The Role of Octopus in Marlin

The Role of Octopus in Marlin

Marlin is a new DRM system for solving the interoperability challenges facing the consumer electronics industry. Octopus is a set of specifications and a reference implementation for a simple, open, and flexible digital rights management (DRM) engine for implementing core DRM functions in different applications.

1 Introduction

This document introduces Octopus as it is used in the Marlin DRM system. The document begins with a discussion of Octopus' goals and benefits to a potential adopter of Marlin specifications. The largest section focuses on the specifics of the architecture; this covers both the conceptual underpinnings and practical application of Octopus in providing DRM functionality on CE devices and services.

2 Goals

Octopus is a general purpose toolkit designed to meet several goals:

- *Simplicity*. The Octopus DRM engine uses a minimalist stack-based Virtual Machine (VM) for executing Control Programs (programs that enforce content usage governance policies).

- *Modularity*. Octopus is designed to function as simple modules combined and integrated into a larger DRM-enabled application.

- *Flexibility*. Because of its modular design, Octopus can be used in a wide variety of software environments, from embedded devices to general-purpose PCs.

- *Open*. As a part of Marlin, Octopus code modules and APIs can be implemented by adopters in virtually any programming language in systems that they control completely. Neither Marlin nor Octopus force users to adopt particular content formats or restrict content encoding in any way.

- *Semantically Agnostic*. The Octopus DRM model leverages a simple relationship graph that turns authorization requests into queries about the structure of a graph. The vertices in the graph represent entities in the system and directed edges represent relationships between these entities, but Octopus is not, and does not need to be, aware of what those vertices and edges represent in any particular application. In Octopus, the vertices are referred to as "Nodes" and the edges as "Links".

- *Seamless Integration with Web Services*. The Octopus client was designed from the beginning with web services in mind, and can use such services in several ways. For example, vertices and edges in the authorization graph can be dynamically discovered through services. That said, the architecture itself does not depend on web services.

- *Simplified Key Management*. In many cases, the authorization graph topology can be reused to simplify the derivation of content protection keys without requiring cryptographic retargeting. This is an optional but powerful feature of Octopus – it can also integrate with other key management systems.

- *Separation of Governance, Encryption, and Content*. The controls that govern content are logically distinct from the cryptographic information used to enforce the governance. Controls and cryptographic information are again both logically distinct from content and content formats.

3 Value Proposition

Marlin leverages the benefits of Octopus DRM which include the following:

- Rapid deployment
- Scalable and future proof
- Balances security with utility – recommends policies to govern content rather than heavy crypto (in fact, it is crypto agnostic)
- Lets the adopter retain control of many aspects of the implementation…
- Opens up CE environment to multiple services – disrupts the dominant silo-model

4 Architecture Overview

Marlin specifications define ways to make content available to a user so that it is consistently governed, regardless of the distribution channel. This consistent behavior of content is enabled through the use of an Octopus-based DRM system.

Octopus technology is central to Marlin as it enables content to be shared on a variety of mediacapable devices while ensuring that the use of the content is governed according to usage rights granted by the content owner. Octopus is designed to efficiently support arbitrarily complex content licensing models without adding complexity to the underlying implementation. This is a result of its design decision to be agnostic of the semantics of the relationships formulated by the license issuer or other entities in the overall system.

This section describes Octopus licenses, how Octopus is implemented in the Client, Octopus objects, and Scuba key distribution.

4.1 Octopus Licenses

An Octopus license is a set of objects that govern the use of content and convey the information necessary for obtaining the content encryption keys. The content and the license are logically separate but are bound together by internal references (using object identifiers). A license can apply to more than one item of content, and more than one license can apply to any single item of content.

A typical Octopus licensee is a user. In Marlin, the licensee is represented by an Octopus User Node, which is a cryptographically-certified identity. When the licensee is represented by an Octopus User Node, then the content is considered to be targeted to the user. Content may be cryptographically *bound* (keyed) to one Octopus Node representing a device or a user, yet *targeted* through authorization rules to another Octopus Node (device or user).

An Octopus Control is a component of an Octopus license that contains an executable program known as a control program. In the context of a license, a Marlin DRM Client uses the Octopus Control to arbitrate access to content by executing the control program. The control program has entry points for governing specific actions; Marlin specifies the actions that must be supported by an Octopus implementation. For example, the Octopus control program would have an entry point for the PLAY action. When a Marlin DRM Client application attempts to play content the Octopus control program's PLAY entry point would be invoked to determine whether the content can be played.

