[MS-DRMCD]:

Windows Media Digital Rights Management (WMDRM): MTP Command Extension

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.

Revision Summary

Date	Revision History	Revision Class	Comments
7/16/2010	0.1	New	First Release.
8/27/2010	0.1	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2010	0.1	None	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	0.1	None	No changes to the meaning, language, or formatting of the technical content.
1/7/2011	0.1	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2011	0.1	None	No changes to the meaning, language, or formatting of the technical content.
3/25/2011	0.1	None	No changes to the meaning, language, or formatting of the technical content.
5/6/2011	0.1	None	No changes to the meaning, language, or formatting of the technical content.
6/17/2011	0.2	Minor	Clarified the meaning of the technical content.
9/23/2011	0.2	None	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	1.0	Major	Updated and revised the technical content.
3/30/2012	1.0	None	No changes to the meaning, language, or formatting of the technical content.
7/12/2012	1.0	None	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	1.0	None	No changes to the meaning, language, or formatting of the technical content.
1/31/2013	2.0	Major	Updated and revised the technical content.
8/8/2013	3.0	Major	Updated and revised the technical content.
11/14/2013	3.0	None	No changes to the meaning, language, or formatting of the technical content.
2/13/2014	4.0	Major	Updated and revised the technical content.
5/15/2014	4.0	None	No changes to the meaning, language, or formatting of the technical content.
6/30/2015	5.0	Major	Significantly changed the technical content.
10/16/2015	5.0	None	No changes to the meaning, language, or formatting of the technical content.
7/14/2016	6.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction

This protocol is designed to support digital rights management for portable consumer electronic devices. This protocol can be used to enable consumers to experience audio and/or video on portable devices, while still protecting the rights of the content owner.

This protocol enables client devices to locally receive and store DRM licenses and enforce DRM rights and policies. For example, devices leveraging this protocol can maintain play counts, expiration, and metering information. Licenses can be acquired both through a PC with indirect license acquisition, or directly from a content service using direct license acquisition. Examples of client devices where this protocol can be used include:

- Portable media players
- Mobile phones
- PDAs
- Set top boxes for Internet based video on-demand (VOD) and IPTV services

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

attribute: A characteristic of some object or entity, typically encoded as a name/value pair.

authentication: The ability of one entity to determine the identity of another entity.

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.

certificate template: A list of attributes that define a blueprint for creating an X.509 **certificate**. It is often referred to in non-Microsoft documentation as a "certificate profile". A **certificate template** is used to define the content and purpose of a digital certificate, including issuance requirements (certificate policies), implemented X.509 extensions such as application policies, key usage, or extended key usage as specified in [X509], and enrollment permissions. Enrollment permissions define the rules by which a **certification authority (CA)** will issue or deny certificate requests. In Windows environments, **certificate templates** are stored as objects in the Active Directory and used by Microsoft enterprise **CAs**.

certification authority (CA): A third party that issues **public key certificates**. Certificates serve to bind public keys to a user identity. Each user and certification authority (CA) can decide whether to trust another user or CA for a specific purpose, and whether this trust should be transitive. For more information, see [RFC3280].

challenge: A piece of data used to authenticate a user. Typically a challenge takes the form of a nonce.

cipher: A cryptographic algorithm used to encrypt and decrypt files and messages.

content: Multimedia data. **content** is always in ASF, for example, a single ASF music file or a single ASF video file. Data in general. A file that an application accesses. Examples of content include web pages and documents stored on either web servers or SMB file servers.

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].

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

Digital Rights Management (DRM): A set of technologies that provides control over how a given piece of protected content can be used.

digital signature: A message authenticator that is typically derived from a cryptographic operation by using an asymmetric algorithm and private key. When a symmetric algorithm is used for this purpose, the authenticator is typically referred to as a Message Authentication Code (MAC).

direct license acquisition: The process by which a web-enabled device requests a license directly from the license server over a network.

domain: A set of users and computers sharing a common namespace and management infrastructure. At least one computer member of the set must act as a domain controller (DC) and host a member list that identifies all members of the domain, as well as optionally hosting the Active Directory service. The domain controller provides **authentication** of members, creating a unit of trust for its members. Each domain has an identifier that is shared among its members. For more information, see [MS-AUTHSOD] section 1.1.1.5 and [MS-ADTS].

elliptic curve cryptography (ECC): A **public-key** cryptosystem that is based on high-order elliptic curves over finite fields. For more information, see [IEEE1363].

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

error code: An integer that indicates success or failure. In Microsoft implementations, this is defined as a Windows error code. A zero value indicates success; a nonzero value indicates failure.

exchange: A pair of messages, consisting of a request and a response.

flags: A set of values used to configure or report options or settings.

Greenwich mean time (GMT): Time measured at the Greenwich Meridian Line at the Royal Observatory in Greenwich.

group: A named collection of users who share similar access permissions or roles.

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.

HRESULT: An integer value that indicates the result or status of an operation. A particular HRESULT can have different meanings depending on the protocol using it. See [MS-ERREF] section 2.1 and specific protocol documents for further details.

indirect license acquisition: The process of transferring a license between two local devices. For example, licenses are indirectly acquired from a computer to a mobile device such as a cell phone, Smartphone, PDA, or portable media player.

key: In cryptography, a generic term used to refer to cryptographic data that is used to initialize a cryptographic algorithm. **Keys** are also sometimes referred to as keying material.

key exchange: A synonym for key establishment. The procedure that results in shared secret keying material among different parties. Key agreement and key transport are two forms of **key exchange**. For more information, see [CRYPTO] section 1.11, [SP800-56A] section 3.1, and [IEEE1363] section 3.

license synchronization: The process of requesting **updates** for licenses that have expired or become invalid. Once the device is connected to the indirect license acquisition host, the MTP protocol [MTP] is used to request a **license synchronization challenge** from the device.

Media Transfer Protocol (MTP): **MTP** is used to manage **content** on any portable device with storage. The primary purpose of **MTP** is to facilitate communication between devices that connect to a computer or other host, **exchange** data, and then disconnect for standalone use. A secondary purpose of **MTP** is to enable command and control of a connected device. This includes remote control of device functionality, monitoring of device-initiated events, and reading and setting of device properties. For more information, see [MTP].

message: A data structure representing a unit of data transfer between distributed applications. A message has message properties, which may include message header properties, a message body property, and message trailer properties.

negotiation: A series of exchanges. The successful outcome of a **negotiation** is the establishment of one or more security associations (SAs). For more information, see [RFC2408] section 2.

policy: The description of actions permitted for a specified set of **content**, and restrictions placed on those actions. Restrictions are described in the license associated with the **content**.

private key: One of a pair of keys used in public-key cryptography. The private key is kept secret and is used to decrypt data that has been encrypted with the corresponding public key. For an introduction to this concept, see [CRYPTO] section 1.8 and [IEEE1363] section 3.1.

protected content: (1) 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).

(2) **Content** for which usage is governed by policies specified in a license.

proxy: A computer, or the software that runs on it, that acts as a barrier between a network and the Internet by presenting only a single network address to external sites. By acting as a go-between that represents all internal computers, the proxy helps protects network identities while also providing access to the Internet.

public key: One of a pair of keys used in public-key cryptography. The public key is distributed freely and published as part of a digital certificate. For an introduction to this concept, see [CRYPTO] section 1.8 and [IEEE1363] section 3.1.

public key infrastructure (PKI): The laws, policies, standards, and software that regulate or manipulate certificates and public and private keys. In practice, it is a system of digital certificates, **certificate authorities (CAs)**, and other registration authorities that verify and authenticate the validity of each party involved in an electronic transaction (3). For more information, see [X509] section 6.

public-private key pair: The association of a public key and its corresponding private key when used in cryptography. Also referred to simply as a "key pair". For an introduction to public-private key pairs, see [IEEE1363] section 3.

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

removable media: Any type of storage that is not permanently attached to the computer. A persistent storage device stores its data on media. If the media can be removed from the device, the media is considered removable. For example, a floppy disk drive uses removable media.

revocation list: The list of identifiers of software or hardware components to which protected content cannot flow. Different content protection systems typically have different formats for representing **revocation lists**.

root certificate: A self-signed **certificate** that identifies the **public key** of a root **certification authority (CA)** and has been trusted to terminate a **certificate chain**.

secure clock server (SCS): Used to synchronize device times with the current global time using the secure clock protocol **messages** as specified in section 2.2.2.

Secure Sockets Layer (SSL): A security protocol that supports confidentiality and integrity of messages in client and server applications that communicate over open networks. SSL uses two keys to encrypt data-a **public key** known to everyone and a private or secret key known only to the recipient of the message. SSL supports server and, optionally, client **authentication** using X.509 **certificates**. For more information, see [X509]. The SSL protocol is precursor to Transport Layer Security (TLS). The TLS version 1.0 specification is based on SSL version 3.0 [SSL3].

service: A process or agent that is available on the network, offering resources or services for clients. Examples of services include file servers, web servers, and so on.

session key: A symmetric key that is derived from a master key and is used to encrypt or authenticate a specific media stream by using the Secure Real-Time Transport Protocol (SRTP) and Scale Secure Real-Time Transport Protocol (SSRTP).

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.

SHA-1 hash: A hashing algorithm as specified in [FIPS180-2] that was developed by the National Institute of Standards and Technology (NIST) and the National Security Agency (NSA).

standard license: A stand-alone, self-contained license. A **standard license** is not **updated** by a leaf license.

symmetric key: A secret key used with a cryptographic symmetric algorithm. The key needs to be known to all communicating parties. For an introduction to this concept, see [CRYPTO] section 1.5.

time service: A system service that implements support for synchronizing a computer's local time with a **time source**.

time source: A component that possesses a clock and that makes the clock's time available to other components for synchronization. For more information, see "reference source" in [RFC1305].

transaction identifier (TID): A GUID that uniquely identifies a transaction between a portable device and a WMDRM **service**.

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].

update: An add, modify, or delete of one or more objects or attribute values. See originating update, replicated update.

XML: The Extensible Markup Language, as described in [XML1.0].

XML Schema (XSD): A language that defines the elements, attributes, namespaces, and data types for **XML** documents as defined by [XMLSCHEMA1/2] and [W3C-XSD] standards. An XML schema uses **XML** syntax for its language.

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.

[FIPS180-2] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-2, August 2002, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

[FIPS198-1] FIPS PUBS, "The Keyed-Hash Message Authentication Code (HMAC)", FIPS PUB 198-1, http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

[IEEE1363] Institute of Electrical and Electronics Engineers, "Standard Specifications for Public-Key Cryptography", 1363-2000, August 1999, http://grouper.ieee.org/groups/1363/

[ISO-8601] International Organization for Standardization, "Data Elements and Interchange Formats - Information Interchange - Representation of Dates and Times", ISO/IEC 8601:2004, December 2004, http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40874&ICS1=1&ICS2=140&ICS3=30

Note There is a charge to download the specification.

[MS-DRMND] Microsoft Corporation, "Windows Media Digital Rights Management (WMDRM): Network Devices Protocol".

[MS-DRM] Microsoft Corporation, "Digital Rights Management License Protocol".

[MS-ERREF] Microsoft Corporation, "Windows Error Codes".

[RFC1738] Berners-Lee, T., Masinter, L., and McCahill, M., Eds., "Uniform Resource Locators (URL)", RFC 1738, December 1994, http://www.ietf.org/rfc/rfc1738.txt

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

[RFC2459] Housley, R., Ford, W., Polk, W., and Solo, D., "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 2459, January 1999, http://www.rfc-editor.org/rfc/rfc2459.txt

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.rfc-editor.org/rfc/rfc2616.txt

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000, http://www.rfc-editor.org/rfc/rfc2818.txt

[RFC3174] Eastlake III, D., and Jones, P., "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001, http://www.ietf.org/rfc/rfc3174.txt

[XML] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", W3C Recommendation 16 August 2006, edited in place 29 September 2006, http://www.w3.org/TR/2006/REC-xml-20060816/

1.2.2 Informative References

[CR-WMDRM] Microsoft Corporation, "Compliance and Robustness Rules for Windows Media DRM", http://wmlicense.smdisp.net/wmdrmcompliance/

[MSDN-MAS] Microsoft Corporation, "Hosting a Metering Aggregation Service", http://msdn.microsoft.com/en-us/library/bb614605(VS.85).aspx

[MTP] Microsoft Corporation, "Media Transfer Protocol Enhanced", September 2006, http://www.microsoft.com/downloads/details.aspx?FamilyID=fed98ca6-ca7f-4e60-b88c-c5fce88f3eea&displaylang=en

[SCHNEIER] Schneier, B., "Applied Cryptography, Second Edition", John Wiley and Sons, 1996, ISBN: 0471117099, http://www.wiley.com/WileyCDA/WileyTitle/productCd-0471117099.html

[X9.62] American National Standards Institute, "Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62:2005, 2005, http://webstore.ansi.org/ansidocstore/product.asp?sku=ANSI+X9%2E62%3A2005

Note There is a charge to download the specification.

1.3 Protocol Overview

This protocol is designed to facilitate portable devices that store and play back audio/video **content**.

These are the major components that are part of this protocol:

- Device
- Licensing Service
- Metering Aggregation Service
- Indirect License Acquisition Host
- Content Provider Media Service
- Secure Time Service

1.3.1 Device

The device serves as the client in the system. The device is responsible for generating the different types of **challenges** to other roles and ultimately presents the media to a user. This protocol is intended for use in portable media players, but can be used in any portable device.

1.3.2 Licensing Server

The licensing server maintains a list of licenses containing permissions associated to that media (content) under **digital rights management (DRM)** protection. The main function of the licensing server is to provide information that enables and confirms those rights associated with the protected media.

1.3.3 Metering Aggregation Server

Metering requires coordination between the content provider (the content owner and license issuer). A metering aggregation server collects and processes metering data.

1.3.4 Indirect License Acquisition Host

The indirect license acquisition host is used by the device as a **proxy** to obtain its license. A user connects the device to a local computer acting as an indirect license acquisition host, and the device connects through this host to obtain a license. In some cases, the indirect license acquisition host performs a DLA operation and forwards the results to the device. In other cases, the indirect license acquisition host already possesses its own license and has been given the rights to create a derived license. A derived license grants the indirect license acquisition host the right to make a secondary (and perhaps more restricted) license to send to the device.

1.3.5 Secure Clock Server

Portable devices are required to be able to validate time-based licenses against a correct time reference in order to support a time-based model of content usage. One way to implement this requirement is by using a secure clock. A secure clock is a clock that cannot be modified by any entity other than a secure time server. This section focuses on using a secure clock, and provides information about how to set a secure clock on a portable device. A complete specification of the secure clock protocol is documented in section 3.3.

Before playing **protected content (2)**, a portable device verifies that a valid license is present for the content. If the license is time-based, the device then ensures that the current time is within the validity period of the license, using the time according to the device's secure clock.

By default, the secure clocks of new out-of-the-box devices are in the unset state. Therefore, before playing protected content, the device ensures its secure clock is set as described in section 3.2.3 Initialization. Portable devices can set a secure clock indirectly by connecting to the indirect license acquisition host, or directly by communicating with a **secure clock server**.

1.4 Relationship to Other Protocols

HTTP/HTTPS: This protocol can use HTTP [RFC2616] or HTTPS [RFC2818] as a transport. For transport rules regarding HTTP/HTTPS, see section 2.1.

MTP: This protocol can use MTP [MTP] as a transport.

1.5 Prerequisites/Preconditions

This protocol extends [MS-DRMND]: Windows Media Digital Rights Management (WMDRM): Network Devices Protocol Specification.

1.6 Applicability Statement

This protocol is applicable for provisioning and managing DRM protected content (1) on portable devices where continuous connection to a network is not desired as a requirement to play protected content.

1.7 Versioning and Capability Negotiation

There is no facility for version or capability **negotiation**. To be compliant with this protocol, all components understand and correctly process all messages as specified in this document.

The use of HTTPS security is negotiated as specified in section 2.1.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

This protocol can be used with the following transport protocols:

- Media Transport Protocol [MTP]
- Hypertext Transfer Protocol [RFC2616]
- Hypertext Transfer Protocol Secure [RFC2818]

HTTPS Security: If a **Uniform Resource Locator (URL)** is used for a connection contains "https", the client SHOULD use **Secure Sockets Layer (SSL)** for the connection. If SSL is used, the client SHOULD check the server's **certificate** to ensure it is current, matches the **domain**, and is properly signed by a trusted authority. The client MUST follow HTTP [RFC2616] redirections during session. If a server responds with "HTTP 301 (Moved)" or "HTTP 302 (Redirect)", the client MUST use this redirect URL as the new URL and start again by submitting the request to the redirect URL.

2.2 Common Message Syntax

2.2.1 Common Data Types

2.2.1.1 Deviation from XML Standard

The **XML messages** sent in this protocol do not use the standard XML identifying header. The XML encoding is indicated in the following code snippet.

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

2.2.1.2 Common XML Elements

The following section contains XML schema elements common to many DRM-CD messages.

When composing XML messages for this protocol, it is necessary to understand how these messages deviate from [XML], the XML standard. The differences are covered in this document in section 3.1.5.

DATA elements: The **DATA** elements in this protocol identify the portion of a message which is authenticated using an **elliptic curve cryptography (ECC)** message signature. ECC message signature **authentication** is specified in section 2.2.1.6. The nature of the **DATA** element varies for each message.

2.2.1.2.1 MID

The MID element contains a base64-encoded (as documented in Digital Rights Management License Protocol Specification [MS-DRM] section 2.2.1.1) DRM_ID structure. This indicates the Metering ID, which is used to identify a specific MAS.

```
<xs:element name="MID" type="xs:string"/>
```

2.2.1.2.2 MSDRM_SIGNATURE_VALUE

The MSDRM_SIGNATURE_VALUE element contains a base64-encoded string (as documented in [MS-DRM] section 2.2.1.1) representing the signature of the data contained in the **DATA** complex type. The signature is computed by the mechanism specified in section 2.2.1.6.

```
<xs:element name="MSDRM_SIGNATURE_VALUE" type="xs:string" />
```

2.2.1.2.3 RECORDS

The RECORDS element is a sequence of string elements consistent with the type of message within which it has been incorporated.

