MEDIA-S

SYSTEM DESIGN DOCUMENT

Document Date: April 3, 2003
Author: SideSpace Solutions, Inc.
WWW: http://www.sidespace.com/products/medias/

TABLE OF CONTENTS

PURPOSE

This document is intended to outline ways to add Digital Rights Management (DRM) features to multimedia files. The first multimedia file type to be examined is Ogg Vorbis (OGG) audio files and therefore the remainder of this document will deal with Ogg Vorbis media encryption. An overview of the project goals will first be provided, followed by a technical discussion of how DRM can be implemented on Ogg Vorbis files.

This early version of the document will hopefully serve as a springboard for discussion of implementing DRM in Ogg Vorbis, and will evolve depending on feedback from the Ogg Vorbis community.

This document will provide a high-level overview of the Media-S system architecture. Two other documents, the "Media-S End User Guide" and the "Media-S Developer's Guide" will be built from the foundation provided by this document.

OGG VORBIS

Ogg Vorbis describes itself as a "fully Open, non-proprietary, patent-and-royalty-free, general-purpose compressed audio format for mid to high quality audio". OGG is very popular in the open source community and is rapidly gaining acceptance in both mainstream hardware and software audio circles. OGG has been used as the sound engine in several games, and is generally agreed to be superior to other audio formats at low bitrates.

OGG is being used for this DRM implementation because it is an open format, and this project is designed to provide an open DRM solution. Licensing issues make the use of MP3 or WMA audio formats undesirable for an open, standards-based DRM solution.

DIGITAL RIGHTS MANAGEMENT

DRM technology is used to provide users with access to content that the users would otherwise not be able to use. Examples of successful DRM systems in use today are American commercial satellite receivers (e.g. DirecTV), and major-label online music systems such as PressPlay and Rhapsody.

While many users complain that DRM is a poor solution to the complex issues of the fair-use of copyrighted content; to not offer a viable, open DRM solution ignorant of the current state of events.

Media-S

Media-S is designed to be an open, freely available DRM solution for multimedia content. The solution being proposed in this document is a framework based on open standards and meant to be easy to "plug-in" to either existing applications, or easily customized for more specialized applications. While Media-S could be used to protect any type of multimedia file, this implementation will concentrate on the securing of Ogg Vorbis audio files.

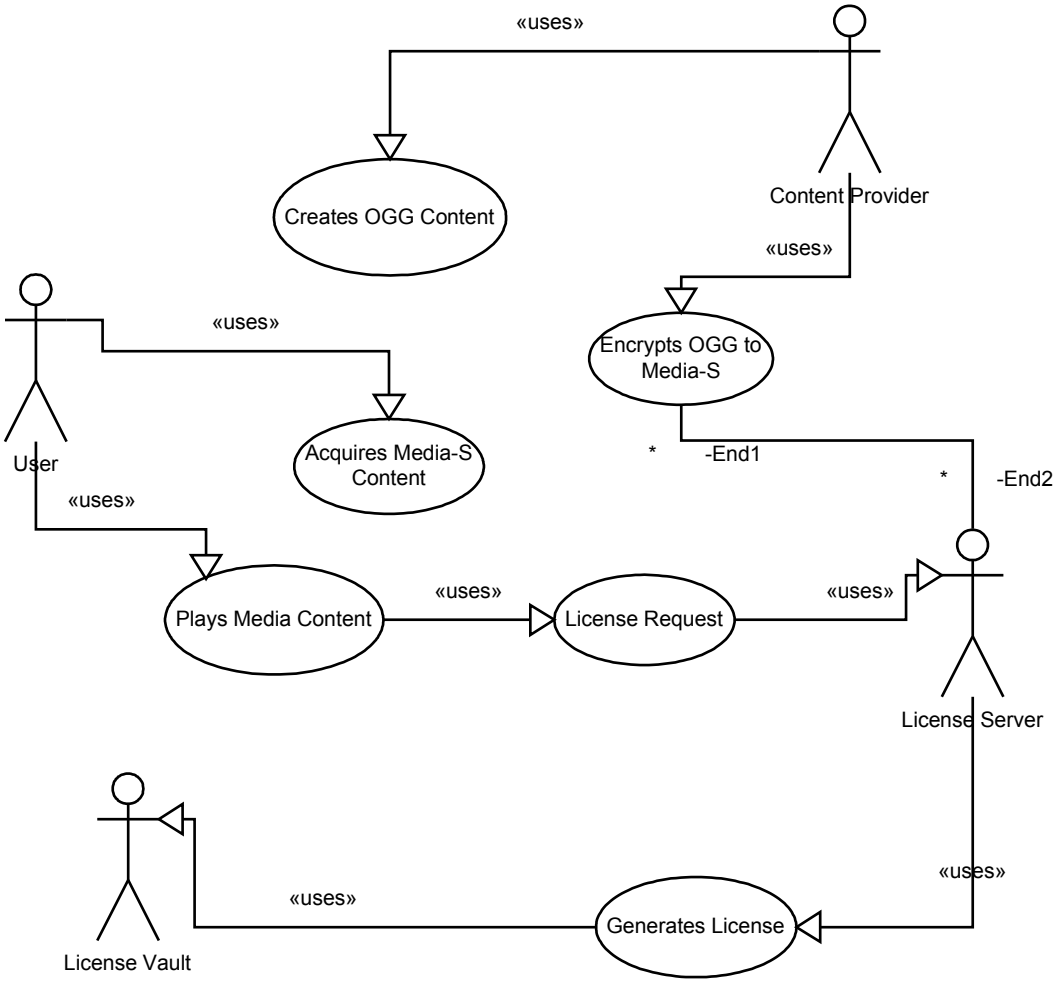
Media-S is dual licensed under the GPL and a binary-only license. Details of these licenses can be found on the Media-S website.