When the license is targeted to the user, a typical control program will attempt to discover a path of Links from the current node (typically the media rendering entity) to the User Node identified in the control program. Initially, the search is rooted in a Personality Node, which in Octopus defines properties of a device or client application. The control program invokes the system call

Octopus.Links.IsNodeReachable and supplies the identity of the targeted User Node as a

parameter. The Octopus engine is free to implement this system call any way it chooses.Links are semantic-agnostic; the meaning of a Link is determined by the Link issuer and by the controls that act upon the Link. The semantics of a path of Links between a device and a user can be used to indicate device ownership by that user. In such a scenario, linking a device to the identity of the User Node can enable the device to access all content licensed to that particular user. More generally, the graph that is formed by the nodes and links – i.e., the topology – is a reflection of the business models and policies that the system enforces.

An Octopus Link contains information that describes which node the link originated from and which node the link is going to. Links are integrity-protected so that unauthorized relationships cannot be forged.To enforce a particular policy, Octopus Links may contain an Octopus Control that behaves similarly to the way controls work in a license. As a result, an Octopus Control that is carried in a link can evaluate whether the link is valid within a given context.

The graph formed by the Octopus Link topology is very flexible. For example, multiple devices can be linked to a user, allowing the user to consume content on all his/her linked devices. Conversely, it is also possible to link a single device to multiple users, allowing multiple members of a household to view their content from a common device such as a Set Top Box (STB) (see Figure 1 below).

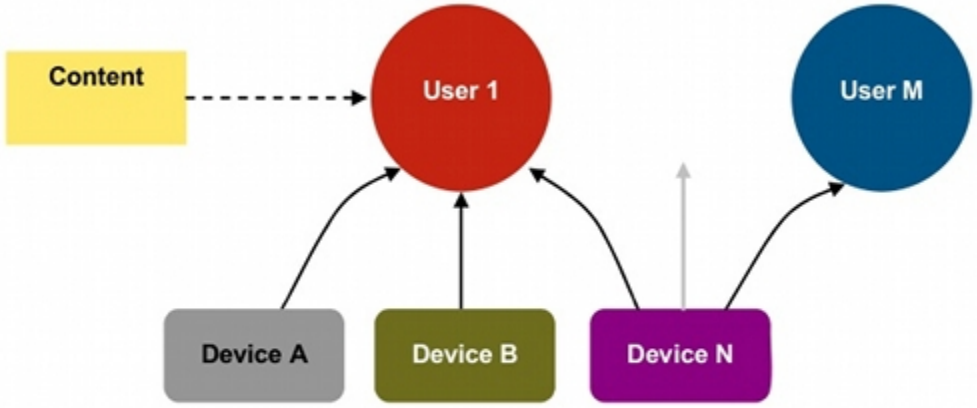

Figure 1: Simple Device/User Topology

Both the Marlin Broadband and Broadcast Delivery System specifications define a similar topology where a set of devices is linked to a user. However, devices are not required to be directly linked to a user. Intermediate nodes can be inserted into the path from the device to the User Node; for that matter, the User Node can also be an intermediate. The Marlin Broadband Delivery System specification leverages this latter approach to realize a topology that supports a subscription model. Specifically, there is a Subscription Node and a Link which creates the path between the User Node and the Subscription Node.

A strategy for decoupling the management of devices from the User Node is to include an intermediate node that represents a set of devices. In Marlin, this intermediate node is referred to as a *Domain Node.* The Link between a device and a Domain Node is managed by a *Domain Manager.* A Domain Manager exposes a set of services, the DRM Object Provider service and the Domain Information Provider service, which together are responsible for generating and distributing the link between a device and a domain. The Domain Manager regulates which devices are

members of the domain, the consequences of joining or leaving the domain, and other policies for governing the domain configuration. Once a device is added to a domain, it can play any content

belonging to that domain (subject to provisions in the license) by building a link graph between the Personality Node of the device (or client application) and the target Node in the License's Octopus Control.

An elaborate example of how these Links make content accessible to a user is shown in Figure 2, below.