The contents of the RECORDS element are encrypted with the ECC **public key** of the metering server's certificate, as specified in section 2.2.1.6. It is then base64 encoded as specified in [MS-DRM] section 2.2.1.1. The RECORDS element is stored as a string in both the METER_CHALLENGE_DATA and METERDATA_RESPONSE_DATA messages, specified in sections 2.2.3.2.2 and 2.2.3.3.2 respectively.

```
<xs:element name="RECORDS" type="xs:string"/>
```

2.2.1.2.4 TID

The TID element contains a base64-encoded string holding a DRM_ID structure as specified in [MS-DRM] section 2.2.1.1. This structure identifies a specific unique transaction.

```
<xs:element name="TID" type="xs:string" />
```

2.2.1.2.5 URL

The URL element contains a string pointing to a Uniform Resource Locator (URL) as defined by [RFC1738].

```
<xs:element name="URL" type="xs:string" />
```

2.2.1.3 Common XML Complex Types

2.2.1.3.1 CERTIFICATE

The CERTIFICATE complex type, and its corresponding **private key**, are used to generate an ECC message signature as defined in section 2.2.1.6. This element identifies a **CERT** structure or a **PKCERT** structure. The structure included depends on the protocol used.

The included structure is always base64-encoded as specified in [MS-DRM] section 2.2.1.1.

```
<xs:complexType name="CERTIFICATE">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name ="private" type="xs:int" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

private: If present, this **attribute** MUST be set to "1" to make the certificate's public key private. This means that the certificate's public key is private and is not distributed. If the value is missing or set to any other value, the certificate's public key is not private and can be distributed.

2.2.1.3.2 CERTIFICATECHAIN

The CERTIFICATECHAIN complex type identifies a **certificate chain**, containing one or more CERTIFICATE elements, and is described by the following XML schema element.

```
<xs:complexType name="CERTIFICATECHAIN">
  <xs:sequence>
    <xs:element name="CERTIFICATE" type="CERTIFICATE" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
```

CERTIFICATE: Contains one or more CERTIFICATE elements as defined in section 2.2.1.3.1.

If there are multiple CERTIFICATE child elements in the node, the first CERTIFICATE child element MUST be the certificate issued by the root authority (CR). The second CERTIFICATE child element is required to be the license server certificate (CS). Subsequent CERTIFICATE child elements comprise a certificate chain downward from CS.

2.2.1.3.3 HASHALGORITHM

The HASHALGORITHM complex type describes the hashing mechanism used when computing a SIGNATURE element, over the contents of a DATA tag.

The HASHALGORITHM contains a HASHALGTYPE enumeration.

```
<xs:complexType name="HASHALGORITHM">
  <xs:attribute name="type" type="HASHALGTYPE" />
</xs:complexType>
```

type: Contains a HASHALGTYPE enumeration as specified in 2.2.1.4.1. The type of hashing algorithm to be used to generate a **signature** element.

2.2.1.3.4 KID

The KID complex type contains the **Key** identifier for the target of a specific set of actions.The KID format allows multiple actions.The number of actions is the only action qualifier the device uses. For metering aggregation, the action is always "play". This element contains the action and the qualifiers to be applied to the action.

```
<xs:complexType name="KID">
  <xs:attribute name="value"/>
  <xs:element name="ACTION" type="xs:string">
    <xs:attribute name="value"/>
  </xs:element>
</xs:complexType>
```

value: Specifies the action. MUST be set to "b64encoded kid", indicating a play content action.

ACTION: Contains a string indicating the maximum number of times the content is allowed to be played. The content MUST NOT be played more than this number of times.

ACTION.value: Contains the action to be performed. For example, the action for metering aggregation is always "play".

2.2.1.3.5 SIGNALGORITHM

The SIGNALGORITHM complex type contains the signing algorithm used for computing a signature, which is represented by a SIGNALGTYPE enumeration.

```
<xs:complexType name="SIGNALGORITHM">
  <xs:attribute name ="type" type="SIGNALGTYPE"/>
</xs:complexType>
```

type: Contains a SIGNALGTYPE enumeration as specified in section 2.2.1.4.3. This indicates the algorithm used for computing a signature as specified in section 2.2.1.6.

2.2.1.3.6 SIGNATURE

The SIGNATURE complex type calls out the HASHALGTYPE and SIGNALGTYPE values used in signature computation. The SIGNATURE element is otherwise analogous to the MSDRM_SIGNATURE_VALUE element. In practice, these values are **SHA1** and **MSDRM**, precisely the primitives used for creating a MSDRM_SIGNATURE_VALUE, and detailed in section 2.2.1.6. The resulting signature is stored in a SIGNATURE_VALUE element as a base64-encoded string as specified in [MS-DRM] section 2.2.1.1. This string contains the generated signature.

SIGNATURE elements are required to be accompanied by a CERTIFICATECHAIN element.

```
<xs:complexType name="SIGNATURE">
  <xs:sequence>
    <xs:element name="HASHALGORITHM" type="HASHALGORITHM" />
    <xs:element name="SIGNALGORITHM" type="SIGNALGORITHM" />
    <xs:element name="VALUE" type="SIGNATURE_VALUE" />
  </xs:sequence>
</xs:complexType>
```

HASHALGORITHM: The hash algorithm used to generate the signature, as specified in section 2.2.1.4.1 HASHALGTYPE enumeration.

SIGNALGORITHM: The signing algorithm used to generate the signature, as specified in section 2.2.1.3.5 SIGNALGORITHM element.

VALUE: The base64-encoded string containing the generated signature. The base64 encoding is specified in [MS-DRM] section 2.2.1.1.

2.2.1.3.7 SIGNATURE_VALUE

The SIGNATURE_VALUE complex type contains a signature, as specified in section 2.2.1.6. The MSDRM signature is computed using the methods specified in section 2.2.1.6. The string is base64 encoded as specified in [MS-DRM] section 2.2.1.1.

```
<xs:complexType name="SIGNATURE_VALUE">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name ="private" type="xs:int" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

private: An optional attribute that provides metadata about the signature value. When present, MUST be set to "1", indicating that the value is kept private.

2.2.1.4 Common XML Enumerations

2.2.1.4.1 HASHALGTYPE

The HASHALGTYPE enumeration identifies the type of hashing algorithm used when generating an ECC signature. The only supported algorithm is SHA1.

```
<xs:simpleType name="HASHALGTYPE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="SHA"/>
  </xs:restriction>
</xs:simpleType>
```

SHA: The SHA1 hashing algorithm is used to generate the ECC signature. For information on SHA1 see [RFC3174].

2.2.1.4.2 MESSAGETYPE

The MESSAGETYPE enumeration is used in Windows Media Digital Rights Management (WMDRM): MTP Command Extension messages to determine if a message contains a challenge, or response message. A challenge message is a request for information required to fulfill a process and a response message is sent as a reply to a challenge message.

```
<xs:simpleType name="MESSAGETYPE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="challenge"/>
    <xs:enumeration value="response"/>
  </xs:restriction>
</xs:simpleType>
```

challenge: A request for information.

response: A reply to a request for information.

2.2.1.4.3 SIGNALGTYPE

The SIGNALGTYPE enumeration represents the possible algorithm choices for generating a signature. This protocol only supports the MSDRM signature scheme, as specified in section 2.2.1.6.

```
<xs:simpleType name="SIGNALGTYPE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="MSDRM"/>
  </xs:restriction>
</xs:simpleType>
```

MSDRM: The scheme used to generate a signature. The MSDRM scheme is the only value supported by DRM-CD.

2.2.1.5 Common Binary Structures

2.2.1.5.1 DRM_ID

The DRM_ID binary structure contains a 16-byte opaque structure indicating a unique identity bound to a specific DRM-enabled device.

<table>
<tr><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>1</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>2</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>3</th><th></th></tr>
<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>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>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>0</th><th>1</th></tr>
<tr><td colspan="32">GUID (16 bytes)</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

GUID (16 bytes): A byte array containing a unique, opaque sequence of bytes. The opaque sequence of bytes MUST be reliably unique when generated. The sequence MUST be guaranteed to be unique across all devices and all applications.

2.2.1.5.2 KIPub

The KIPub structure is the public key representing the root **certificate authority (CA)** key. This is used to sign the **root certificate** in the server certificate chain (**CS**). **KIPub** is specified in the following byte sequence.

<table>
<tr><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>1</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>2</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>3</th><th></th></tr>
<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>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>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>0</th><th>1</th></tr>
<tr><td colspan="8">0x4D</td><td colspan="8">0xBF</td><td colspan="8">0xD9</td><td colspan="8">0x0D</td></tr>
<tr><td colspan="8">0xD9</td><td colspan="8">0x6E</td><td colspan="8">0x8C</td><td colspan="8">0x9E</td></tr>
<tr><td colspan="8">0x32</td><td colspan="8">0x5F</td><td colspan="8">0x4F</td><td colspan="8">0x3D</td></tr>
<tr><td colspan="8">0xEC</td><td colspan="8">0xA9</td><td colspan="8">0x84</td><td colspan="8">0x59</td></tr>
<tr><td colspan="8">0x6B</td><td colspan="8">0x5E</td><td colspan="8">0x06</td><td colspan="8">0x86</td></tr>
<tr><td colspan="8">0xE7</td><td colspan="8">0xE2</td><td colspan="8">0xC2</td><td colspan="8">0x8B</td></tr>
<tr><td colspan="8">0xDE</td><td colspan="8">0x14</td><td colspan="8">0x4B</td><td colspan="8">0x29</td></tr>
<tr><td colspan="8">0x2C</td><td colspan="8">0xEC</td><td colspan="8">0x4D</td><td colspan="8">0x1D</td></tr>
<tr><td colspan="8">0x76</td><td colspan="8">0xFD</td><td colspan="8">0x5A</td><td colspan="8">0x14</td></tr>
<tr><td colspan="8">0x90</td><td colspan="8">0x3A</td><td colspan="8">0x10</td><td colspan="8">0x77</td></tr>
</table>

2.2.1.6 Cryptographic Characteristics

ECC keys: Windows Media Digital Rights Management (WMDRM): MTP Command Extension protocols use 160 bit ECC keys as defined in [MS-DRM] section 2.2.1.3.

ECC signature creation: MSDRM signatures are created using ECDSA [X9.62] over curve ECC_1. This algorithm is defined in its entirety in [IEEE1363]. The ECC_1 curve is specified in [MS-DRM] section 2.2.1.2.

WMDRM Certificates: WMDRM certificates are used as specified in [MS-DRM] in section 2.2.2.

Cryptographic parameters: This protocol uses cryptographic parameters specified in [MS-DRM] section 2.2.1.2.

2.2.2 Secure Clock Protocol Messages

The secure clock protocol is used to ensure that the clock on the device is synchronized with time as specified in section 3.3 Secure Clock Server Details.

This section defines secure clock messages in **XML Schema (XSD)** format. Prior to being transmitted, each secure clock XML message MUST be encoded as specified in [MS-DRM] section 2.2.1.1 when sent. When the messages are received, they MUST be decoded as specified in [MS-DRM] section 2.2.1.1.

2.2.2.1 Petition for a Secure Clock Challenge

The client MUST issue a new petition for each new secure clock challenge. The secure clock server MAY deny a secure clock challenge that has not been petitioned. Therefore, it is important that the client does not cache petitions.

The petition request is a plain HTTP/1.0 GET request, as shown by this example:

```
GET PetitionPath HTTP/1.0
User-Agent: Client-User-Agent
```

If the petition URL contains "https", the client SHOULD use SSL for the connection. If SSL is used, the client SHOULD check the server's certificate to ensure it is current, matches the domain, and is properly signed by a trusted authority.

2.2.2.2 Secure Clock Challenge Petition Response

If the secure clock server responds with HTTP 200 (OK), the client reads the entire body of the response for the secure clock challenge URL. If successful, the client proceeds to the next step, submitting the secure clock challenge.

A valid petition response has the following format:

```
HTTP/1.1 302 Found
Server: Microsoft-IIS/5.0
Location: URL
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 191
```

2.2.2.3 Secure Clock Challenge POST Request

The client submits a secure clock challenge request to the secure clock challenge URL, retrieved from the petition response, and waits for the server's response. The secure clock challenge request is an HTTP/1.0 POST request as shown below:

```
POST ChallengePath HTTP/1.0
Accept: Accept: */*
Content-Type: application/x-www-form-urlencoded
Content-Length: 376
User-Agent: Client-User-Agent
```

```
Proxy-Connection: Keep-Alive
Pragma: no-cache
```

The client MUST be prepared to follow HTTP [RFC2616] redirections during the petition. If the secure clock server responds with HTTP 301 (Moved) or 302 (Redirect), the client MUST use this redirect URL as the new secure clock challenge URL and start again by submitting the secure clock challenge to the redirected URL.

2.2.2.4 Secure Clock Challenge Message

The secure clock challenge message is used to start a device clock synchronization procedure.

```
<xs:element name="DRMCLOCK" type="DRMCLOCK_CHALLENGE" />
```

DRMCLOCK_CHALLENGE: A DRMCLOCK_CHALLENGE complex type, as specified in section 2.2.2.4.1.

2.2.2.4.1 DRMCLOCK_CHALLENGE

The DRMCLOCK_CHALLENGE complex type is the basis for a secure clock challenge message.

```
<xs:complexType name="DRMCLOCK_CHALLENGE">
  <xs:sequence>
    <xs:element name="DATA" type="SECURECLOCK_CHALLENGE_DATA"/>
  </xs:sequence>
  <xs:attribute name ="type" type="MESSAGETYPE"/>
</xs:complexType>
```

DATA: Contains a SECURECLOCK_CHALLENGE_DATA complex type as specified in section 2.2.2.4.2.

type: The message type. MUST be set to "CHALLENGE".

2.2.2.4.2 SECURECLOCK_CHALLENGE_DATA

The SECURECLOCK_CHALLENGE_DATA complex type contains the location of the secure clock server, as well and the transaction ID generated by the device. The transaction ID is used to associate the resultant response message with the challenge message.

```
<xs:complexType name="SECURECLOCK_CHALLENGE_DATA">
  <xs:sequence>
    <xs:element name="URL" type="xs:string"/>
    <xs:element name="TID" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

URL: A URL element, as specified in section 2.2.1.2.5. Identifies the URL of the secure clock server. This URL value is present in the device certificate.

TID: Contains the **transaction ID**. This unique value is generated by the device and associated with a single transaction. This value MUST NOT be re-used in future messages or transactions.

2.2.2.5 Secure Clock Response Message

The secure clock response message is used to supply clock data from a trusted clock server so that the device can synchronize its internal clock to ensure the timely application of content licensing **policy**.

```
<xs:element name="DRMCLOCK_RESPONSE" type="DRMCLOCK_RESPONSE" />
```

DRMCLOCK_RESPONSE: A DRMCLOCK_RESPONSE complex type, as specified in section 2.2.2.5.1.

2.2.2.5.1 DRMCLOCK_RESPONSE

The DRMCLOCK_RESPONSE complex type is used to construct the response to a secure clock challenge message as described in section 2.2.2.4.

```
<xs:complexType name="DRMCLOCK_RESPONSE">
  <xs:choice>
    <xs:sequence>
      <xs:element name="DATA" type="SECURECLOCK_RESPONSE_DATA"/>
      <xs:element name="CERTIFICATECHAIN" type="CERTIFICATECHAIN" />
      <xs:element name="SIGNATURE" type="SIGNATURE" />
    </xs:sequence>
    <xs:element name="ERROR" type="xs:int"/>
  </xs:choice>
  <xs:attribute name ="type" type="MESSAGETYPE"/>
</xs:complexType>
```

DATA: Contains a SECURECLOCK_RESPONSE_DATA complex type, as specified in section 2.2.2.5.2.

CERTIFICATECHAIN: Contains the certificate(s) associated with the private ECC key. MUST be used to sign the **DATA** element in the DRMCLOCK_RESPONSE message (section 3.3.5.3).

SIGNATURE: Contains a secure signature for this message. The signature is computed by the server using the private key from the secure clock certificate, using the ECC signature creation mechanisms specified in section 2.2.1.6. The corresponding public key from the device certificate is used to validate the signature. Signature validation is described in [MS-DRM] section 3.2.5.1.

The signature is computed over the entire **DATA** element, which is composed of SECURECLOCK_RESPONSE_DATA as documented in section 2.2.2.5.2.

ERROR: An **HRESULT** value, as specified in [MS-ERREF]. A nonzero value MUST be sent when error conditions occur. This is used when an **error code** is returned. This type of response is sent by the secure clock server if an error is encountered when constructing the response.

type: The type of message. MUST be set to "RESPONSE".

2.2.2.5.2 SECURECLOCK_RESPONSE_DATA

The SECURECLOCK_RESPONSE_DATA complex type is the payload for a response to the SECURECLOCK_CHALLENGE message. It contains information used to ensure that the device's clock conforms to policy. Policy is specified within the license.

```
<xs:complexType name="SECURECLOCK_RESPONSE_DATA">
  <xs:sequence>
    <xs:element name="TID" type="xs:string" />
    <xs:element name="GMTTIME" type="xs:string" />
    <xs:element name="REFRESHDATE" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

TID: Identifies a unique transaction between the device and the secure clock server. The same value MUST be used from the device's secure clock challenge message. This value MUST NOT be re-used in future messages or transactions.

GMTIME: A string representation of **Greenwich mean time (GMT)** time, indicating the current GMT at the time the message was sent. This string is derived using the time conversion method specified in [ISO-8601]. This value is used to set the device's internal clock. MUST be set to the secure clock server's **time** when sent.

REFRESHDATE: Contains a string representation of the refresh date, in GMT time. This element MUST be set to the secure clock server's time, plus a policy configurable time delta. When this date is reached, the device MUST request resynchronization with the secure clock server.

This string is derived using the time conversion method specified in section 3.3.5.3. The device SHOULD use this value to determine when its secure clock is in need of refresh. This action SHOULD cause the indirect license acquisition host to perform a new secure clock synchronization procedure. In a non-tethered scenario, the device synchronizes with a secure clock after this date.

2.2.3 Metering Protocol Messages

The metering aggregation server is used to coordinate the usage of content against the license associated with the content. This section specifies the messages used to interact with the metering aggregation server as specified in section 3.4.

2.2.3.1 Metering Certificates

In order to generate a metering challenge message, the device MUST have pre-knowledge of an appropriate metering certificate for each MID presented in a content license. Each metering certificate contains a unique MID. A device will only be able to utilize content with licenses that contain the same MID as that found in the metering certificate.

2.2.3.1.1 METERCERT

The METERCERT complex type forms the basis for a metering certificate.

```
<xs:complexType name="METERCERT">
  <xs:sequence>
    <xs:element name="DATA" type="METERCERT_DATA" />
    <xs:element name="CERTIFICATECHAIN" type="CERTIFICATECHAIN" />
    <xs:element name="SIGNATURE" type="SIGNATURE" />
  </xs:sequence>
  <xs:attribute name="version" type="xs:string"/>
</xs:complexType>
```

DATA: MUST contain a METERCERT_DATA complex type, as specified in section 2.2.3.1.2.

CERTIFICATECHAIN: MUST contain the certificate whose private ECC key has been used to sign the **DATA** element.