IMPLEMENTATION OVERVIEW

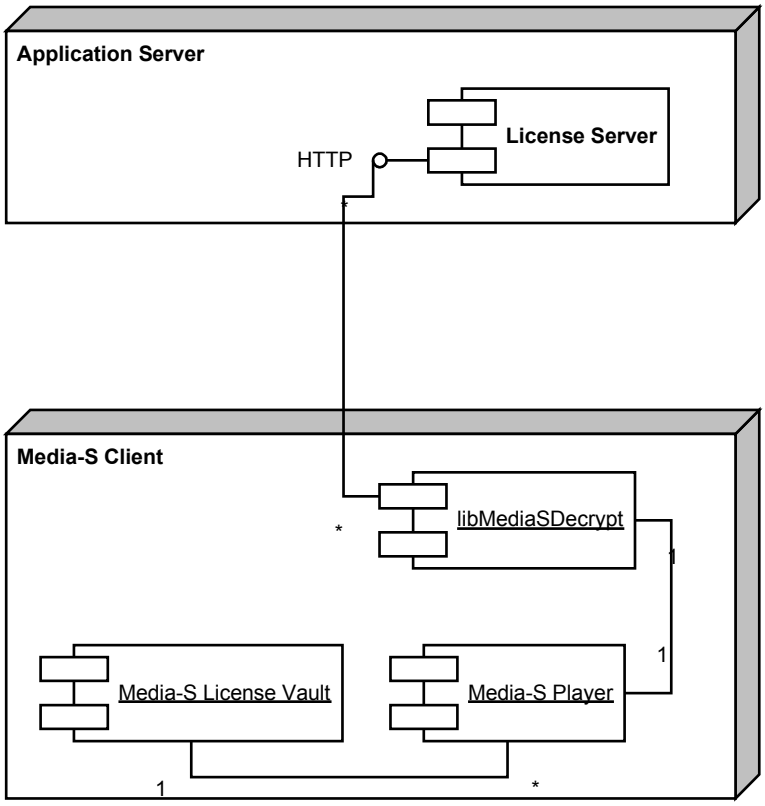
At a high-level, implementing Media-S will require the following system components:

- Encryption program that will take as input a normal OGG file and encrypt the file according to user preferences to create an Media-S file. This program will be used by content providers and will be knows as “oggsenc”.
- Player program that can acquire a content license and decrypt and play Media-S files. This player will be functionally equivalent to ogg123. For this player to work, it will require the use of a “license vault” that will store and manage a user's licenses. This program will be known as “oggs123”.
- License Server that can grant licenses for specific content. License servers can be run by content providers to securely modify remote content licenses once a license has expired.

These components will interact as pictured in the following use case diagram:



A component diagram of the Media-S system is below:



This document will proceed to provide a technological overview of Media-S before documenting each of the above use cases and actors in detail.

TECHNICAL FOUNDATION

The above system components require a development plan that emphasizes security and portability. Thankfully, security and portability are a hallmark of many open-source projects that will be used as key pieces in the Media-S solution. Therefore the following components will be used to create Media-S:

- XML: Licenses and data in the license vault will be stored as encrypted XML
- OpenSSL: The OpenSSL security toolkit will be used for all data encryption routines.
- HTTP: License server communication will be implemented using HTTPS with OpenSSL certificate validation.

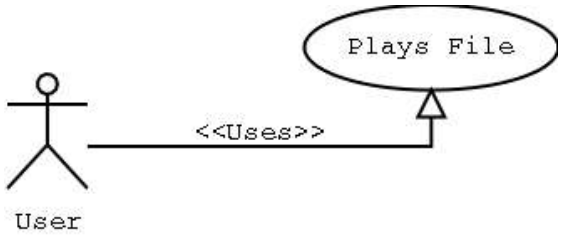
The combination of these technologies provide a portable and industry-accepted security foundation.

USE CASES

The use case diagram presented earlier in this document provided a good overview

of the Media-S system from a user perspective. This section will detail various use cases in more detail.

The first use case to examine is very simple. This use case is just that of a user trying to play Media-S content:



PLAYS FILE USE CASE

Description:

A user wishes to play an Media-S file.

Assumptions:

- The Media-S player clearly conveys its status to the user

Preconditions:

- The user has installed an Media-S player
- The user has acquired Media-S content

Postconditions:

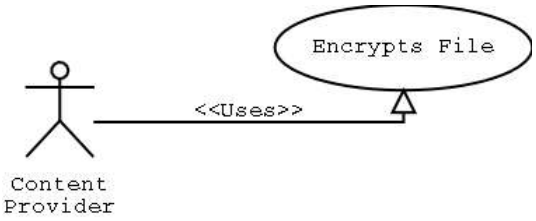
- The user either successfully plays the file, or is given a clear description of any errors that occurred.

