VUDRM Documentation

VUALTO

Jan 25, 2022

Contents

The VUDRM platform provides a variety of services that allow you to protect your OTT video content; It has device agnostic support for Microsoft PlayReady, Google Widevine, and FairPlay Streaming and is supported by all major players.

If you have any questions or need assistance in using this platform, please contact info@vualto.com.

CHAPTER 1

Contents:

1.1 USER GUIDE

Our VUDRM Admin page gives you the tools you will need to test our DRM services with your content. If you need assistance or clarification in relation to anything in this guide, please contact support@vualto.com.

1.1.1 Login

To access the VUDRM Admin site, please go to the Admin Login page - login using the credentials on your onboarding documentation.

1.1.2 Dashboard

Once you have successfully logged in, you will be redirected to the `Dashboard` page.

The dashboard will show you an overview of all DRM licenses served with your content.

By clicking on the `Range` dropdown field, you can set custom time frames to get an overview for that time period.

1.1.3 Configuration

Client Info

The `Client Info` section contains the details of how your account is configured, such as which DRM providers are enabled and which URLs have been set to retrieve each service's DRM licenses.

If you require amendments to your account, please contact support@vualto.com.

Client Info - Edit default token policies

At the bottom of the `Client Info` page, you can view each DRM service's default token policy. This can be edited to give more control over how a service will act by default.

Clicking `Edit Token Policy` will load the JSON editor.

Please ensure that the JSON has been entered correctly, otherwise you will be informed of an error and will not be able to save the amended policy.

When the amends are complete, click `Close` and then `Save`.

Each DRM service has limitations to how their policy can be altered. Please read the links below for more details:

- DRM Policy
- PlayReady DRM Policy
- Fairplay DRM Policy
- Widevine DRM Policy

VUDRM Token

In this section, you can create, validate, decrypt, encode, and decode VUDRM Tokens.

There are a variety of templates to choose from which will automatically populate the `Policy` text box, please ensure that you replace all default values (such as `{"polend":"DD-MM-YYYY HH:MM:SS"}` to `{"polend":"09-05-2021 12:30:00"}`). When the default values have been changed, you can then generate a token with those parameters.

The buttons on the page do the following:

- `Generate` - This will generate a token to include the policy you have entered
- `Validate Policy` - This will ensure the policy is valid
- `Validate Token` - Validate the token created
- `Decrypt Token` - Decrypts the token
- `Encode Token` - URL Encodes the token
- `Decode Token` - URL Decodes the token

Once a token has been generated, a new button will appear - `Use Token To Test Your Stream`.

Clicking `Use Token To Test Your Stream` will load the `Test Your Stream` player with the generated token in the `VUDRM Token` text box.

For an in depth guide of how our tokens work - please refer to the VUDRM Token documentation.

VUDRM Encryption Keys

Within this section, you can call the `CPIX Key Provider` to fetch VUDRM Encryption Keys in CPIX XML document format as well as the CPIX keys as JSON which is generated at the same time, both of which are available to copy.

To create an Encryption key, do the following:

- Select which Key Provider API you wish to use by using the `Key Provider API`
- Select which DRM Providers by changing the relevant providers button to `YES`

- Enter a relevant name in the `Content ID` text field
- Click `Generate`
- Click `Copy` to copy the newly created key

We also have our `Legacy JSON Key provider` (available on the `Key Provider API` dropdown) to use if your product requires it. However, we recommend you use the `CPIX Key Provider` whenever possible.

For more information - please refer to our Encryption Key Provision documentation.

Test Your Stream

Integrated into the VUDRM Admin site is our internal demo player which can be used for testing your streams.

A VUDRM Token with an open policy is loaded into the `VUDRM Token` field used by the player. You can copy and paste a token you created in the `VUDRM Token` section of the `Info` page. This can be used to test content with your variation of the token. Alternatively, clicking the `Use Token To Test Your Stream` button within the `VUDRM Token` page will automatically enter that token into the Token field used by the player.

The fields are as follows:

- `VUDRM Token` - Token used by the player for the stream
- `DASH Stream URL` - DASH stream to test
- `HLS Stream URL` - HLS stream to test

If you select the `Use Custom License Server URLs?` button, you will be presented with three fields available to edit:

- `Widevine License Server URL`
- `Playready License Server URL`
- `Fairplay License Server URL`

If these custom fields have values, they will override the default License Server URLs set within the client configuration.

Clicking `Load Player` will then load the content with the conditions of the filled in fields.

1.1.4 User

On the `User` section of the VUDRM Admin site, you have the option of changing your password.

1.1.5 Health

The `Health` page gives an overview of the general health for each DRM or DRM related service we provide. If one of these services are down, it will be down throughout all regions.

By default, the page is set to auto-refresh every 60 seconds, but you can toggle this to `No` if not required.

