valid extensions listed in section 2.6.6.

The uri attribute value shall not be empty.
2.6.5.3 CT_ClassificationExtensionList

The CT_ClassificationExtensionList type describes the contents of the extLst element (see section 2.6.4.5).

```xml
<xsd:complexType name="CT_ClassificationExtensionList">
    <xsd:sequence>
        <xsd:element name="ext" type="CT_ClassificationExtension" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

2.6.5.4 CT_ClassificationLabel

The CT_ClassificationLabel type describes the contents of the label element (see section 2.6.4.4).

Each CT_ClassificationLabel contains sensitivity label metadata that was added as part of classifying content using one or more sensitivity label. The attributes shall be written in the order presented here. Unless otherwise specified, these all shall be written exactly as specified when the sensitivity label was created and not updated in any way subsequent to that until the sensitivity label is removed or a different sensitivity label is associated with the corresponding content. Unless otherwise specified, the values are case sensitive. Optional values shall be omitted and shall not be present with an empty value except as noted with the descriptions for particular attributes.

```xml
<xsd:complexType name="CT_ClassificationLabel">
    <xsd:attribute name="id" type="xsd:string" use="required"/>
    <xsd:attribute name="enabled" type="xsd:boolean" use="required"/>
    <xsd:attribute name="method" type="xsd:string" use="required"/>
    <xsd:attribute name="siteId" type="ST_ClassificationGuid" use="required"/>
    <xsd:attribute name="contentBits" type="xsd:unsignedInt" use="optional"/>
    <xsd:attribute name="removed" type="xsd:boolean" use="required"/>
</xsd:complexType>
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>xsd:string</td>
<td>The id attribute value corresponds to the unique identifier of the sensitivity label as specified in policy. This value shall not be empty. This value ought to be set to the same value as siteId attribute for removed labels. This shall be written in lowercase. If a 128 bit integer, as it typically is, it ought to be written as a ST_ClassificationGuid type.</td>
</tr>
<tr>
<td>enabled</td>
<td>xsd:boolean</td>
<td>The enabled attribute value indicates whether the sensitivity label represented by this element is enabled. Removed labels ought not to be enabled.</td>
</tr>
<tr>
<td>method</td>
<td>xsd:string</td>
<td>The method attribute value represents the assignment method for the sensitivity label. This value shall be empty (method=&quot;&quot;) if the removed attribute is 1. The method attribute value ought to be &quot;Standard&quot; or &quot;Privileged&quot;. When converting sensitivity label metadata from some other source, the following guidelines shall govern which to use:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Standard: Use for any sensitivity label that was not directly applied by the user. This includes any default labels, automatically applied labels.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Privileged: Use for any sensitivity label that was directly applied by the user. This includes any manually applied sensitivity labels as well as recommended or mandatory labeling or any feature where the user decides which sensitivity label to apply.</td>
</tr>
</tbody>
</table>
| siteId      | ST_ClassificationGuid | The siteId attribute value represents the Azure Active Directory (Azure AD) site identifier corresponding to the sensitivity label policy which describes the sensitivity label. There shall only be at most
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>one label element with a given siteId. When converting sensitivity label metadata from some other source which contains both parent and child sensitivity labels, only the child sensitivity label shall be included.</td>
</tr>
</tbody>
</table>
| contentBits | xsd:unsignedInt | The contentBits attribute value represents a decimal DWORD ([MS-DTYPE]) describing the types of content marking that ought to be applied to a file. This value can be 0 when present. This value is a logical OR of the following values:  
- Header: 1  
- Footer: 2  
- Watermark: 4  
- Encryption: 8  
Other values are unused, and reserved for future use, and shall be ignored on read and shall not be set on write but it is possible to be set by future versions. Implementations ought to omit this attribute for removed labels. |
| removed   | xsd:boolean  | The removed attribute value indicates whether the sensitivity label was removed. A value of 0 (for example) means the sensitivity label is applied to the content. A value of 1 (for example) means the sensitivity label is not applied to the content. When a sensitivity label is removed, instead of removing the label element from the xml there ought to be one label element with a removed attribute to indicate for the corresponding tenant that its sensitivity labels shall not be converted from older locations (see section 2.6.3) it might be stored in the file format, such as custom document properties [MS-OI29500] section 2.1.31. |