Steps:

1. The user invokes the player via a command line (or GUI) to play the Media-S file
2. The Media-S player verifies the user has permission to play the file
3. The Media-S player acquires a license to play the file if necessary, and updates the license vault
4. The player either plays the file, or gives a meaningful error describing the problem

Benefitting Actor:

- User



CREATES Media-S CONTENT

Description:

A content provider would like to create some protected content using Media-S

Assumptions:

- The Media-S encoder conveys its status to the user

Preconditions:

- The content provider has unencrypted Media-S content that they wish to protect
- The content provider has SSL certificates that will be used to sign the content
- The content provider is aware of how they would like to license the content

Postconditions:

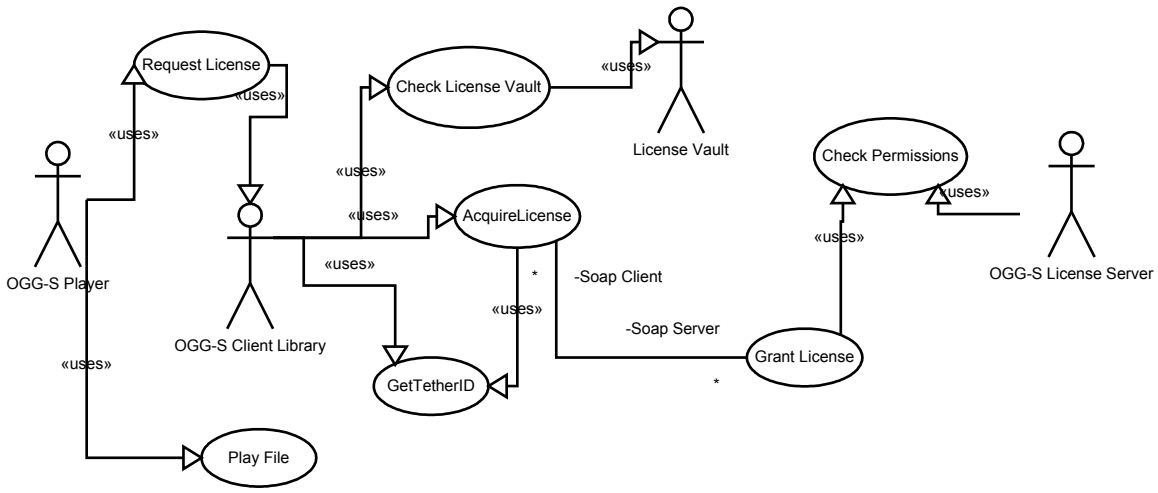
- The content provider creates an Media-S file, or receives an error message explaining why the action failed.

Steps:

1. The content provider invokes the Media-S encoder via a command line or GUI
2. The Media-S encoder asks the content provider what permissions they would like to implement
3. The Media-S encoder encrypts the file

Benefitting Actor:

- Content Provider



REQUEST LICENSE USE CASE

Description:

The Media-S player has to request a license to play an Media-S file

Assumptions:

- The Media-S player uses the standard license vault and encryption libraries
- The Media-S player would like permission to play the Media-S file
- The Media-S player will be responsible for updating the license vault

Preconditions:

- The player is aware of protected content
- The license vault is accessible to the client library
- The client library has permission to remotely acquire a license

Postconditions:

- A license is either granted to the player, or an error code is returned

Steps:

1. The Media-S player invokes the correct Media-S client library for the protected file
2. The Media-S client library looks up the license status for this piece of protected content in the license vault
3. If no valid license exists in the license vault, the client library connects to the license server to attempt and acquire a valid license
4. The status of the license request is conveyed to the player

Benefitting Actor:

- Media-S Player

CHECK LICENSE VAULT USE CASE

Description:

When the Media-S client library receives a request for an action on a protected file, the license vault is first queried to determine what existing licenses exist for the content.

Assumptions:

- The Media-S player uses the standard license vault and encryption libraries
- The operation being queried for is permission to play the protected file

Preconditions:

- The license vault is accessible to the client library

Postconditions:

- A license either exists for the protected content or not

Steps:

1. The Media-S client library verifies the license vault exists
2. The Media-S client library opens the license vault and looks for a key that is the SHA1 sum of the protected file
3. The Media-S client library gets the key to decrypt the key's value
4. The actual license is decrypted into XML
5. The license is checked for play permission, expiration, etc.
6. A return code is sent back to the player by the client library

Benefitting Actor:

- Media-S Player

GET TETHER ID USE CASE

Description:

The Media-S client library has to be capable of creating a unique user and machine-specific tether ID. Creating this ID is a function of the client library so that vendor-specific encryption routines can be used in generating the tether ID. The tether ID is used by the license vault and the license server to uniquely identify the user.

Assumptions:

- The Media-S player uses the standard license vault and encryption libraries
- The tether ID that is generated is a one-way hash, it cannot be decrypted to gather information about a user

Preconditions:

- None

Postconditions:

- A unique string is generated that links a specific user to a specific machine

Steps:

1. The Media-S client library creates a unique string based on the user and machine characteristics

Benefitting Actor:

- Media-S Client Library

ACQUIRE LICENSE USE CASE

Description:

When the Media-S client library cannot fulfill the request for the license from the license vault, it must attempt to acquire a license from a trusted license server.

