Encrypted Object Extension

Version 1.1.1i

ABSTRACT:

"Publication of this Working Draft for review and comment has been approved by the Cloud Storage Technical Working Group. This draft represents a "best effort" attempt by the Cloud Storage Technical Working Group to reach preliminary consensus, and it may be updated, replaced, or made obsolete at any time. This document should not be used as reference material or cited as other than a 'work in progress.' Suggestion for revision should be directed to http:/snia.org/feedback."

Working Draft

November 16, 2016

USAGE

DISCLAIMER

Revision History

Date	Version	By	Comments
2015-06-22	1.1a	CDMI TWG	Initial draft for TWG review
2015-06-23	1.1b	CDMI TWG	Updates from TWG review
2015-11-19	1.1c	CDMI TWG	Updates to support JOSE representations
2015-12-16	1.1d	CDMI TWG	Updates to address review comments
2016.01.19	1.1e	CDMI TWG	Updates to address review comments
2016-03-14	1.1f	CDMI TWG	Updates to incorporate review comments
2016-08-25	1.1g	CDMI TWG	Updates to incorporate implementation feedback
2016-10-03	1.1.1h	CDMI TWG	Updates to incorporate SDC Plugfest findings and to update for the CDMI 1.1.1 ISO version in preparation for merging.
2016-11-16	1.1.1i	CDMI TWG	Updates to separate out value signing KMS IDs from object signing KMS IDs, and added process for signing and verifying whole-object signatures.

Encrypted Object Extension

Overview

Cloud storage systems are often used to store encrypted objects. While no additional support is required for a CDMI system to handle encrypted objects where all encryption and decryption functionality is performed by the clients, there is significant value in being able to permit the cloud to encrypt and decrypt objects. Encryption and decryption enables cloud-based processing of object plaintext, plaintext delivery to clients, and the ability to encrypt cloud-side.

This extension defines a series of operations that can be performed against CDMI objects, where the value of the object is a valid Cryptographic Message Syntax (CMS) data structure, or where the value of the object is a valid JSON Web Encryption (JWE) data structure.

These operations include:

- Create a new encrypted object
- Delete an existing encrypted object
- Encrypt an existing non-encrypted object
- Decrypt an existing encrypted object
- Re-encrypt an existing encrypted object with a different key/algorithm
- Access the ciphertext of an encrypted object
- Access the plaintext of an encrypted object
- Update the plaintext of an existing encrypted object

This extension also defines metadata specifying how an encrypted object is encrypted and signed and how keys are obtained from a remote key management system using a standard such as KMIP.

It is important to note that this extension only protects the ‘value’ of CDMI object; it does not hide the existence of an object, nor does it protect the contents or presence of the object name, ID, or any other object fields and metadata.

Encrypted objects can be created client-side and server-side, and are stored in CMS or JWE format. The subset of the CMS and JWE standard used is specified later in this extension.

Modifications to the CDMI 1.1.1 spec:

1) In Clause 2, add reference to CMS and JWE:

RFC 5652, Cryptographic Message Syntax (CMS) - https://www.ietf.org/rfc/rfc5652.txt

RFC 7515, JSON Web Signatures (JWS) - https://www.ietf.org/rfc/rfc7515.txt

RFC 7516, JSON Web Encryption (JWE) - https://www.ietf.org/rfc/rfc7516.txt

RFC 7518, JSON Web Algorithms (JWA) - https://www.ietf.org/rfc/rfc7518.txt

2) In Clause 3, add the following terms:

JOSE
JavaScript Object Signing and Encryption

JWA
JSON Web Algorithm

JWE
JSON Web Encryption

JWS
JSON Web Signing

3) In Clause 6.1, add to end:

Ciphertext representation of encrypted objects are created, accessed, and updated by explicitly specifying a MIME type "application/cms" or "application/jose+json". Otherwise, a plaintext representation is created, accessed, and updated. For more details on encrypted objects, see clause 23.