The world map contains region specific services. A regions health colour will change in the following circumstances:

- `GREEN` - All services are operational
- `AMBER` - 1 or more services are not operational
- `RED` - All services are not operational

1.2 DEVELOPER DOCUMENTATION

The purpose of this documentation is to explain the usage and functionality of the various VUDRM services. If you have any questions or need assistance in using this platform, please contact info@vualto.com.

1.2.1 PACKAGING GUIDES

AWS Media Services

In order to use our SPEKE API you will need to set up an API gateway that adds the header `API-KEY` with your API key and with your client name in place of `client-name` in "https://speke.vudrm.tech/client-name/speke". For more information on how to setup the API Gateway see the *API Gateway* section.

Media Packager

1. Go to AWS Media Packager.

- Log into the AWS console
- Either type "Media Packager" into the search box and then select "MediaPackager" or open the "Services" drop down in the top left of the screen and select "MediaPackager" from under the "Media Services" header.

2. Select or create a channel.

- To create a channel press the "Create" button in the top right corner, enter an ID for the channel and press create.

3. Add an endpoint to the channel.

- Press the "Add/edit endpoints" button
- Press the "Add" button to create a new endpoint

4. Assign an ID to the endpoint.

- You can also add a description of the endpoint if needed

5. Set the "Manifest Name" to "manifest".

6. Set the type of stream.

- Use DASH-ISO if you want to use Playready or Widevine DRM
- Use Apple HLS if you want to use Fairplay DRM

7. Select the option to “Encrypt content”.

8. Add a Resource ID.

9. Add appropriate System IDs.

- If you wish to use Widevine DRM use “edef8ba9-79d6-4ace-a3c8-27dcd51d21ed” as the System ID (only if output group is DASH ISO)
- If you wish to use Playready DRM use “9a04f079-9840-4286-ab92-e65be0885f95” as the System ID (only if output group is DASH ISO)
- If you wish to use Fairplay DRM use “94ce86fb-07ff-4f43-adb8-93d2fa968ca2” as the System ID (only if output group is APPLE HLS)

10. Set the URL to that of your API gateway.

11. Add an appropriate Role ARN.

12. Press the “Save” button under the list of endpoint.

13. To test the content, copy the endpoints URL.

14. Load into a player to show the content working.

Widevine

- Open “https://admin.vudrm.tech/” in a new Google Chrome tab and log in, if you do not have a log in please contact support@vualto.com
- Select “Configuration” from the left hand navigation menu and then select the “Test Your Stream” tab
- Paste the endpoints URL into the “DASH Stream URL” box
- Press “Load Player”

Media Convert

1. Go to AWS S3.

- Log into the AWS console
- Either type “S3” into the search box and then select “S3” or open the “Services” drop down in the top left of the screen and select “S3” from under the “Storage” header.

2. Create or select a bucket.

3. Create a new folder.

- Press the “Create folder” button

- Enter a name for the folder (Remember this as you will need it later)
- Press the "Save" button

4. Go to AWS Elemental MediaConvert in a new tab.

- Log into the AWS console in a new tab
- Either type "Media Convert" into the search box and then select "MediaConvert" or open the "Services" drop down in the top left of the screen and select "MediaConvert" from under the "Media Services" header.

5. Create a new job by clicking the "Create job" button in the top right of the screen.

- If you cannot see the "Create job" button ensure you can see the "Recent jobs" list by clicking the icon with 3 lines in the top left of the screen and then selecting "jobs"

6. Set "Input 1" to your source clip.

7. Add an output group by clicking the "Add" button on the left of the screen next to "Output groups".

- If you are using **Widevine** or **Playready** select "DASH ISO"
- If you are using **Fairplay** select "Apple HLS"

8. Set the destination S3 bucket by pressing the "Browse" button.

- Select your bucket from the "S3 bucket" drop down
- Select the folder you created earlier from the "Location" drop down (Should be something like "YOUR-BUCKET/YOUR-FOLDER-NAME/")
- Confirm selection by pressing "Choose" button

9. After selecting a destination with the "Browse" button, click in the "Destination" box and add "manifest" to the end of the destination.

10. Set the "Segment control" to "Segmented files" by clicking the drop down under "Segment control" and clicking "Segmented files".

11. Add DRM to the output group. (Only use one of the below)

Widevine (only if output group is DASH ISO)

- Press the toggle next to "DRM Encryption" to add the DRM information.
- Add a "Resource ID"
- Set the "System ID" to "edef8ba9-79d6-4ace-a3c8-27dcd51d21ed"
- Set the "URL" to that of your API gateway

Playready (only if output group is DASH ISO)

- Press the toggle next to “DRM Encryption” to add the DRM information.
- Add a “Resource ID”
- Set the “System ID” to “9a04f079-9840-4286-ab92-e65be0885f95”
- Set the “URL” to that of your API gateway

Fairplay (only if output group is Apple HLS)