Assumptions:

- The Media-S player uses the standard license vault and encryption libraries
- The Media-S file's content provider is running an Media-S license server that is protected via HTTPS
- The Media-S file specifies the URL of the license server
- The Media-S file's header can be decrypted by the client library

Preconditions:

- The license vault is accessible to the client library
- The license vault does not contain a valid license for the Media-S file

Postconditions:

- The license vault is either updated to contain a valid license for the Media-S file, or the failure to acquire a license is communicated to the client library

Steps:

1. The Media-S client library decrypts the header of the Media-S file
2. The client library reads the “License Server URL” from the header XML
3. The client library authenticates the server by sending a unique challenge and verifying the expected response
4. A request is sent to the license server with the inputs being the user's unique tether ID and the SHA1 content hash. This request is sent using HTTPS.

5. A response is received from the license server
6. If granted, the resulting license is stored in the user's license vault

Benefitting Actors:

- Media-S Player
- Content Provider

GRANT LICENSE USE CASE

Description:

To easily and automatically grant licenses to clients, the content provider can setup a license server to handle requests. The specific interfaces that this license server has to support are detailed later into this document.
The license server is a simple web application secured using HTTPS. Clients are authenticated via OpenSSL digital certificates.

Assumptions:

- The content provider encodes the license acquisition URL in each piece of content it produces
- The license server is protected via HTTPS
- The web server supports the query interface described later in this document

Preconditions:

- The web server has the ability to grant content licenses
- The Media-S client library has acquired a valid ticket to the license server

Postconditions:

- The license vault is either updated to contain a valid license for the Media-S file, or the failure to acquire a license is communicated to the client library

Steps:

1. The Media-S client library gets the user's TetherID and unique content ID
2. A request is sent to the license server with the inputs being the user's unique tether ID and the SHA1 content hash. This request is sent using HTTPS.
3. The license server determines whether the user has the requested right for this content.
4. A response is received from the license server
5. If granted, the resulting license is stored in the user's license vault

Benefitting Actors:

- Media-S Player
- Content Provider

CHECK PERMISSIONS USE CASE

Description:

The license server is a simple web server secured using HTTPS. Clients are authenticated via OpenSSL certificates.
To generate a license, the server must be able to determine what permissions a user has for a piece of content. The license server will receive as input the user id, content id, and other extra data provided by the client library. The license server must deliver to the client a proper license or a denial based on this information.

Assumptions:

- The license server performs the permission checking
- If a license request is denied, the license server does not have to give a reason for denial.

Preconditions:

- A request has been made to the license server for a license

Postconditions:

- A valid license is returned, or a denial is returned.

Steps:

1. The license server receives a request for a license
2. The license server determines what permissions a user has for a piece of content
3. The license server returns the permission information to the client

Benefitting Actors:

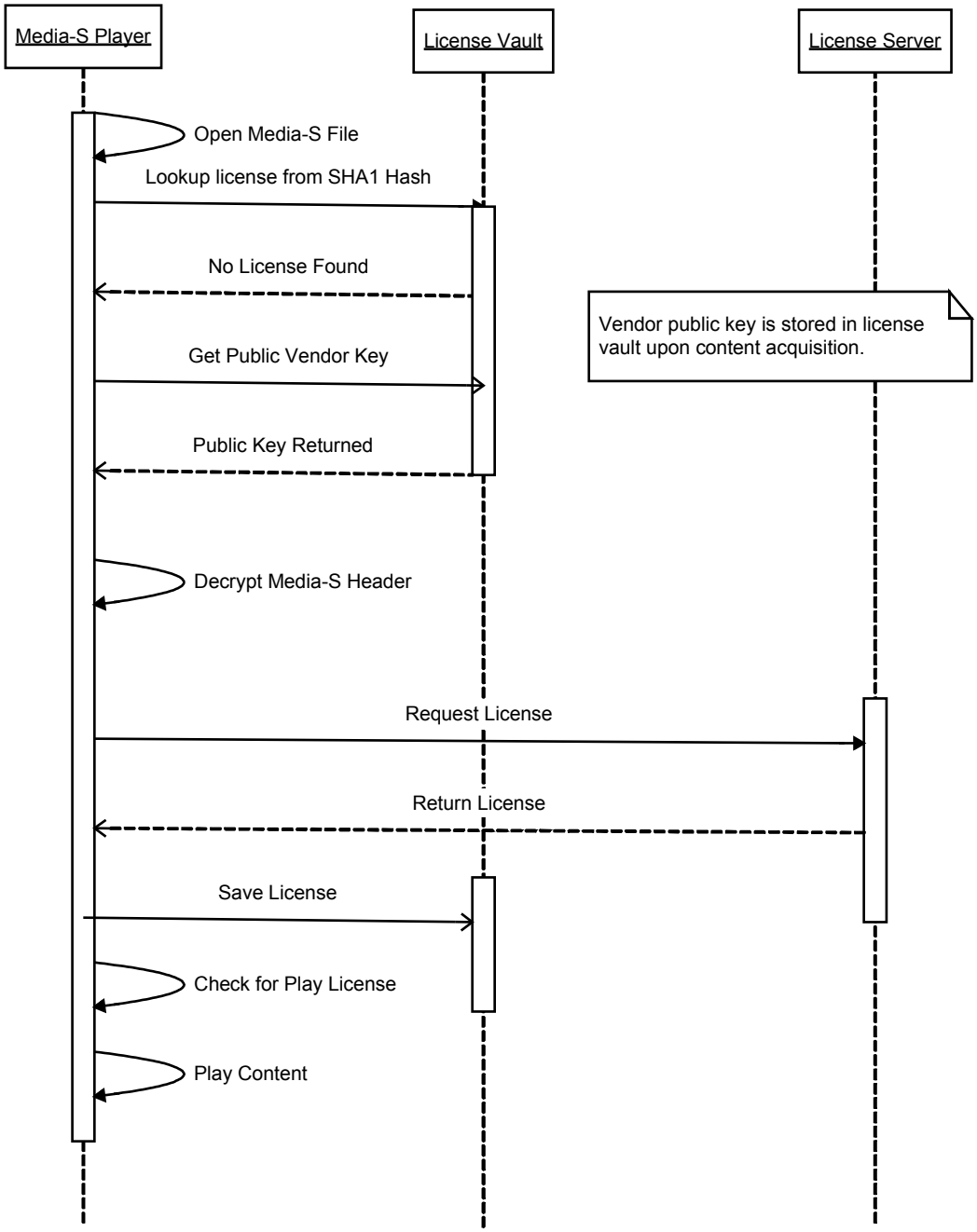
- Media-S Player
- Content Provider

COMPONENT DETAILS

Now that the components of Media-S have been previewed, the implementation details of each component will be examined. From the above overview, there are five key components of the Media-S system:

1. Media-S Player, the actual player that uses the Media-S client library (oggs123)
2. Media-S Client Library, a client library used by an Media-S player that can be customized by a content provider to provide unique encryption and decryption routines
3. Media-S Encoder, which will use the client library to encrypt a file (oggsenc)
4. License Vault, a DBM file that stores and manages user licenses. Its contents are manipulated by the Media-S client library.
5. License Server, an HTTPS server run by content providers who wish to dynamically serve and renew license requests.

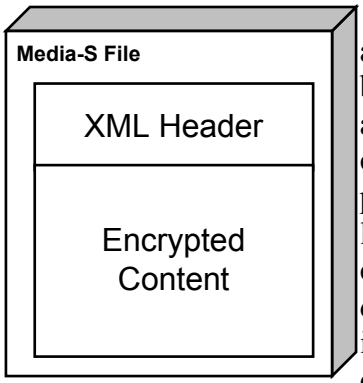
The interaction of these components is described in the following activity diagram:



Media-S FILE FORMAT OVERVIEW

The Media-S file will be an encrypted OGG file with additional header data as shown below. The XML header is encrypting using 3DES encryption. The public key for this encryption is unique per content-provider. Therefore the player has to either be aware of the content provider so it can lookup the public key for the content provider, or be supplied the public key. This content provider and public key data will be stored in

the user's license vault (which is described below).



The XML that is in the header element is detailed in an appendix of this document (although an example is provided below). This XML will contain song meta-data information, as well as a license request URL for the specific piece of content. This license request URL is very important as if the player is unable to retrieve a valid license for the content locally it will use HTTPS to request a valid license for the content. Once a valid license has been requested, it can be downloaded and stored in the license store. In most Media-S implementations it is ideal to give licenses on a per-user per-content basis. To support this type of implementation it is recommended that the Media-S headers be dynamically created each time a file is requested by a user. Using this dynamic addition of license data ensures that licenses can be renewed and revoked on a per-user basis.

The XML in the header will also contain the public key that can be used to decrypt the actual OGG data. The player therefore has to be able to decrypt the header blob in order to play the content.

Here is some sample decrypted XML from the header element using Low security:

```
<XML>
   <MULTIMEDIA TYPE="ogg">
         <HEADER>
            <SONG>
                  <title>Here We Are</title>
                  <artist>Lee Michaels</artist>
                  <copyright>(c) 2002 MFV Records</copyright>
                  ...
            </SONG>
            <ENCRYPTIONINFO>
                  <level>LOW</level>
                  <PUBLICKEY>
                        <data>aAGNQ2626523FLFJkjat </data> --> base64 encoded
                  </PUBLICKEY>
            </ENCRYPTIONINFO>
            <LICENSERENEW>
                  <url>http://www.securecontent.com/license</url>
                  <requestdata>user_id=12415&content_id=15642</requestdata>
            </LICENSERENEW>
         </HEADER>
      </MULTIMEDIA>
</XML>
```

And here is the same header XML using HIGH security:

```
<XML>
   <MULTIMEDIA TYPE="ogg">
         <HEADER>
            <SONG>
                  <title>Here We Are</title>
                  <artist>Lee Michaels</artist>
                  <copyright>(c) 2002 MFV Records</copyright>
                  ...
            </SONG>
            <ENCRYPTIONINFO>
                  <level>HIGH</level>
                  <PUBLICKEY>
                        <data>aAGNQ2626523FLFJkjat </data> --> base64 encoded
                  </PUBLICKEY>
                  <SIGNATURE type="SHA1">
                        <data>asaf2RFJFK5114 </data> --> base64 encoded
                  </SIGNATURE>
            </ENCRYPTIONINFO>
```

```
            <LICENSERENEW>
                  <url>http://www.securecontent.com/license</url>
                  <requestdata>user_id=12415&content_id=15642</requestdata>
            </LICENSERENEW>
         </HEADER>
      </MULTIMEDIA>
</XML>
```

The EncryptionInfo element listed in the above license serves several purposes. First, it sets a level of protection that the decoding application must conform to. This level of security is needed to prevent malicious users from developing programs that simply save the unencrypted byte-stream to disk. The levels of encryption are defined below:

LEVEL	DESCRIPTION
Low	The low level is the default of the oggs123 application. This level means that the unencrypted media byte-stream is available to third-party developers. Also, the encrypted Media-S content can be decrypted via the public key specified in the PUBLICKEY element. Because this security is minimal, content encrypted with a low level of security is best suited for development environments only.
High	High security means that the Media-S content can only be decoded by digitally signed applications. The Media-S file in question must also be digitally signed. The decoding application must ensure that the digital signature of the Media-S file matches a known value. This level of encryption ensures that content can only be decoded by applications that are not malicious in nature, and is the recommended value for most content.