4) In Clause 6.2.8, add examples 2 and 3:

EXAMPLE 2 PUT to the container URI to create an encrypted object:

```
PUT /MyContainer/MyEncryptedObject.txt HTTP/1.1
Host: cloud.example.com
Content-Type: application/cms
Content-Length: 1425

<CMS Encrypted Object>
```

The following shows the response:

```
HTTP/1.1 201 Created
```

EXAMPLE 3 PUT to the container URI to create an encrypted object:

```
PUT /MyContainer/MyEncryptedObject2.txt HTTP/1.1
Host: cloud.example.com
Content-Type: application/jose+json
Content-Length: 1425

<JWE Encrypted Object JSON>
```

The following shows the response:

```
HTTP/1.1 201 Created
```

5) In Clause 6.3.3, add to end of table 8:

Header	Type	Description	Requirement
Accept	Header String	"*/*" or a value as per clause 5.13.2 "Content-type negotiation". If the object has a mimetype of "application/cms" or "application/jose+json", and the mimetype "application/cms" or "application/jose+json" is included in the Accept mimetype, the CDMI server shall return the CMS or JOSE value in the response message body. Otherwise, the decrypted plaintext shall be returned in the response message body, along with the encapsulated mimetype in the Content-Type response header. If decryption is not possible, an error result code shall be returned. (See Clause 23 – Encrypted Objects) If the Accept mimetype list includes "*/*" before "application/cms" and/or "application/jose+json", the server will first try to return the decrypted plaintext, and shall return the CMS or JOSE value when decryption fails. If the Accept mimetype list excludes "*/*", decrypted plaintext shall only be returned if the encapsulated mimetype is included in the Accept mimetype list.	Optional

6) In Clause 6.3.8, add examples 3 through 5:

EXAMPLE 3 GET to the data object URI to always return the ciphertext of an encrypted object:

```
GET /MyContainer/MyEncryptedObject.txt HTTP/1.1
Host: cloud.example.com
Accept: application/cms, application/jose+json
```

The following shows the response:

```
HTTP/1.1 200 OK
Content-Type: application/cms
Content-Length: 1425

<CMS Encrypted Object>
```

EXAMPLE 4 GET to the data object URI to read the plaintext of an encrypted object, if possible; otherwise, get the ciphertext:

```
GET /MyContainer/MyEncryptedObject.txt HTTP/1.1
Host: cloud.example.com
Accept: */*, application/cms, application/jose+json
<Header credentials used to authenticate and access the decryption
key>
```

The following shows the response:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 252

<Decrypted contents of Encrypted Value>
```

EXAMPLE 5 GET to the data object URI to read the plaintext of an encrypted object:

```
GET /MyContainer/MyEncryptedObject.txt HTTP/1.1
Host: cloud.example.com
<Header credentials used to authenticate and access the decryption
key>
```

The following shows the response:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 252

<Decrypted contents of Encrypted Value>
```

7) In Clause 8.1, add to end:

CDMI data object operations only permit management operations and access to the ciphertext of encrypted objects. For more details on encrypted objects, see clause 23.

8) In Clause 8.2.9, add examples 5 and 6:

EXAMPLE 5 PUT to the container URI to create an encrypted object:

```
PUT /MyContainer/MyEncryptedObject.txt HTTP/1.1
Host: cloud.example.com
Content-Type: application/cdmi-object
X-CDMI-Specification-Version: 1.1

{
   "mimetype" : "application/cms",
   "metadata" : {
      "cdmi_enc_key_id" : "testkey"
   },
   "valuetransferencoding" : "base64"
   "value" : "<CMS Encrypted Object in Base64>"
}
```

The following shows the response:

```
HTTP/1.1 201 Created
```

EXAMPLE 6 PUT to the container URI to create an encrypted object:

```
PUT /MyContainer/MyEncryptedObject2.txt HTTP/1.1
Host: cloud.example.com
Content-Type: application/cdmi-object
```

```
X-CDMI-Specification-Version: 1.1

{
   "mimetype" : "application/jose+json",
   "metadata" : {
      "cdmi_enc_key_id" : "77c7e2b8-6e13-45cf-8672-617b5b45243a"
   },
   "valuetransferencoding" : "json",
   "value" : {
       "protected": "eyJhbGciOiJkaXIiLCJraWQiOiI3N2M3ZTJi
                     OC02ZTEzLTQ1Y2YtODY3Mi02MTdiNWI0NTI0
                     M2EiLCJlbmMiOiJBMTI4R0NNIn0",
       "iv": "refa467QzzKx6QAB",
       "ciphertext": "JW_i_f52hww_ELQPGaYyeAB6HYGcR559l9T
                      YnSovc23XJoBcW29rHP8yZOZG7YhLpT1bjF
                      uvZPjQS-m0IFtVcXkZXdH_lr_FrdYt9HRUY
                      kshtrMmIUAyGmUnd9zMDB2n0cRDIHAzFVeJ
                      UDxkUwVAE7_YGRPdcqMyiBoCO-FBdE-Nceb
                      4h3-FtBP-c_BIwCPTjb9o0SbdcdREEMJMyZ
                      BH8ySWMVi1gPD9yxi-aQpGbSv_F9N4IZAxs
                      cj5g-NJsUPbjk29-s7LJAGb15wEBtXphVCg
                      yy53CoIKLHHeJHXex45Uz9aKZSRSInZI-wj
                      sY0yu3cT4_aQ3i1o-tiE-F8Ios61EKgyIQ4
                      CWao8PFMj8TTnp",
       "tag": "vbb32Xvllea2OtmHAdccRQ",
       "cty": "text/plain"
    }
}
```