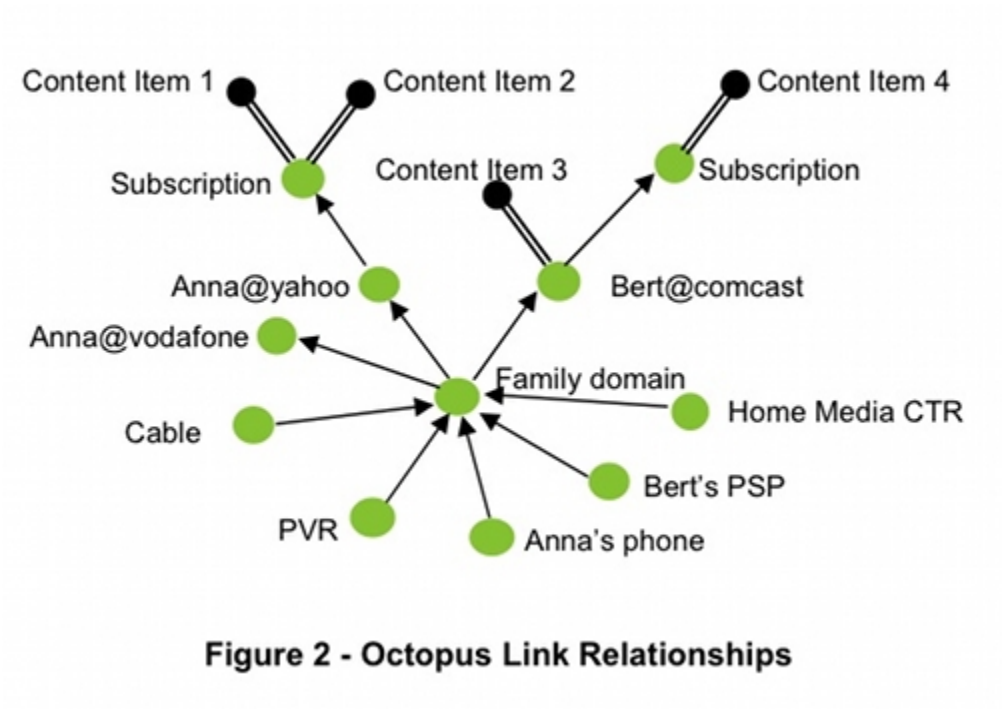

Figure 2 - Octopus Link Relationships

In the above figure, Anna's phone can access Content Item 2, because:

- There is a link from her phone to the family domain (the phone is registered as a member of the domain)
- There is a link from the family domain to Anna's identity at Yahoo! (the family domain is registered with her account, so it can play content associated with the account)
- There is a link from Anna's Yahoo! identity to a Yahoo! music subscription service (With her account, Anna has a music subscription at Yahoo!)
- There is a link from the Subscription to Content Item 2 (Rights to Content Item 2 are included in the subscription services)

4.2 Octopus in the Client

The Octopus engine is part of a DRM client. Figure 3 below illustrates elements in a sample DRM client with an Octopus engine.