The discussions below demonstrate these security levels in more detail.

LICENSE VAULT OVERVIEW

Working hand-in-hand with the Media-S decoding mechanism (such as the oggs123 player) is the license vault. The license vault is a local encrypted database that stores content provider and license information for each file. This DBM file itself is not encrypted, however by default each of the key values is encrypted using 1024-bit 3DES encryption using a key generated from specific user and machine information. The use of machine-specific information ties a specific license vault to a particular user on a particular machine, and it is this encryption that provides the “tethering” aspect of Media-S.

For low security applications, the actual license vault is an encrypted DBM file whose keys are base64 representations of SHA1 checksums for each file. Each key corresponds to an XML string that details the license permissions for the file. The key values themselves are encrypted using a global shared secret.

High security applications are free to implement other license vault protection schemes on the client, with the understanding that "hiding" some license vault information may make other applications unable to decode the content.

Although this XML is detailed in the appendix, the following example outlines a sample configuration:

```
<XML>
   <LICENSES>
      <CONTENTPROVIDER>
            <PUBLICKEY>
                  <data>aAGNQ2626523FLFJkjat </data> --> base64 encoded
            </PUBLICKEY>
            <name>MFV Records</name>
      </CONTENTPROVIDER>
      <LICENSE>
            <expiration>10/21/2002</expiration>
            <PERMISSIONS>
                  <PERMISSION>
                        <type>PLAY</type>
                        <active>TRUE</active>
                  </PERMISSION>
                  <PERMISSION>
                        <type>STREAM-TO-FRIEND</type>
                        <active>TRUE</active>
                        <expiration>10/15/2002</expiration>
                  </PERMISSION>
            </PERMISSIONS>
      </LICENSE>
   </LICENSES>
</XML>
```

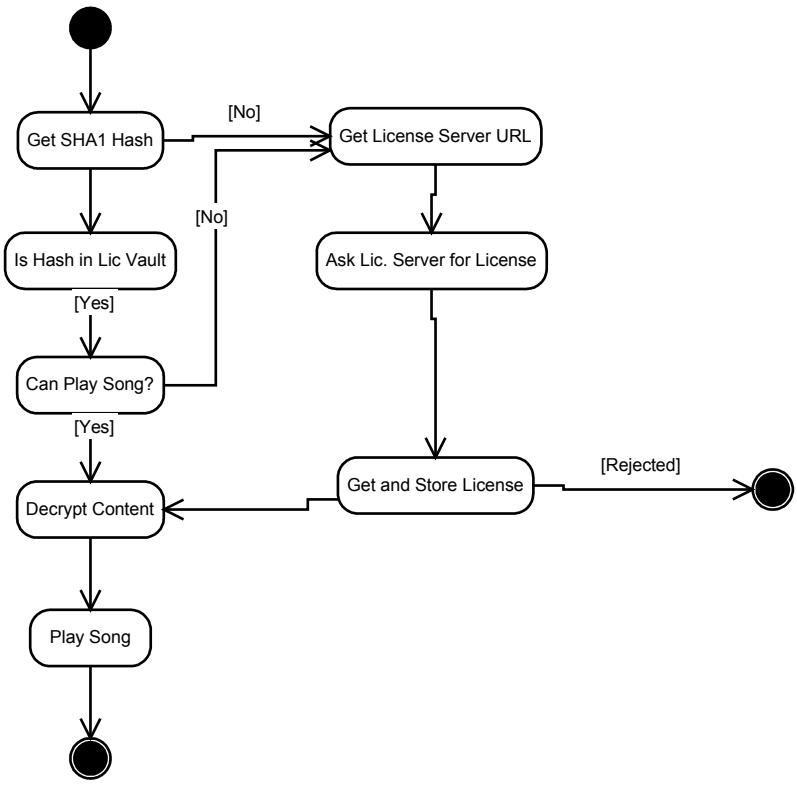
A few notes about the above license:

- Permissions can be defined on an application-by-application basis. There is no global set of permissions that an application needs to support. The permissions that oggs123 supports are detailed later in this document.
- Permissions can expire at different times. To determine if a license is expired, the lowest effective date is used. For instance, if a user wants to play the above file they can do so until 10/21/2002. However they can only stream the file to a friend until 10/15/2002.

Streamed content data is also stored in the license vault with the checksum being the URL of the content to access.

Since the license vault is a normal file, the ability to back-up and restore the license vault exists.

An activity diagram of acquiring a license is below:



LICENSE SERVER OVERVIEW

When a license does not exist for content in the license store, the license acquisition URL is retrieved from the content and queried for a license using the requestdata element. Communication to the license server is handled by HTTPS to ensure that communication cannot be intercepted or forged. The inputs to the license server are:

- SHA1 sum of the content being requested (required)
- User's Machine-specific Public Key (required)
- User ID (optional)
- Content ID (optional)

If content is not user-specific (for example a free time-limited preview of a song) then just the SHA1 sum of the content can be used to request a license. However, most Media-S implementations will require specific user information to be provided as well to match content to a specific user/machine (for example only ensuring subscribed users can access private content).