SIGNATURE: Contains the ECC signature of the METERCERT_DATA. The signature MUST validate against the public key extracted from the accompanying CERTIFICATECHAIN element.

version: Indicates the version of the metering data format. This element is set to "1.0" for this version of the protocol.

2.2.3.1.2 METERCERT_DATA

The METERCERT_DATA message specifies authentication elements for the metering certificate. These are additional elements containing information to strengthen the authentication of the metering data beyond that offered by the Metering certificates specified in [MS-DRM]. This is used to ensure the source of the information is valid.

```
<xs:complexType name="METERCERT_DATA">
  <xs:sequence>
    <xs:element name="MID" type="xs:string"/>
    <xs:element name="PUBLICKEY" type="xs:string"/>
    <xs:element name="URL" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

MID: Identifies the MID to associate with the metering certificate.

PUBLICKEY: Contains a base64-encoded ECC public key, in the PUBKEY format ([MS-DRM] section 2.2.1.6). The **PUBKEY** is the public portion of a **public-private key pair** in ECC_1. Base64 encoding is performed as specified in [MS-DRM] section 2.2.1.1. This identifies the public key for the metering certificate, which is used for encrypting the Metering Protocol messages.

URL: A URL element, as specified in section 2.2.1.2.5. Identifies the URL for the metering aggregation server.

2.2.3.2 Metering Challenge Message

A metering challenge message used by the device to send data to the metering aggregation server about the usage of content on the device.

```
<xs:element name="METERDATA" type= "METERDATA_CHALLENGE" />
```

METERDATA: A METERDATA_CHALLENGE complex type, as specified in section 2.2.3.2.1.

2.2.3.2.1 METERDATA_CHALLENGE

The METERDATA_CHALLENGE complex type is the basis for a metering challenge message.

```
<xs:complexType name="METERDATA_CHALLENGE">
  <xs:sequence>
    <xs:element name="DATA" type="METER_CHALLENGE_DATA" />
    <xs:element name="MSDRM_SIGNATURE_VALUE" type="xs:string" />
    <xs:element name="CERTIFICATE" type="CERTIFICATE" />
  </xs:sequence>
  <xs:attribute name ="type" type="MESSAGETYPE"/>
</xs:complexType>
```

DATA: Contains METERDATA_CHALLENGE_DATA.

MSDRM_SIGNATURE_VALUE: Contains the signature value. This element is computed by the server using the fallback device certificate contained in the accompanying CERTIFICATE element. The signature is computed over the entire **DATA** element, which is composed of METER_CHALLENGE_DATA.

CERTIFICATE: This element contains a **PKCERT** structure ([MS-DRM] section 2.2.1.5) corresponding to the fallback device certificate used to sign the **DATA** element of the message. The **PKCERT** structure is base64-encoded as documented in [MS-DRM] section 2.2.1.1.

type: Indicates the type of message. MUST be set to "CHALLENGE".

2.2.3.2.2 METER_CHALLENGE_DATA

The METER_CHALLENGE_DATA complex type contains the list of records the device requires to ensure its license data is current.

```
<xs:complexType name="METER_CHALLENGE_DATA">
  <xs:sequence>
    <xs:element name="URL" type="xs:string"/>
    <xs:element name="MID" type="xs:string"/>
    <xs:element name="TID" type="xs:string"/>
    <xs:element name="PARTIALDATA" minOccurs="0" type="xs:string"/>
    <xs:element name="RECORDS" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

URL: A URL element, as specified in section 2.2.1.2.5. Identifies the URL of the metering aggregation server, which services the MID for the content license. This value is present in the device certificate, which also contains the public key for the metering aggregation server.

MID: Identifies the MID which is being tracked.

TID: The transaction identifier (TID). This is a unique value generated by the device, which MUST NOT be replayed in future messages/transactions.

PARTIALDATA: Optional element. When present, this element is used to indicate whether more data is available. When there is no more data, this element MAY be present and set to "0". If **PARTIALDATA** is not present, then this is equivalent to **PARTIALDATA** being present with a value of "0". There is no more data available.

Value	Meaning
"0"	No more data is available.
"1"	More data is available.

RECORDS: Contains a list of KIDs, as specified in section 2.2.1.3.4. The list is encrypted with the public ECC key of the metering certificate for the target MID, as described in section 2.2.1.6. The result MUST be base64-encoded as documented in [MS-DRM] section 2.2.1.1, and sent in the RECORDS element as a METERCERT_RECORD_DATA (section 2.2.3.2.3).

2.2.3.2.3 METERCERT_RECORD_DATA

The METERCERT_RECORD_DATA complex type contains a list of KIDs for use in the RECORDS element. The list is encrypted with the public ECC key of the metering certificate for the target MID, as specified in section 2.2.1.6.

This section specifies the METERCERT_RECORD_DATA complex type in unencrypted form. The METERCERT_RECORD_DATA complex type MUST be base64-encoded before it is placed in the RECORDS element. Base64 encoding is specified in [MS-DRM] section 2.2.1.1.

```
<xs:complexType name="METERCERT_RECORD_DATA">
  <xs:sequence maxOccurs="unbounded">
    <xs:element name="KID" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

KID: Contains the KIDs as documented in section 2.2.1.3.4.

2.2.3.3 Metering Response Message

The metering response message is sent in response to the metering challenge message (section 2.2.3.2). It contains a list of actions the device SHOULD take to ensure content is kept in compliance with the content's license. The content upon which the actions are taken can be identified through the KID associated with the license for the content.

```
<xs:element name="METERDATA" type= "METERDATA_RESPONSE" />
```

METERDATA: A METERDATA_RESPONSE complex type, as specified in section 2.2.3.3.1.

2.2.3.3.1 METERDATA_RESPONSE

The METERDATA_RESPONSE complex type is the basis for a metering response message. This information included ensures the device license data is current.

```
<xs:complexType name="METERDATA_RESPONSE">
  <xs:sequence>
    <xs:element name="DATA" type="METERDATA_RESPONSE_DATA" />
    <xs:element name="MSDRM_SIGNATURE_VALUE" type="xs:string" />
    <xs:element name="METERCERT" type="METERCERT" />
  </xs:sequence>
  <xs:attribute name ="type" type="MESSAGETYPE"/>
</xs:complexType>
```

DATA: Contains METERDATA_RESPONSE_DATA, as specified in section 2.2.3.3.2.

MSDRM_SIGNATURE_VALUE: Contains a signature computed using the private key from the metering certificate contained in the accompanying METERCERT element. The signature is computed over the entire **DATA** element, as described in section 2.2.1.6.

METERCERT: Contains the certificate whose private key was used to sign the **DATA** element.

type: Indicates the message type. MUST be set to "RESPONSE".

2.2.3.3.2 METERDATA_RESPONSE_DATA

The METERDATA_RESPONSE_DATA complex type contains a list of identifiers with associated records.

```
<xs:complexType name="METERDATA_RESPONSE_DATA">
  <xs:sequence>
    <xs:attribute name="MID" type="xs:string"/>
    <xs:attribute name="TID" type="xs:string"/>
    <xs:attribute name="PARTIALDATA" type="xs:string" use="optional" default= "1"/>
    <xs:element name="RECORDS" type="xs:string"/>
    <xs:element name="COMMAND" type="COMMAND" />
  </xs:sequence>
</xs:complexType>
```

MID: Contains the MID being tracked. The MID being tracked MUST match the value presented in the metering challenge message and MUST contain the MID used in the accompanying METERCERT element.

TID: Contains the TID being tracked. MUST match the TID from the metering challenge message (section 2.2.3.2).

PARTIALDATA: Optional element. When present, this element is used to indicate whether more data is available. When there is no more data, **PARTIALDATA** MAY be present and set to "0". If **PARTIALDATA** is not present then this is equivalent to **PARTIALDATA** being present with a value of "0". There is no more data available.

Value	Meaning
"0"	No more data is available.
"1"	More data is available.

RECORDS: Contains a list of KIDs. Each KID is associated with a content license. The list of KIDs is encrypted with the public ECC key of the metering certificate for the target MID, as described in section 2.2.1.6, and then base64 encoded.

COMMAND: A COMMAND enumeration, as specified in section 2.2.3.3.3. MUST be set to "RESET".

2.2.3.3.3 COMMAND

The COMMAND enumeration is used to direct the behavior of the device upon receiving a metering protocol response message by giving it a command. The only supported command is "RESET", as specified below.

```
<xs:simpleType name="COMMAND">
  <xs:restriction base="xs:string">
    <xs:enumeration value="RESET"/>
  </xs:restriction>
</xs:simpleType>
```

RESET: This command triggers a reset of the metering data on the store for the given MID.

2.2.4 Synchronization List Messages

The synchronization list (DRMSYNCLIST) facilitates the **update**/refresh process when the device connects with a host.

2.2.4.1 DRMSYNCLIST

The DRMSYNCLIST message is a synchronization list. This list is a quick reference of every license that needs to be replaced or updated.

An entry is added to the synchronization list for every **Type 2** and **Type 3** licenses on the system. The synchronization list transmitted up to a host provides an immediate reference to those licenses that need freshening, and licenses that require a replacement by another license.

State Data: State data is different depending on the type of license.

Type 2 (expiring) licenses: State data is not maintained because the license has a built-in expiration date.

Type 3 (state-dependent) licenses: State data is maintained. State determines expiration date, play counts remaining, or both. State must be retrieved to determine whether the license needs to be updated/refreshed.

```
<xs:complexType name="DRMSYNCLIST">
  <attribute name="type" type="xs:string">
  <xs:element name="RECORDS">
    <xs: element name="KID" type="xs:string">
  </xs:element>
</xs:complexType>
```

type: Indicates the message type. MUST be set to "challenge".

RECORDS.KID: Contains a base64 encoded KID.

2.2.4.1.1 License Refresh

System licenses are cast into three categories with corresponding renewal actions: The license structure and usage is specified in [MS-DRM] section 2.2.2.3.

License Category	Description	Refresh Action
Type 1	Permanent license, grants an unlimited right to play / use	Permanent license, never requires a renewal. Once this license has been downloaded, it's good forever.
Type 2	Temporary license, limits are fundamental to the license itself (for example, expiration date)	Temporary license with expiration - a new license with the same KID must be downloaded.
Type 3	Temporary license, limits are based on system state data (for example, play count limits or relative time limit)	Temporary license with state limits A host computer can have instances where it creates a derived device license with deliberate limitations, such as play count limits or relative time limits (for example, the license can be good for the next 30 days). For this case, the host will be able to create new licenses as needed.

2.2.4.1.2 Synchronization List Message - Client

Creation of the list is the responsibility of the portable device. Multiple licenses or multiple updates over time can update the same KID entry in the synchronization list (DRMSYNCLIST). The license format is specified in [MS-DRM] section 2.2.2.3.

Explicit removal of a license through DeleteLicense can result in the removal of an entry from the synchronization list.

Entries in the synchronization list reflect that the content could expire and not that the content has already expired. A KID is added to the synchronization list when a license first arrives on the system. The evaluation will be based only on the PLAY action.

2.2.4.1.3 Synchronization List Message - Host

An outside application initiates the sync operation. This synchronization command also passes in the threshold data to the device. Examples of threshold data include play count and hour values. The device processes the request and returns a synchronization list (DRMSYNCLIST) to the Host. The license format is specified in [MS-DRM] section 2.2.2.3.

The synchronization list consists of all licenses on the device that exceed the "Threshold" and thus requires a license side-load. The host iterates through the synchronization list and side-loads the license to the device.

2.2.4.2 DRMSYNCLIST_CHALLENGE

The DRMSYNCLIST_CHALLENGE complex type contains a sequence of SYNCLIST_CHALLENGE_DATA records. These records of type SYNCLIST_CHALLENGE_DATA are described in section SYNCLIST_CHALLENGE_DATA.

```
<xs:complexType name="DRMSYNCLIST_CHALLENGE">
  <xs:sequence maxOccours="unbounded">
    <xs:element name="RECORDS" type="SYNCLIST_CHALLENGE_DATA"/>
  </xs:sequence>
  <xs:attribute name ="type" type="MESSAGETYPE"/>
</xs:complexType>
```

RECORDS: Contains a SYNCLIST_CHALLENGE_DATA complex type, as specified in section 2.2.4.3.

type: Indicates the message type. MUST be set to "CHALLENGE".

2.2.4.3 SYNCLIST_CHALLENGE_DATA

The SYNCLIST_CHALLENGE_DATA complex type contains a sequence of KIDs upon which the License Server is requested to take specific action on behalf of the device to determine if the content is still within the policy as described in the content's associated license.

```
<xs:complexType name="SYNCLIST_CHALLENGE_DATA">
  <xs:sequence maxOccours="unbounded">
    <xs:element name="KID" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

KID: Contains the information as documented in section 2.2.1.3.4.

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

3.1.1.1 Cryptographic Requirements

The following data needs to be present prior to implementing any of these protocols. This data is licensed from a license acquisition server and kept as permanent state data.

KSpriv: The private server cryptographic key.

CS: The server certificate chain.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Processing Events and Sequencing Rules

The device and server roles specified for this protocol MUST have the **KSpriv** key available and MUST have the same **CS** keys.

To ensure the timely synchronization of the device's secure clock, a time delta is manageable via policy setting. When the **REFRESHDATA** with the time delta is reached, the device MUST request a time synchronization with the secure clock server.

3.1.5.1 Cryptographic Algorithms and Characteristics

The following algorithms are used by the protocol specified in this document.

Block Cipher: The block **cipher** used is **Data Encryption Standard (DES)** with 56-bit keys in 8 byte blocks.

Stream Cipher: The stream cipher used is **Rivest Cipher 4 (RC4)** ([SCHNEIER] section 17.1) with 64-bit keys with Byte Streams.

Public Key Infrastructure (PKI): The **PKI** key used is ECC with 160/320-bit private key/public key pair. It is strongly recommended that the portable device store the private key securely.

Hash Algorithm: WM-DRMCD compliant devices use the **Secure Hashing Algorithm 1 (SHA-1)** [FIPS180-2].

HMAC Algorithm: DRM-CD compliant devices and servers currently use **Hash-based Message Authentication Code (HMAC)** [FIPS198-1].

The following table specifies characteristics for the cryptographic algorithms used in this protocol.

Component	Certificate	Algorithm	Key length	Input/Output	Use
DRM	Proprietary	Elliptic Curve	PrivKey: 160-bit PubKey: 360 bit	16 byte block / 80 bytes	**Digital signature** , **Key Exchange**
DRM	n/a	RC4	64-bit	Byte stream / Byte stream	Data **Encryption** (transient)
DRM	n/a	CBC-MAC	48-bytes	Byte blocks / 8 Bytes	Data Encryption (transient)
License Server	Proprietary	Elliptic Curve	PrivKey: 160-bit PubKey: 360 bit	16 byte block / 80 bytes	License signing, Data Encryption, Signature Verification
Content Server	n/a	RC4-DES-SHA1	64/56-bit		Data Encryption (persistent)

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Device Details

3.2.1 Abstract Data Model

The following ADM elements are present on the client.

Device Identity: When devices are created, they are given a unique serial number to prevent device cloning.

Real-Time Clock: Devices SHOULD implement a real-time clock. This allows the use of the secure clock **service**.

Refresh Time: The amount of time that can elapse before the device needs to set its **Real-TimeClock** by contacting the secure clock server (section 3.2.5.5).

Policy: The device uses policy defined in the license associated with the media (section 3.2.5.6) to securely play existing and new Protected content (1).

3.2.1.1 Persistent Local Storage

Local storage on the device is required to persist content and **DRM Data Stores**.

When a user acquires protected content (1) on a portable device, a license is issued to the device in one of the following two ways:

- Direct license acquisition (section 3.2.5.4)
- Indirect license acquisition (section 3.6)

In both cases, the license is bound to the device certificate, and then the license is placed in a secured storage location on the device.

To play protected content, the device searches the license store located in a secured storage location for licenses. When a license is found, the device private key is used to decrypt the **content key** contained in the license, and then the **content key** decrypts the content for playback. This private key is unique to the device. Therefore, a license is only valid on the device to which it is issued.

This process is straightforward when a device has only one location to store content and licenses. However, when a device supports removable storage media, such as flash memory cards and USB sticks, this process presents several implications that affect the design of the portable device. The device needs to store licenses in a way that ensures the user can successfully play protected content.

There are two types of **DRM Data Store**:

Permanent DRM Data Store: A permanent local storage mechanism built in to the device. This storage is always available to the device.

Temporary DRM Data Store: Removable data storage that can be attached to the device temporarily. This storage is available if and only if it is physically attached to the device.

When content is placed on a **DRM Data Store**, the corresponding license has to be placed either in the **permanentDRM Data Store** on the same **temporaryDRM Data Store**. This ensures the content can be played when it is available. If the license is placed on a removable **DRM Data Store** that is separate from the corresponding protected content, the user cannot play the protected content from the device unless both media are attached to the device.

As a general rule, a protected content file and its license are kept in the same **DRM Data Store.**

Licenses are stored securely using techniques independent of the format used by local storage medium. This can be done using a secured storage location on the device.

When a license is issued, the license is bound to the device certificate of the device, even when the license is located on **removable media**. This prevents secure content from being insecurely replayed on a separate device.

Copied licenses are not valid licenses. Although protected content files can be copied, protected content requires a license to be issued for the media at its new location. Therefore, devices that are not web-enabled need to connect to a computer and use the appropriate implementation-specific mechanism to transfer the content and issue licenses to the device.

3.2.1.2 Cryptographic Keys

The device needs to have the following keys and certificates ready in order to use this protocol.

KCpriv: The private client cryptographic key.

KSpub: The public server cryptographic key.

CA: The client application certificate. This is required for the leaf certificate only. This secures the indirect licensing ceremony and ensures that the client application is authorized to participate in an indirect licensing ceremony. This ceremony is described in section 3.6 of this document.

CM: The client machine certificate. This ensures that the client machine is authorized to participate in the indirect licensing ceremony. This ceremony is described in section 3.6 of this document.

KIPub: The KIPub public key representing the root certificate authority (CA) key, as specified in section 2.2.1.5.2.

3.2.2 Timers

None.

3.2.3 Initialization

Upon initialization, the device SHOULD ensure its clock is correct by updating its clock using the process as described in section 3.2.5.5.1.

3.2.3.1 Creating a Device Certificate

When DRM is initialized on the device, a unique device certificate with a unique device key pair MUST be created for that specific device:

1. The device creates a unique public-private device key pair.
2. The device creates a unique device certificate with the following stored in the device certificate:
   - The device public key is stored in the device certificate.
   - The device private key is encrypted with a hash of the group private key, and then is stored in the device certificate.
3. The device certificate is stored on the device.