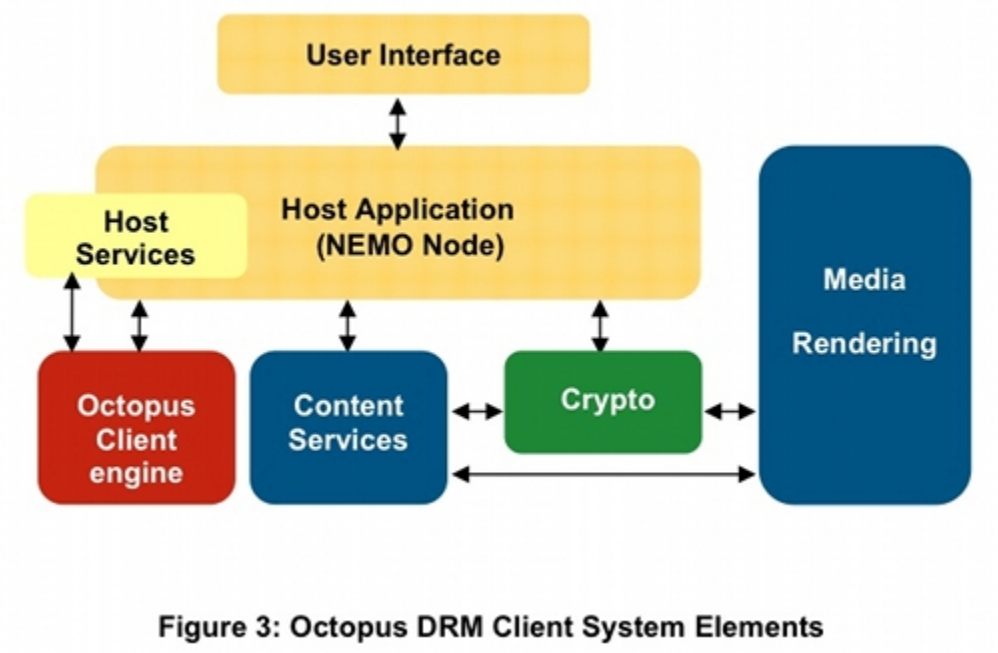

Figure 3: Octopus DRM Client System Elements

Typically, a host application serves the following functions:

- Provides a user interface by which the user requests access to protected content, and with which users interact to see metadata about the content, or error information;
- Manages interactions with the file system or storage facilities;
- Recognizes the protected content object format;
- Requests the DRM engine to evaluate the DRM objects that make up the license (including running the control program) to see if permission to access the content can be granted;
- Provides cryptographic services that perform cryptographic functions that the DRM engine needs such as verifying any required signatures.
- Requests the DRM engine to provide the key to the protected content
- Decrypts and interacts with the media rendering services to render content.

4.3 Octopus Objects

To enable secure and governed content consumption, Marlin relies on the use and exchange of trusted and cryptographically protected Octopus objects and the execution of embedded control programs.

Octopus Objects include:

- **(Octopus) Nodes**: represent DRM system entities such as users or devices.
- **Links**: represent directed relationships between Nodes. When used to express business rules, a Link relationship often translates as “belongs to.” For example, a control that permits rendering if User Node X is reachable following such links from the rendering device indicates a requirement for the device to belong to a certain user.
- **Protected Content Files**: represent the protected media data.
- **License Component Objects**: govern access to the protected content. As Figure 4 illustrates, a license is a collection of the following objects:
    - *ContentKey object*: includes encrypted key data
    - *Protector object*: binds content to ContentKey object
    - *Control object*: includes and protects the control program. The control program is a small piece of executable bytecode that runs on a virtual machine. It checks conditions for content access, and whether certain Nodes are “reachable” using Octopus Links
    - *Controller object:* binds ContentKey objects and Control objects
    - *Metadata:* provides human-readable information to describe the conditions required by the license.
- **Agents**: independent objects that carry control programs. An agent is distributed to an Octopus engine to accomplish some function such as writing into a DRM client’s secure store. An agent is often sent as a consequence of contacting a remote service or executing a remote control program. For example, an agent can be used to affect a content move operation, to initialize a counter or to deregister a node.

The Control object and Controller object are both signed so that the Octopus engine can verify that the control information is from a trusted source before giving the host application permission to access the content.

Control programs are expressed in Plankton bytecode. Plankton supports arbitrarily complex programs that test for conditions such as the following:

- *Time-based conditions:* Compare client time to some value or values specified in the bytecode; this can be any set of time comparisons;
- *Targeting a particular Node:* Check whether a certain Node is reachable from the Node where the Octopus engine is executing. This is a crucial concept in Octopus as it provides

support for such models as Domains (see below) and other types of memberships.

- *Testing if certain Node attributes match specified values:* Check the Personality Node and render if the Node attributes are acceptable. For example, whether the rendering capabilities meet fidelity requirements.

 6.

- *Testing if the security-related metadata at the client is up-to-date:* For example, this can be used to check whether the client has an acceptable version of the client software and accurate time. This function relies on an assertion from a Data Certification Service.

- *State-based conditions:* Licenses can refer to a secure database with state information. For example, this allows rights that involve counters (number of plays, number of exports) and content rights to be moved between devices or exported from the domain.

Using these types of conditions, a Plankton control object can express rules that govern how content can be rendered, transferred and exported. Figure 4 depicts the relation between the various Octopus Objects that make up a license and how the license is bound to the content.