2.6.5.5 CT_ClassificationLabelList

The CT_ClassificationLabelList type describes the contents of the labelList element (see section 2.6.4.3).

```xml
<xsd:complexType name="CT_ClassificationLabelList">
  <xsd:sequence>
    <xsd:element name="label" type="CT_ClassificationLabel" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="extLst" type="CT_ClassificationExtensionList" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

2.6.6 LabelInfo Stream Extensions

There are currently no extensions defined for the CT_ClassificationExtensionList type (see section 2.6.5.3).

2.7 MsoDataStore

Writers shall not create both an IsRedundantDataStorePromotion (section 2.7.1) storage and an IsModifiedDataStorePromotion (section 2.7.2) storage.

If a process changes any value in any MsoDataStore subordinate stream (1) without an associated change to the Custom XML Data Storage and related ([ISO/IEC29500-1:2016] sections 15.2.5 and 15.2.6) within the encrypted streams (1) of the compound file (1) referred to by the IRMDS (section 1.3.2) structure therein, writers:
- Shall remove the **IsRedundantDataStorePromotion** storage as the **MsoDataStore** subordinate streams (1) are no longer completely represented in the encrypted streams (1).

- Shall create the **IsModifiedDataStorePromotion** storage as the **MsoDataStore** subordinate streams (1) represent updated content that was originally represented in the encrypted streams (1).

**NOTE:** The presence or absence of **IsRedundantDataStorePromotion** and **IsModifiedDataStorePromotion** storages allow a process to determine if it can safely decrypt the IRMDS object and discard the IRMDS structures (if **IsRedundantDataStorePromotion** is present), if it considers the streams (1) subordinate to the **MsoDataStore** storage as superseding the content in the IRMDS structures (if **IsModifiedDataStorePromotion** is present), or if it retains the IRMDS structures for interoperability (if neither storage is present).

**Example 1:** If a document represented by an IRMDS object is uploaded to a Microsoft SharePoint Online document library protected by Information Rights Management (IRM), and the **IsRedundantDataStorePromotion** storage is not present, then SharePoint Online will not decrypt the IRMDS object and will operate solely on the IRMDS object itself.

**Example 2:** If a document represented by an IRMDS object is uploaded to a SharePoint document library protected by Information Rights Management (IRM), and the **IsRedundantDataStorePromotion** storage is present, a SharePoint Online process modifies a stream (1) subordinate to the **MsoDataStore** storage and removes the **IsRedundantDataStorePromotion** storage and creates the **IsModifiedDataStorePromotion** storage. The content for the Custom XML Data Storage ([ISO/IEC29500-1:2016] sections 15.2.5 and 15.2.6) in the encrypted streams (1) of the IRMDS object is now out-of-date, and the streams (1) subordinate to the **MsoDataStore** storage MUST now supersede the content in the encrypted streams (1).

### 2.7.1 **IsRedundantDataStorePromotion** Storage

The **IsRedundantDataStorePromotion** storage, when present as a child of the root storage object, indicates the subordinate streams (1) of the **MsoDataStore** storage represent content that is identically represented in the encrypted streams (1) of the compound file (1) referred to by the IRMDS (section 1.3.2) structure therein.

Readers can choose to preserve, edit, or discard the subordinate streams (1) when the **IsRedundantDataStorePromotion** storage is present.

Writers shall not create the **IsRedundantDataStorePromotion** storage if the subordinate **MsoDataStore** streams (1) are not completely represented in the encrypted streams (1).

### 2.7.2 **IsModifiedDataStorePromotion** Storage

The **IsModifiedDataStorePromotion** storage, when present as a child of the root storage object, indicates the subordinate streams (1) of the **MsoDataStore** storage represent subsequently updated content that was originally represented in the encrypted streams (1) of the compound file (1) referred to by the IRMDS (section 1.3.2) structure therein.

Readers shall consider the streams (1) subordinate to the **MsoDataStore** storage as superseding any Custom XML Data Storage **MsoDataStore** ([ISO29500-1:2016 §15.2.4 and §15.2.6] content represented in the encrypted streams (1).

Writers shall not create the **IsModifiedDataStorePromotion** storage if the unmodified subordinate **MsoDataStore** streams (1) are not completely represented in the encrypted streams (1).
2.8 EncryptedSIHash Stream

The EncryptedSIHash stream (1), when present as a child of the root storage object, indicates information about the \x05SummaryInformation stream (1) ([MS-OSHARED] section 2.3.3.2.1).

The EncryptedSIHash stream (1) shall only be present on encrypted files.