The following shows the response:

```
HTTP/1.1 201 Created
```

9) In Clause 8.4.4, add to the end of the "mimetype" row in table 30:

Header	Type	Description	Requirement
Mimetype	JSON String	<existing content> If this field is set to "application/cms" or "application/jose+json", the CDMI server shall encrypt or re-encrypt the value of the object in place, using the key specified by the "cdmi_enc_key_id" metadata item. If the "cdmi_enc_key_id" metadata item is not present, the object ID shall be used as the key identifier. The mimetype of the plaintext shall be stored in the CMS or JWE JSON representation. If a "cdmi_enc_value_sign_id" metadata item is present, the encrypted object shall also be signed. If this field is changed from "application/cms" or "application/jose+json" to any other mimetype, the CDMI server shall decrypt the value of the object in place, replacing the specified mimetype with the mimetype of the	Optional

Header	Type	Description	Requirement
		encrypted object, if stored as part of the encrypted object. For more details on encrypted objects, see clause 23.	

10) In Clause 8.4.8, add examples 11 and 12:

EXAMPLE 11   PUT to the data object URI to encrypt an existing object:

```
PUT /MyContainer/MyDataObject.txt HTTP/1.1
Host: cloud.example.com
Content-Type: application/cdmi-object
X-CDMI-Specification-Version: 1.1

{
   "mimetype" : "application/cms",
   "metadata" : {
      "cdmi_enc_key_id" : "testkey"
   }
}
```

The following shows the response:

```
HTTP/1.1 204 No Content
```

EXAMPLE 12   PUT to the data object URI to decrypt an existing encrypted object:

```
PUT /MyContainer/MyEncryptedObject.txt HTTP/1.1
Host: cloud.example.com
Content-Type: application/cdmi-object
X-CDMI-Specification-Version: 1.1

{
   "mimetype" : "text/plain"
}
```

The following shows the response:

```
HTTP/1.1 204 No Content
```

11) In Clause 12.1.1, Add new rows at end of table "Table 100 - System-Wide Capabilities"

Capability Name	Type	Description
cdmi_enc_cms	JSON String	If present and "true", this capability indicates that the cloud storage system supports operations against the contents of CMS encrypted objects.
cdmi_enc_jwe	JSON String	If present and "true", this capability indicates that the cloud storage system supports operations against the contents of JWE encrypted objects.
cdmi_enc_inplace	JSON	If present and "true", this capability indicates that the cloud

Capability Name	Type	Description
	String	storage system supports operations to encrypt and decrypt objects in place, including updates.
cdmi_enc_access	JSON String	If present and "true", this capability indicates that the cloud storage system supports operations to decrypt objects on access.
cdmi_cms_encryption	JSON Array of JSON Strings	If present, this capability lists which CMS ContentEncryptionAlgorithmIdentifier encryption algorithms are supported for operations against the contents of CMS encrypted objects.
cdmi_cms_digest	JSON Array of JSON Strings	If present, this capability lists which CMS MessageAuthenticationCodeAlgorithm digest algorithms are supported for operations against the contents of CMS encrypted objects.
cdmi_cms_signature	JSON Array of JSON Strings	If present, this capability lists which CMS SignatureAlgorithmIdentifier signature algorithms are supported for operations against the contents of CMS encrypted objects.
cdmi_jwe_enc	JSON Array of JSON Strings	If present, this capability lists which JOSE "enc" encryption algorithms are supported for operations against the contents of JWE encrypted objects, as defined in RFC 7518.
cdmi_jwe_alg	JSON Array of JSON Strings	If present, this capability lists which JOSE "alg" encryption algorithms are supported for operations against the contents of JWE encrypted objects, as defined in RFC 7518.
cdmi_jws_alg	JSON Array of JSON Strings	If present, this capability lists which JOSE "alg" encryption algorithms are supported for operations against the contents of JWS signatures, as defined in RFC 7518.

