Universal DRM (uDRM) Technical Specification

Version: 1.0

Kaltura
open source video

Kaltura Business Headquarters

250 Park Avenue South, 10th Floor, New York, NY 10003

Tel.: +1 800 871 5224

Contents

1 Overview 3
1.1 About This Document 3
1.2 Intended Audience 3
1.3 Abbreviations 3
1.4 Referenced Documents 3
2 uDRM Module Specifications 4
2.1 Architecture Diagram 4
2.2 Supported DRM Providers 4
2.3 uDRM APIs 5
2.3.1 Authentication 5
2.3.2 Custom Data 5
2.3.3 HTTP Methods and Samples 5
2.3.4 License 8
2.3.5 Encrypt 10
2.4 Supported Policies 10
2.4.1 Playready 10
2.4.2 Widevine 11
2.4.3 Apple Fairplay 11
2.5 Security and Fault Tolerance 11
2.5.1 IP Protection 11
2.5.2 Key Obfuscation 11
2.5.3 MPAA Auditing 11
2.5.4 Fault Tolerance 11
2.6 Error Codes 12
2.6.1 Format 12
2.6.2 Supported Response Codes 12
3 Use Cases 13
3.1 Kaltura MediaPrep 13
3.2 External Encryption Modules 13

Release History

Version	Owner / Writer	Date	Comments
1.0 draft A	Arik Gaisler	August 2015	New document
1.0 draft B	Yocheved Abram	20 Sept 2015	General editing

1 Overview

1.1 About This Document

This document provides the technical specifications for the Kaltura uDRM module that enables integration with the uDRM API interface.

1.2 Intended Audience

This document is intended for technical architects and developers who are involved in integrating with the Kaltura uDRM encryption and license server APIs.

1.3 Abbreviations

The following abbreviations are used throughout this document.

Abbreviation	Meaning
API	Application Programming Interface
BE	Backend
CAS	Conditional Access System
CDN	Content Delivery Network
CENC	Common Encryption
CRUD	Create, Read, Update, and Delete
DB	Database
DRM	Digital Rights Management
HDCP	High Bandwidth Digital Content Protection
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
OTT	Over The Top
OVP	Open Video Platform
REST	Representational State Transfer
SDK	Software Development Kit
TBD	To Be Defined
uDRM	Universal DRM
URL	Uniform Resource Locator
XDCR	Cross Data Center Replication

1.4 Referenced Documents

The following documents are referenced throughout this document.

Doc Number	Doc Title
	SAE16 Qualification Report
	Couchbase XDCR Data Encryption

2 uDRM Module Specifications

2.1 Architecture Diagram

Figure 1 presents the uDRM architecture.

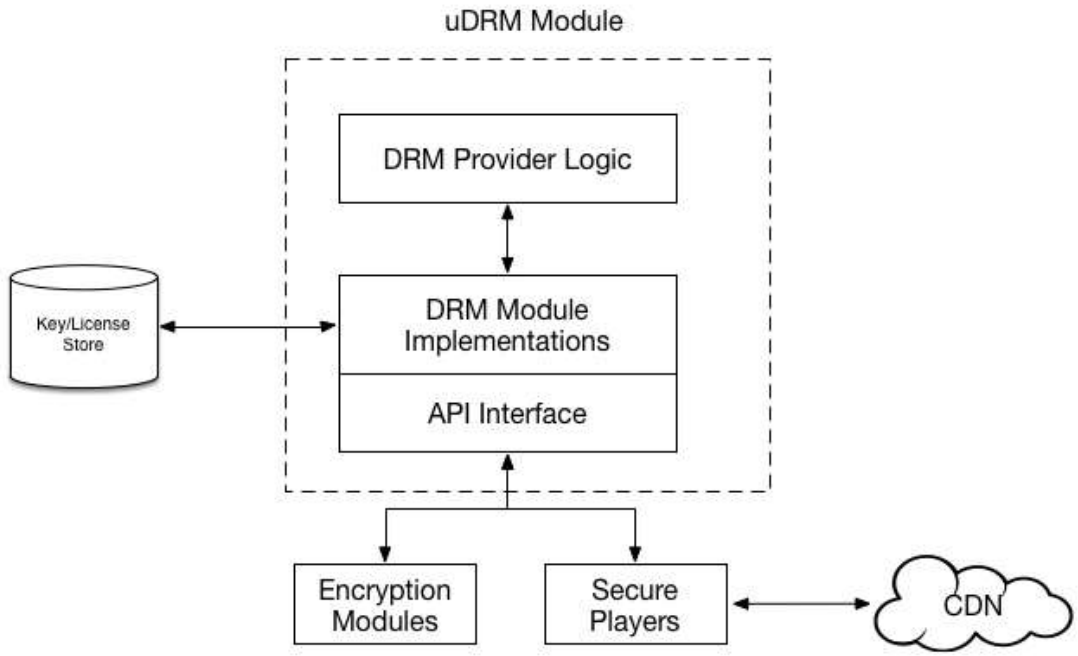

Figure 1: uDRM Architecture

The components in the uDRM module include the following:

- **Encryption Modules –** internal Kaltura encryption modules (whether as part of the on-the-fly packager or pre-encryption jobs) or external encryption modules (also can be pre-encryption processes or part of the on-the-fly packaging modules)
- **Secure Players –** Kaltura Secure Player or external players integrating with the uDRM license interface
- **API Interface –** uDRM API interface
- **DRM Module Implementations –** business logic layer of the uDRM module
- **DRM Provider Logic –** DRM servers and SDK hosted or proxied by the Kaltura uDRM server for generating DRM specific licenses and keys
- **Key/License Store –** uDRM DB (Couchbase) for key storage

2.2 Supported DRM Providers

The Kaltura uDRM module currently supports the following DRM formats and DRM providers:

- CENC
  - Playready
  - Widevine (Widevine Modular)
- Playready (Smooth Stream)
- Widevine (Classic)
- Apple Fairplay
- Adobe Primetime (TBD)

2.3 uDRM APIs

The Kaltura uDRM module exposes a REST API interface supporting the following operations:

- Encrypt – generating an encryption key for the relevant encryption module
- License – generating a license for the relevant DRM client
- Policy – CRUD (create/read/update/delete) operations on DRM policies

2.3.1 Authentication

Each request to the uDRM interface includes a signature authenticating the request to the relevant provider.

The signature is constructed in the following manner

```
Signature = Base64(SHA1(private_key + data))
```

Where:

- `private_key` – a private key used by the specific DRM client (whether encryption process or DRM client)
- `data` – the data passed in the API request (can be the POST data or the query string data, excluding the signature, depending on the request type)

2.3.2 Custom Data

All requests to uDRM endpoints contain a custom data object. The custom data is passed as part of the query parameters, or as part of the JSON post data (see specific API tables for details) in Base64.

The custom data is of the following format:

```
{
    "ca_system": "OTT",
    "account_id": "167",
    "content_id": "file_name.ism",
    "files": "",
    "user_token" :"12345",(optional)
    "key": "CsjUtytiiiieee", (optional)
    "key_id": "file_external_id" (optional),
    "udid", :"12345678" (optional)
}
```

Where:

- `ca_system`: OTT/OVP – static parameter (configured as part of the client onboarding)
- `account_id`: the account identifier (provisions as part of the account onboarding)
- `content_id`: the content external ID
- `files`: empty (future use)
- `user_token`: the user ID (optional)
- `key`: for external encryption processes (see section 2.3.3) supplying a pre-generated key. Base64 encoded.
- `key_id`: for external encryption processes (see section 2.3.3) supplying a pre-generated key ID. Base64 encoded.
- `Udid` : the device udid

2.3.3 HTTP Methods and Samples

GET, POST, PUT, DELETE

HTTP Method - GET

Description - Returns content data, if it exists.

Data - In query string:

```
?custom_data={custom_data}
```

Sample Request

```
{
    "ca_system": "OTT",
    "account_id": "167",
    "content_id": "file_name.ism",
    "files": ""
}
```

Sample Response

```
{
  "pssh": [
    {
      "data": "pssh data”,
      "uuid": "<uuid>
    }
  ],
  "file": null,
  "license_url": "<base64 la_url>
  "key": "<key>
  "key_id": "<base 64 of key ID=",
  "content_id": “<content id>
}
```

HTTP Method - POST

Description - Creates a new key (if content exists, returns existing content)

Data - As JSON In POST data

Sample Request

```
{
    "ca_system": "OTT",
    "account_id": "167",
    "content_id": "file_name.ism",
    "files": ""
}
```

Sample Response

```
{
  "pssh": [
    {
      "data": "pssh data”,
      "uuid": "<uuid>
    }
  ],
  "file": null,
  "license_url": "<base64 la_url>
  "key": "<key>
  "key_id": "<base 64 of key ID=",
  "content_id": “<content id>
}
```

HTTP Method - PUT

Description - Creates a new key, overrides if exists

Data - As JSON in post data

If data contains the fields key and key_id, it creates a new entry using these parameters. If not, it generates independently (as in the case with POST) and overrides it if the key already exists.