The license server will return an XML license (in the form of a string) which can then be properly encrypted and placed into the user's license vault.

To be an Media-S compliant license server, the license server must be able to generate a license via HTTPS given the following query string parameters:

NAME	DESCRIPTION
content	Base64 encoded SHA1 content hash
user	The user's unique tether-ID. This value is also base64 encoded.

Validation of client and server is done using digital certificates with OpenSSL. The server must verify that the client's certificate is valid, and the client must ensure that the server certificate is correct.

Media-S PLAYER DETAIL

The Media-S format is designed to be player agnostic so implementing Media-S support into a player will just involve linking to a decryption and licensing library that exports the functions described below. Different decryption and licensing schemes can be used for each application so long as the libraries expose the functions described below.

OGGS123 is the default application for playing back Ogg Vorbis encrypted Media-S content. This player is a simple extension of the OGG123 player using the default Media-S library for license retrieval and decryption. This section will outline the functions that this libraries exposes, and how an application developer can override this default library at runtime to provide application-specific encryption code within the Media-S framework.

The core decryption and license management routines are handled by a library called libmedias. This library can be either statically or dynamically added to an application. If a dynamic implementation is chosen, it is recommended that the application ensure that the SHA1 checksum of the library is expected.

The actual manual page for oggs123 will be provided in a separate “User Guide” document.

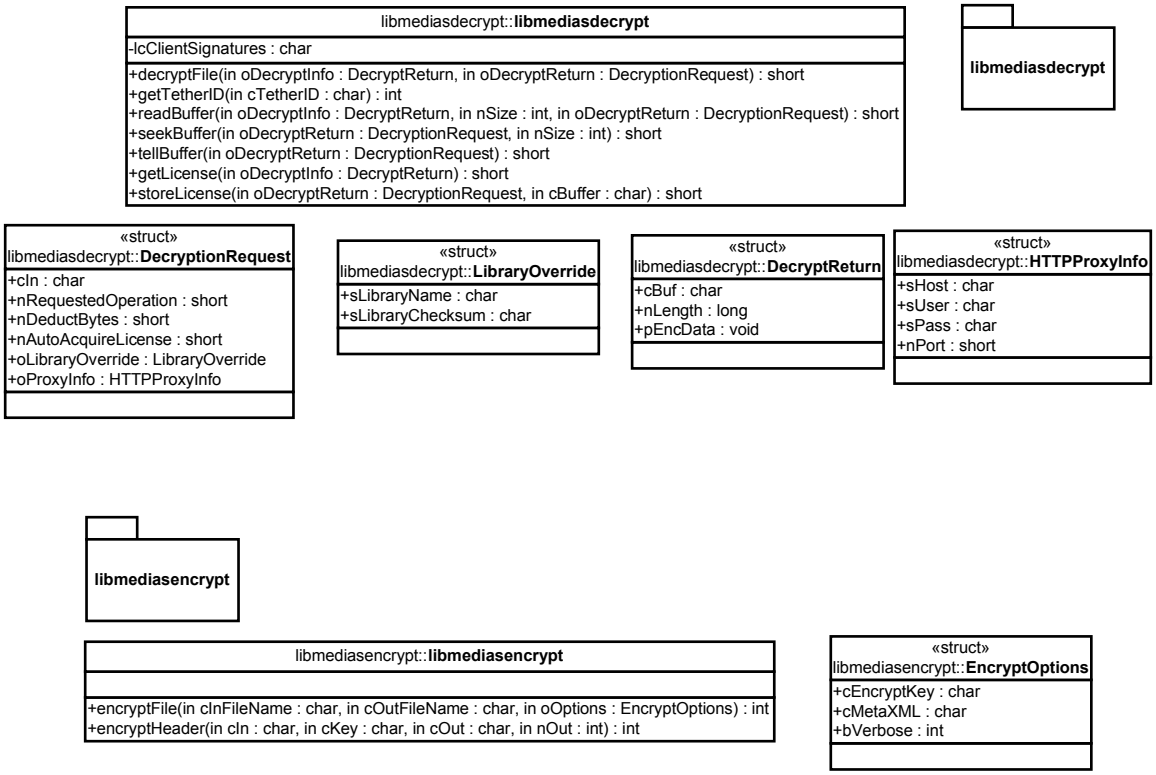
Media-S CLIENT LIBRARY DETAIL

The Media-S player described above, oggs123, relies on a client library to do the actual license management and decryption functions. Ideally, any Media-S player will house the license management and decryption functions in a client library that is either statically or dynamically linked to the player application.

Content providers interested in the most secure Media-S solution will prefer to statically link in a single Media-S client library, while developers may prefer to debug and test using a shared library.

Regardless, each client library exposes the following structures and methods:

The above functions and structures are apt to change during development, therefore their details are not presented here, but will be covered in a subsequent development guide.



Media-S ENCODER DETAIL

To create Media-S files, content providers will use a tool called “oggsenc”, which will use libmedias to encrypt normal OGG files. This tool will be used by content providers to create protected Media-S files that can be customized on a per-user basis.

The actual functionality of oggs123 will be described in a separate “Media-S User Guide”.

RISKS

The following risks have been identified with this implementation:

- A weakness in the license vault is that if a user copies their license vault to another location on the computer, plays content, then copies the license vault back to its original location then there is no reliable way to determine that the license vault has changed. Possible solutions to this problem are being researched.