The EncryptedSIHash stream (1), when created or updated, shall be composed entirely of one EncryptedPropertyStreamInfo structure (section 2.10), with a Checksum (section 2.10) computed using the entire \x05SummaryInformation stream (1) contents, exclusive of stream (1) properties.

Writers that do not write the \x05SummaryInformation stream (1) shall not write the EncryptedSIHash stream (1).

Writers that update the \x05SummaryInformation stream (1) without an associated change to any applicable related document properties within the encrypted streams (1) of the compound file (2) referred to by the IRMDS structure therein shall not update the EncryptedSIHash stream (1).

NOTE: The presence and contents of the EncryptedSIHash stream (1) allows a process to determine if it can safely decrypt the IRMDS and discard the \x05SummaryInformation stream (1) (EncryptedSIHash stream (1) is present and the CRC32 matches the computed CRC32 of the \x05SummaryInformation stream (1)).

2.9 EncryptedDSIHash Stream

The EncryptedDSIHash stream (1), when present as a child of the root storage object, indicates information about the \x05DocumentSummaryInformation stream (1) ([MS-OSHARED] section 2.3.3.2.2 and [MS-OSHARED] section 2.3.3.2.3).

The EncryptedDSIHash stream (1) shall only be present on encrypted files.

The EncryptedDSIHash stream (1), when created or updated, shall be composed entirely of one EncryptedPropertyStreamInfo structure (section 2.10), with a Checksum (section 2.10) computed using the entire \x05DocumentSummaryInformation stream (1) contents, exclusive of stream (1) properties.

Writers that do not write the \x05DocumentSummaryInformation stream (1) shall not write the EncryptedDSIHash stream (1).

Writers that update the \x05DocumentSummaryInformation stream (1) without an associated change to any applicable related document properties within the encrypted streams (1) of the compound file (2) referred to by the IRMDS structure therein shall not update the EncryptedDSIHash stream (1).

NOTE: The presence and contents of the EncryptedDSIHash stream (1) allows a process to determine if it can safely decrypt the IRMDS and discard the \x05DocumentSummaryInformation stream (1) (EncryptedDSIHash stream (1) is present and the CRC32 matches the computed CRC32 of the \x05DocumentSummaryInformation stream (1)).

2.10 EncryptedPropertyStreamInfo Structure

<table>
<thead>
<tr>
<th>StreamId</th>
<th>Version</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
StreamId (1 byte): A number which identifies this structure as content in either the 
EncryptedSIHash (section 2.8) or EncryptedDSIHash (section 2.9) stream (1). The value shall be 0xAB. The EncryptedPropertyStreamInfo shall be considered invalid for all other values.

Version (1 byte): A number which identifies the stream schema. A value of 0x00 indicates the format given here. Readers shall ignore the stream (1) if some other value is present.

Checksum (4 bytes): A value that shall be a CRC32 computed using the MsoCrc32Compute ([MS-OSHARED] section 2.4.3) algorithm of the entire contents of the corresponding \x05SummaryInformation or \x05DocumentSummaryInformation stream (1), stored little-endian.

Reserved (variable): Undefined and shall be ignored if Version is 0x00; reserved for future use.
3 Structure Examples

This section provides examples of the following structures:

- Office binary data file structures with corresponding hexadecimal and graphical representation.

The example for the ECMA-376 document [ECMA-376] contains the following streams (1) and storages:

- **0x06DataSpaces** storage:
  - **Version** stream (1) containing a DataSpaceVersionInfo structure as specified in section 3.1.
  - **DataSpaceMap** stream (1) containing a DataSpaceMap structure as specified in section 3.2.
  - **DataSpaceInfo** storage:
    - **DRMEncryptedDataSpace** stream (1) containing a DataSpaceDefinition structure as described in section 3.3.
  - **TransformInfo** storage:
    - **0x06Primary** stream (1) containing an IRMDSTransformInfo structure as described in section 3.4.
    - **EUL-ETRHA1143ZLUDD412YTI3M5CTZ** stream (1) containing an EndUserLicenseHeader structure and a certificate chain as described in section 3.5.
  - **EncryptedPackage** stream (1).
  - **0x05SummaryInformation** stream (1).
  - **0x05DocumentSummaryInformation** stream (1).

Note that not all of the streams (1) and storages in the file, including the 0x05SummaryInformation stream (1) and 0x05DocumentSummaryInformation stream (1), are specified in the IRMDS structure, and examples are not provided for those streams (1) in this section. OLE compound files conforming to this structure frequently contain other storages and streams (1).

3.1 Version Stream

This section provides an example of a Version stream (1) that contains a DataSpaceVersionInfo structure (section 2.1.5).