This sequence is initiated whenever DRM is started on the device. For example, DRM can be initialized when the device first starts, or the first time a computer tries to transfer protected content (1) to the device.

The following diagram shows how a device certificate and a device key pair are created when the device is initialized.

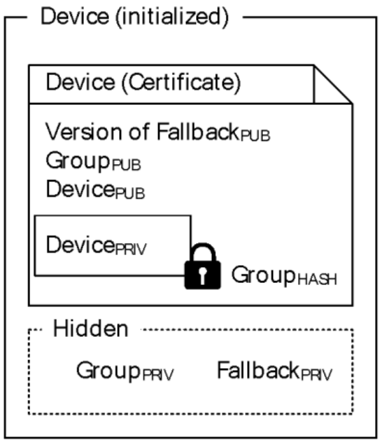

Figure 1: Initialized Device Certificate

3.2.4 Higher-Layer Triggered Events

Refresh Time Event: This event initiates the use of the secure clock server (section 3.3).

Once the secure clock on the device has been set, it will periodically be reset according to **Refresh Time** set using the **REFRESHDATE** from a previously stored secure clock response. Once the **REFRESHDATE** has elapsed, a **Refresh Time Event** occurs.

When a **Refresh Time Event** occurs, the device SHOULD attempt to set its clock by contacting the secure clock server (section 3.3) the next time it is connected to the indirect license acquisition host, or has established internet access.

3.2.5 Processing Events and Sequencing Rules

Devices SHOULD implement a **Real-Time Clock** and refresh it after each specified **Refresh Time** interval. The ability to properly restrict access to content based on a specific date requires this functionality.

Devices MUST support the **policy** as described in the content's license to play existing and new Protected content (1).

Cryptographic keys MUST be present as described in section 3.2.1.2 Cryptographic Keys.

Group certificate key pair: This key pair is shared by all devices of the same model.<1>

The **group** certificate private key is stored in a manner so that the key is opaque to the user on the device. The group certificate private key is used to sign the device certificate.

Device key pair: This key pair is generated by the device at run time and is used to secure information on the device.

The device public key is stored in the device certificate, and is used by licensing servers to encrypt **content keys** for licenses that are issued specifically for this device.

The device private key is encrypted and stored in the device certificate. The device private key is used to decrypt the **content key** in licenses during the **decryption** process Group Keys.

Fallback certificate key pair: This key pair is also shared by all devices of the same model. <2> This key pair is used for acquiring licenses from versions of Media Rights licensing servers that do not support Media DRM for portable devices.

The version of the fallback public key is stored in the device certificate.

The fallback private key is stored (hidden) on the device and is used to decrypt the **content key** for those licenses that are issued to the fallback certificate.

3.2.5.1 Local Storage and Acquiring Content and Licenses

The device MUST maintain persistent content and **DRM Data Stores**. For more information, see section 3.2.1.1.

Licenses MUST be stored on either a permanent **DRM Data Store** or in the same temporary **DRM Data Store** as the corresponding content. Licenses SHOULD always be kept in the same temporary **DRM Data Store** as the corresponding content. Each individual content license MUST be bound to the device certificate of the device. Copied licenses MUST NOT be considered valid.

3.2.5.2 Acquiring a License

A device can acquire a license in two ways:

- Direct license acquisition is used when a device has an Internet connection and can acquire licenses directly from a licensing server.
- Indirect license acquisition is used when a device connects to a computer to synchronize content. When protected content (1) is transferred from the computer to the device, the DRM component of the media player issues a license to the device. The new device license is based on the original license on the computer.

3.2.5.3 License Acquisition Keys and Certificates

The license acquisition process uses key pairs as follows:

1. The device requests a license and sends its device certificate to the license issuer.
2. The license issuer validates the device certificate and verifies that the device can enforce the rights required by the content license.
3. The license issuer uses the device public key (located in the device certificate) and encrypts the **content key**. The license issuer then creates a license with the appropriate rights and includes the encrypted **content key**. Only the device with the matching device private key can decrypt the **content key** and play the content by using this license.
4. The license is stored in the license store on the device.

The following diagram shows how the license issuer uses the device key pair to encrypt a license during license acquisition.

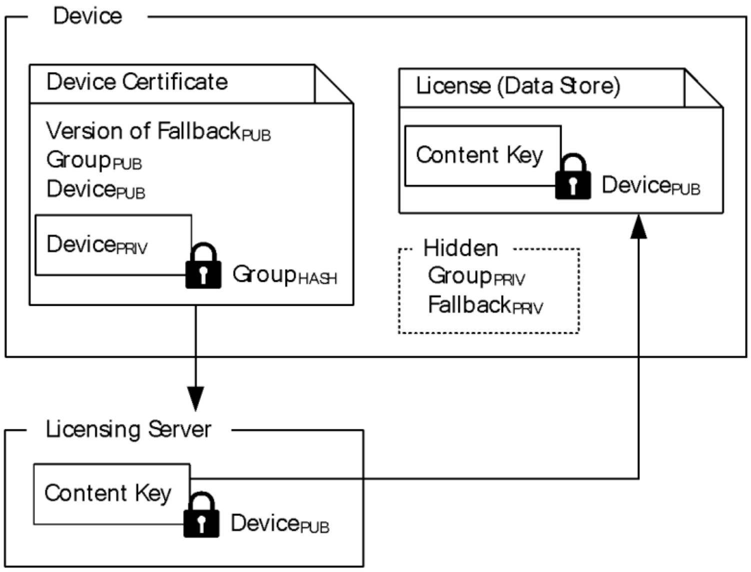

Figure 2: Using the device key pair to encrypt a license

3.2.5.4 Direct License Acquisition

For devices that acquire licenses directly over the internet, changes have been made to improve playback performance without requiring changes to the **direct license acquisition (DLA)** protocol as follows:

The device acquires a license from a Media Rights Manager license server using the existing DLA protocol.

1. The device derives its **symmetric key** from its private key using the **SHA-1** algorithm.
2. The device decrypts the **content key** using its private key.
3. The device re-encrypts the **content key** using its symmetric key.
4. The device regenerates the license hash using SHA-1 and HMAC using its symmetric key.
5. The device stores the license in the license store.

3.2.5.5 Managing the Device's Secure Clock

3.2.5.5.1 Initializing Secure Clock

If a secure clock is unset on a device, or has not been yet been verified via the secure clock protocol, the device MUST attempt to set the clock using the secure clock protocol.

3.2.5.5.2 Issuing a Petition for a Secure Clock Challenge

The client MUST issue a new petition for each new secure clock challenge. The secure clock server MAY deny a secure clock challenge that has not been petitioned. Therefore, it is important that the client does not cache petitions.

The client obtains the petition URL from stored device configuration. In typical DRM-CD implementations, the URL is stored in the device certificate, and is retrieved by parsing the device certificate's **SECURECLOCK** XML node. The petition URL is located in the URL tag under the **SECURECLOCK** XML node.

The client submits a petition request to the petition URL and waits for the server's response. The petition request is a plain HTTP/1.0 GET request, as specified in section 2.2.2.1.

If the petition URL contains "https", the client SHOULD use SSL for the connection. If SSL is used, the client SHOULD check the server's certificate to ensure it is current, matches the domain, and is properly signed by a trusted authority.

The client MUST be prepared to follow HTTP [RFC2616] redirections during the petition. If the secure clock server responds with "HTTP 301 (Moved)" or "302 (Redirect)", the client MUST use this redirect URL as the new petition URL and start again by submitting the petition request to the redirect URL.

If the secure clock server responds with "HTTP 200 (OK)", the client reads the entire body of the response for the secure clock challenge URL. If successful, the client proceeds to the next step, submitting the secure clock challenge.

A valid petition response has the format as specified in section 2.2.2.2.

3.2.5.5.3 Submitting a Secure Clock Challenge

The client MUST then submit the secure clock challenge to the secure clock server. To generate the secure clock challenge, use the **DRM_MGR_ClkGenerateChallenge** function. Then, the client receives a secure clock response message.

3.2.5.5.4 Posting request to the secure clock challenge URL

The client submits a secure clock challenge request to the secure clock challenge URL and waits for the server's response. The secure clock challenge request is an HTTP/1.0 POST request as shown in section 2.2.2.3.

The client obtains the secure clock challenge URL from the petition. There MUST be one petition for each secure clock challenge.

Typically, SSL is not used in this step, but the client MUST be prepared to handle SSL if the secure clock challenge URL uses it. If SSL is used, the client SHOULD check the secure clock server's certificate to ensure it is current, matches the domain, and is properly signed by a trusted authority. Optionally, the client can verify that the certificate belongs to a known Microsoft secure clock server.

3.2.5.5.5 Setting a Secure Clock

A secure clock on a device is one that is set by accessing a web-based secure clock server and cannot be changed by the end user. The secure clock protocol uses a public-private key pair for communication between the server and the device.

On those devices that support a secure clock, the process of setting the clock is as follows:

1. For devices that are not web-enabled, when the device connects to a computer, the media player retrieves the version and device certificate to determine whether the device has a secure clock. Then, the media player requests the device clock status to determine whether the clock needs to be set.

2. For devices that are web-enabled, the device determines whether its clock requires setting.

3. If the device has a secure clock that needs to be set, the device sends a secure clock challenge (directly, or indirectly through the media player on the computer) to the secure clock server.

    1. The secure clock server sends a secure time response, signed with a secure clock private key, to the device, directly or indirectly through the media player on the computer.

    2. The device verifies the signature on the secure time response by using the secure clock public key and sets its clock.

The following diagram shows the keys that are used when setting the device's secure clock.

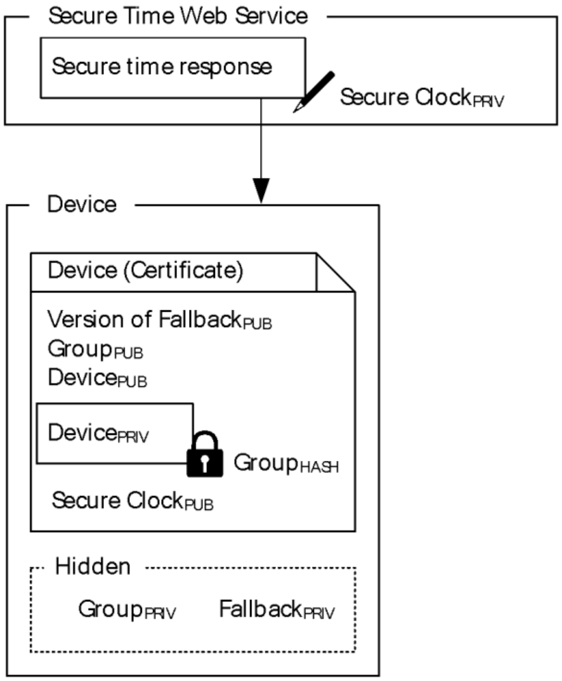

Figure 3: Keys used on the devices secure clock submitting the secure clock challenge

3.2.5.5.6 Create a Secure Clock Challenge

Once a petition URL has been retrieved, the client builds a secure clock challenge message. To compose the message, a randomly generated, unique TID is created, and the petition URL is copied into the message. The message type is set to "challenge".

Finally, the entire message is base64-encoded as documented in [MS-DRM] section 2.2.1.1, and sent to the secure clock server.

3.2.5.5.7 Posting a Secure Clock Challenge

The client posts a secure clock challenge as specified in section 2.2.2.3.

The client MUST be prepared to follow HTTP [RFC2616] redirections during the petition. If the secure clock server responds with "HTTP 301 (Moved)" or "302 (Redirect)", the client MUST use this redirect URL as the new secure clock challenge URL and start again by submitting the secure clock challenge to the redirected URL.

If the secure clock server responds with "HTTP 200 (OK)", the client reads the entire body of the response for the secure clock challenge URL. If successful, the client follows the procedure as documented in the section 3.2.5.5.5.

Upon receiving a secure clock response message, the device first base64-decodes the response as documented in [MS-DRM] section 2.2.1.1, putting the message into its raw XML format.

It then verifies the SIGNATURE element over the **DATA** element using the accompanying CERTIFICATECHAIN. The device MAY also verify that the certificates in the CERTIFICATECHAIN validate to a trusted certificate root for secure clock processing.

The SIGNATURE element MAY subsequently be validated against its own configuration data, which contains the public key of its configured secure clock server. In typical Windows Media Digital Rights Management (WMDRM): MTP Command Extension implementations, the secure clock server's public key is stored in the device certificate, and is retrieved by parsing the device certificate's **SECURECLOCK** XML node. The public key is located in the PUBLICKEY tag under the **SECURECLOCK** XML node.

Once the **DATA** element has been validated, the TID is validated against the TIDs of secure clock challenges previously issued by the device. If the TID cannot be located, the device MUST stop processing the secure clock response message.

Upon validating the TID, the device SHOULD synchronize its internal clock to the value in the **GMTTIME** element. The device SHOULD also store the time value from the **REFRESHDATE** element to indicate the time a new secure clock challenge is sent.

3.2.5.6 Metering on the Device

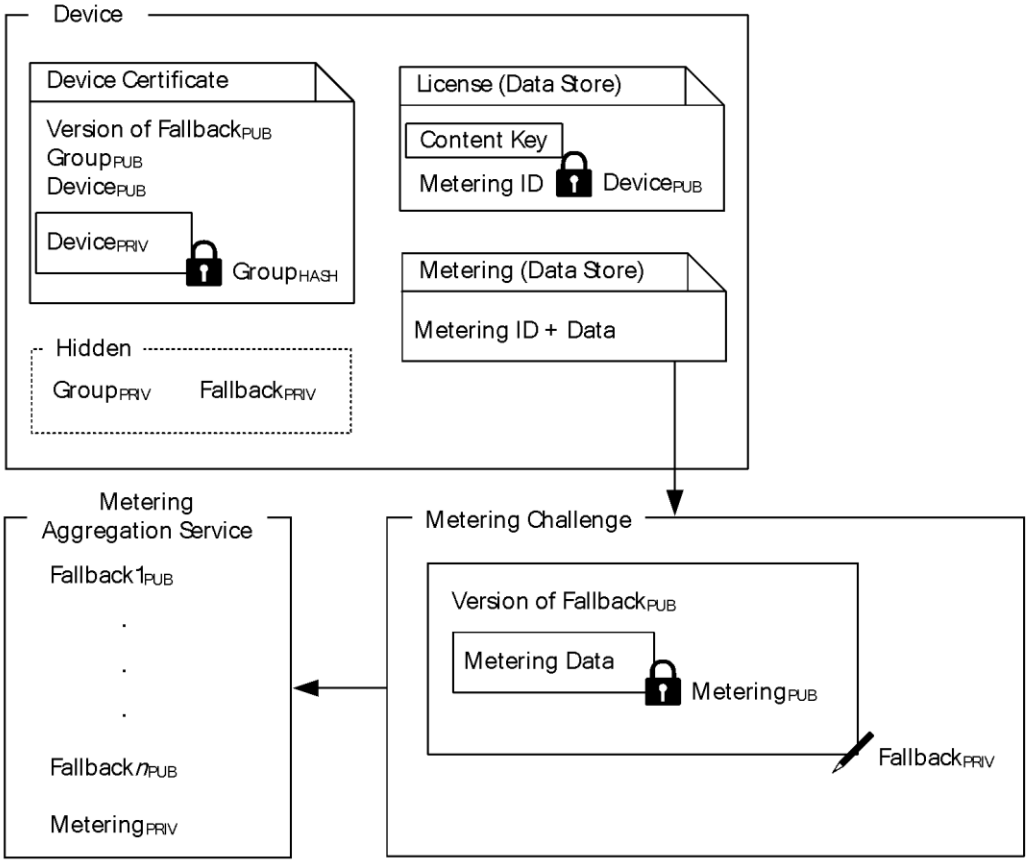

Figure 4: Metering flow with key usage

3.2.5.6.1 Reporting Metering Data

When content requires metering, its license contains a *metering ID*. Each time the content is used, the count and type of action are recorded in the metering section of the data store with the corresponding metering ID. The data must be reported periodically to the metering aggregation server URL. When a device connects to a computer, the metering application or plug-in (provided by the content service) requests metering data, and then sends it to a metering aggregation server.

The process for reporting this metering data is as follows:

1. The metering application or plug-in requests metering data from the device and sends a metering certificate , which contains a metering public key, a metering ID, and a URL for its metering aggregation server.

2. The device collects the metering data for that metering ID from the data store and encrypts it with the metering public key.

3. The device generates a metering challenge containing the fallback public key version and the encrypted metering data, and then signs the challenge with the fallback private key.

4. The metering challenge is sent by the metering application or plug-in to the metering aggregation server.

5. The metering aggregation server verifies the signature on the metering challenge, and then decrypts the metering data using the metering private key.

3.2.5.6.2 GetMeterChallenge Creation

When a device requires a GetMeterChallenge, it extracts the METERCERT from the message, and begins to build a metering challenge message.

The TID element MUST be set to a randomly generated, unique value, base64 encoded as documented in [MS-DRM] section 2.2.1.1.

The MID element MUST be set to the MID value extracted from the METERCERT.

The URL element MUST be set to the URL value extracted from the METERCERT.

The RECORDS element contains encrypted data, base64 encoded as documented in [MS-DRM] section 2.2.1.1 for all content statistics corresponding to the target MID. It is encrypted with the public key of the METERCERT, using the mechanism described in section 2.2.1.6.

The **DATA** element is then signed using the device's private fallback key using the ECC signature creation method (section 2.2.1.6). The generated signature MUST be base64-encoded as documented in [MS-DRM] section 2.2.1.1, and used to populate the MSDRM_SIGNATURE_VALUE. The corresponding device fallback certificate is base64-encoded as documented in [MS-DRM] section 2.2.1.1, and used to populate the CERTIFICATE element. As noted above, the CERTIFICATE element MUST have the private attribute set to "1".

The MESSAGETYPE MUST be set to "challenge".

Finally, the entire metering challenge message is returned to the Metering Plugin over **MTP** [MTP], which forwards it to the MAS using HTTP [RFC2616].

3.2.5.6.3 Meter Response Processing

Once the device receives the metering response message, it MUST validate element data as follows.

- The TID from the response MUST match an unfinished TID from a challenge emitted by the device.
- The MID from the response MUST match the MID in the METERCERT element, and the stored metering certificate originally presented by the indirect license acquisition host over MTP [MTP]. Additionally, MSDRM_SIGNATURE_VALUE element MUST be base64-encoded as documented in [MS-DRM] section 2.2.1.1 and MUST be verified over the entire **DATA** element as described in section 2.2.1.6 using the METERCERT.
- The COMMAND element MUST be set to "RESET".
- The MESSAGETYPE MUST be set to "RESPONSE".

