

## The Role of NEMO in Marlin

THE MARLIN DEVELOPER COMMUNITY, INCLUDING, INTERTRUST, PANASONIC, PHILIPS, SAMSUNG AND SONY MAKE NO REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, CONCERNING THE COMPLETENESS, ACCURACY, OR APPLICABILITY OF ANY INFORMATION CONTAINED IN THIS DOCUMENT. THE MARLIN DEVELOPER COMMUNITY, INCLUDING, INTERTRUST, PANASONIC, PHILIPS, SAMSUNG AND SONY DISCLAIM ALL LIABILITY OF ANY KIND WHATSOEVER, EXPRESS OR IMPLIED, ARISING OR RESULTING FROM THE RELIANCE OR USE BY ANY PARTY OF THIS DOCUMENT. THIS DOCUMENT IS PROVIDED TO YOU "AS IS". THE MARLIN DEVELOPER COMMUNITY, INCLUDING, INTERTRUST, PANASONIC, PHILIPS, SAMSUNG AND SONY MAKE NO REPRESENTATIONS CONCERNING THE APPLICABILITY OF ANY PATENT, COPYRIGHT OR OTHER PROPRIETARY RIGHT OF A THIRD PARTY TO THE INFORMATION CONTAINED IN THIS DOCUMENT OR ITS USE. THE RECEIPT OR ANY USE OF THIS DOCUMENT OR ITS CONTENTS DOES NOT IN ANY WAY CREATE BY IMPLICATION, ESTOPPEL OR OTHERWISE, ANY LICENSE OR RIGHT TO ANY PATENT, COPYRIGHT, TRADEMARK OR TRADE SECRET OF THE MARLIN DEVELOPER COMMUNITY, INCLUDING, INTERTRUST, PANASONIC, PHILIPS, SAMSUNG AND SONY, WHICH ARE OR MAY BE ASSOCIATED WITH THE IDEAS, TECHNIQUES, CONCEPTS OR EXPRESSIONS CONTAINED HEREIN.

# The Role of NEMO in Marlin

Using Marlin entails performing networked transactions for accessing content and DRM objects. NEMO (Networked Environment for Media Orchestration) standardizes Marlin's message-layer security, including entity authentication and formats for authorization attributes (roles).

## 1 Introduction

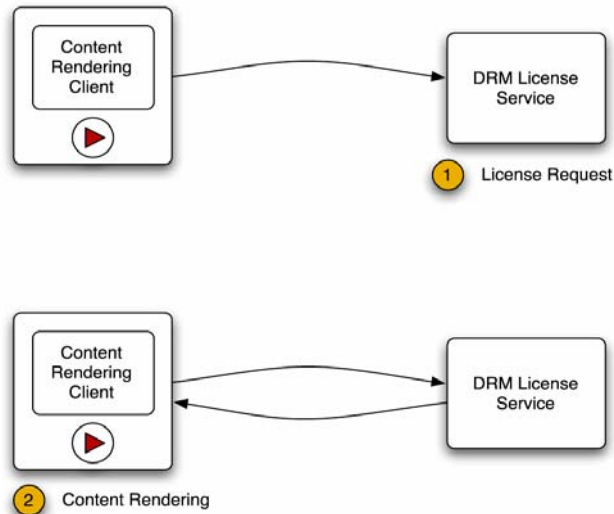
This document introduces the Networked Environment for Media Orchestration (NEMO) as it is used in the Marlin DRM system. The document begins with a discussion of DRM transactions, which provides a natural basis for understanding the respective roles played by the NEMO and Octopus technologies in Marlin. The largest section focuses on the specifics of the NEMO stack, addressing authentication, authorization, and message security in turn.

## 2 Decomposing DRM

The transactions that occur in a DRM system can be separated into two general categories based on the type of information accessed, acquired, or manipulated:

- *Content Access Transactions* involve direct access to or manipulation of media content or other sensitive information protected by the DRM system. Examples of content access transactions include rendering a protected video clip, burning a copy of a protected audio track to a compact disc, or moving a protected file to a portable device. Content access transactions involve direct access to a content protection key and are performed at the point of consumption under the direction of a user.
- *Object Transactions* are transactions in which a user or system acquires or interacts with objects defined by the DRM system that in some way govern access to protected content. Such objects include DRM licenses, membership tokens, revocation lists, and so forth. One or more object transactions are usually required before all of the collateral necessary to perform a content access transaction is available. Object transactions are typically characterized by the use of some type of communications network to assemble DRM objects at the point of consumption.