Sample Request

```
{
    "ca_system": "OTT",
    "account_id": "167",
    "content_id": "file_name.ism",
    "files": "",
```

```
    “key” : “CsjUtytiiiieee”,
    “key_id” : “file_external_id”
}
```

Sample Response

```
{
  "pssh": [
    {
      "data": "pssh data”,
      "uuid": "<uuid>
    }
  ],
  "file": null,
  "license_url": "<base64 la_url>
  "key": "<key>
  "key_id": "<base 64 of key ID=",
  "content_id": “<content id>
}
```

HTTP Method - DELETE

Description - Deletes a key entry if it exists

Data - JSON as part of the POST data

Sample Request

```
{
    "ca_system": "OTT",
    "account_id": "167",
    "content_id": "file_name.ism",
    "files": "",
     “key” : “CsjUtytiiiieee”,
     “key_id” : “file_external_id”
}
```

Figure 2 illustrates a typical encryption flow.

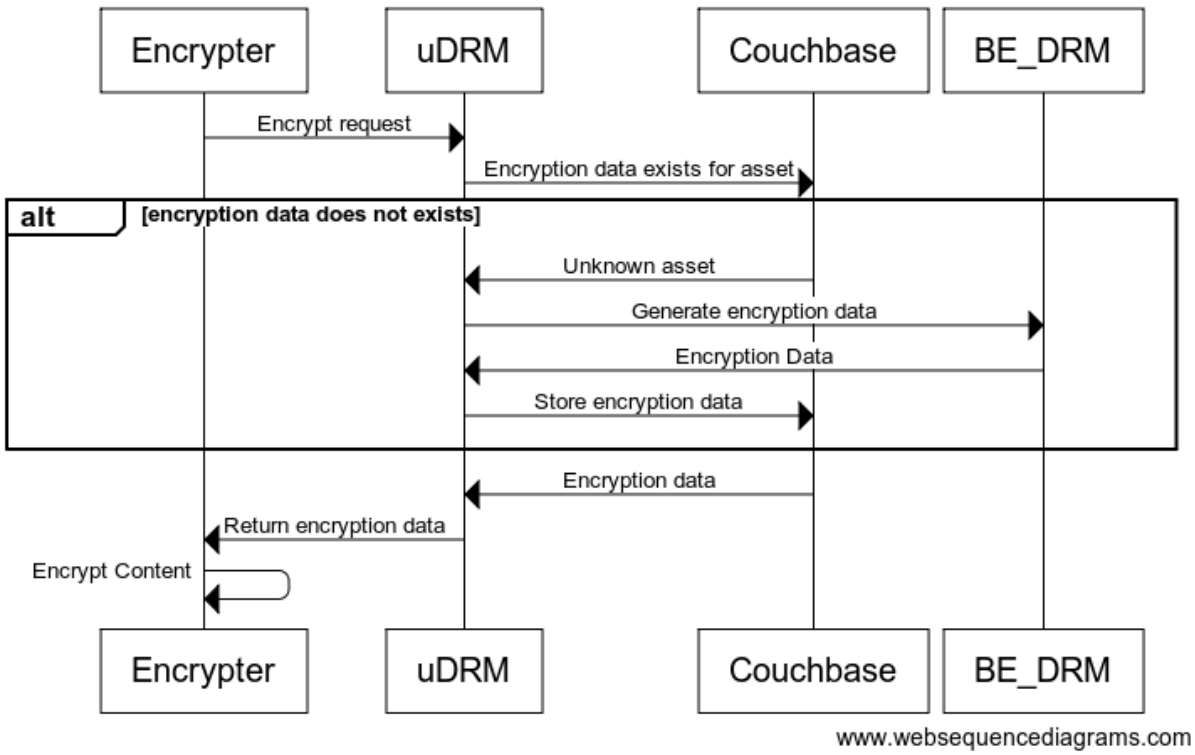

Figure 2: Typical Encryption Flow

1. The encryption component initializes the encryption flow by requesting encryption data from the uDRM encryption interface. This component can be the Kaltura encryption process, or an external one integrating with the uDRM module.
2. The uDRM module checks if encryption data exists for this asset.
   a. If it exists, the uDRM module returns the data.
   b. If it does not exist:
      - The uDRM module calls BE_DRM implementation (e.g. Playready/Widevine) to generate the encryption data.
      - The uDRM module stores the data in an internal data store (Couchbase).
      - The uDRM module returns the data to the encryption component.