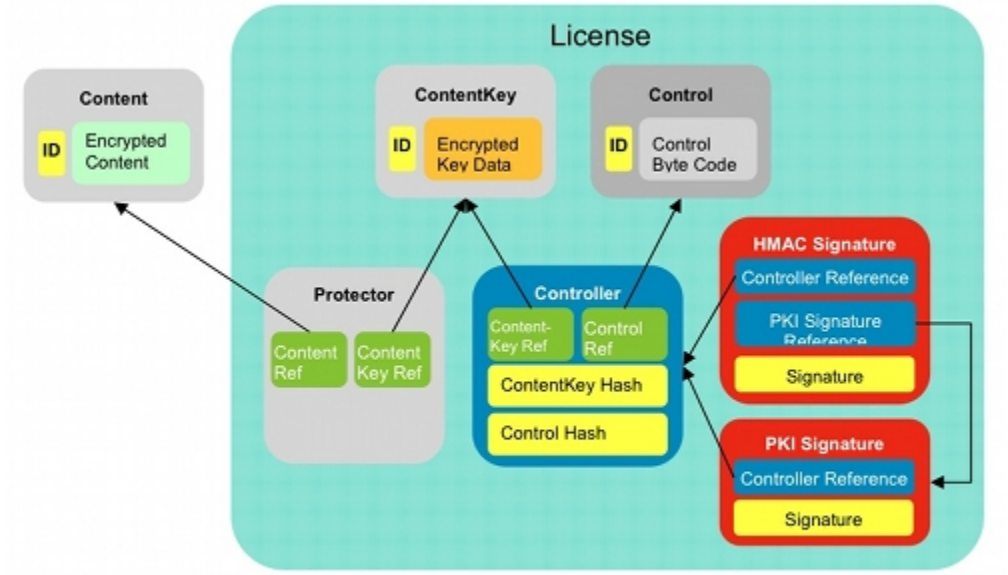

Figure 4: Octopus License Structure

4.4 Key Distribution

The Scuba key distribution system is designed to be a natural fit to the Octopus Architecture. The basic principle behind Scuba is to use Octopus Link objects for distributing keys in addition to their primary purpose of establishing relationships between Node objects. Recall that an Octopus Control object contains a control program that decides if a requested operation should be granted or not. That control program often checks that a specific Octopus node is reachable via a collection of Octopus links. Scuba key distribution leverages the same collection of links to make keys available to the Octopus Engine that is executing the control.

Each Octopus Node object in a Marlin deployment has Scuba keys. These keys are used to encrypt content keys and other nodes' Scuba keys. Each Octopus Link object created for use in the same

deployment contains some cryptographic Scuba data payload that allows key information to be derived when chains of links are processed by the Octopus Engine.

Given a collection of links from a node A to a node Z, any Octopus entity that has access to the private Scuba keys of A also has access to the private Scuba keys of Z. Having access to Z's private Scuba keys gives the node access to any content key encrypted with those keys. A piece of protected content is said to be *bound* to a Node X if there is a corresponding Content Key object encrypted with the Scuba key of Node X. Using Scuba, an Entity's Octopus Engine can access any piece of content that is bound to a reachable Node.

4.4.1 Nodes, Entities and Scuba Keys

In the context of Scuba key distribution, nodes are simple data objects, not active participants. Examples of nodes are media players, devices, content packagers, etc. Active participants are called *Entities*. An Entity that consumes content uses an Octopus Engine and manages at least one Node object that constitutes its Octopus Personality. This entity has access to all the data of the Node objects it manages, including all the private information of those objects.

Node objects that participate in a Scuba key distribution system contain Scuba keys as part of their data. Each node N has 3 keys:

- Scuba Public Key - *Kpub[N]*: This is the public part of a pair or public/private keys for the public key cipher. This key comes with a certificate so that its credentials can be verified by entities that want to cryptographically bind confidential information to it.

- Scuba Private Key - *Kpriv[N]*: This is the private part of the public/private key pair. The Entity that manages that node is responsible for ensuring that this private key is kept secret. However, unlike private keys used for authentication in a Public Key Infrastructure (PKI), Scuba private keys may be shared with other Entities via Octopus Links.

- Scuba Symmetric Key - *Ks[N]*: This is a key to be used with the symmetric cipher. This key is confidential, and the Entity that manages the node is responsible for keeping it secret. However, this key may be shared with other Entities via Octopus Links.

5 Conclusions

Octopus' simple, open, and flexible digital rights management (DRM) engine for implementing core DRM functions in different applications is a natural fit for Marlin. The Octopus architecture enables Marlin to solve the interoperability challenges facing the consumer electronics industry.