- Press the toggle next to “DRM Encryption” to add the DRM information.
- Set “Encryption-method” to “SAMPLE-AES”
- Set “Key provider type” to “SPEKE”
- Set “Initialization vector in manifest” to “Exclude”
- Add a “Resource ID”
- Set the “System ID” to “94ce86fb-07ff-4f43-adb8-93d2fa968ca2”
- Set the “URL” to that of your API gateway
- Set the “Constant initilization vector” to “00000000000000000000000000000000”

12. Configure the outputs “Outputs”. (Use the same one as above)

Widevine

- Press the “Add output” button.
- Set the “Name modifier” for “Output 1” to “video”
- Set the “Name modifier” for “Output 2” to “audio”
- Select “Output 1” from the “Output groups” section on the left of the screen
- Select “Video” if not already selected and set the “Bitrate (bits/s)” to “5000000”
- Select “Audio 1” and click the “Remove audio” button
- Select “Output 2” from the “Output groups” section on the left of the screen
- Select “Video” if not already selected and click the “Remove video” button

Playready

- Press the “Add output” button.
- Set the “Name modifier” for “Output 1” to “video”
- Set the “Name modifier” for “Output 2” to “audio”
- Select “Output 1” from the “Output groups” section on the left of the screen
- Select “Video” if not already selected and set the “Bitrate (bits/s)” to “5000000”
- Select “Audio 1” and click the “Remove audio” button

- Select “Output 2” from the “Output groups” section on the left of the screen
- Select “Video” if not already selected and click the “Remove video” button

Fairplay

- Set the “Name modifier” for “Output 1” to “video-audio”
- Select “Output 1” from the “Output groups” section on the left of the screen
- Select “Video” if not already selected and set the “Bitrate (bits/s)” to “5000000”

13. Set the “IAM role” in the “Job settings”.

- Click “Settings” under “Job settings”
- Select an appropriate IAM from the dropdown under “IAM role”

14. Press the “Create” button in the bottom right of the screen.

15. Wait approx. 5 minutes.

- On the “Job summary” page you can press the “Refresh” button to check if your job is complete. (You will know it has finished once there is a finish time)

16. To test the content, return to S3 and locate the folder you made earlier.

17. Right click the folder you created a select the “Make public” option followed by pressing the “Make public” button.

18. Then open your folder and click the manifest file.

- If you selected “DASH ISO” as the output group, the file should be called “manifest.mpd”
- If you selected “Apple HLS” as the output group, the file should be called “manifest.m3u8”

19. Load into a player to show the content working. (Use the same one as above)

Widevine

- Open “https://admin.vudrm.tech/” in a new Google Chrome tab and log in, if you do not have a log in please contact support@vualto.com
- Select “Configuration” from the left hand navigation menu and then select the “Test Your Stream” tab
- Return to your S3 tab and copy the link from the bottom of the page and paste it into “DASH Stream URL” box, it should look something like “https://s3-eu-west-1.amazonaws.com/YOUR-BUCKET/YOUR-FOLDER-NAME/manifest.mpd”
- Press “Load Player”

API Gateway

Importing the API from Swagger

After navigating to API Gateway Service in AWS and clicking `Create API`, simply click `Import` for a REST API. Then ensure the protocol is set to `REST` and `Import from Swagger or Open API 3` is selected. You can then paste the JSON below and click `Import`.

```
{
  "swagger": "2.0",
  "info": {
    "title": "speke proxy"
  },
  "basePath": "/speke",
  "schemes": [
    "https"
  ],
  "paths": {
    "/": {
      "post": {
        "produces": [
          "application/json"
        ],
        "parameters": [
          {
            "name": "API-KEY",
            "in": "header",
            "required": false,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "200 response"
          }
        }
      }
    }
  }
}
```

API Gateway setup

Once imported you will need to set up the post method by clicking `Set up now`.

- Set `Integration Type` to be `HTTP`
- Check `Use HTTP Proxy integration`
- Set the `Endpoint URL` to be `https://speke.vudrm.tech/<your-client-name>/speke`

With these options set click `Create`.

Then select `Integration Request` and under the `HTTP Headers` section add a header with the name `API-KEY` and `Mapped from` set to `'<your-api-key>'` ensuring to include the single quotes surrounding your API key.

Now in the `Actions` dropdown select `Deploy API`.

From the `Deployment stage` dropdown select `[New Stage]`, and enter an appropriate name for the stage in the box below.

You will then be sent to the stage editor for the stage you have just created and deployed to. On this page there will be your `Invoke URL`. This is the URL that can be used by other AWS services to make requests to our SPEKE API.

NGINX VOD Module

Our CPIX Key Provider API has an endpoint that is dedicated to providing the key information needed to integrate with Kaltura's NGINX VOD Module.