If all of the above conditions are met, the device SHOULD reset content playback statistics using the list of KIDs in the response associated with the MID.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

3.3 Secure Clock Server Details

The secure clock server hosts a secure **time service** on the internet which is reachable via HTTP [RFC2616] and HTTPS [RFC2818]. The device communicates directly with the secure clock server if it is web enabled, or uses an indirect license acquisition host over MTP as described in the document MTP [MTP].

3.3.1 Abstract Data Model

time: The secure clock server keeps track of the current GMT time.

3.3.2 Timers

None.

3.3.3 Initialization

None.

3.3.4 Higher-Layer Triggered Events

None.

3.3.5 Processing Events and Sequencing Rules

3.3.5.1 Cryptographic Requirements

Cryptographic keys MUST be present as specified in section 3.1.1.1.

Secure Clock Key Pair: This key pair is generated on the secure clock server.

Secure Clock Public Key: The secure clock public key can be stored in the device **certificate template** or at any location accessible to the device at runtime. The public key is used to verify the signature on the secure time response.

Secure Clock Private Key: The secure clock private key is used by the secure clock server to sign the secure time response.

3.3.5.2 Processing Secure Clock Challenge

Upon receiving the secure clock challenge message, the server first decodes the message, which was base64-encoded as documented in [MS-DRM] section 2.2.1.1, to its XML format. The server then retrieves the URL and TID from the challenge. It then moves to the next step, creating a secure clock response message.

3.3.5.3 Secure Clock Response Creation

After receiving the secure clock challenge message, the secure clock server begins to compose a secure clock response message.

The secure clock server retrieves GMT time from its authoritative **time source**, and translates it to ZULU time, as described in Date and Time in PD Messages. This is placed in the **GMTTIME** element. A policy configurable time delta is added to the current time, formatted into ZULU time, and placed in the **REFRESHDATE** element of the message.

The TID is copied from the challenge, and placed in the TID element.

A Digital Rights Management signature is computed over the **DATA** element, base64- encoded as documented in [MS-DRM] section 2.2.1.1, and then placed into the SIGNATURE element of the message. The certificate whose private key was used in the signature operation is base64-encoded as documented in [MS-DRM] section 2.2.1.1, placed in a CERTIFICATE element, which is then inserted into the CERTIFICATECHAIN element of the message.

The MESSAGETYPE attribute is set to RESPONSE.

The entire XML message is base64-encoded as documented in [MS-DRM] section 2.2.1.1 for transmission to the client.

The secure clock response is then returned to the device over the chosen protocol. The error is processed as documented in section 3.2.5.5.5.

3.3.5.4 Challenge and Petition Flow

The general process of setting a secure clock is summarized in the following diagram.

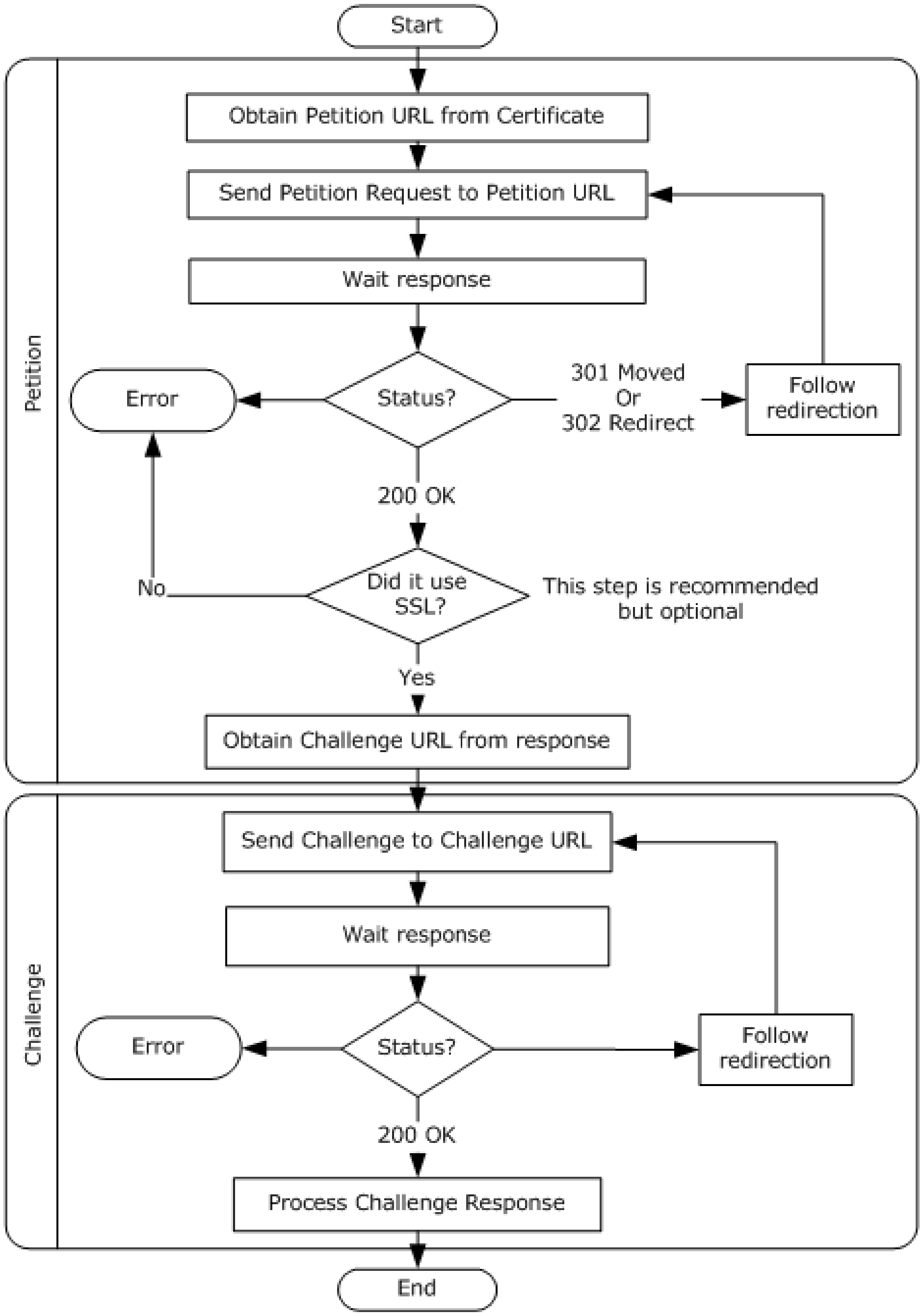
Start
Petition
Obtain Petition URL from Certificate
Send Petition Request to Petition URL
Wait response
Error
Status?
301 Moved
Or
302 Redirect
Follow redirection
200 OK
No
Did it use SSL?
This step is recommended but optional
Yes
Obtain Challenge URL from response
Challenge
Send Challenge to Challenge URL
Wait response
Error
Status?
Follow redirection
200 OK
Process Challenge Response
End

Figure 5: Setting a secure clock

3.3.6 Timer Events

None.

3.3.7 Other Local Events

None.

3.4 Metering Aggregation Server Details

The Metering Aggregation Server (MAS) hosts a metering protocol service on the internet which is reachable via HTTP [RFC2616] and HTTPS [RFC2818]. The device communicates directly with the MAS, if it is web enabled, or uses an indirect license acquisition host over MTP [MTP].

The **license synchronization** server retrieves the license synchronization challenge over MTP [MTP], and uses it to renew expired or invalid licenses. ere this is a requirement. Metering content usage is defined as counting the number of times a content file is used. Metering data is collected over time as a user plays content on the device. Periodically, such as when the user connects the device to a computer, a metering application or plug-in on the computer requests metering data from the device and sends it to a metering aggregation server. If the device is disconnected from the computer for a long time, a large quantity of metering data can accumulate. When this happens, it can take a long time to upload all the metering data from the device. This section contains best practices for original equipment manufacturers (OEMs) describing how to improve the metering experience, performance, and functionality for end users.

The implementer will want to consider the following best practices when implementing metering for portable devices:

1. Use the latest version of the Windows Media DRM for Portable Devices Porting Kit.

2. The metering requests from a computer SHOULD be returned by the device within one minute. If the device cannot return metering data within the time-out period, the metering report fails and the user receives an error message from the metering application or a metering plug-in on the computer. In the event the metering data is not processed, the metering data continues to accumulate and is never cleared from the device.

3. Reduce the size of the metering challenge that is sent in each metering report. When calling the **DRM_MGR_GenerateMeterChallenge** function, pass a value for *pbChallenge* to limit the size of the buffer. The metering challenge data can then be divided into smaller amounts, and will require multiple transactions to complete the report.

4. The optimal size of this buffer depends on the processing power and transfer speed of the device. The buffer SHOULD be small enough that the data can be sent before the time-out limit is reached. The OEM will need to test multiple values, and test a worst-case scenario to find the optimal value.

5. Partial transactions are **flagged** as such when the **DRM_MGR_ProcessMeterResponse** function is called to clear the data store of the reported data (DRM_METER_RESPONSE_PARTIAL is returned in *pfFlagsOut*). The metering application on the computer SHOULD repeat the process until 0 is returned, indicating that the entire process is complete.

The following figure shows a conceptual overview of metering message flow.

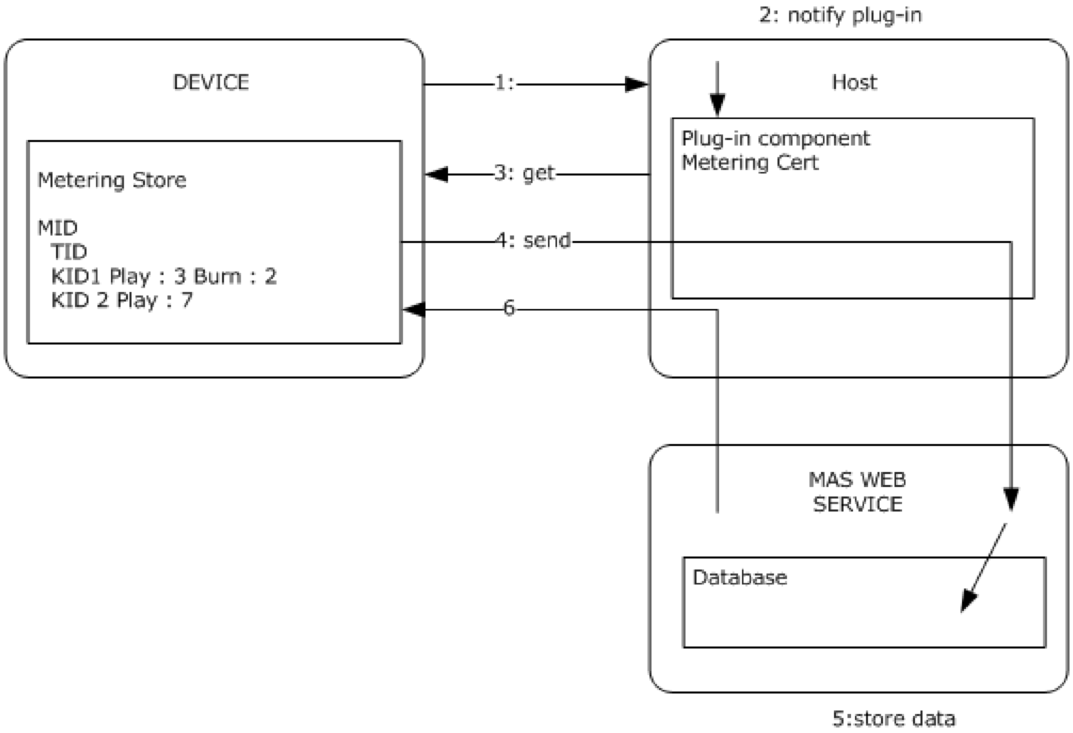

Figure 6: Conceptual metering overview

The metering flow is documented in section 3.4.5.2.

3.4.1 Abstract Data Model

None.

3.4.2 Timers

None.

3.4.3 Initialization

None.

3.4.4 Higher-Layer Triggered Events

None.

3.4.5 Processing Events and Sequencing Rules

3.4.5.1 Cryptographic Requirements

The cryptographic requirements common to all roles in this protocol are defined in section 3.1.1.1 in this document.

3.4.5.2 Metering Aggregation Flow

1. The basic process of metering is as follows. The numbered steps correspond to numbers in Figure 7 below. The metering aggregation server gives the license issuer a metering certificate, which contains a metering ID and a URL indicating where metering data MUST be reported. Then, the license issuer includes the metering certificate in those licenses for the content that requires metering.

2. The device acquires the requested content (when requested by the end-user) and receives a license for the content through a **standard license**-acquisition process.

3. A media player on a computer or device opens this content license. The DRM component of the media player records metering data. The metering data includes a tally of the number of times the content has been used, the type of action performed, and the metering ID. Note that when recording metering data, the media player uses the license key ID to tally the counted actions. Thus, to track metering information for individual content items, each content item MUST be identified with a unique key ID.

4. The metering aggregation server periodically requests metering data for a specific metering ID and then sends the metering data to a corresponding metering aggregation server. If a web-enabled device is used, the device MAY send the metering data directly to the metering aggregation server.

   Note Metering data can only be decrypted by the metering aggregation server that owns the metering certificate containing the metering ID.

5. If a user subscribes to multiple music services and downloads metered content from each service the metering component associated with each service requests metering data using its metering ID. Only the metering data corresponding to that metering ID is gathered, and then sent to the metering aggregation server associated with the metering ID. It is important to note that a transaction ID is created at this time for the items that are reported.

6. After the metering aggregation server receives and processes the metering data, the server returns a response to the media player, prompting the player to clear the metering data that was reported. The transaction ID is saved until a reporting transaction has been completed. If the process of reporting metering data is interrupted, the transaction ID identifies which data to resend, thereby ensuring that a transaction is complete before clearing the data store.

The following is a diagram specifying both the order and direction of the metering information flow:

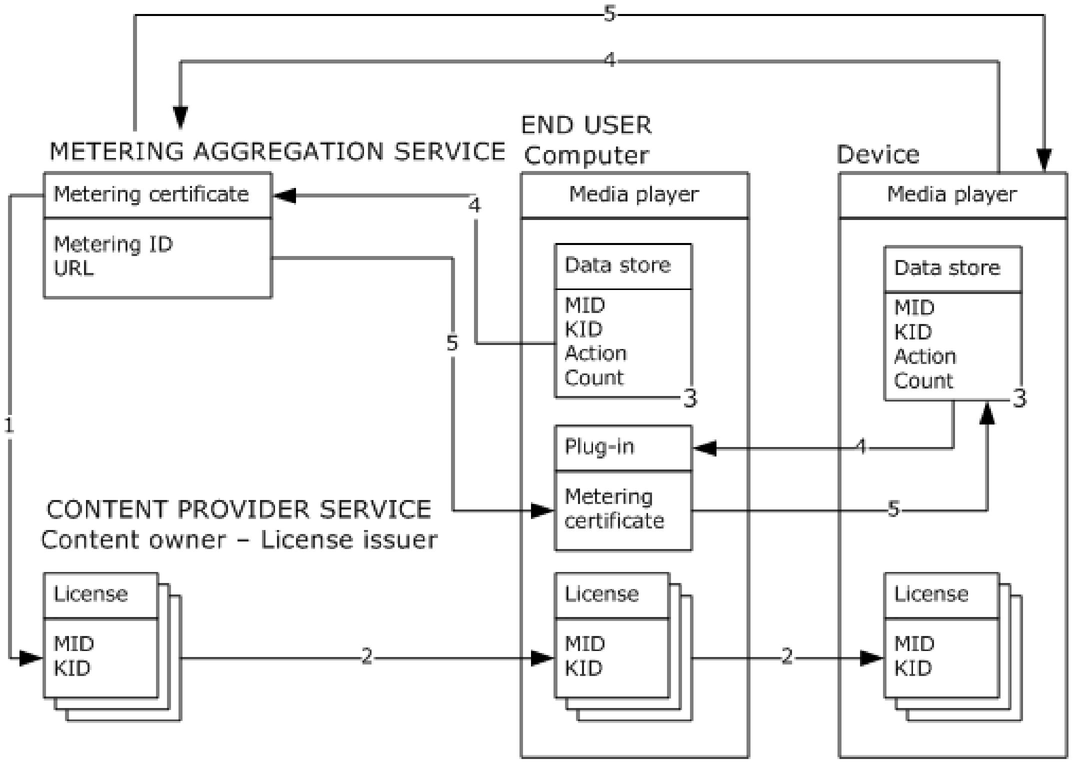

Figure 7: Metering aggregation process flow

3.4.5.3 Meter Challenge Processing

After the MAS receives the metering challenge message, it MUST first verify the signature of the contents of the **DATA** element using the CERTIFICATE element and MSDRM_SIGNATURE_VALUE element. These elements are base64-encoded as documented in [MS-DRM] section 2.2.1.1. The MAS MUST then build a digital signature using the MSDRM ECC signature creation algorithms and process (section 2.2.1.6).

Additionally, the MAS [MSDN-MAS] MUST verify that the CERTIFICATE element chains to a trusted root certificate key using certificate chain validation mechanisms as described in [RFC2459].

The base64-decoded MID in the challenge MUST match the MID in the metering aggregation server's preconfigured metering certificate.

The contents of the RECORDS element are then base64-decoded and decrypted using the MAS's metering certificate's private key using the algorithm described in section 2.2.1.6. The decrypted information SHOULD be stored in the MAS's database for tracking content statistics.

Finally, using data from the challenge, the MAS creates a metering response message, as detailed in section 3.4.5.4.

3.4.5.4 Meter Response Creation

Once the metering challenge message has been verified, the MAS builds a metering response message.

The TID MUST be copied from the challenge, base64-encoded as documented in [MS-DRM] section 2.2.1.1, and placed in the TID element.

The MID MUST be copied from the challenge, base64-encoded as documented in [MS-DRM] section 2.2.1.1, and placed in the MID element.

The COMMAND element MUST be set to "RESET".

The MESSAGETYPE MUST be set to "RESPONSE".

The decrypted and decoded RECORDS element from the metering challenge message MUST be encrypted with the public key from the CERTIFICATE element of the metering challenge message. The encrypted data MUST be base64-encoded as documented in [MS-DRM] section 2.2.1.1 and placed in the RECORDS element of the metering response message.

The MAS's [MSDN-MAS] preconfigured metering certificate is placed in the METERCERT element. The metering certificate's private key is then used to sign the entire **DATA** element as described in section 2.2.1.6. The resultant signature MUST be base64-encoded as documented in [MS-DRM] section 2.2.1.1, and placed into the MSDRM_SIGNATURE_VALUE element.