The response format for encryption key requests have the format defined in section 2.3.4.

2.3.4 License

The flow for requesting a license is as pictured in Figure 3.

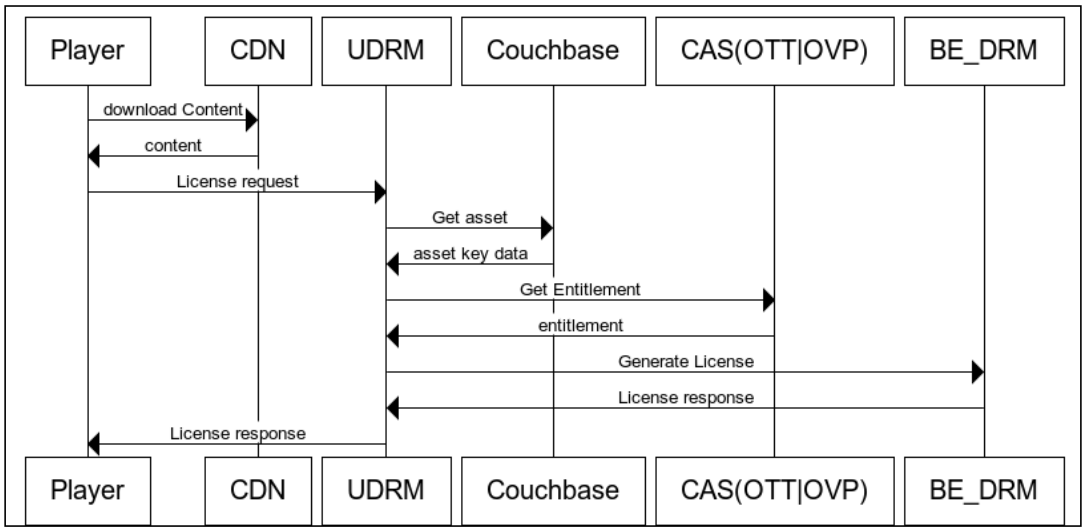

Figure 3: License Generation Flow Diagram

1. The player downloads content from relevant the CDN.
2. The player generates the license URL with the relevant signature and data, and requests a license from the uDRM license acquisition API.
3. The uDRM module retrieves the asset from the internal data store (Couchbase data store).
4. The uDRM retrieves entitlements from the relevant entitlements BE (CAS).
5. Once the entitlement is received, a license is generated by the relevant underlying DRM backend (for instance, Playready server, Widevine SDK).
6. The uDRM module returns the license to the relevant player.

All requests to the license endpoints are with the following URL scheme:

`HTTP(S)://{BASE_UDRM_LOCATION}/{DRM_PROVIDER}/LICENSE?SIGNATURE={SIGNATURE}`

Where:

- `BASE_UDRM_LOCATION`: constant uDRM prefix
- `DRM_PROVIDER`: a DRM provider for license acquisition
- `SIGNATURE`: authentication signature

Sample Request

`https://udrm.kaltura.com/cenc/widevine/license?signature={signature}`

The uDRM license endpoint supports the operations defined in the following table.

HTTP Method - POST

Description - HTTP Method - POST

Description - Creates a new key (if content exists, returns existing content)

Data - As JSON In POST data

Sample Request

```
{
    "ca_system": "OTT",
    "account_id": "167",
    "content_id": "file_name.ism",
    "files": ""
}
```

Sample Response

```
{
```

```
  "pssh": [
    {
      "data": "pssh data”,
      "uuid": "<uuid>
    }
  ],
  "file": null,
  "license_url": "<base64 la_url>
  "key": "<key>
  "key_id": "<base 64 of key ID=",
  "content_id": “<content id>
}
```

Description - As JSON In POST data

Data

In query string

```
?custom_data={custom_data}
```

Sample Request

```
{
   "ca_system": "OTT",
   "account_id": "167",
   "content_id": "file_name.ism",
   "files": "",
   “user_id” : “123456”
}
```

2.3.5 Encrypt

All requests to the encrypt use the following URL scheme:

```
HTTP(S)://{BASE_UDRM_LOCATION}/{ENCRYPTION_PROVIDER}/ENCRYPTION?SIGNATURE=
{SIGNATURE}
```

Where:

- `BASE_UDRM_LOCATION`: Constant uDRM prefix
- `ENCRYPTION_PROVIDER`: A DRM provider for encryption.
- `SIGNATURE`: Authentication signature

Sample Request

```
https://udrm.kaltura.com/cenc/widevine/encryption?signature={signature}
```

The following operations are supported on the encryption endpoint.