12) In Clause 16.3, add new row at end of table "Table 118 – Storage System Metadata":

Metadata Name	Type	Description	Requirement
cdmi_enc_signature	JSON Object	Contains a JWS compact serialization of a signature for the entire object (value and metadata). See section 23.7 for more details.	Optional

13) In Clause 16.4, add new row at end of table "Table 119 – Data System Metadata":

Metadata Name	Type	Description	Requirement
cdmi_enc_key_id	JSON String	Contains a unique key identifier (e.g., KMIP Identifier) for the symmetric key used to encrypt and decrypt the object.	Optional

Metadata Name	Type	Description	Requirement
cdmi_enc_value_sign_id	JSON String	Contains a unique key identifier (e.g., KMIP Identifier) for the private key used for signing the value of the object.	Optional
cdmi_enc_value_verify_id	JSON String	Contains a unique key identifier (e.g. KMIP Identifier) for the public key or certificate chain used for verifying the signature of the value of the object.	Optional
cdmi_enc_object_sign_id	JSON String	Contains a unique key identifier (e.g., KMIP Identifier) for the private key used for signing the entire object.	Optional
cdmi_enc_object_verify_id	JSON String	Contains a unique key identifier (e.g. KMIP Identifier) for the public key or certificate chain used for verifying the signature of the entire object.	Optional

14) Add new clause 23, "Encrypted Objects"

23.1 Overview

A cloud storage system may optionally implement additional operations against encrypted objects. Support for these operations are indicated by the presence of the cloud storage system-wide capabilities for encrypted objects.

Encrypted object operations include the ability to encrypt, re-encrypt, and decrypt objects that are already stored in the cloud (in-place), to sign and verify the signature of encrypted objects, and to access and update the plaintext associated with encrypted objects.

The CDMI International Standard does not specify the method by which keys are managed. Key management services are provided by an external key management system (KMS), and the use of the KMIP standard is given as an example of how a CDMI server interacts with an external KMS.

CDMI objects can contain values that are encrypted. Operations against an encrypted CDMI object are only supported if the encrypted object value is a valid CMS or JWE JSON format. The CMS or JWE JSON object shall include an embedded mimetype of the encrypted object. For JWE, the "cty" header shall be used for this purpose.

23.2 Encryption Operations

The state transition diagram for encrypted objects is shown in Figure 14:

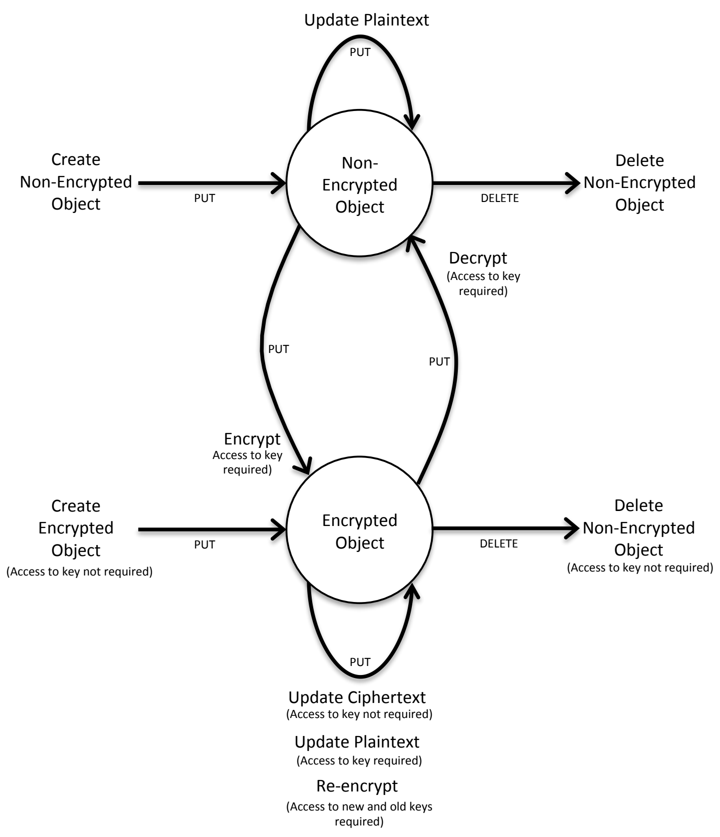
Update Plaintext
PUT
Create Non-Encrypted Object
PUT
Non-Encrypted Object
DELETE
Delete Non-Encrypted Object
Decrypt (Access to key required)
PUT
PUT
Encrypt Access to key required)
Create Encrypted Object (Access to key not required)
PUT
Encrypted Object
DELETE
Delete Non-Encrypted Object (Access to key not required)
PUT
Update Ciphertext (Access to key not required)
Update Plaintext (Access to key required)
Re-encrypt (Access to new and old keys required)