Finally, the completed metering response message is sent to the device for processing. Device processing is specified in section 3.2.5.6.3.

3.4.6 Timer Events

None.

3.4.7 Other Local Events

None.

3.5 Licensing Server Details

3.5.1 Abstract Data Model

Content key: The **content key** is a symmetric key that is used to encrypt content . This key is included in the content license and is used by the player to decrypt the content. When a license is issued to a device, the **content key** is encrypted with a device public key.

3.5.2 Timers

None.

3.5.3 Initialization

The licensing server MUST have a **content key**. For more information, see section 3.3.1.

3.5.4 Higher-Layer Triggered Events

None.

3.5.5 Processing Events and Sequencing Rules

3.5.5.1 Cryptographic Requirements

Cryptographic keys MUST be present as specified in section 3.1.1.1 Cryptographic Keys.

For more information about how to create the device certificate, see [CR-WMDRM].

3.5.5.2 Evaluate the License for Count and Expiration Criteria

The server MUST compare a **MaxCount** supplied by the caller and a count extracted from the aggregated license data. The server MUST perform processing, based on that comparison, as follows.

1. Check to see if the **MaxCount** policy is equal to **IGNORE**.
    1. If so, then perform a sync. After the sync, no further processing is required.
    2. Otherwise, check to see if the **License Count** is less than or equal to the **MaxCount** supplied by the caller.
        1. If so, no sync is required. No further processing is required.
        2. Otherwise, a sync is required. After the sync, no further processing is required.

The following diagram depicts the processing described in this section.

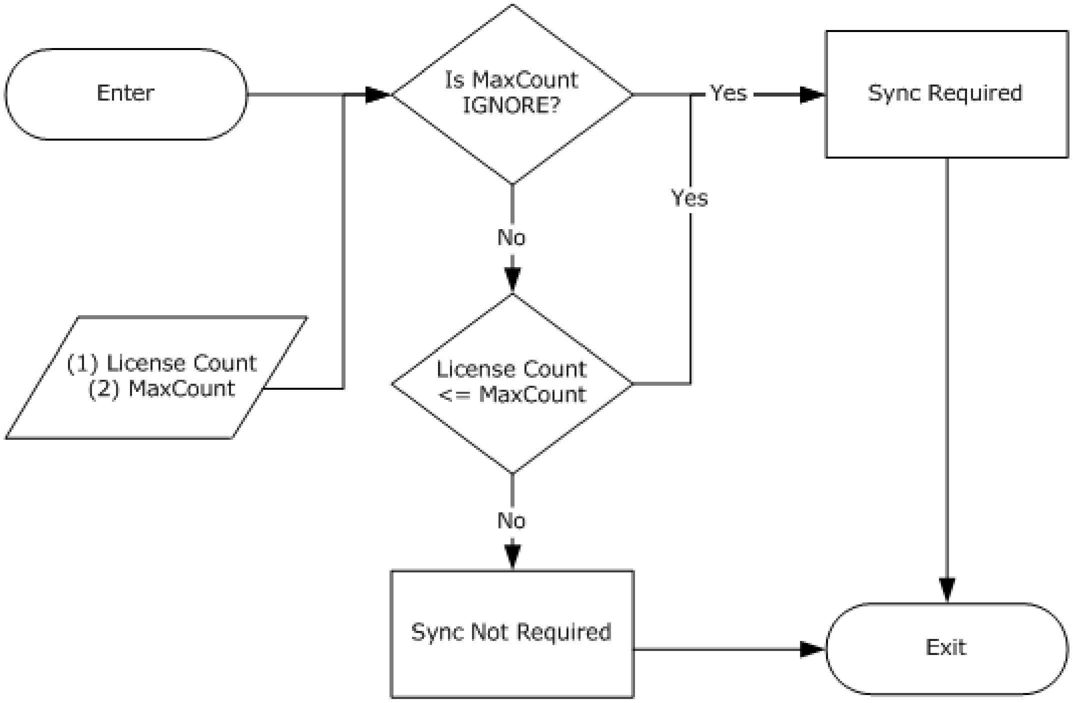

Figure 8: License play count validation

If both synchronization criteria are set to **IGNORE** then all KIDs for all licenses are returned, even for those KIDs that cannot expire. Otherwise the license is evaluated according to the following table.

License state	Criteria for inclusion in synchronization list
WM_DRM_LICENSE_STATE_COUNT	Count Test
WM_DRM_LICENSE_STATE_UNTIL	Hours Test on until-date
WM_DRM_LICENSE_STATE_FROM_UNTIL	Hours Test on from-date Hours Test on until-date
WM_DRM_LICENSE_STATE_COUNT_FROM	Hours Test on from-date Count Test

License state	Criteria for inclusion in synchronization list
WM_DRM_LICENSE_STATE_COUNT_FROM_UNTIL	Hours Test on from-date Count Test Hours Test on until-date
WM_DRM_LICENSE_STATE_COUNT_UNTIL	Hours Test on until-date Count Test
WM_DRM_LICENSE_STATE_EXPIRATION_AFTER_FIRSTUSE	Count Test with 1 as MaxCount
WM_DRM_LICENSE_STATE_NORIGHT	Sync always
WM_DRM_LICENSE_STATE_FROM	Sync Never
WM_DRM_LICENSE_STATE_UNLIM	Sync Never

- The date is extracted from the aggregated license state data.
- The threshold is expressed in hours.

The server MUST perform processing as follows:

1. Check to see if the **MaxHours** policy is set to **IGNORE**.
    1. If so, a sync is required. After the sync, no further processing is required.
2. Otherwise:
    1. Set the threshold time to the current time (**Now**) plus **MaxHours**.
    2. Check if the license date is less than or equal to the threshold time.
        1. If so, a sync is required. After the sync, no further processing is required.
        2. Otherwise, no further processing is required.

The following diagram depicts the processing described in this section.

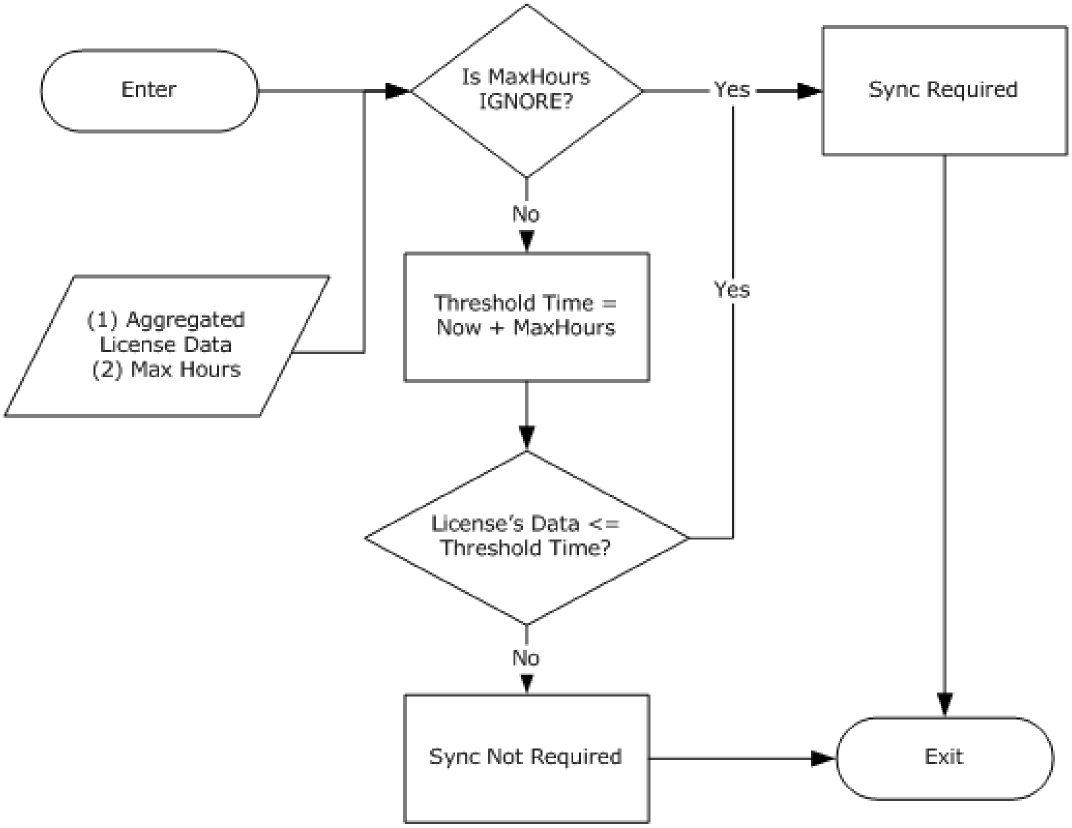
Enter
Is MaxHours IGNORE?
Yes
Sync Required
No
(1) Aggregated License Data
(2) Max Hours
Threshold Time = Now + MaxHours
Yes
License's Data <= Threshold Time?
No
Sync Not Required
Exit

Figure 9: Evaluating the expiration characteristics of a license

3.5.6 Timer Events

None.

3.5.7 Other Local Events

None.

3.6 Indirect License Acquisition Host Details

3.6.1 Abstract Data Model

The DRM component of the player acts as a local license server and issues a license to the device. **Indirect license acquisition** is the process of transferring a license between two local devices. For example, licenses might be transferred between a computer and a cell phone, Smartphone, PDA, or portable media player.

The license is bound to the device certificate, and then the license is placed in a secured storage location on the device. The device can play content as described in section 3.2.

3.6.2 Timers

None.

3.6.3 Initialization

None

3.6.4 Higher-Layer Triggered Events

None.

3.6.5 Processing Events and Sequencing Rules

3.6.5.1 Cryptographic Requirements

Cryptographic keys MUST be present as specified in section 3.1.1.1.

3.6.5.2 Retrieving Metering Data From a Device

When a device is connected to the indirect license acquisition host, a series of Metering components MAY query metering data from the device by using the GetMeterChallenge process, sending a METERCERT corresponding to the MID(s) being serviced by the Metering Plugin.

3.6.5.3 Licensing Acquisition

The following illustrates the **exchange** process of copying DRM licenses between an indirect license acquisition host and device during indirect license acquisition.

1. The following steps are taken during indirect license acquisition:

    1. The indirect license acquisition host requests the device certificate.

    2. The device sends the device certificate, and the indirect license acquisition host validates it by checking against the certificate **revocation list**.

    3. The indirect license acquisition host creates a random session ID and an RC4 **session key**. The indirect license acquisition host encrypts the session key with the device public key from the device certificate.

2. Assuming the AllowCopy right is enabled in the content license:

    1. The indirect license acquisition host verifies the device is capable of receiving the license. For instance, the device supports required features such as metering and enforcement of expiration dates.

    2. The indirect license acquisition host derives the device license—a license that is suitable for the device with similar or a subset of rights.

    3. The indirect license acquisition host encrypts the **content key** using the session key created in step 3.

    4. The indirect license acquisition host creates a hash of the license using SHA-1 and HMAC using the session key.

    5. The indirect license acquisition host builds a LICENSERESPONSE message including the encrypted=false attribute on each LICENSE node in the license response.

    6. The indirect license acquisition host calls the SetLicenseResponse MTP extension on the device. As part of the parameters, it includes the session key, session ID, and the DRM license.

3. The device processes the response to SetLicenseResponse:
    1. The device derives a device symmetric key from the device private key using the SHA-1 algorithm.
    2. From the secure store, the device retrieves the previously stored session ID and encrypted session key (encrypted with the device symmetric key).
4. The device compares the session ID in the secure store and the session ID in the response:
    1. If they match, the device uses the device symmetric key to decrypt the session key retrieved from the secure store.
    2. If they do not match, the device uses the device private key to decrypt the session key from the response, re-encrypts the session key using the device symmetric key, and stores the session ID and re-encrypted session key in the secure store.
    3. The device decrypts the **content key** using the session key from step 1-c.
    4. The device re-encrypts the **content key** using the device symmetric key.
    5. The device regenerates the license hash using SHA-1 and HMAC using the device symmetric key.
    6. The device stores the license in the license store.

3.6.6 Timer Events

None.

3.6.7 Other Local Events

None.

4 Protocol Examples

4.1 Secure Clock Protocol Examples

4.1.1 Obtain a petition URL from the device certificate.

The client obtains the petition URL from the device certificate by parsing the device certificate XML. The petition URL is located in the URL tag under the SECURECLOCK tag.

4.1.2 Submit a petition request to the petition URL

The client submits a petition request to the petition URL and waits for the server's response. The petition request is a plain HTTP/1.0 GET request, as shown by this example:

```
GET PetitionPath HTTP/1.0
User-Agent: Client-User-Agent
```

If the petition URL contains "https", the client can use SSL for the connection. (For non-SSL transport, remove the "s" in "https" from the URL.) If SSL is used, the client could check the server's certificate to ensure it is current, matches the domain, and is properly signed by a trusted authority.

4.1.3 Handle redirections

The client has to be prepared to follow redirections during the petition. If the secure clock server responds with "HTTP 301 (Moved)" or "HTTP 302 (Redirect)", the client can use this redirect URL as the new petition URL and start again by submitting the petition request to it.

Typically a petition has at least one redirection, however, there could be more than one redirection before the client receives the actual secure clock response.

4.1.4 Read the petition response

If the secure clock server responds with "HTTP 200 (OK)", the client reads the entire body of the response for the secure clock challenge URL. If successful, the client proceeds to the next step, submitting the secure clock challenge.

A valid petition response has the following format as defined in section secure clock challenge petition response:

```
HTTP/1.1 302 Found
Server: Microsoft-IIS/5.0
Location: URL
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 191
```

It is recommended that clients ensure that the response was transmitted via an SSL connection, even though SSL is not required.

If SSL is used, it is recommended that the client check the secure clock server's certificate to verify the following.

1. Ensure the response is current.

2. Ensure the response matches the domain.

3. Ensure that the response is properly signed by a trusted authority.

The client can verify that the certificate belongs to a known Microsoft secure clock server.

4.1.5 Secure Clock Challenge Message

The secure clock challenge message is specified in the section 2.2.2.4 DRMCLOCK_CHALLENGE.

1. Get URL from device certificate (see **SECURECLOCK.URL** in device certificate). Get the secure clock public key from device certificate or from a location accessible by the device at runtime.

2. Generate TID, persist, base64 encode.

3. Generate message in XML.

4. Base64 encode entire XML.

5. Await response for 5 minutes, or give up.

6. Process response = verify TID is correct.

7. Verify message signature against the secure clock public key.

8. Set the clock.

```
<DRMCLOCK type="challenge">
  <DATA>
    <URL>http://go.microsoft.com/fwlink/?LinkId=25817</URL>
    <TID>z4TomeJh3H5YfRfOoMRYtg==</TID>
  </DATA>
</DRMCLOCK>
```

4.1.6 Secure Clock Response Message

The secure clock response message is specified in section 2.2.2.5.

1. Accept request from client.

2. Base64 decode to get raw XML.

3. Grab TID, add to **DATA** node XML.

4. Get system time as **ZULU** time string, add to **DATA** node XML.

5. Add in refresh time according to policy, add to **DATA** node XML.

6. Sign contents of data node, as described in section 2.2.1.6.

7. Add base64-encoded secure clock certificate to CERTIFICATE/CERTIFICATECHAIN XML.

8. Base64 encode entire XML.

9. Send to client.

```
<DRMCLOCK type="response">
  <DATA>
    <TID>Fu2bjE77rxKeq54ck8bRPQ==</TID>
    <GMTTIME>#20091299 66:44:31Z#</GMTTIME>
    <REFRESHDATE>#20101199 66:44:31Z#</REFRESHDATE>
  </DATA>
  <CERTIFICATECHAIN>
<CERTIFICATE>AAEAADgAAAC5z9yihdTTDj6gWuC2hovQQiLzAxNdjiupld!06zjVyJCPSjOLwjkI5SmqrnEDjo2H!KJV
4qk2CQ0gMwyurmC1oOVc1uT64vQ!uoHobiFeg2NjDB8AAAABAAAAAQAAAAE=</CERTIFICATE>
  </CERTIFICATECHAIN>
  <SIGNATURE>
    <HASHALGORITHM type="SHA" />
    <SIGNALGORITHM type="MSDRM" />
    <VALUE private="1">QpMlhhZasfEysaNSYHIpM9avLmg2vsFvGS37TpdycX6RhShCAsNhXw==</VALUE>
  </SIGNATURE>
</DRMCLOCK>
```

4.1.6.1 Handle Secure Clock Challenge redirections

Typically, a secure clock challenge is not redirected because any redirections took place during the petition. Nevertheless, if the secure clock server responds to the secure clock challenge request with "HTTP 301 (Moved)" or "302 (Redirect)", the client is required to follow the redirection by using the redirect URL as the new secure clock challenge URL, and then start again by posting the secure clock challenge to it.

4.1.6.2 Processing the Secure Clock Challenge Response

The secure clock response message is specified in section 2.2.2.5.

If the secure clock server responds with "HTTP 200 (OK)", the client reads the entire body of the response, treating it as the secure clock challenge response.

A valid secure clock challenge response has the following format:

```
HTTP/1.1 200 OK
Connection: Keep-Alive
Content-Length: 1620
Expires: Expiration-GMT-Time
Date: Response-GMT-Time
Server: Microsoft-IIS/6.0
Content-Transfer-Encoding: base64
Cache-Control: no-cache; no-transform; no-store; must-revalidate; max-age=0; s-maximage=0
Pragma: no-cache
```

Note that the client does not need to enforce that a secure clock challenge response came through an SSL connection.

4.2 Metering Protocol Examples

These are samples of messages used during a metering protocol session.