```
00000000: 3C 00 00 00 4D 00 69 00 6F 00 66 00 2E 00 43 00 6F 00 74 00 61 00 72 00 73 00
0000010: 01 00 00 00 4D 00 69 00 6F 00 66 00 2E 00 43 00 6F 00 74 00 61 00 72 00 73 00
0000020: 01 00 00 00 4D 00 69 00 6F 00 66 00 2E 00 43 00 6F 00 74 00 61 00 72 00 73 00
0000030: 01 00 00 00 4D 00 69 00 6F 00 66 00 2E 00 43 00 6F 00 74 00 61 00 72 00 73 00
0000040: 01 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00
```
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 (1) 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 6E 00 64 00  6D 00 65 00 74 00 65 00
00000030:  64 00 44 00 61 00 74 00  13 00 00 00 00 00 00 00
00000040:  00 00 00 06 00 00 00 00  00 00 00 00 00 00 00 00
```

HeaderLength
EntryCount

MapEntries (variable)

... 

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

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

**MapEntries (variable):** The contents of the `DataSpaceMapEntry` 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 64 00 50 00 61 00 6C 00 61 00 2A 00
00000030: 00 00 00 44 00 52 00 4D 00 6E 00 63 00 72 00 74
00000040: 44 00 74 00 61 00 53 00 70 00 61 00 63 00 00 00
```

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

**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 (1).
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 (1) 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 (1) in the \0x06DataSpaces\DataSpaceInfo storage (section 2.2.2) that contains a DataSpaceDefinition structure (section 2.1.7).

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 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 (1) 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).
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): "\{C7DFACD-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 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 (1) (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 | 00000010: | 5A 41 42 76 41 48 63 41 |
| 00000020: | 63 77 42 6C 41 48 41 41 | 00000030: | 62 67 42 30 41 47 38 41 |
| 00000040: | 5A 41 42 76 41 48 63 41 | 00000050: | 6C 20 76 65 72 73 69 6F |
| 00000060: | 5A 41 42 76 41 48 63 41 | 00000070: | 49 4E 3D 22 31 2E 30 22 3F |
| 00000080: | 3E 3C 3F 78 6D 6C 20 76 65 72 73 69 6F 3D 22 31 2E 30 22 3C 43 45 52 54 49 46 49 43 41 54 45 43 48 41 00000070: | 50 41 42 59 41 48 41 54 51 41 75 41 44 49 41 00000080: | 49 4E 3D 22 31 2E 30 22 3F |
| 00000090: | 50 41 42 59 41 48 41 54 51 41 75 41 44 49 41 000000A0: | 49 4E 3D 22 31 2E 30 22 3C 43 45 52 54 49 46 49 43 41 54 45 43 48 41 000000B0: | 50 41 42 59 41 48 41 54 51 41 75 41 44 49 41 000000C0: | 49 4E 3D 22 31 2E 30 22 3C 43 45 52 54 49 46 49 43 41 54 45 43 48 41 000000D0: | 50 41 42 59 41 48 41 54 51 41 75 41 44 49 41 000000E0: | 49 4E 3D 22 31 2E 30 22 3C 43 45 52 54 49 46 49 43 41 54 45 43 48 41 000000F0: | 50 41 42 59 41 48 41 54 51 41 75 41 44 49 41 00000100: | 49 4E 3D 22 31 2E 30 22 3C 43 45 52 54 49 46 49 43 41 54 45 43 48 41 00000110: | 50 41 42 59 41 48 41 54 51 41 75 41 44 49 41 00000120: | 49 4E 3D 22 31 2E 30 22 3C 43 45 52 54 49 46 49 43 41 54 45 43 48 41 00000130: | 50 41 42 59 41 48 41 54 51 41 75 41 44 49 41 00000140: | 49 4E 3D 22 31 2E 30 22 3C 43 45 52 54 49 46 49 43 41 54 45 43 48 41 00000150: | 50 41 42 59 41 48 41 54 51 41 75 41 44 49 41 00000160: | 49 4E 3D 22 31 2E 30 22 3C 43 45 52 54 49 46 49 43 41 54 45 43 48 41 00000170: | 50 41 42 59 41 48 41 54 51 41 75 41 44 49 41 00000180: | 49 4E 3D 22 31 2E 30 22 3C 43 45 52 54 49 46 49 43 41 54 45 43 48 41 |
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 (1) 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 47 38 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 47 30 41
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 3 | 0 | 1 |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **Length** |
| **ID_String.Length (variable)** |
|   ...   |
| **ID_String.Data (variable)** |
|   ...   |

**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.Length and ID_String.Data together form a UTF-8-LP-P4 structure (section 2.1.3).

**ID_String.Data (variable):** "VwBpAG4AZABvAhAcwA6AHUAcwBlAHcAcwA6AHUAQABjAG8AbgB0AG8AcwBVAC4AYwBvAG0A" 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 (1) (section 2.2.7).