You can request encryption keys in the format required by the NGINX VOD Module by performing the following command.

```
  curl --location --request GET 'https://cpix.vudrm.tech/v1/keys/<client-name>/nginx/
↪<content-id>' \
    --header 'api-key: <api-key>'
```

This will return the following response:

```
[
    {
        "pssh": [
            {
                "data": "<base64 encoded PlayReady PSSH data>",
                "uuid": "9a04f079-9840-4286-ab92-e65be0885f95"
            },
            {
                "data": "<base64 encoded Widevine PSSH data>",
                "uuid": "edef8ba9-79d6-4ace-a3c8-27dcd51d21ed"
            }
        ],
        "key": "<base64 encoded encryption key>",
        "key_id": "<base64 encoded key id>",
        "iv": "<base64 encoded iv>"
    }
]
```

Below you can find a full example of an NGINX config file that will integrate the VOD module with VUDRM. This example is a modified version of the example given by The New York Times in their repo containing the Docker files necessary to run the VOD module as a Docker container.

This modifications to the example config are as follows: You will need the addition of this to activate just in time DRM on all endpoints.

```
vod_drm_enabled on;
vod_drm_request_uri "/<client-name>/nginx/<content-id>";
vod_drm_upstream_location /drm;
```

The addition of this proxy pass inorder to add your api key to the request made to our CPIX Key Provider API.

```
location /drm {
  internal;
  proxy_pass "https://cpix.vudrm.tech/v1/keys";
  proxy_set_header API-KEY <api-key>;
}
```

This will give you support for DASH with Widevine and PlayReady. To support HLS with Fairplay you will need to add the following to your hls endpoint.

```
vod_hls_encryption_method sample-aes;
vod_hls_encryption_key_uri "skd://fairplay-license.vudrm.tech/v2/license/<content-id>
↪";
vod_hls_encryption_key_format "com.apple.streamingkeydelivery";
vod_hls_encryption_key_format_versions "1";
```

NGINX Config - Full Example

```
worker_processes  auto;

events {
        use epoll;
}

http {
        log_format  main  '$remote_addr $remote_user [$time_local] "$request" '
                '$status "$http_referer" "$http_user_agent"';

        access_log  /dev/stdout  main;
        error_log   stderr debug;

        default_type  application/octet-stream;
        include       /usr/local/nginx/conf/mime.types;

        sendfile    on;
        tcp_nopush  on;
        tcp_nodelay on;

        vod_mode                            local;
        vod_metadata_cache                  metadata_cache 16m;
        vod_response_cache                  response_cache 512m;
        vod_last_modified_types             *;
        vod_segment_duration                9000;
        vod_align_segments_to_key_frames    on;
        vod_dash_fragment_file_name_prefix "segment";
        vod_hls_segment_file_name_prefix    "segment";

        vod_drm_enabled on;
        vod_drm_request_uri "/vualto-demo/nginx/test-nginx";
        vod_drm_upstream_location /drm;

        vod_manifest_segment_durations_mode accurate;

        open_file_cache          max=1000 inactive=5m;
        open_file_cache_valid    2m;
        open_file_cache_min_uses 1;
        open_file_cache_errors   on;

        aio on;

        server {
                listen 80;
                server_name localhost;
```

(continues on next page)

(continued from previous page)

```
                root /opt/static;

                location ~ ^/videos/.+$ {
                        autoindex on;
                }

                location /hls/ {
                        vod hls;
                        vod_hls_encryption_method sample-aes;
                        vod_hls_encryption_key_uri "skd://fairplay-license.vudrm.tech/
↪v2/license/test-nginx";
                        vod_hls_encryption_key_format "com.apple.streamingkeydelivery
↪";
                        vod_hls_encryption_key_format_versions "1";

                        alias /opt/static/videos/;
                        add_header Access-Control-Allow-Headers '*';
                        add_header Access-Control-Allow-Origin '*';
                        add_header Access-Control-Allow-Methods 'GET, HEAD, OPTIONS';
                }

                location /thumb/ {
                        vod thumb;
                        alias /opt/static/videos/;
                        add_header Access-Control-Allow-Headers '*';
                        add_header Access-Control-Allow-Origin '*';
                        add_header Access-Control-Allow-Methods 'GET, HEAD, OPTIONS';
                }

                location /dash/ {
                        vod dash;
                        alias /opt/static/videos/;
                        add_header Access-Control-Allow-Headers '*';
                        add_header Access-Control-Allow-Origin '*';
                        add_header Access-Control-Allow-Methods 'GET, HEAD, OPTIONS';
                }

                location /drm {
                        internal;
                        proxy_pass "https://cpix.vudrm.tech/v1/keys";
                        proxy_set_header API-KEY <api-key>;
                }
        }
}
```

Shaka Packager

Setup using Docker

Setup Shaka Packager using the commands below:

```
# Pull and run shaka packager
docker pull google/shaka-packager
docker run -v /<file_path_to_content>/:/media -it --rm google/shaka-packager
```

(continues on next page)

(continued from previous page)

```
# Test that setup is correct by getting stream info
packager input=/media/<name_of_content>.mp4 --dump_stream_info
```

If successful, the expected output of the stream dump should look like this:

```
File "/media/test.mp4":
Found 1 stream(s).
Stream [0] type: Video
 codec_string: avc1.640028
 time_scale: 12288
 duration: 17078784 (1389.9 seconds)
 is_encrypted: false
 codec: H264
 width: 1920
 height: 810
 pixel_aspect_ratio: 1:1
 trick_play_factor: 0
 nalu_length_size: 4

Packaging completed successfully.
```

Get CPIX Documents

CPIX Documents are required to give Shaka Packager the necessary information in order to encrypt your media.