4.2.1 Metering Certificate (METERCERT) Example

This is an example of a metering certificate (METERCERT). For details on the construction and use of a metering certificate, see section 2.2.3.1.

```
<METERCERT version="1.0">
  <DATA>
    <MID>pIKx8Wsy5aYRQN8LLIIcTQ==</MID>
    <PUBLICKEY>nm0yQ6CXLcQIBf*Zc*VKGL!SziEdTT!ehGxDotUazwFtE7X40IlqAQ==</PUBLICKEY>
    <URL>http://www.mymeterservice.com</URL>
  </DATA>
  <CERTIFICATECHAIN>
<CERTIFICATE>AAEAADgAAAB17bfg!cqDLB2clHGU7P6zkrOCVUddX95UYkqpjqvoIr4ezqw2zgwfrvKR1dq!EzdGD8ND
2Ihkn0OPEoWZZKCwgidp7Y5SHR!NFJJazdPWfBQEBB4AAAABAAAAAQAAAAE=</CERTIFICATE>
  </CERTIFICATECHAIN>

  <SIGNATURE>
    <HASHALGORITHM type="SHA" />
    <SIGNALGORITHM type="MSDRM" />
    <VALUE>dvVMtj5p9o7I5IK2XIYwkQoF1Rli4cgpS1UkCjBx8KIr7SrPE9!0Vg==</VALUE>
  </SIGNATURE>
</METERCERT>
```

4.2.2 Metering Challenge Message Example

This is an example of a metering challenge message. For details about the construction and usage of a metering challenge message, refer to section 2.2.3.2.

```
<METERDATA type="challenge">
  <DATA>
    <URL>http://www.mymeterservice.com</URL>
    <MID>pIKx8Wsy5aYRQN8LLIIcTQ==</MID>
    <TID>+TWDCx03NDPCcFcMd3D9gA==</TID>

<RECORDS>dsHxczcnNtNumCAoCRTqFiJICxf7JFCMckanxPdCWC3ZutRVL9pyMPeirHdN3v/dAeSHwnarI6o7Sc9PHKqW
L5X/JUKFsIZOf+CJsNcHWAs1XJl/NsUZgsbGSJaQPQ47U9aBJulL0X9npTyX/kxONEqydms/Cl9RfURvPUyFiAtgC7Rd2
PVtUkHRQa+DfbiJwHAB3ItjLeOp3DNlTdA3C0dPoSsQ0hKjrVFOJP8INroPcqDEEDI7VsOHv/6zfxafGXuuDZMT9NY2Xz
d5SAV1vFn/EzPRd+XogB1EKwaZ5UwTOOpxQ0GjY4fJy4UJ5owmhjiuEFSI2yBIXFyuCsNvtRuqioqBjSawW+g/5iT5Eni
0svTQEj4V1ilqCEZjAMyiUPszNwm61kaoLOwbhBxQjgkPtxjKVQ6Zh95/5VGCfBtolnXzWxNpCbVYpEXnk1F1IlKh/4sg
YzOJD7J/tg54pykK5v/3tceHZURe8MiogNK46TqpNmJ4KMstkZo=</RECORDS>
  </DATA>

<MSDRM_SIGNATURE_VALUE>rvkPi7249p2QXdo0bfBYlGgQnlheNpPrUN3uOOxwf0QAzUFWWJnASg==</MSDRM_SIGNAT
URE_VALUE>
  <CERTIFICATE
private="1">2rVLgWBb3FkbJZdM34mOzbhEEiiBnvOFoNv0blUT9/SOMzA/POmNCgIDAAI9Q1iuZc3Tlvwey3ocJyjux
UdaP4rUdoQ5jWH9rhXa537ASm352xJT</CERTIFICATE>
</METERDATA>
```

4.2.3 Metering Response Message Example

This is an example of a metering response message. For details about the construction and usage of a metering response message refer to section 2.2.3.3.

```
<METERDATA type="response">
  <DATA>
    <MID>pIKx8Wsy5aYRQN8LLIIcTQ==</MID>
    <TID>+TWDCx03NDPCcFcMd3D9gA==</TID>

<RECORDS>Q!xmtgAnDChG9b*2N974XCVefiPa0WVFlqITtYMwYrnDlzM9c8jQctzvWDNVYJEyQAEHlG1KAiNHpTEXDy9J
c8XDX0oRCriM0!m!1tU7FGnOUNdhpO39dJNmMvgltRsMgI9yR!LwyadB7NhVIOqFwM0Gdm5Ty7c0tglB5iGciwW0419iw
wHMpgHZH3e0xH3n5qq*bYiCQM5642t5KTCXlGGrCFRm1vU6egCxqdw9trBCoBObeXxuy3vglpLPr1hXuqloIIkR6m5PJ!
J!kcNZLM1ioeIgemMmvoKnz0tVAAuH8TgL*ZnMPJATY3wEiVh3J436p3FRplR0zsJ1PlXU*cIfJuR6QHPO</RECORDS>
    <COMMAND>RESET</COMMAND>
  </DATA>

<MSDRM_SIGNATURE_VALUE>dVlgX!BqfK8HVKXH7168BiqEXGUmqppNBd49Qa5Ij7*PL2f8HOa3Ug==</MSDRM_SIGNAT
URE_VALUE>
  <METERCERT version="1.0">
    <DATA>
      <MID>pIKx8Wsy5aYRQN8LLIIcTQ==</MID>
      <PUBLICKEY>nm0yQ6CXLcQIBf*Zc*VKGL!SziEdTT!ehGxDotUazwFtE7X40IlqAQ==</PUBLICKEY>
      <URL>http://www.mymeterservice.com</URL>
    </DATA>
    <CERTIFICATECHAIN>

<CERTIFICATE>AAEAADgAAAB17bfg!cqDLB2clHGU7P6zkrOCVUddX95UYkqpjqvoIr4ezqw2zgwfrvKR1dq!EzdGD8ND
2Ihkn0OPEoWZZKCwgidp7Y5SHR!NFJJazdPWfBQEBB4AAAABAAAAAQAAAAE=</CERTIFICATE>
    </CERTIFICATECHAIN>
    <SIGNATURE>
      <HASHALGORITHM type="SHA"></HASHALGORITHM>
      <SIGNALGORITHM type="MSDRM"></SIGNALGORITHM>
      <VALUE>dvVMtj5p9o7I5IK2XIYwkQoF1Rli4cgpS1UkCjBx8KIr7SrPE9!0Vg==</VALUE>
    </SIGNATURE>
  </METERCERT>
</METERDATA>
```

4.3 Cryptographic Test Vectors for Algorithms

4.3.1 Target Platform Addressing

The size of a byte in bits is dependent on the addressing of the target platform. The OEM ensures its cryptographic algorithms are isolated from the platform's native byte size by using the declared type DRM_NATIVE_BYTE. The OEM can then define DRM_NATIVE_BYTE to the native byte type of the target platform and use macros to provide further abstraction.

4.4 Message Wire Line Examples

These as samples of real line captures of specific messages used in this protocol. These are included to show how these messages will look if captured as part of a network monitoring session.

4.4.1 SecureClockChallenge

These are hexadecimal representations of messages captured during a secure clock session. For details regarding the construction and usage of the secure clock challenge Message refer to section 2.2.2.

The messages are shown both in plain text XML and as base64-encoded formats. This is an example of a secure clock challenge message. For details about the construction and usage of a secure clock challenge message, refer to section 2.2.2.4.

4.4.1.1 SecureClockChallenge-XML

```
00000000: FF FE 3C 00 44 00 52 00  4D 00 43 00 4C 00 4F 00  ..<.D.R.M.C.L.O.
00000010: 43 00 4B 00 20 00 74 00  79 00 70 00 65 00 3D 00  C.K. .t.y.p.e.=.
00000020: 22 00 63 00 68 00 61 00  6C 00 6C 00 65 00 6E 00  ".c.h.a.l.l.e.n.
00000030: 67 00 65 00 22 00 3E 00  3C 00 44 00 41 00 54 00  g.e.".>.<.D.A.T.
00000040: 41 00 3E 00 3C 00 55 00  52 00 4C 00 3E 00 68 00  A.>.<.U.R.L.>.h.
00000050: 74 00 74 00 70 00 3A 00  2F 00 2F 00 67 00 6F 00  t.t.p.:././.g.o.
00000060: 6D 00 69 00 63 00 72 00  6F 00 73 00 6F 00 66 00  m.i.c.r.o.s.o.f.
00000070: 74 00 63 00 6F 00 6D 00  2F 00 66 00 77 00 6C 00  t.c.o.m./.f.w.l.
00000080: 69 00 6E 00 6B 00 2F 00  3F 00 4C 00 69 00 6E 00  i.n.k./.?.L.i.n.
00000090: 6B 00 49 00 64 00 3D 00  32 00 35 00 38 00 31 00  k.I.d.=.2.5.8.1.
000000A0: 37 00 3C 00 2F 00 55 00  52 00 4C 00 3E 00 3C 00  7.<./.U.R.L.>.<.
000000B0: 54 00 49 00 44 00 3E 00  7A 00 34 00 54 00 6F 00  T.I.D.>.z.4.T.o.
000000C0: 6D 00 65 00 4A 00 68 00  33 00 48 00 35 00 59 00  m.e.J.h.3.H.5.Y.
000000D0: 66 00 52 00 66 00 4F 00  6F 00 4D 00 52 00 59 00  f.R.f.O.o.M.R.Y.
000000E0: 74 00 67 00 3D 00 3D 00  3C 00 2F 00 54 00 49 00  t.g.=.=.<./.T.I.
000000F0: 44 00 3E 00 3C 00 2F 00  44 00 41 00 54 00 41 00  D.>.<./.D.A.T.A.
00000100: 3E 00 3C 00 2F 00 44 00  52 00 4D 00 43 00 4C 00  >.<./.D.R.M.C.L.
00000110: 4F 00 43 00 4B 00 3E 00                           O.C.K.>.
```

4.4.1.2 SecureClockChallenge - Base-64 Encoded

```
00000000: 50 41 42 45 41 46 49 41  54 51 42 44 41 45 77 41  PABEAFIATQBDAEwA
00000010: 54 77 42 44 41 45 73 41  49 41 42 30 41 48 6B 41  TwBDAEsAIAB0AHkA
00000020: 63 41 42 6C 41 44 30 41  49 67 42 6A 41 47 67 41  cABlAD0AIgBjAGgA
00000030: 59 51 42 73 41 47 77 41  5A 51 42 75 41 47 63 41  YQBsAGwAZQBuAGcA
00000040: 5A 51 41 69 41 44 34 41  50 41 42 45 41 45 45 41  ZQAiAD4APABEAEEA
00000050: 56 41 42 42 41 44 34 41  50 41 42 56 41 46 49 41  VABBAD4APABVAFIA
00000060: 54 41 41 2B 41 47 67 41  64 41 42 30 41 48 41 41  TAA+AGgAdAB0AHAA
00000070: 4F 67 41 76 41 43 38 41  5A 77 42 76 41 43 34 41  OgAvAC8AZwBvAC4A
00000080: 62 51 42 70 41 47 4D 41  63 67 42 76 41 48 4D 41  bQBpAGMAcgBvAHMA
00000090: 62 77 42 6D 41 48 51 41  4C 67 42 6A 41 47 38 41  bwBmAHQALgBjAG8A
000000A0: 62 51 41 76 41 47 59 41  64 77 42 73 41 47 6B 41  bQAvAGYAdwBsAGkA
000000B0: 62 67 42 72 41 43 38 41  50 77 42 4D 41 47 6B 41  bgBrAC8APwBMAGkA
000000C0: 62 67 42 72 41 45 6B 41  5A 41 41 39 41 44 49 41  bgBrAEkAZAA9ADIA
000000D0: 4E 51 41 34 41 44 45 41  4E 77 41 38 41 43 38 41  NQA4ADEANwA8AC8A
000000E0: 56 51 42 53 41 45 77 41  50 67 41 38 41 46 51 41  VQBSAEwAPgA8AFQA
000000F0: 53 51 42 45 41 44 34 41  65 67 41 30 41 46 51 41  SQBEAD4AegA0AFQA
00000100: 62 77 42 74 41 47 55 41  53 67 42 6F 41 44 4D 41  bwBtAGUASgBoADMA
00000110: 53 41 41 31 41 46 6B 41  5A 67 42 53 41 47 59 41  SAA1AFkAZgBSAGYA
00000120: 54 77 42 76 41 45 30 41  55 67 42 5A 41 48 51 41  TwBvAE0AUgBZAHQA
00000130: 5A 77 41 39 41 44 30 41  50 41 41 76 41 46 51 41  ZwA9AD0APAAvAFQA
00000140: 53 51 42 45 41 44 34 41  50 41 41 76 41 45 51 41  SQBEAD4APAAvAEQA
00000150: 51 51 42 55 41 45 45 41  50 67 41 38 41 43 38 41  QQBUAEEAPgA8AC8A
00000160: 52 41 42 53 41 45 30 41  51 77 42 4D 41 45 38 41  RABSAE0AQwBMAE8A
00000170: 51 77 42 4C 41 44 34 41                           QwBLAD4A
```

4.4.2 SecureClockResponse

The messages are shown both in plain text XML and as base64-encoded formats.

This is an example of a secure clock response message. For details about the construction and usage of a secure clock response message refer to section 2.2.2.5.

4.4.2.1 SecureClockResponse-XML

```
00000000: FF FE 3C 00 3F 00 78 00  6D 00 6C 00 20 00 76 00  ..<.?.x.m.l. .v.
00000010: 65 00 72 00 73 00 69 00  6F 00 6E 00 3D 00 22 00  e.r.s.i.o.n.=.".
00000020: 31 00 2E 00 30 00 22 00  20 00 3F 00 3E 00 20 00  1...0.". .?.>. .
00000030: 3C 00 44 00 52 00 4D 00  43 00 4C 00 4F 00 43 00  <.D.R.M.C.L.O.C.
00000040: 4B 00 20 00 74 00 79 00  70 00 65 00 3D 00 22 00  K. .t.y.p.e.=.".
00000050: 72 00 65 00 73 00 70 00  6F 00 6E 00 73 00 65 00  r.e.s.p.o.n.s.e.
00000060: 22 00 3E 00 3C 00 44 00  41 00 54 00 41 00 3E 00  ".>.<.D.A.T.A.>.
00000070: 3C 00 54 00 49 00 44 00  3E 00 46 00 75 00 32 00  <.T.I.D.>.F.u.2.
00000080: 62 00 6A 00 45 00 37 00  37 00 72 00 78 00 4B 00  b.j.E.7.7.r.x.K.
00000090: 65 00 71 00 35 00 34 00  63 00 6B 00 38 00 62 00  e.q.5.4.c.k.8.b.
000000A0: 52 00 50 00 51 00 3D 00  3D 00 3C 00 2F 00 54 00  R.P.Q.=.=.<./.T.
000000B0: 49 00 44 00 3E 00 3C 00  47 00 4D 00 54 00 54 00  I.D.>.<.G.M.T.T.
000000C0: 49 00 4D 00 45 00 3E 00  23 00 32 00 30 00 30 00  I.M.E.>.#.2.0.0.
000000D0: 39 00 31 00 32 00 39 00  39 00 20 00 36 00 36 00  9.1.2.9.9. .6.6.
000000E0: 3A 00 34 00 34 00 3A 00  33 00 31 00 5A 00 23 00  :.4.4.:.3.1.Z.#.
000000F0: 3C 00 2F 00 47 00 4D 00  54 00 54 00 49 00 4D 00  <./.G.M.T.T.I.M.
00000100: 45 00 3E 00 3C 00 52 00  45 00 46 00 52 00 45 00  E.>.<.R.E.F.R.E.
00000110: 53 00 48 00 44 00 41 00  54 00 45 00 3E 00 23 00  S.H.D.A.T.E.>.#.
00000120: 32 00 30 00 31 00 30 00  31 00 31 00 39 00 39 00  2.0.1.0.1.1.9.9.
00000130: 20 00 36 00 36 00 3A 00  34 00 34 00 3A 00 33 00   .6.6.:.4.4.:.3.
00000140: 31 00 5A 00 23 00 3C 00  2F 00 52 00 45 00 46 00  1.Z.#.<./.R.E.F.
00000150: 52 00 45 00 53 00 48 00  44 00 41 00 54 00 45 00  R.E.S.H.D.A.T.E.
00000160: 3E 00 3C 00 2F 00 44 00  41 00 54 00 41 00 3E 00  >.<./.D.A.T.A.>.
00000170: 3C 00 43 00 45 00 52 00  54 00 49 00 46 00 49 00  <.C.E.R.T.I.F.I.
00000180: 43 00 41 00 54 00 45 00  43 00 48 00 41 00 49 00  C.A.T.E.C.H.A.I.
00000190: 4E 00 3E 00 3C 00 43 00  45 00 52 00 54 00 49 00  N.>.<.C.E.R.T.I.
000001A0: 46 00 49 00 43 00 41 00  54 00 45 00 3E 00 41 00  F.I.C.A.T.E.>.A.
000001B0: 41 00 45 00 41 00 41 00  44 00 67 00 41 00 41 00  A.E.A.A.D.g.A.A.
000001C0: 41 00 43 00 35 00 7A 00  39 00 79 00 69 00 68 00  A.C.5.z.9.y.i.h.
000001D0: 64 00 54 00 54 00 44 00  6A 00 36 00 67 00 57 00  d.T.T.D.j.6.g.W.
```

```
000001E0: 75 00 43 00 32 00 68 00  6F 00 76 00 51 00 51 00  u.C.2.h.o.v.Q.Q.
000001F0: 69 00 4C 00 7A 00 41 00  78 00 4E 00 64 00 6A 00  i.L.z.A.x.N.d.j.
00000200: 69 00 75 00 70 00 6C 00  64 00 21 00 30 00 36 00  i.u.p.l.d.!.0.6.
00000210: 7A 00 6A 00 56 00 79 00  4A 00 43 00 50 00 53 00  z.j.V.y.J.C.P.S.
00000220: 6A 00 4F 00 4C 00 77 00  6A 00 6B 00 49 00 35 00  j.O.L.w.j.k.I.5.
00000230: 53 00 6D 00 71 00 72 00  6E 00 45 00 44 00 6A 00  S.m.q.r.n.E.D.j.
00000240: 6F 00 32 00 48 00 21 00  4B 00 4A 00 56 00 34 00  o.2.H.!.K.J.V.4.
00000250: 71 00 6B 00 32 00 43 00  51 00 30 00 67 00 4D 00  q.k.2.C.Q.0.g.M.
00000260: 77 00 79 00 75 00 72 00  6D 00 43 00 31 00 6F 00  w.y.u.r.m.C.1.o.
00000270: 4F 00 56 00 63 00 31 00  75 00 54 00 36 00 34 00  O.V.c.1.u.T.6.4.
00000280: 76 00 51 00 21 00 75 00  6F 00 48 00 6F 00 62 00  v.Q.!.u.o.H.o.b.
00000290: 69 00 46 00 65 00 67 00  32 00 4E 00 6A 00 44 00  i.F.e.g.2.N.j.D.
000002A0: 42 00 38 00 41 00 41 00  41 00 41 00 42 00 41 00  B.8.A.A.A.A.B.A.
000002B0: 41 00 41 00 41 00 41 00  51 00 41 00 41 00 41 00  A.A.A.A.Q.A.A.A.
000002C0: 41 00 45 00 3D 00 3C 00  2F 00 43 00 45 00 52 00  A.E.=.<./.C.E.R.
000002D0: 54 00 49 00 46 00 49 00  43 00 41 00 54 00 45 00  T.I.F.I.C.A.T.E.
000002E0: 3E 00 3C 00 2F 00 43 00  45 00 52 00 54 00 49 00  >.<./.C.E.R.T.I.
000002F0: 46 00 49 00 43 00 41 00  54 00 45 00 43 00 48 00  F.I.C.A.T.E.C.H.
00000300: 41 00 49 00 4E 00 3E 00  3C 00 53 00 49 00 47 00  A.I.N.>.<.S.I.G.
00000310: 4E 00 41 00 54 00 55 00  52 00 45 00 3E 00 3C 00  N.A.T.U.R.E.>.<.
00000320: 48 00 41 00 53 00 48 00  41 00 4C 00 47 00 4F 00  H.A.S.H.A.L.G.O.
00000330: 52 00 49 00 54 00 48 00  4D 00 20 00 74 00 79 00  R.I.T.H.M. .t.y.
00000340: 70 00 65 00 3D 00 22 00  53 00 48 00 41 00 22 00  p.e.=.".S.H.A.".
00000350: 3E 00 3C 00 2F 00 48 00  41 00 53 00 48 00 41 00  >.<./.H.A.S.H.A.
00000360: 4C 00 47 00 4F 00 52 00  49 00 54 00 48 00 4D 00  L.G.O.R.I.T.H.M.
00000370: 3E 00 3C 00 53 00 49 00  47 00 4E 00 41 00 4C 00  >.<.S.I.G.N.A.L.
00000380: 47 00 4F 00 52 00 49 00  54 00 48 00 4D 00 20 00  G.O.R.I.T.H.M. .
00000390: 74 00 79 00 70 00 65 00  3D 00 22 00 4D 00 53 00  t.y.p.e.=.".M.S.
000003A0: 44 00 52 00 4D 00 22 00  3E 00 3C 00 2F 00 53 00  D.R.M.".>.<./.S.
000003B0: 49 00 47 00 4E 00 41 00  4C 00 47 00 4F 00 52 00  I.G.N.A.L.G.O.R.
000003C0: 49 00 54 00 48 00 4D 00  3E 00 3C 00 56 00 41 00  I.T.H.M.>.<.V.A.
000003D0: 4C 00 55 00 45 00 20 00  70 00 72 00 69 00 76 00  L.U.E. .p.r.i.v.
000003E0: 61 00 74 00 65 00 3D 00  22 00 31 00 22 00 3E 00  a.t.e.=.".1.".>.
000003F0: 51 00 70 00 4D 00 6C 00  68 00 68 00 5A 00 61 00  Q.p.M.l.h.h.Z.a.
00000400: 73 00 66 00 45 00 79 00  73 00 61 00 4E 00 53 00  s.f.E.y.s.a.N.S.
00000410: 59 00 48 00 49 00 70 00  4D 00 39 00 61 00 76 00  Y.H.I.p.M.9.a.v.
00000420: 4C 00 6D 00 67 00 32 00  76 00 73 00 46 00 76 00  L.m.g.2.v.s.F.v.
00000430: 47 00 53 00 33 00 37 00  54 00 70 00 64 00 79 00  G.S.3.7.T.p.d.y.
00000440: 63 00 58 00 36 00 52 00  68 00 53 00 68 00 43 00  c.X.6.R.h.S.h.C.
00000450: 41 00 73 00 4E 00 68 00  58 00 77 00 3D 00 3D 00  A.s.N.h.X.w.=.=.
00000460: 3C 00 2F 00 56 00 41 00  4C 00 55 00 45 00 3E 00  <./.V.A.L.U.E.>.
00000470: 3C 00 2F 00 53 00 49 00  47 00 4E 00 41 00 54 00  <./.S.I.G.N.A.T.
00000480: 55 00 52 00 45 00 3E 00  3C 00 2F 00 44 00 52 00  U.R.E.>.<./.D.R.
00000490: 4D 00 43 00 4C 00 4F 00  43 00 4B 00 3E 00        M.C.L.O.C.K.>.
```