These two types of transactions define two points of governance that are relevant to every DRM system. Figure 1 shows a typical pair of interactions in which a DRM-enabled client first obtains and then evaluates a DRM license that provides content access.



**Figure 1: (a) A DRM-enabled content rendering client requests a DRM license from an appropriate DRM license service (b) The DRM license is evaluated locally in order to provide access to content.**

DRM systems require that both content access and object transactions are performed in a manner that prevents unauthorized access to content and creation of objects that protect the content. However, the security concerns for the two types of transactions are naturally different. For example:

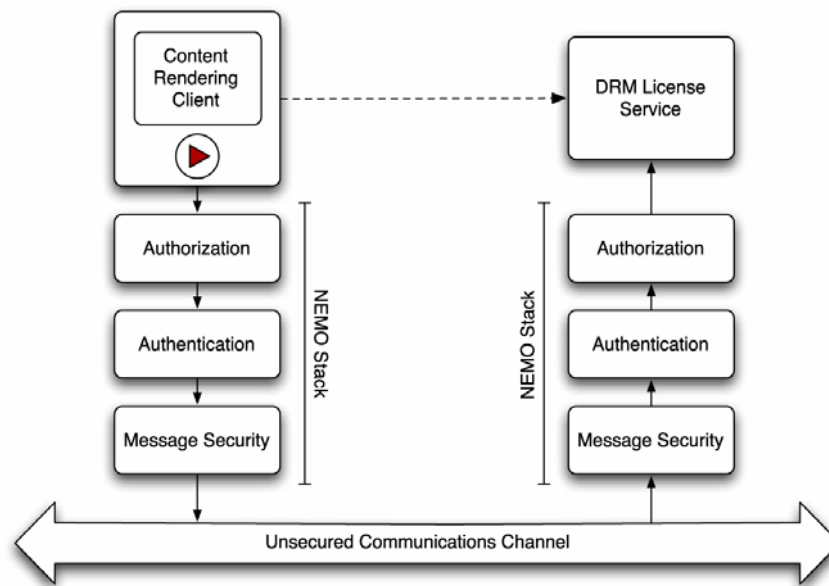
- *Content Access Transactions* may require authenticating a human principal, checking a secure render count, evaluating a DRM license to derive a content protection key, etc. A major threat against legitimate execution of a content access transaction is breach of the tamper resistant boundary that protects the objects and data inside.
- *Object Transactions* usually involve a communications channel between the entity that requires the DRM object and the entity that can provide it. As such, object transactions must be secured against communications-based threats such as man-in-the-middle attacks, replay attacks, denial-of-service attacks, and attacks in which unauthorized entities acquire DRM objects that they should not legitimately possess.

In general, object transactions involve authentication of two interacting entities, the protection of messages passed between them, and authorization of the transaction. The primary purpose of such transactions is to gather integrity-protected DRM objects from legitimate sources so that content access transactions can be performed. From the perspective of a content access transaction, the mechanisms by which legitimate DRM objects are obtained and the collateral information used in obtaining them are irrelevant; these mechanisms can and should be invisible to the content access itself. This natural separation of concerns leads to a layered communications model that clearly distinguishes the trusted communications framework from applications that are built on top of it.

### 3 The NEMO Stack

The simplified license acquisition and consumption example shown in Figure 1 above obscures many important details. For example, it does not show how the DRM license service verifies that the entity requesting a DRM license is in fact a legitimate DRM client and not a malicious entity attempting to obtain an unauthorized license or to deny service to legitimate clients by consuming network bandwidth and processing power. Nor does it show how sensitive information is protected for confidentiality and integrity as it moves through the communications channels connecting the client and service.

A more complete view of this transaction is shown in Figure 2, below.



**Figure 2: A more detailed view of the object access transaction shown in Figure 1a. The dotted line represents the logical transaction from the point of view of the application-layer content rendering client and DRM license server. The stack below represents the layers of processing required to guarantee trusted and protected delivery between the two endpoints.**

In Figure 2 a rendering client requests a license from a DRM license server. The dotted line in the diagram indicates that the original source and ultimate consumer of the information are the content rendering client and the DRM license server, respectively. In reality, the message payload is handled by several layers of processing interposed between the application-layer logic and the unsecured communications channel connecting the two endpoints.