You can retrieve these documents from the VUDRM Admin site.

For more information, please view our documentation on CPIX Document Requests and our CPIX Key Provider API.

Examples of Each Document

CPIX Document

Below is an example of a CPIX Document with all `DRM Providers` selected.

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc" xmlns:xsi=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:cpix="urn:dashif:org:cpix"␣
↪xsi:schemaLocation="urn:dashif:org:cpix cpix.xsd">
    <cpix:ContentKeyList>
        <cpix:ContentKey kid="11111111-1111-1111-1111-111111111111" explicitIV=
↪"MjlkMzNlYzVjYWIyZmRmZg==" commonEncryptionScheme="cenc">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:PlainValue>3wFFjPspsydY/t2uSPeE6w==</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
    </cpix:ContentKeyList>
    <cpix:DRMSystemList>
        <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
            <cpix:PSSH><PlayReady_PSSH></cpix:PSSH>
```

(continues on next page)

(continued from previous page)

```
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
            <cpix:PSSH><Widevine_PSSH></cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="33333333-3333-3333-33333-33333333333" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
            <cpix:URIExtXKey><FairPlay_URIExtXKey</cpix:URIExtXKey>
            <cpix:HLSSignalingData playlist="master"><master_playlist_value></
↪cpix:HLSSignalingData>
            <cpix:HLSSignalingData playlist="media"><media_playlist_value></
↪cpix:HLSSignalingData>
        </cpix:DRMSystem>
    </cpix:DRMSystemList>
    <cpix:ContentKeyUsageRuleList></cpix:ContentKeyUsageRuleList>
</cpix:CPIX>
```

Package your content

Now you have retrieved the relevant documentation, you can package content with Shaka Packager.

The values from the CPIX documents will need to be copied and pasted to the corresponding section in the examples below.

For more information, please refer to the documentation from Google on Shaka Packager.

Packaging Content with Widevine and PlayReady DRM

Below is an example of how we can send a request to Shaka Packager to encrypt and package our content with Widevine and PlayReady DRM.

Please note, the `Widevine_PSSH` from the CPIX Document is in Base64 format. Shaka Packager requires this to be in Hex format which can be achieved by running the following command `echo '<Widevine_PSSH>' | od -A n -t x1 | sed 's/ *//g'`:

```
  packager \
    in=/media/<name-of-content>.mp4,stream=audio,output=/media/<name-of-output-content>
↪.mp4,drm_label=AUDIO \
    in=/media/<name-of-content>.mp4,stream=video,output=/media/<name-of-output-content>
↪.mp4,drm_label=VIDEO \
    --enable_raw_key_encryption \
    --keys label=AUDIO:key_id=<key_id_hex_value>:key=<content_key_hex_value>,
↪label=VIDEO:key_id=<key_id_hex_value>:key=<content_key_hex_value> \
    --pssh <Widevine_PSSH> \
    --protection_systems Widevine,PlayReady \
    --mpd_output /media/<name-of-manifest>.mpd
```

Packaging Content with FairPlay DRM

Below is an example of how we can send a request to Shaka Packager to encrypt and package our content with FairPlay DRM.

```
  packager \
    in=media/<name-of-content>.mp4,stream=audio,output=media/<name-of-output-content>.
↪mp4,drm_label=AUDIO \
    in=media/<name-of-content>.mp4,stream=video,output=media/<name-of-output-content>.
↪mp4,drm_label=VIDEO \
    --protection_scheme cbcs \
    --enable_raw_key_encryption \
    --keys label=AUDIO:key_id=<key_id_hex_value>:key=<content_key_hex_value>,
↪label=VIDEO:key_id=<key_id_hex_value>:key=<content_key_hex_value> \
    --protection_systems FairPlay \
    --iv <iv_hex_value> \
    --hls_master_playlist_output media/<name-of-manifest>.m3u8 \
    --hls_key_uri <fairplay_laurl_value>
```

Testing packaged content

Upload your packaged files and manifest to AWS S3 or an equivalent hosting service. You can now test your newly packaged content with our player on the VUDRM Admin site.

For more information, please view our documentation on how to Test Your Stream.

You can confirm a relevant license request has been made by checking the network tab of the browsers dev tools.

Unified Streaming

The recommended approach to integrate with USP will depend on your exact use case. If you will be requesting CPIX documents regularly, e.g. for use with live content, it is recommended to use our CPIX Edge Reverse Proxy to retrieve CPIX documents. If you will be making less frequent calls for CPIX documents, e.g. for one time packaging, you can use either our CPIX Edge Reverse Proxy or request CPIX documents from our CPIX Key Provider API directly.

If you would prefer to not use a CPIX document at all you can also use the *JSON formatted response*.

mp4split

With a CPIX document

A CPIX document can be used in the below mp4split commands to encrypt content. For a full list CPIX mp4split options see Unified Streamings full documentation.

You can encrypt content with a CPIX document stored locally by doing the following commands.