```
00000040:  94 BE 00 00 3C 3F 78 6D
00000050:  6C 20 76 65 72 73 69 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 41 41 64 67 42 6L 41 47 55 41
00000090:  50 51 41 69 41 45 4D 41 62 77 42 75 41 48 55 41
000000a0:  64 51 42 79 41 48 41 41 62 77 42 7A 41 47 55 41
000000b0:  64 51 42 79 41 48 41 41 62 77 42 7A 41 47 55 41
000000c0:  64 51 42 79 41 48 41 41 62 77 42 7A 41 47 55 41
000000d0:  64 51 42 79 41 48 41 41 62 77 42 7A 41 47 55 41
```

93 / 119
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:

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

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

   ```
   PABYAHIATQBMACAAdgBlAHIAcwBpAG8AbgA9ACIAIgAgAHgAbQ8sAG4Acw9ACIAIgAgAHAdQBYAHAAwBzAGUAQPAIEMAwbBuAHQAZQBhuAHQAZQBMgkAywb1AG4AcwBlACIAFgASAEITwBEAfKAKAIAB0AHkACAltADAhAIgBMAEAQwBFAE4AwBFACIAIA2AGUAcBzAGkAbwBuAD0IAgAzAC4AMAIAAD4APABJFAJMAwVBAAARABBAAEKAATQB...
   ```

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 IACwBpAG8AbgA9ACIAIgAgAHgAbQ8sAG4Acw9ACIAIgAgAHAdQBYAHAAwBzAGUAQPAIEMAwbBuAHQAZQB uAHQAZQBMgkAywb1AG4AcwBlACIAFgASAEITwBEAfKAKAIAB0AHkACAltADAhAIgBMAEAQwBFAE4AwBFACIAIA2AGUA cBzAGkAbwBuAD0IAgAzAC4AMAIAAD4APABJFAJMAwVBAAARABBAAEKAATQB...
   ```
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 (1).

```
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 00 00 00 4D 00 69 00
00001440:  63 00 72 00 6F 00 73 00 20 00 42 00 73 00 79 00
00001450:  70 00 6F 00 67 00 72 00 61 00 70 00 68 00 12 00
00001460:  69 00 63 00 20 00 50 00 72 00 6F 00 76 00 64 00
00001470:  72 00 20 00 76 00 31 00 30 00 3E 00 00 00 00 00
```

| 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>
<thead>
<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>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaltSize</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Salt (variable)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EncryptedVerifier (16 bytes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VerifierHashSize</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EncryptedVerifierHash (variable)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</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 (1) containing detailed information used to initialize the cryptography that is used to encrypt the EncryptedPackage stream (1).

```
00001800: 02 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 00 4D 00 69 00 63 00 72 00 6F 00 66 00
00001830: 74 00 6F 00 70 00 67 00  72 00 61 70 00 69 00 45
00001840: 6E 00 68 00 61 00 6E 00  20 00 41 00 45 00 53 00
00001850: 20 00 43 00 72 00 79 00  70 00 74 00 6F 00 67 00
00001860: 20 00 50 00 72 00 6F 00  76 00 69 00 64 00 20 20
00001870: 61 00 6E 00 64 00 20 00  41 00 45 00 53 00 20 00
00001880: 43 00 72 00 79 00 70 00  6F 00 67 00 72 00 61 00
00001890: 70 00 69 00 20 00 28 00  50 00 72 00 6F 00 74 00
000018A0: 70 00 74 00 79 00 70 00  65 00 29 00 00 00 10 00
000018B0: 92 25 50 F6 B6 4F FE 5B D3 96 DF 5E E9 17 DA 3A
000018C0: BF 86 E1 8F 64 9D 17 D0 A5 41 D9 45 00018D0: E9 17 DA 3A BF 86 E1 8F 64 9D 17 D0 A5 41 D9 45
000018E0: 0B B9 50 46 D3 91 41 84
```

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 (0x00000004) 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 (0x00000004) 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 3F 07 00 00 00 00
4D 00 69 00 63 00 72 00 6F 00 73 00 00 6F 00 66 00
74 00 20 00 45 00 6E 00 66 00 63 00 61 00 6E 00 63 00
65 00 64 00 20 00 53 00 41 00 45 00 30 00 20 00 61 00
6E 00 64 00 20 00 41 00 4D 00 53 00 20 00 20 00 43 00
72 00 79 00 70 00 74 00 6F 00 66 00 69 00 6D 00 70 00 61 00
70 00 68 00 69 00 6D 00 20 00 50 00 20 00 20 00 61 00
76 00 69 00 64 00 65 00 72 00 20 00 28 00 50 00 46 00
72 00 6F 00 74 00 6F 00 73 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">
```

---

98 / 119

[MS-OFFCRYPTO] - v20211005
Office Document Cryptography Structure
Copyright © 2021 Microsoft Corporation
Release: October 5, 2021
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 (1).

```
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
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23| 24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39| 40|
| EncryptionVersionInfo
| Salt (16 bytes)
| ... |
| ... |
| EncryptedVerifier (16 bytes)
| ... |
| ... |
| EncryptedVerifierHash (16 bytes)
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)

PasswordKeyEncryptor (Agile Encryption)
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
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<encryption xmlns="http://schemas.microsoft.com/office/2006/encryption"
  <keyData
    saltSize="16"
    blockSize="16"
    keyBits="128"
    hashSize="20"
    cipherAlgorithm="AES"
    cipherChaining="ChainingModeCBC"
    hashAlgorithm="SHA-1"
    saltValue="/a4iWqPyIvE2cUolJMKrIw=="/>