The processing layers that separate the application layer components from the unsecured communications channel are referred to collectively as *the NEMO stack*. The NEMO stack can be thought of as a secure messaging framework that guarantees integrity protected, confidential delivery of messages between trusted endpoints. The layered stack model provides two distinct advantages:

- Designers of the application layer logic do not need to expend effort developing the underlying secure communications mechanisms that connect endpoints. The trusted messaging infrastructure is a common design pattern that, once designed, can be

deployed in many different situations regardless of the application layer logic that they are supporting.

- The messaging framework itself can remain agnostic to the precise semantics of the messages it is conveying and focus its efforts on preventing communications-related attacks and attacks on the authenticity of the messaging endpoints.

The NEMO stack consists of several distinct layers of processing as outlined below.

### **3.1 Authentication**

In NEMO, messaging endpoints may be *authenticated*. Authentication is a process by which a given endpoint demonstrates to another that it has been given a valid name by an authority trusted for this purpose. The naming authority must be trusted by the relying endpoint in a transaction; establishing such an authority is typically undertaken by the organizations deploying the trusted technology.

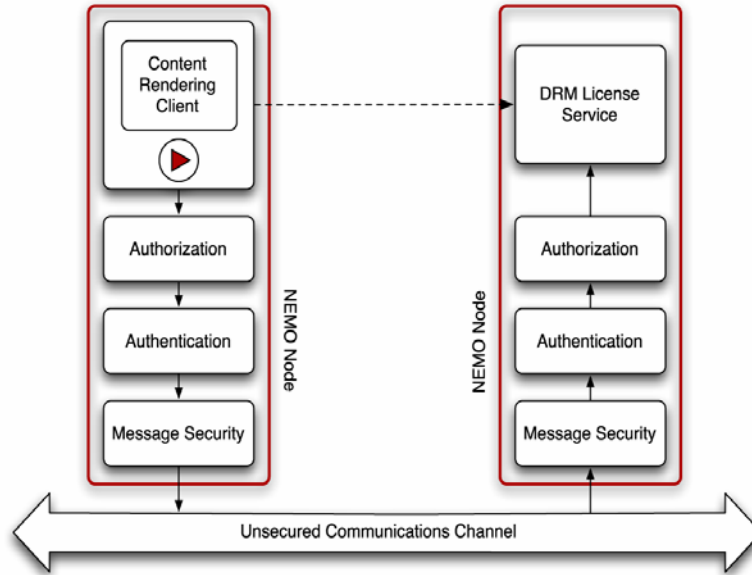
A common mechanism for demonstrating possession of a valid name uses public key cryptography and digital signatures. Using this approach, an entity is provided with three pieces of information:

1. A *distinguished name* that provides an identifier for the entity
2. An *asymmetric key pair*, consisting of a public key and a secret private key
3. A digitally signed *certificate* that asserts that the holder of the private key has the given distinguished name.

The certificate binds the distinguished name and the private key. Any entity that uses the private key to sign a piece of information is trusted to have the given distinguished name. The signature can be verified using only the public key. Authentication in Marlin is based on the X.509v3 standard described by [RFC3280].

Since any entity that can demonstrate possession of a certified private key is trusted to have the distinguished name mentioned in the certificate, protecting the private key used to sign information becomes an important consideration. In effect, the ability to use the private signing key defines the boundaries of the entity identified by the distinguished name. At the application layer, senders and recipients need to know that messages originate from trusted counterparts. As such, it is important that the application layer logic itself be part of the authenticated entity. For this reason, the NEMO stack and the application layers that rely upon it are enclosed in a trust boundary referred to as a *NEMO node*, as shown in Figure 3, below.

In practice, a NEMO node is the entity identified by a given distinguished name. Typically, any subsystem contained within the NEMO trust boundary shares access to the private message signing key.



**Figure 3: The communications stack of Figure 2 with the NEMO node trust boundary shown in red.**

### 3.2 Authorization

The authentication mechanism described above proves to distributed messaging endpoints that their correspondent's identities are trustworthy. In many applications, this information is too coarse — more detailed information about the capabilities and properties of the endpoints is required to make policy decisions about certain transactions. For example, in the context of Figure 1, the content rendering client needs to know not only that it is communicating with an authenticated endpoint, but also with a service that has been deemed competent to provide valid DRM license objects.