Figure 14 – Encrypted Object State Transitions

The following eight encryption operations are defined:

23.2.1 Create a new encrypted object

Client-encrypted objects shall be stored to a CDMI server using a standard HTTP or CDMI PUT operation, as described in clauses 6.2 and 8.2. The client shall indicate that an object is encrypted by specifying a mimetype of "application/cms" or "application/jose+json".

A client may register an encryption key, signing keys and/or verification keys with a Key Management System (KMS), and may indicate the Key IDs in cdmi_enc_key_id, cdmi_enc_value_sign_id, cdmi_enc_object_sign_id, cdmi_enc_value_verify_id, and/or cdmi_enc_object_verify_id metadata items. This allows the CDMI server to access the keys from the KMS on behalf of a client, when needed.

Creating an encrypted objects on a CDMI server does not require any encryption-specific capabilities to be supported, and is backwards compatible with earlier versions of the CDMI standard. This permits encrypted objects to be stored and transferred by CDMI servers that do not support encryption-specific functionality.

23.2.2 Delete an encrypted object

Encrypted objects shall be deleted using a standard HTTP or CDMI DELETE operation, as described in clauses 6.5 and 8.5. Any client with sufficient permissions shall be permitted to delete an encrypted object, regardless of if they can access the decryption keys.

23.2.3 Encrypt an unencrypted object

Existing unencrypted objects shall be encrypted in-place by performing a CDMI PUT operation, as described in clause 8.4, that changes the object mimetype to "application/cms" or "application/jose+json" and specifies a cdmi_enc_key_id metadata item. The client may also specify a cdmi_enc_value_sign_id and/or cdmi_enc_value_verify_id metadata item to indicate that the object is to be signed, and to provide signature verification information.

The CDMI Server shall use the client's credentials (which are included in HTTP headers, and are out of scope of this International Standard) to retreive the encryption and signing keys, and encryption and signing algorithm information from the KMS, and shall use the keys to encrypt and sign the value of the object. The mimetype of the encrypted value is stored in the CMS wrapper, or in a "cty" field of the JWE JSON.

23.2.4 Decrypt an encrypted object

Existing encrypted objects shall be decrypted in-place by performing a CDMI PUT operation, as described in clause 8.4, that changes the object mimetype from "application/cms" or "application/jose+json" to the original mimetype as specified in the CMS wrapper, or in the "cty" field of the JWE JSON. Specifying any other fields or metadata shall return a 400 Bad Request result code.

The CDMI Server shall use the client's credentials (which are included in HTTP headers, and are out of scope of this International Standard) to retreive the encryption, signing and verification keys, and encryption, signing and verification algorithm information from the KMS, and shall use the keys to decrypt and verify the encrypted value and user metadata included in the object.

23.2.5 Re-encrypt an encrypted object

Existing encrypted objects shall be re-encrypted in-place by performing a CDMI PUT operation, as described in clause 8.4, that retains the object mimetype of "application/cms" or "application/jose+json", or changes the object mimetype from "application/cms" to "application/jose+json", or vice-versa. The client shall also specify a new cdmi_enc_key_id, cdmi_enc_value_sign_id and/or cdmi_enc_value_verify_id metadata item to indicate the new key(s) to be used. Specifying any other fields or metadata shall return a 400 Bad Request result code.