  <dataIntegrity
    encryptedHmacKey="uwpAEFPX1hQyD2O01k22lhjvNW0ECyAAPuOwDygsfY="
    encryptedHmacValue="uf66bJjtrYjOjSFqRkqkNQY9NjNQqUI+xcxk8Q8y4mko="/>
</encryption>
```
<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="pps6BlbmqCFgopsmlrWnQ=="
            encryptedVerifierHashInput="JYU4Q0u2BhqzQA5D4J/voA=="
            encryptedVerifierHashValue="eB2jX5mvहbJ+9Q7ffEc+6X2Mdyz2gлHOXXvT9Pn6nK+w="
            encryptedKeyValue="2F86HG+xV5nGa27DE1gqg=="/>
    </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:** ":uwpAEFW1hQyD2001kz1lhhjevNw0ECyAA0u2OxDYgsfY=" specifies the encrypted copy of the randomly generated value used when generating the encryption key.

**encryptedHmacValue:** ":uf6HbJjtryJOjSFqrkqkNQy9NjNQUP1+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 encryptedVerifierHashInput attribute encoded as specified in section 2.3.4.13.

encryptedVerifierHashValue: "eB2jX5mvhBJ+9Q7fFC+6X2Mydz2gIHOXx0T9Pn6nK+w=" specifies the encryptedVerifierHashValue encoded as specified in section 2.3.4.13.

encryptedKeyValue: "2F86HG+xV3nGa27DElgqgw==" specifies the encryptedKeyValue encoded as specified in section 2.3.4.13.

3.12 LabelInfo Stream

This section provides example LabelInfo streams (2) (section 2.6.4).

The example below has both applied sensitivity labels (id="{1e5447a7-0548-4856-b1af-4c253c3ad016}", id="{989510b7-9e58-40d7-98bf-60b911cb0ea9}", and id="{f5ad6165-baa2-492b-b3cc-e7925d2faba7}") and removed sensitivity labels (id="{30863317-b7ea-4af9-ba77-23ca2107d146}" and id="{ec916be2-2331-4c51-b61a-32f19fd0a1e3}").

<?xml version="1.0" encoding="utf-8" standalone="yes"?><clbl:labelList xmlns:clbl="http://schemas.microsoft.com/office/2020/mipLabelMetadata"><clbl:label id="{1e5447a7-0548-4856-b1af-4c253c3ad016}" enabled="1" method="Privileged" siteId="{f44a5b26-9898-45b5-9c1e-62a1a758d3c}" contentBits="15" removed="0" />
<clbl:label id="{30863317-b7ea-4af9-ba77-23ca2107d146}" enabled="0" method="" siteId="{30863317-b7ea-4af9-ba77-23ca2107d146}" removed="1" />
<clbl:label id="{989510b7-9e58-40d7-98bf-60b911cb0ea9}" enabled="1" method="Standard" siteId="{d1b22d3d-8585-53a6-abc3-e803c7e8d2a}" contentBits="0" removed="0" />
<clbl:label id="{ec916be2-2331-4c51-b61a-32f19fd0a1e3}" enabled="0" method="" siteId="{ec916be2-2331-4c51-b61a-32f19fd0a1e3}" removed="1" />
<clbl:label id="{f5ad6165-baa2-492b-b3cc-e7925d2faba7}" enabled="1" method="Privileged" siteId="{1dc653f2-77f1-4cac-9644-656982d12f12}" contentBits="0" removed="0" />
</clbl:labelList>

This example provides a extLst element (section 2.6.4.5) showing two separate hypothetical features.

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 (1), 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 (1) are not encrypted.
Key stream (1) reuse can occur in document data streams (1), potentially with known plaintext, implying that certain portions of encrypted data can be either directly extracted or trivially retrieved.

Key stream (1) reuse occurs multiple times within the RC4 CryptoAPI Encrypted Summary stream (1).

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 (1), 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 (1) might not be encrypted.

- Depending on the application, key stream (1) 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 (1) and storages are not subject to signing. Tampering with these streams (1) 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 updates to those products.

- 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 2016
- Microsoft Office 2019
- Microsoft Office SharePoint Server 2007
- Microsoft SharePoint Server 2010
- Microsoft SharePoint Server 2013
- Microsoft SharePoint Server 2016
- Microsoft SharePoint Server 2019
- Microsoft Office 2021
- Microsoft SharePoint Server Subscription Edition

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates 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
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).
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.

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.
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.

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].

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 2013 set a Version.vMajor value of 0x004.

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>57</td>
<td>57</td>
</tr>
<tr>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>59</td>
<td>59</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>62</td>
<td>62</td>
</tr>
<tr>
<td>63</td>
<td>63</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>66</td>
<td>66</td>
</tr>
<tr>
<td>67</td>
<td>67</td>
</tr>
<tr>
<td>68</td>