4.4.2.2 SecureClockResponse- Base64 Encoded

```
00000000: 50 41 42 45 41 46 49 41  54 51 42 44 41 45 77 41  PABEAFIATQBDAEwA
00000010: 54 77 42 44 41 45 73 41  49 41 42 30 41 48 6B 41  TwBDAEsAIAB0AHkA
00000020: 63 41 42 6C 41 44 30 41  49 67 42 79 41 47 55 41  cABlAD0AIgByAGUA
00000030: 63 77 42 77 41 47 38 41  62 67 42 7A 41 47 55 41  cwBwAG8AbgBzAGUA
00000040: 49 67 41 2B 41 44 77 41  52 41 42 42 41 46 51 41  IgA+ADwARABBAFQA
00000050: 51 51 41 2B 41 44 77 41  56 41 42 4A 41 45 51 41  QQA+ADwAVABJAEQA
00000060: 50 67 42 47 41 48 55 41  4D 67 42 69 41 47 6F 41  PgBGAHUAMgBiAGoA
00000070: 52 51 41 33 41 44 63 41  63 67 42 34 41 45 73 41  RQA3ADcAcgB4AEsA
00000080: 5A 51 42 78 41 44 55 41  4E 41 42 6A 41 47 73 41  ZQBxADUANABjAGsA
00000090: 4F 41 42 69 41 46 49 41  55 41 42 52 41 44 30 41  OABiAFIAUABRAD0A
000000A0: 50 51 41 38 41 43 38 41  56 41 42 4A 41 45 51 41  PQA8AC8AVABJAEQA
000000B0: 50 67 41 38 41 45 63 41  54 51 42 55 41 46 51 41  PgA8AEcATQBUAFQA
000000C0: 53 51 42 4E 41 45 55 41  50 67 41 6A 41 44 49 41  SQBNAEUAPgAjADIA
000000D0: 4D 41 41 77 41 44 6B 41  4D 51 41 79 41 44 6B 41  MAAwADkAMQAyADkA
000000E0: 4F 51 41 67 41 44 59 41  4E 67 41 36 41 44 51 41  OQAgADYANgA6ADQA
000000F0: 4E 41 41 36 41 44 4D 41  4D 51 42 61 41 43 4D 41  NAA6ADMAMQBaACMA
00000100: 50 41 41 76 41 45 63 41  54 51 42 55 41 46 51 41  PAAvAEcATQBUAFQA
00000110: 53 51 42 4E 41 45 55 41  50 67 41 38 41 46 49 41  SQBNAEUAPgA8AFIA
00000120: 52 51 42 47 41 46 49 41  52 51 42 54 41 45 67 41  RQBGAFIARQBTAEgA
00000130: 52 41 42 42 41 46 51 41  52 51 41 2B 41 43 4D 41  RABBAFQARQA+ACMA
```

```
00000140: 4D 67 41 77 41 44 45 41  4D 41 41 78 41 44 45 41  MgAwADEAMAAxADEA
00000150: 4F 51 41 35 41 43 41 41  4E 67 41 32 41 44 6F 41  OQA5ACAANgA2ADoA
00000160: 4E 41 41 30 41 44 6F 41  4D 77 41 78 41 46 6F 41  NAA0ADoAMwAxAFoA
00000170: 49 77 41 38 41 43 38 41  55 67 42 46 41 45 59 41  IwA8AC8AUgBFAEYA
00000180: 55 67 42 46 41 46 4D 41  53 41 42 45 41 45 45 41  UgBFAFMASABEAEEA
00000190: 56 41 42 46 41 44 34 41  50 41 41 76 41 45 51 41  VABFAD4APAAvAEQA
000001A0: 51 51 42 55 41 45 45 41  50 67 41 38 41 45 4D 41  QQBUAEEAPgA8AEMA
000001B0: 52 51 42 53 41 46 51 41  53 51 42 47 41 45 6B 41  RQBSAFQASQBGAEkA
000001C0: 51 77 42 42 41 46 51 41  52 51 42 44 41 45 67 41  QwBBAFQARQBDAEgA
000001D0: 51 51 42 4A 41 45 34 41  50 67 41 38 41 45 4D 41  QQBJAE4APgA8AEMA
000001E0: 52 51 42 53 41 46 51 41  53 51 42 47 41 45 6B 41  RQBSAFQASQBGAEkA
000001F0: 51 77 42 42 41 46 51 41  52 51 41 2B 41 45 45 41  QwBBAFQARQA+AEEA
00000200: 51 51 42 46 41 45 45 41  51 51 42 45 41 47 63 41  QQBFAEEAQQBEAGcA
00000210: 51 51 42 42 41 45 45 41  51 77 41 31 41 48 6F 41  QQBBAEEAQwA1AHoA
00000220: 4F 51 42 35 41 47 6B 41  61 41 42 6B 41 46 51 41  OQB5AGkAaABkAFQA
00000230: 56 41 42 45 41 47 6F 41  4E 67 42 6E 41 46 63 41  VABEAGoANgBnAFcA
00000240: 64 51 42 44 41 44 49 41  61 41 42 76 41 48 59 41  dQBDADIAaABvAHYA
00000250: 55 51 42 52 41 47 6B 41  54 41 42 36 41 45 45 41  UQBRAGkATAB6AEEA
00000260: 65 41 42 4F 41 47 51 41  61 67 42 70 41 48 55 41  eABOAGQAagBpAHUA
00000270: 63 41 42 73 41 47 51 41  49 51 41 77 41 44 59 41  cABsAGQAIQAwADYA
00000280: 65 67 42 71 41 46 59 41  65 51 42 4B 41 45 4D 41  egBqAFYAeQBKAEMA
00000290: 55 41 42 54 41 47 6F 41  54 77 42 4D 41 48 63 41  UABTAGoATwBMAHcA
000002A0: 61 67 42 72 41 45 6B 41  4E 51 42 54 41 47 30 41  agBrAEkANQBTAG0A
000002B0: 63 51 42 79 41 47 34 41  52 51 42 45 41 47 6F 41  cQByAG4ARQBEAGoA
000002C0: 62 77 41 79 41 45 67 41  49 51 42 4C 41 45 6F 41  bwAyAEgAIQBLAEoA
000002D0: 56 67 41 30 41 48 45 41  61 77 41 79 41 45 4D 41  VgA0AHEAawAyAEMA
000002E0: 55 51 41 77 41 47 63 41  54 51 42 33 41 48 6B 41  UQAwAGcATQB3AHkA
000002F0: 64 51 42 79 41 47 30 41  51 77 41 78 41 47 38 41  dQByAG0AQwAxAG8A
00000300: 54 77 42 57 41 47 4D 41  4D 51 42 31 41 46 51 41  TwBWAGMAMQB1AFQA
00000310: 4E 67 41 30 41 48 59 41  55 51 41 68 41 48 55 41  NgA0AHYAUQAhAHUA
00000320: 62 77 42 49 41 47 38 41  59 67 42 70 41 45 59 41  bwBIAG8AYgBpAEYA
00000330: 5A 51 42 6E 41 44 49 41  54 67 42 71 41 45 51 41  ZQBnADIATgBqAEQA
00000340: 51 67 41 34 41 45 45 41  51 51 42 42 41 45 45 41  QgA4AEEAQQBBAEEA
00000350: 51 67 42 42 41 45 45 41  51 51 42 42 41 45 45 41  QgBBAEEAQQBBAEEA
00000360: 55 51 42 42 41 45 45 41  51 51 42 42 41 45 55 41  UQBBAEEAQQBBAEUA
00000370: 50 51 41 38 41 43 38 41  51 77 42 46 41 46 49 41  PQA8AC8AQwBFAFIA
00000380: 56 41 42 4A 41 45 59 41  53 51 42 44 41 45 45 41  VABJAEYASQBDAEEA
00000390: 56 41 42 46 41 44 34 41  50 41 41 76 41 45 4D 41  VABFAD4APAAvAEMA
000003A0: 52 51 42 53 41 46 51 41  53 51 42 47 41 45 6B 41  RQBSAFQASQBGAEkA
000003B0: 51 77 42 42 41 46 51 41  52 51 42 44 41 45 67 41  QwBBAFQARQBDAEgA
000003C0: 51 51 42 4A 41 45 34 41  50 67 41 38 41 46 4D 41  QQBJAE4APgA8AFMA
000003D0: 53 51 42 48 41 45 34 41  51 51 42 55 41 46 55 41  SQBHAE4AQQBUAFUA
000003E0: 55 67 42 46 41 44 34 41  50 41 42 49 41 45 45 41  UgBFAD4APABIAEEA
000003F0: 55 77 42 49 41 45 45 41  54 41 42 48 41 45 38 41  UwBIAEEATABHAE8A
00000400: 55 67 42 4A 41 46 51 41  53 41 42 4E 41 43 41 41  UgBJAFQASABNACAA
00000410: 64 41 42 35 41 48 41 41  5A 51 41 39 41 43 49 41  dAB5AHAAZQA9ACIA
00000420: 55 77 42 49 41 45 45 41  49 67 41 2B 41 44 77 41  UwBIAEEAIgA+ADwA
00000430: 4C 77 42 49 41 45 45 41  55 77 42 49 41 45 45 41  LwBIAEEAUwBIAEEA
00000440: 54 41 42 48 41 45 38 41  55 67 42 4A 41 46 51 41  TABHAE8AUgBJAFQA
00000450: 53 41 42 4E 41 44 34 41  50 41 42 54 41 45 6B 41  SABNAD4APABTAEkA
00000460: 52 77 42 4F 41 45 45 41  54 41 42 48 41 45 38 41  RwBOAEEATABHAE8A
00000470: 55 67 42 4A 41 46 51 41  53 41 42 4E 41 43 41 41  UgBJAFQASABNACAA
00000480: 64 41 42 35 41 48 41 41  5A 51 41 39 41 43 49 41  dAB5AHAAZQA9ACIA
00000490: 54 51 42 54 41 45 51 41  55 67 42 4E 41 43 49 41  TQBTAEQAUgBNACIA
000004A0: 50 67 41 38 41 43 38 41  55 77 42 4A 41 45 63 41  PgA8AC8AUwBJAEcA
000004B0: 54 67 42 42 41 45 77 41  52 77 42 50 41 46 49 41  TgBBAEwARwBPAFIA
000004C0: 53 51 42 55 41 45 67 41  54 51 41 2B 41 44 77 41  SQBUAEgATQA+ADwA
000004D0: 56 67 42 42 41 45 77 41  56 51 42 46 41 43 41 41  VgBBAEwAVQBFACAA
000004E0: 63 41 42 79 41 47 6B 41  64 67 42 68 41 48 51 41  cAByAGkAdgBhAHQA
000004F0: 5A 51 41 39 41 43 49 41  4D 51 41 69 41 44 34 41  ZQA9ACIAMQAiAD4A
00000500: 55 51 42 77 41 45 30 41  62 41 42 6F 41 47 67 41  UQBwAE0AbABoAGgA
00000510: 57 67 42 68 41 48 4D 41  5A 67 42 46 41 48 6B 41  WgBhAHMAZgBFAHkA
00000520: 63 77 42 68 41 45 34 41  55 77 42 5A 41 45 67 41  cwBhAE4AUwBZAEgA
00000530: 53 51 42 77 41 45 30 41  4F 51 42 68 41 48 59 41  SQBwAE0AOQBhAHYA
00000540: 54 41 42 74 41 47 63 41  4D 67 42 32 41 48 4D 41  TABtAGcAMgB2AHMA
00000550: 52 67 42 32 41 45 63 41  55 77 41 7A 41 44 63 41  RgB2AEcAUwAzADcA
00000560: 56 41 42 77 41 47 51 41  65 51 42 6A 41 46 67 41  VABwAGQAeQBjAFgA
00000570: 4E 67 42 53 41 47 67 41  55 77 42 6F 41 45 4D 41  NgBSAGgAUwBoAEMA
00000580: 51 51 42 7A 41 45 34 41  61 41 42 59 41 48 63 41  QQBzAE4AaABYAHcA
```

```
00000590: 50 51 41 39 41 44 77 41  4C 77 42 57 41 45 45 41  PQA9ADwALwBWAEEA
000005A0: 54 41 42 56 41 45 55 41  50 67 41 38 41 43 38 41  TABVAEUAPgA8AC8A
000005B0: 55 77 42 4A 41 45 63 41  54 67 42 42 41 46 51 41  UwBJAEcATgBBAFQA
000005C0: 56 51 42 53 41 45 55 41  50 67 41 38 41 43 38 41  VQBSAEUAPgA8AC8A
000005D0: 52 41 42 53 41 45 30 41  51 77 42 4D 41 45 38 41  RABSAE0AQwBMAE8A
000005E0: 51 77 42 4C 41 44 34 41                          QwBLAD4A
```

5 Security

5.1 Security Considerations for Implementers

SHA-1 Hashing: The use of SHA-1 hashing is critical to this protocol. Although SHA-1 has been shown to be vulnerable to collision through that use of specially crafted plaintext, the "one-wayness" of SHA-1 has not been compromised. This protocol relies on the "one-way" characteristic of SHA-1.

ECC Curves: This protocol does not utilize a standard curve for its ECC. The curve utilized by this protocol is fully disclosed in the document [MS-DRM].

Unique device identifiers: It is recommend that every device be provisioned with a unique serial number to prevent device cloning, as specified in section 3.2.1.

Real-time clock: It is recommended that every device implement a real-time clock, as specified in section 3.2.1.

HTTPS Transport: The HTTPS transport is recommended for use when available, as specified in section 2.1.

Secure Storage: The availability of a user opaque storage location on the device is crucial to the security of this protocol. The secure storage is used to hold and protect cryptographic keys and licenses.

Certified Cryptographic Library: The cryptographic library used in conjunction with this protocol is recommended to be FIPS certified and to pass NIST standard cryptographic test vectors to ensure both security and interoperability.

5.2 Index of Security Parameters

Security parameter	Section
Certificates	2.2.3.1
Cryptographic Keys	3.1.1.1
Digital Signatures	2.2.1.6
DRM Common Cryptographic Parameters	[MS-DRM] section 2.2.1.2
Hashing Algorithms	2.2.1.3.3, 2.2.1.4.1
Encryption Algorithms	2.2.1.3.5, 2.2.1.4.3

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs.

- Windows XP operating system Service Pack 1 (SP1)
- Windows Vista operating system
- Windows 7 operating system
- Windows Server 2003 operating system
- Windows Server 2008 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system
- Windows 10 v1511 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 3.2.5: This key pair can be generated by the OEM using the GenerateDacRequest.exe tool available from Microsoft.

<2> Section 3.2.5: This key pair can be generated by the OEM using the GenerateDacRequest.exe tool available from Microsoft.

7 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- The removal of a document from the documentation set.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the technical content of the document is identical to the last released version.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated**.

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
6 Appendix A: Product Behavior	Added Windows 10 v1511 to applicability list.	Y	Content update.

8 Index