The CDMI Server shall use the client's credentials (which are included in HTTP headers, and are out of scope of this International Standard) to retreive both the original encryption and signing keys using the original metadata values, and the new encryption and signing keys using the new metadata values from the KMS, and shall use these keys to decrypt, verify, encrypt and sign the value of the object, as needed.

If an encrypted object does not have an existing cdmi_enc_key_id metadata item, does not have a "kid" header, and no keys are associated with the Object ID, the specified metadata shall be added to the object, and no re-encryption operation shall be performed.

23.2.6 Access ciphertext of an encrypted object

The ciphertext content of an encrypted object shall be read by performing an HTTP GET operation, as described in clause 6.3, with an Accept header value of "application/cms" or "application/jose+json", depending on the mimetype of the encrypted object.

The ciphertext content of an encrypted object shall also be read by performing a CDMI GET operation, as described in clause 8.3.

23.2.7 Access plaintext of an encrypted object

The plaintext value of an encrypted object shall be read by performing an HTTP GET operation, as described in clause 6.3, with an Accept header value other than "application/cms" or "application/jose+json", typically "*/*. Object plaintext cannot be transparently accessed using a CDMI GET.

The CDMI Server shall use the client's credentials (which are included in HTTP headers, and are out of scope of this International Standard) to retreive the encryption, signing and verification keys, and encryption, signing and verification algorithm information from the KMS, and shall use the keys to decrypt and verify the encrypted value included in the object.

When an encrypted object is decrypted for access, the plaintext shall not be retained or cached by the CDMI server.

23.2.8 Update plaintext of an encrypted object

The plaintext value of an encrypted object shall be modified by performing an HTTP PUT operation, as described in clause 6.4, with an Content-Type header value other than "application/cms" or "application/jose+json", typically "*/*"., depending on the mimetype of the encrypted object. Object plaintext cannot be transparently modified using a CDMI GET.

The CDMI Server shall use the client's credentials (which are included in HTTP headers, and are out of scope of this International Standard) to retreive the encryption, signing and verification keys, and encryption, signing and verification algorithm information from the KMS, and shall use the keys to decrypt and verify the encrypted value, update the value, and re-encrypt/re-sign the updated value.

When an encrypted object is decrypted for update, the plaintext shall not be retained or cached by the CDMI server.

23.2.9 Other CDMI Operations

Other operations specifed by this International Standard (such as copying, serializing, querying, etc) treat an encrypted value the same way as a non-encrypted value.

23.3 Example Uses of Encrypted Objects

Encrypted objects can be used with CDMI systems in the following ways:

- **Passthrough** – A client may store an encrypted object in any format in a CDMI server, with the ciphertext being accessible to the server and to other authorized clients. No access to the plaintext is provided. Passthrough use is compatible with all CDMI systems and is useful when the clients manage all security-related operations and want to protect against potentially untrustworthy clouds.
- **Server-side encryption and signing** – A client may instruct a CDMI server that supports encrypted object operations to take an existing CDMI object and encrypt or encrypt and sign it in place into CMS or JWE JSON representation, where the value of the object is persistently stored from that point on in an encrypted format. Server-side

encryption and signing is useful when clients trust the CDMI server and want to increase object security without having to re-upload the data.

- **Server-side decryption** – A client may instruct a CDMI server that supports encrypted object operations to take an existing CDMI object and decrypt it in place from a CMS or JWE JSON representation, where the value of the object is persistently stored from that point on in a decrypted format. Server-side decryption is useful when a client trusts the CDMI server and wants to decrease object security without having to re-upload the data.
- **Client access decryption** – A CDMI server may automatically attempt to decrypt an encrypted object when accessed via HTTP. Client access decryption is useful to provide transparent access to authorized HTTP clients without requiring modifications to the HTTP clients.
- **Cloud access decryption** – A CDMI server may automatically decrypt encrypted objects when it has access to the decryption keys. Cloud access decryption is useful for cloud-resident data processing performed by the CDMI server, such as virus scanning, query, and analytics.
- **Signature verification** – A CDMI server can automatically verify signatures that are attached to encrypted objects that include a signature. Signature verification is useful for detecting corruption or alteration before delivering data to a client.

23.4 KMS Integration

The encryption key is obtained from the KMS using a unique identifier that is stored in the cdmi_enc_key_id metadata item associated with the encrypted object. If this metadata item is not present, the CDMI object ID shall be used to locate the key.