<td>68</td>
</tr>
<tr>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td>73</td>
<td>73</td>
</tr>
<tr>
<td>74</td>
<td>74</td>
</tr>
<tr>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>76</td>
<td>76</td>
</tr>
<tr>
<td>77</td>
<td>77</td>
</tr>
<tr>
<td>78</td>
<td>78</td>
</tr>
<tr>
<td>79</td>
<td>79</td>
</tr>
<tr>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>81</td>
<td>81</td>
</tr>
<tr>
<td>82</td>
<td>82</td>
</tr>
<tr>
<td>83</td>
<td>83</td>
</tr>
<tr>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td>87</td>
<td>87</td>
</tr>
<tr>
<td>88</td>
<td>88</td>
</tr>
<tr>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>90</td>
<td>90</td>
</tr>
<tr>
<td>91</td>
<td>91</td>
</tr>
<tr>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>93</td>
<td>93</td>
</tr>
<tr>
<td>94</td>
<td>94</td>
</tr>
<tr>
<td>95</td>
<td>95</td>
</tr>
<tr>
<td>96</td>
<td>96</td>
</tr>
<tr>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>98</td>
<td>98</td>
</tr>
<tr>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td>103</td>
<td>103</td>
</tr>
<tr>
<td>104</td>
<td>104</td>
</tr>
<tr>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>106</td>
<td>106</td>
</tr>
<tr>
<td>107</td>
<td>107</td>
</tr>
<tr>
<td>108</td>
<td>108</td>
</tr>
<tr>
<td>109</td>
<td>109</td>
</tr>
<tr>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>112</td>
<td>112</td>
</tr>
<tr>
<td>113</td>
<td>113</td>
</tr>
<tr>
<td>114</td>
<td>114</td>
</tr>
<tr>
<td>115</td>
<td>115</td>
</tr>
</tbody>
</table>

--

[MS-OFFCRYPTO] - v20211005
Office Document Cryptography Structure
Copyright © 2021 Microsoft Corporation
Release: October 5, 2021
DataSpaceReferenceComponent structure – data spaces 25
DataSpaceVersionInfo structure – data spaces 23
Details
\ signature stream 70
\(0x06DataSpaces\)DataSpaceInfo storage (section 2.2.2.29, section 2.3.4.3 38)
\(0x06DataSpaces\)DataSpaceMap stream (section 2.2.1, section 2.3.4.1 38)
\(0x06DataSpaces\)TransformInfo storage 38
\(0x06DataSpaces\)TransformInfo storage for ECMA-376 documents 30
\(0x06DataSpaces\)TransformInfo storage for Office binary documents 29
EncryptedPackage stream 39
EncryptionInfo stream (agile encryption) 44
EncryptionInfo stream (extensible encryption) 40
EncryptionInfo stream (standard encryption) 39
xmlsignatures storage 77
binary document digital signatures structure 67
binary document password verifier derivation Method 1 60
binary document password verifier derivation Method 2 63
binary document write protection Method 1 65
binary document write protection Method 2 65
binary document write protection Method 3 66
binary document XOR array initialization Method 1 60
binary document XOR array initialization Method 2 64
binary document XOR data transformation Method 1 62
binary document XOR data transformation Method 2 65
CertificateInfo structure - CryptoAPI digital signature 68
CryptoAPI digital signature CertificateInfo structure 68
CryptoAPI digital signature generation 70
CryptoAPI digital signature structure 70
data encryption (agile encryption) 52
data spaces structure 21
DataIntegrity generation (agile encryption) 52
DataSpaceDefinition structure 26
DataSpaceMap structure 24
DataSpaceMapEntry structure 25
DataSpaceReferenceComponent structure 25
data encryption (standard encryption) 42
DataSpaceVersionInfo structure 23
document write protection structure 65
ECMA-376 document encryption 38
ECMA-376 document encryption key generation (standard encryption) 42
ECMA-376 document write protection 65
encryption key derivation 59
encryption key generation (agile encryption) 49
encryption structure 33
EncryptionHeader structure 34
EncryptionHeaderFlags structure 34
EncryptionTransformInfo structure 27
EncryptionVerifier structure 36
end-user license stream 31
EndUserLicenseHeader structure 32
ExtensibilityHeader structure 31
File structure 21
idOfficeObject object element 73
idPackageObject object element 73
Information Rights Management Data Space structure 28
initialization vector generation (agile encryption) 50
IRMSTransformInfo structure 31
ISO write protection method 66
KeyInfo element 73
Length-Prefixed Padded Unicode String (UNICODE-LP-P4) structure 22
Length-Prefixed UTF-8 String (UTF-8-LP-P4) structure 23
LicenseID 32
Office binary document RC4 CryptoAPI encryption 53
Office binary document RC4 encryption 58
password verification - Office binary document RC4 CryptoAPI encryption 57
password verification - Office binary document RC4 encryption 59
password verification - XOR obfuscation 65
password verification - Office binary document RC4 encryption 43
password verification generation - Office binary document RC4 CryptoAPI encryption 57