The NEMO stack provides a mechanism for asserting, conveying, and applying policy based on more fine-grained attributes about authenticated NEMO nodes via its *authorization* mechanism. Using this mechanism, NEMO nodes that already possess authentication credentials are assigned *role assertions* that associate a named set of capabilities with the distinguished name of the node. For example, Marlin defines the role names

```
urn:marlin:core:role:drm-client
```

and

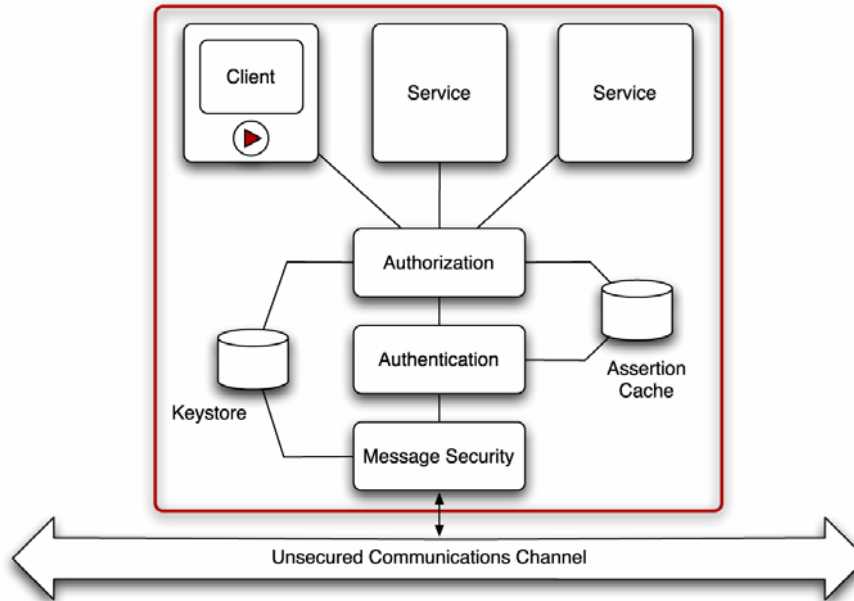
```
urn:marlin:broadband:role:license-service
```

for the DRM client and DRM license server, respectively.

The named roles are intended to convey specific capabilities held by a node. In practice, roles are attached to a NEMO node by asserting an association between the node's distinguished name and the role name. As with authentication certificates, which associate keys with distinguished names, role assertions used for authorization are signed by a trusted role authority that may be different from the name issuer. Inside a NEMO node, role assertions are verified along with the

authentication credentials as a condition for granting access to a messaging endpoint's application layer.

A NEMO node may hold as many role attributes as are required by the application being built on top of NEMO. The example in Figure 4 below shows a NEMO node with multiple roles: one role that indicates the ability to function as a DRM client and two service roles. For example, one node may be simultaneously a DRM Client, a DRM Object Provider and a Security Data Provider.



**Figure 4: A NEMO node with three roles supporting one client and two services for application layer functionality.**

NEMO specifies the use of SAML 1.1 assertions [SAML] for node attributes . Marlin specifications define which role attributes can be assigned to nodes.

### **3.3 Message Security**

The bottom layer of the NEMO stack is the message security layer, which provides integrity, confidentiality, and freshness protection for messages, and mitigates the risk of attacks on the communications channel such as replay attacks. In the message security layer:

- Messages between application layer processes are signed using the NEMO node's private message signing key, providing integrity protection and resistance to man-in-the-middle attacks.
- Messages are encrypted using a public key held by the destination NEMO node. This guarantees that unintended recipients cannot read messages intercepted in transit.
- Nonces and timestamps are added to the message, providing immunity to replay attacks and facilitating proofs of liveness between the messaging endpoints.

NEMO requires support for AES symmetric encryption, RSA public key cryptography, and SHA-256 signature digests, and has mechanisms to signal other algorithms in messages. Marlin requires support for only this core set of algorithms.

## 4 References

[RFC3280] R. Housley *et al.*, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC 3280, <http://www.ietf.org/rfc/rfc3280.txt>

[SAML1.1] Eve Maler, Prateek Mishra and Rob Philpott, eds., *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1*, <http://www.oasis-open.org/committees/download.php/3406/oasis-%20saml-core-1.1.pdf>

[Marlin core] Marlin – Core System Specification.

[NEMOArch] InterTrust. *NEMO Architecture Specification v0.95*.

[NEMOMessage] InterTrust. *NEMO Message Bindings v0.95*.

[NEMOTrust] Intertrust, *NEMO Trust Management Bindings*.

[NEMOSecurity] Intertrust, *NEMO Security Bindings*.