When a client requests that an operation be performed that requires accessing the key for the object, the CDMI server evaluates the credentials provided by the client to determine if the client is authorized to perform the requested operation. If the operation is permitted, the CDMI server retrieves the key from the KMS to complete the requested operation. To retrieve the key, the client may be required to provide additional information in the HTTP request that the CDMI server can then use to authenticate to the KMS.

The CDMI International Standard does not specify the mechanism by which the CDMI server communicates with the KMS. In this International Standard, the KMIP protocol is used as an example. CMS and JWE strings for algorithms, key lengths, etc., need to be mapped to the strings used by the KMS (see KMIP clause 9.1.3.2.7).

All keys are created and managed externally to the CDMI server, typically by the client or a system operating on behalf of the client. As a consequence, the CDMI server requires read-only access to the KMS. The CDMI server shall not cache keys.

23.5 CMS Format

Any valid CMS-formatted data can be stored to a CDMI server. However, encrypted object operations are only defined for the following subset of valid CMS-formatted data.

For encryption operations, the CDMI server shall support the following:

- EnvelopedData
- EncryptedContentInfo

- contentEncryptionAlgorithm value listed in the cdmi_cms_encryption capability of that CDMI server

For signature operations, the CDMI server shall support the following:

- AuthenticatedData
- SignedData
- digestAlgorithms value listed in the cdmi_cms_digest capability of that CDMI server
- SignerInfo
- signatureAlgorithm value listed in the cdmi_cms_signature capability of that CDMI server

The following CMS-formatted data may be ignored: recipientInfos

23.6 JOSE Format

Any valid JWE-formatted data can be stored to a CDMI server. However, encrypted object operations are only defined for the following subset of valid JWE-formatted data.

For encryption operations, the CDMI server shall support the following:

- JWE with Direct Encryption (Symmetric Key from KMS)
- JWE with Key Encryption (Public Key from KMS)

For signature operations, the CDMI server shall support the following:

- JWS RSA (Private Key from KMS)
- JWS ECDSA (Private Key from KMS)
- JWS HMAC-SHA2 (Symmetric Key from KMS)

The following CMS-formatted data may be ignored: Multiple recipients and multiple signatures

23.7 Signature/Digest Verification

If a signature is present as part of the CMS or JWE JSON value, the CDMI server shall verify that the signature of the value is valid before allowing plaintext access or modification.

If a whole-object signature is present, the CDMI server shall verify that the signature contained in the cdmi_enc_signature metadata item is valid before allowing any read operations for the object. Write operations are permitted for an object with an invalid or unverifiable whole-object signature.

When present, a whole-object signature shall be attached as a "cdmi_enc_signature" metadata item in JWS compact format, with the second field (the JWS payload field) replaced with an empty string as described in Appendix F of RFC 7515.

For signature creation and verification, payload field shall be computed using the following process:

1. Create a serialized representation of the CDMI object, as described in clause 15

2. Remove the following metadata items, if present:

- cdmi_atime
- cdmi_acount
- cdmi_enc_signature
- Any *_provided metadata items

3. Sort all JSON objects in the serialized CDMI object according to the following rules:
   - Within each JSON object, name/value pair entries shall be sorted lexicographically by name
   - Within each JSON array, the initial order shall be preserved
4. Remove all JSON whitespace
5. Base64 URL encode, according to the JWS RFC 7515

23.8 Error Handling

If a decryption or signature validation operation is requested against a CDMI object containing an invalid CMS or JWE JSON representation, an HTTP status code of 500 Internal Error shall be returned to the client.

If a decryption or signature validation operation is requested against a CDMI object containing a valid CMS or JWE JSON representation that uses an unsupported algorithm or feature, an HTTP status code of 501 Not Implemented shall be returned to the client.

If a decryption or signature validation operation is requested against a CDMI object containing a valid CMS or JWE JSON representation, but the required keys are temporarily unavailable given the credentials presented, an HTTP status code of 408 Request Timeout shall be returned to the client.

If a decryption or signature validation operation is requested against a CDMI object containing a valid CMS or JWE JSON representation, but the required keys are unavailable given the credentials presented, an HTTP status code of 401 Unauthorized shall be returned to the client.

If a decryption or signature validation operation is requested against a CDMI object containing a valid CMS or JWE JSON representation, valid keys are available, and signature verification fails, an HTTP status code of 403 Forbidden shall be returned to the client.