(Responses are for the CENC flow. Playready Smooth Stream responses are similar – without the pssh section,)

2.4 Supported Policies

Policies are defined in the uDRM interface in advance. Upon content ingestion, the specific pre-defined policies can be attached to the ingested content by using the policy interface.

2.4.1 Playready

Kaltura supports all defined policy controls specified in the Microsoft product compliancy guidelines for the output protection settings defined in the following table.

Field	Allowed Values
Minimum Compressed Digital Audio Output Protection Level	100, 150, 200, 250, 300
Minimum Uncompressed Digital Audio Output Protection Level	100, 150, 200, 250, 300

Field	Allowed Values
Minimum Compressed Digital Video Output Protection Level	400, 500
Minimum Uncompressed Digital Video Output Protection Level	100, 250, 270, 300
Minimum Analog Television Output Protection Level	100, 150, 200

2.4.2 Widevine

Kaltura uDRM supports adding Widevine policies based on the Widevine compliancy standards for the following outputs:

- EMI (CGMS-A)
- APS (Macrovision)
- CIT (Constrained Image Trigger)
- HDCP (High Bandwidth Digital Content Protection)

2.4.3 Apple Fairplay

Kaltura uDRM supports the following policies in the Fairplay CKC response, as defined in the Apple Fairplay development guidelines:

- Video Rental – the FPS implementation will not decrypt the response if the rental has expired
- Video Lease – allocating a lease period per device slot, to control simultaneous playback across multiple FPS enabled devices

2.5 Security and Fault Tolerance

2.5.1 IP Protection

The encryption endpoints are only accessible by whitelisted encryption components verified by Kaltura. These include the internal Kaltura encryption modules and external modules integrating with the uDRM interface.

2.5.2 Key Obfuscation

All encryption keys are encrypted before storing them in the internal uDRM database. The encryption/decryption keys are stored separately in configuration files.

When using the Kaltura encryption modules that perform on-the-fly encryption, in the case of compromise of encrypted content, a key rotation can be performed on-the-fly to change the content encryption keys.

2.5.3 MPAA Auditing

Kaltura is undergoing auditing by the MPAA for the SAE16 qualification report (http://ssae16.com/SSAE16_overview.html) in all of the Kaltura data centers and facilities.

2.5.4 Fault Tolerance

The uDRM module is structured as an *n+m* structure, with DB on-the-fly scalability (using Couchbase clustering).

Data is replicated between all Kaltura data centers using encrypted XDCR (Cross Data Center Replication) as detailed in the Couchbase XDCR data encryption documentation (http://docs.couchbase.com/admin/admin/XDCR/xdcr-dataEncryption.html).

2.6 Error Codes

2.6.1 Format

In the case of an error , the following data format is returned:

```
{
   'message' : "Missing custom data file_id",
   "status_code" : "400",
   "reference" : 12345
}
```

Where :

- `message`: a string message depicting the error
- `status_code`: the status code of the error
- `reference`: an internal reference ID to track the error stack

2.6.2 Supported Response Codes

The following response codes can be returned by the Kaltura uDRM module:

- 200 – successful action
- 400 – invalid call (missing parameter)
- 401 – unauthorized call (error in signature)
- 404 – missing module
- 500 – internal server error

The above codes are returned as status codes in the error code response, along with the HTTP response code.

3 Use Cases

3.1 Kaltura MediaPrep

The most straight forward use case for usage of the Kaltura uDRM module is by leveraging the Kaltura end to end content preparation workflow – MediaPrep. The Kaltura MediaPrep solution is seamlessly integrated with the uDRM encryption process, generating encryption keys for both pre encrypted and on the fly encrypted files, to be used by the Kaltura pre encryption module or the Kaltura on the fly packager respectively

In addition, the Kaltura Universal Studio Player is integrated with the uDRM module for license generation for all uDRM supported DRM providers

When using external players, license requests can be generated according to section 2.3.4

3.2 External Encryption Modules

When using external encryption providers (pre-encryption or on the fly packagers), the integration with the Kaltura uDRM module is with the respective encryption end point (with the exact end point dependant on the DRM provider) – see section 2.3.4.

The encryption data returned by the Kaltura uDRM encryption service can then be used by the external encryption module to encrypt the respective asset

As in the use case above, the license generation flow is handled by the uDRM license endpoint, and is integrated within the Kaltura Universal Studio Player.

When using external players, license requests can beU generated according to section 2.3.4