CONCLUSION

The above document provides an implementation for DRM in OGG files that is both portable and secure. Feedback is encouraged so that a DRM solution can be created that is a viable alternative to proprietary DRM systems.

Once the information in this document reaches achieves some sense of stability, the "Media-S User Guide" and the "Media-S Developer's Guide" can be created.

APPENDIX A: XML DTD DEFINITIONS

1. LICENSE VAULT XML

Each license on a user's machine is stored encrypted in a DBM database known as the License Vault. The keys to this DBM file are the SHA1 hashes of the media files, and each key has as its value an XML license string. This license string has the following DTD:

```
<!ELEMENT LICENSE (CONTENTPROVIDER, LICENSE?)>
<!ELEMENT CONTENTPROVIDER (PUBLICKEY, name,  property*)>
<!ELEMENT property (#PCDATA)>
<!ATTLIST property name #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT LICENSE (expiration, PERMISSIONS*)>
<!ELEMENT PERMISSION (type, active, expiration?,   property*)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT active (TRUE|FALSE)>
```

Sample:

```
<XML>
   <LICENSE>
      <CONTENTPROVIDER>
            <PUBLICKEY>
                  <data>aAGNQ2626523FLFJkjat </data> --> base64 encoded
            </PUBLICKEY>
            <name>MFV Records</name>
            <property name="contact">Mike Smith</property>
      </CONTENTPROVIDER>
      <LICENSE>
            <expiration>10/21/2002</expiration>
            <PERMISSIONS>
                  <PERMISSION>
                        <type>PLAY</type>
                        <active>TRUE</active>
                  </PERMISSION>
                  <PERMISSION>
                        <type>STREAM-TO-FRIEND</type>
                        <active>TRUE</active>
                        <expiration>10/15/2002</expiration>
                  </PERMISSION>
            </PERMISSIONS>
      </LICENSE>
   </LICENSE>
</XML>
```

2. Media-S FILE HEADER XML

The XML License header appears at the top of every Media-S file. This header contains relevant information about the protected content, including song meta-data and license renewel information.

This header is described by the following DTD:

```
<!ELEMENT MULTIMEDIA (HEADER)>
<!ATTLIST MULTIMEDIA type>
<!ELEMENT HEADER  (SONG, ENCRYPTIONINFO, LICENSERENEW?)>
<!ELEMENT SONG (title, author, copyright?, rating?,  property*)>
```

```
<!ELEMENT ENCRYPTIONINFO (level, PUBLICKEY, SIGNATURE?)>

<!ELEMENT level (#PCDATA)>
<!ELEMENT PUBLICKEY (data)>
<!ELEMENT SIGNATURE (data)>
<!ATTLIST SIGNATURE type>
<!ELEMENT data (#PCDATA)>

<!ELEMENT album (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT copyright (#PCDATA)>
<!ELEMENT rating (E|T|M|A)>
<!ELEMENT property (#PCDATA)>
<!ATTLIST property name>
<!ELEMENT PUBLICKEY (data)>
<!ELEMENT LICENSERENEW (url, requestdata?, property*)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT requestdata (#PCDATA)>
```

Sample header:

```
<XML>
      <MULTIMEDIA TYPE="ogg">
         <HEADER>
            <SONG>
                  <title>Here We Are</title>
                  <artist>Lee Michaels</artist>
                  <copyright>(c) 2002 MFV Records</copyright>
                  ...
            </SONG>
            <PUBLICKEY>
                  <data>aAGNQ2626523FLFJkjat </data> --> base64 encoded
            </PUBLICKEY>
            <LICENSERENEW>
                  <url>http://www.securecontent.com/license</url>
                  <requestdata>user_id=12415&content_id=15642</requestdata>
            </LICENSERENEW>
         </HEADER>
      </MULTIMEDIA>
</XML>
```

3. LICENSE SERVER RESPONSE XML

The XML returned by the license server is used by client applications (such as oggs123) to determine what rights a user has to the content in question. If a license is granted, it is stored in the user's license vault.

This response is described by the following DTD:

```
<!ELEMENT LICENSERESPONSES (LICENSERESPONSE+)>
<!ELEMENT LICENSERESPONSE (songhash, userid, LICENSE?)>
<!ELEMENT songhash (#PCDATA)>
<!ELEMENT userid  (#PCDATA)>
<!ELEMENT LICENSE (expiration, PERMISSIONS*)>
<!ELEMENT PERMISSION (type, active, expiration?, property*)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT active (TRUE|FALSE)>
```

Sample response:

```
<XML>
   <LICENSERESPONSES>
      <LICENSERESPONSE>
            <songhash>aAGNQ2626523FLFJkjat </songhash> --> base64 encoded
            <userid>BBaAGNQ2626523FLFJkjat </userid> --> base64 encoded
```

```
            <LICENSE>
                  <expiration>10/21/2002</expiration>
                  <PERMISSIONS>
                        <PERMISSION>
                              <type>PLAY</type>
                              <active>TRUE</active>
                        </PERMISSION>
                        <PERMISSION>
                              <type>STREAM-TO-FRIEND</type>
                              <active>TRUE</active>
                              <expiration>10/15/2002</expiration>
                        </PERMISSION>
                  </PERMISSIONS>
            </LICENSE>
      </LICENSERESPONSE>
   </LICENSERESPONSES>
</XML>
```