password verification generation - Office binary document RC4 encryption 59
password verification generation (standard encryption) 43
PasswordKeyEncryptor generation (agile encryption) 50
protected content stream 32
RC4 CryptoAPI encrypted summary stream 55
RC4 CryptoAPI EncryptedStreamDescriptor structure 55
RC4 CryptoAPI encryption header 53
RC4 CryptoAPI encryption key generation 54
RC4 encryption header 58
SignatureValue element 72
SignedInfo element 72
TimeEncoding structure 67
TransformInfoHeader structure 27
version structure 23
viewer content stream 33
XAdES elements 76
Xmldsig digital signature elements 72
XOR obfuscation 60
Digital signature elements - Xmldsig 72
Digital signatures overview 18
Document write protection
binary document write protection Method 1 65
binary document write protection Method 2 65
binary document write protection Method 3 66
ECMA-376 65
ISO write protection method 66
security implementer considerations 105
Document write protection structure 65
DRMEncryptedDataSpaceStream example 90

E

ECMA-376 document encryption security
Elements

idOfficeObject object 73
idPackageObject object 73
KeyInfo 73
SignatureValue 72
SignedInfo 72
XAdES 76
Xmldsig digital signature 72

Encryption

\x06DataSpaces\DataSpaceInfo storage 38
\x06DataSpaces\DataSpaceMap stream 38
\x06DataSpaces\TransformInfo storage 38
\x06EncryptedPackage stream 39
\EncryptionInfo stream (agile encryption) 44
\EncryptionInfo stream (extensible encryption) 40
\EncryptionInfo stream (standard encryption) 39

40-bit RC4 encryption overview 17

applicability 20

binary document password verifier derivation
Method 1 60

binary document password verifier derivation
Method 2 63

binary document XOR array initialization Method 1 60

binary document XOR array initialization Method 2 64

binary document XOR data transformation Method 1 62

binary document XOR data transformation Method 2 65

CryptoAPI RC4 encryption overview 17
data encryption (agile encryption) 52
DataIntegrity generation (agile encryption) 52

ECMA-376 document encryption key generation (standard encryption) 42

ECMA-376 document encryption key generation (agile encryption) 49

encryption key derivation 59

encryption key generation (agile encryption) 49

EncryptionHeader structure 34

EncryptionHeaderFlags structure 34

EncryptionVerifier structure 36

initialization vector generation (agile encryption) 50

Office binary document RC4 58

Office binary document RC4 CryptoAPI 53

overview 17

password verification - Office binary document RC4

password verification - Office binary document RC4 CryptoAPI 57

password verification – XOR obfuscation 65

password verification (standard encryption) 43

password verifier generation - Office binary document RC4 59

password verifier generation - Office binary document RC4 CryptoAPI 57

password verifier generation (standard encryption) 43

PasswordKeyEncryptor generation (agile encryption) 50

RC4 CryptoAPI encrypted summary stream 55

RC4 CryptoAPI EncryptedStreamDescriptor structure 55

RC4 CryptoAPI encryption header 53

RC4 CryptoAPI encryption key generation 54

RC4 encryption header 58

XOR obfuscation 60

XOR obfuscation overview 17

Encryption header example 99

Encryption key derivation – encryption 59

Encryption key generation (agile encryption) – encryption 49

Encryption structure 33

EncryptionHeader structure – encryption 34

EncryptionHeader Structure example 94

EncryptionHeaderFlags structure – encryption 34

EncryptionTransformInfo structure – data spaces 27

EncryptionVerifier structure – encryption 36

EncryptionVerifier Structure example 96

End-user license stream – IRMDS 31

EndUserLicenseHeader structure – IRMDS 32

EndUserLicenseHeader structure example 93

EUL-ETRHA1143ZLUDD412YT13MSCTZ Stream example 92

Examples 87

\EncryptionInfo Stream 96
\EncryptionInfo Stream (Third-Party Extensible Encryption) 98
\x06Primary Stream 90
certificate chain 93
DataSpaceMap Stream 88
DataSpaceMapEntry structure 89

DRMEncryptedDataSpace Stream 90

encryption header 99

EncryptionHeader Structure 94

EncryptionVerifier Structure 96

EndUserLicenseHeader structure 93

EUL-ETRHA1143ZLUDD412YT13MSCTZ Stream 92

LabelInfo Stream 103

PasswordKeyEncryptor (Agile Encryption) 100

Version Stream 87

Examples overview 87

ExtensibilityHeader structure – IRMDS 31

F

Fields - security index 106

Fields - vendor-extensible 20

File – data spaces 21

G

Glossary 8

I

idOfficeObject object element 73
idPackageObject object element 73

Implementer – security considerations

binary document digital signatures 106
data spaces 104
document write protection 105

ECMA-376 document encryption 104