```
curl -XGET -H "API-KEY: <api-key>" \
  "https://cpix.vudrm.tech/v1/cpix/<client-name>/<content-id>" \
  > drm.cpix

mp4split --license-key=$USP_LICENSE_KEY -o $ismName \
  --cpix=drm.cpix \
  video.ismv audio.isma
```

Or you can reference our CPIX Edge Reverse Proxy directly in the mp4split command as follows.

```
mp4split --license-key=$USP_LICENSE_KEY -o $ismName \
  --cpix="http://local-cpix-proxy/v1/cpix/<client-name>/<content-id>" \
  video.ismv audio.isma
```

With JSON keys

Keys requested in JSON format can be used in the below mp4split to encrypt content. For more information about using mp4split you can vist Unified Streamings full documentation.

```
mp4split --license-key=$USP_LICENSE_KEY -o $ismName \
     --iss.key=$key_id_hex:$content_key_hex \
     --iss.license_server_url="https://playready-license.vudrm.tech/rightsmanager.asmx
↪" \
     --widevine.key=$key_id_hex:$content_key_hex \
     --widevine.license_server_url="https://widevine-license.vudrm.tech/proxy" \
     --widevine.drm_specific_data=$widevine_drm_specific_data \
     --hls.client_manifest_version=4 \
     --hls.key=:$content_key_hex \
     --hls.key_iv=$iv_hex \
     --hls.license_server_url=$fairplay_laurl \
     --hls.playout=sample_aes_streamingkeydelivery \
     video.ismv audio.isma
```

Wowza

The encryption keys provided by the Key Provider APIs are compatible with Wowza.

The following steps detail how to configure Wowza to support VUDRM:

Configure Wowza Stream Settings

1. Stop Wowza appending the session ID to the license server URL:
   - how to control the streaming session id
2. Change the `EXT-X-VERSION` for HLS streaming to `5` by setting the `cupertinoExtXVersion` value:
   - how to change the ext x version

Manually Set VUDRM Settings

This method of integrating VUDRM with Wowza uses key files:

- how to secure mpeg dash
- how to secure apple hls streaming

NB: Widevine, PlayReady, and Fairplay key values can be set in the same file.

An example key file is:

```
mpegdashstreaming-cenc-key-id: MT8jItzhV+ay+3QEtoyIxQ==
mpegdashstreaming-cenc-content-key: EcZ9tM1adkoleC1hwFlpQw==
mpegdashstreaming-cenc-algorithm: AESCTR
mpegdashstreaming-cenc-keyserver-widevine: true
mpegdashstreaming-cenc-keyserver-widevine-system-id: edef8ba9-79d6-4ace-a3c8-
↪27dcd51d21ed
mpegdashstreaming-cenc-keyserver-widevine-pssh-data: Igp3b3d6YS1kZW1vSOPclZsG
mpegdashstreaming-cenc-keyserver-playready: true
mpegdashstreaming-cenc-keyserver-playready-system-id: 9a04f079-9840-4286-ab92-
↪e65be0885f95
```

(continues on next page)

(continued from previous page)

```
mpegdashstreaming-cenc-keyserver-playready-license-url: https://playready-license.
↪vudrm.tech/rightsmanager.asmx
mpegdashstreaming-cenc-keyserver-playready-checksum: Mv5YB5EhHKw=
cupertinostreaming-aes128-method: SAMPLE-AES
cupertinostreaming-aes128-url: skd://fairplay-license.vudrm.tech/license/wowza-demo
cupertinostreaming-aes128-key: 24F79075974B8E1BC8AF576925B8458F
cupertinostreaming-aes128-iv: A140A11A450DBC04E67F39B850C13D41
cupertinostreaming-aes128-iv-include-in-chunklist: false
cupertinostreaming-aes128-key-format: com.apple.streamingkeydelivery
cupertinostreaming-aes128-key-format-version: 1
```

The values are:

- `mpegdashstreaming-cenc-key-id`: The `key_id_big` value from the CENC response.
- `mpegdashstreaming-cenc-content-key`: The `content_key` value from the CENC response.
- `mpegdashstreaming-cenc-algorithm`: Must be set to `AESCTR`.
- `mpegdashstreaming-cenc-keyserver-widevine`: Must be set to `true`.
- `mpegdashstreaming-cenc-keyserver-widevine-system-id`: Must be set to `edef8ba9-79d6-4ace-a3c8-27dcd51d21ed`
- `mpegdashstreaming-cenc-keyserver-widevine-pssh-data`: The `widevine_drm_specific_data` value from the CENC Response.
- `mpegdashstreaming-cenc-keyserver-playready`: Must be set to `true`.
- `mpegdashstreaming-cenc-keyserver-playready-system-id`: Must be set to `9a04f079-9840-4286-ab92-e65be0885f95`
- `mpegdashstreaming-cenc-keyserver-playready-license-url`: The `playready_laurl` value from the CENC Response.
- `mpegdashstreaming-cenc-keyserver-playready-checksum`: The `checksum` value from the CENC Response
- `cupertinostreaming-aes128-method`: Must be set to `SAMPLE-AES`.
- `cupertinostreaming-aes128-url`: The `laurl` value from the Fairplay Response
- `cupertinostreaming-aes128-key`: The `key_hex` value from the Fairplay Response
- `cupertinostreaming-aes128-iv`: The `iv_hex` value from the Fairplay Response
- `cupertinostreaming-aes128-iv-include-in-chunklist`: Must be set to `false`.
- `cupertinostreaming-aes128-key-format`: Must be set to `com.apple.streamingkeydelivery`.
- `cupertinostreaming-aes128-key-format-version`: Must be set to `1`.

VUALTO Wowza DRM API

The VUALTO Wowza DRM API is a small Docker web API application which runs on the Wowza server, allowing the provisioning of keyfiles through a simple REST interface. The VUALTO Wowza DRM API has been tested with Wowza Streaming Engine version 4.7.7.

Running using Docker

In this example, the API is exposed on port 9000 and the API key is set to some GUID:

```
docker run -d --restart always -p 9000:80 \
-e WOWZADRMAPI_APIKEY=93640045-31f1-45c0-aa0d-b5682d7c9ac8 \
-e WOWZADRMAPI_KEYFILESPATH=/mnt/keyfiles \
-e WOWZADRMAPI_GRANDCENTRALURL=https://config.vudrm.tech/v1/clients \
-e WOWZADRMAPI_KEYPROVIDERURL=https://keyprovider.vudrm.tech \
-v [wowza-install-dir]/keys:/mnt/keyfiles \
--name wowza-drm-api vualto/wowza-drm-api:1.0.0
```

Supported actions

`<streamid>` must match the name of the Stream to protect. This will be the name of the .key file created on the server. Details about what name should be used for the key file can be found here: https://www.wowza.com/docs/how-to-secure-mpeg-dash-streaming-using-common-encryption-cenc#dash_cenc

All requests to the API must be authorised with a `x-api-key` header, whose value must match that set by the `WOWZADRMAPI_APIKEY` environment variable. A missing or mismatched `x-api-key` header will yield a `401 Unauthorized` status.

`GET /api/keyfile/<streamid>`

Returns the contents of the keyfile for that Wowza stream if it exists, otherwise a 404.

`POST /api/keyfile/<streamid>`

Create or overwrite the keyfile for the Wowza stream.

Keys can be supplied either automatically using the Vualto DRM platform or manually.

Example body for manual keys:

```
{
    "cenc": {
        "keyid": "RNpS70UjCY5oVHd+4yJoHQ==",
        "contentkey": "lnA03mLuxnL3toiRMxV4Zw=="
    },
    "playready": {
        "laurl": "https://playready-license.vudrm.tech/rightsmanager.asmx",
        "checksum": "pcTs6CEp98A="
    },
    "widevine": {
        "psshdata": "IiRkN2EyOTk1Ni0yOTgwLTQzMzgtYjYyNy04N2MxZjA3OWUwOTFI49yVmwY="
    },
    "fairplay": {
        "key": "CBF76BB43B9A54254A5FC20074A3A53B",
        "iv": "1A3F2AA76D84742DD123E3E50E7BC681",
        "laurl": "skd://fairplay-license.vudrm.tech/license/d7a29956-2980-4338-b627-
↪87c1f079e091"
    }
}
```

Example response of the created keyfile:

```
mpegdashstreaming-cenc-key-id: RNpS70UjCY5oVHd+4yJoHQ==
mpegdashstreaming-cenc-content-key: lnA03mLuxnL3toiRMxV4Zw==
mpegdashstreaming-cenc-algorithm: AESCTR
mpegdashstreaming-cenc-keyserver-playready: true
mpegdashstreaming-cenc-keyserver-playready-system-id: 9a04f079-9840-4286-ab92-
↪e65be0885f95
mpegdashstreaming-cenc-keyserver-playready-license-url: https://playready-license.
↪vudrm.tech/rightsmanager.asmx
mpegdashstreaming-cenc-keyserver-playready-checksum: pcTs6CEp98A=
mpegdashstreaming-cenc-keyserver-widevine: true
mpegdashstreaming-cenc-keyserver-widevine-system-id: edef8ba9-79d6-4ace-a3c8-
↪27dcd51d21ed
mpegdashstreaming-cenc-keyserver-widevine-pssh-data:␣
↪IiRkN2EyOTk1Ni0yOTgwLTQzMzgtYjYyNy04N2MxZjA3OWUwOTFI49yVmwY=
cupertinostreaming-aes128-method: SAMPLE-AES
cupertinostreaming-aes128-url: skd://fairplay-license.vudrm.tech/license/d7a29956-
↪2980-4338-b627-87c1f079e091
cupertinostreaming-aes128-key: CBF76BB43B9A54254A5FC20074A3A53B
cupertinostreaming-aes128-iv: 1A3F2AA76D84742DD123E3E50E7BC681
cupertinostreaming-aes128-iv-include-in-chunklist: true
cupertinostreaming-aes128-key-format: com.apple.streamingkeydelivery
cupertinostreaming-aes128-key-format-version: 1
```

`cenc` is required with either `playready` or `widevine`.

Example body using the VUALTO DRM key provider:

```
{
    "contentId": "mycontentid",
    "client": "my-client-name",
    "clientApiKey": "8f30c7d2-a9b9-474e-907b-97a190abd6c4",
    "drmJson": "{\"drm_provider\": \"VUALTO\"}"
}
```

Example response:

```
mpegdashstreaming-cenc-key-id: jQGIareSAu0jZ5MYUQBQlw==
mpegdashstreaming-cenc-content-key: kMvY/VcE9FnWe61SKUnKYw==
mpegdashstreaming-cenc-algorithm: AESCTR
mpegdashstreaming-cenc-keyserver-playready: true
mpegdashstreaming-cenc-keyserver-playready-system-id: 9a04f079-9840-4286-ab92-
↪e65be0885f95
mpegdashstreaming-cenc-keyserver-playready-license-url: https://playready-license.
↪vudrm.tech/rightsmanager.asmx
mpegdashstreaming-cenc-keyserver-playready-checksum: qYKKs3wVDDk=
mpegdashstreaming-cenc-keyserver-widevine: true
mpegdashstreaming-cenc-keyserver-widevine-system-id: edef8ba9-79d6-4ace-a3c8-
↪27dcd51d21ed
mpegdashstreaming-cenc-keyserver-widevine-pssh-data: IgtteWNvbnRlbnRpZEjj3JWbBg==
cupertinostreaming-aes128-method: SAMPLE-AES
cupertinostreaming-aes128-url: skd://fairplay-license.vudrm.tech/license/mycontentid
cupertinostreaming-aes128-key: 74FF8E038FB0F65AA5D01C52596B99A5
cupertinostreaming-aes128-iv: 59E932087732AA68DF8BCAE63230479E
cupertinostreaming-aes128-iv-include-in-chunklist: false
cupertinostreaming-aes128-key-format: com.apple.streamingkeydelivery
cupertinostreaming-aes128-key-format-version: 1
```

`DELETE /api/keyfile/<streamid>`

Remove the keyfile for the specified Wowza stream, returning it's contents before deletion.

Bento4

Installing Bento4

You can download the binaries for your preferred OS from Bento4.

Add the filepath for the binaries to the Bento4 `bin` folder as a value to your `Path` environemnt variables on your computer.

Get CPIX Documents

CPIX Documents are required to give Bento4 the necessary information in order to encrypt your media.

You can retrieve these documents from the VUDRM Admin site.

For more information, please view our documentation on CPIX Document Requests and our CPIX Key Provider API.

Examples of Each Document

CPIX Document

Below is an example of a CPIX Document with all `DRM Providers` selected.

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc" xmlns:xsi=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:cpix="urn:dashif:org:cpix"␣
↪xsi:schemaLocation="urn:dashif:org:cpix cpix.xsd">
    <cpix:ContentKeyList>
        <cpix:ContentKey kid="11111111-1111-1111-1111-111111111111" explicitIV=
↪"MjlkMzNlYzVjYWIyZmRmZg==" commonEncryptionScheme="cenc">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:PlainValue>3wFFjPspsydY/t2uSPeE6w==</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
    </cpix:ContentKeyList>
    <cpix:DRMSystemList>
        <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
            <cpix:PSSH><PlayReady_PSSH></cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
            <cpix:PSSH><Widevine_PSSH></cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="33333333-3333-3333-33333-33333333333" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
            <cpix:URIExtXKey><FairPlay_URIExtXKey</cpix:URIExtXKey>
            <cpix:HLSSignalingData playlist="master"><master_playlist_value></
↪cpix:HLSSignalingData>
```

(continues on next page)

(continued from previous page)

```
            <cpix:HLSSignalingData playlist="media"><media_playlist_value></
↪cpix:HLSSignalingData>
        </cpix:DRMSystem>
    </cpix:DRMSystemList>
    <cpix:ContentKeyUsageRuleList></cpix:ContentKeyUsageRuleList>
</cpix:CPIX>
```

Package your content

Now you have retrieved the relevant documentation, you can package content with Bento4.

The values from the CPIX documents will need to be copied and pasted to the corresponding section in the examples below.

For more information, please refer to the documentation from Bento4.

Packaging Content with Widevine and PlayReady DRM

Below is an example of how we can send a request to Bento4 to encrypt our content with Widevine and PlayReady DRM and package it:

```
mp4dash \
--encryption-key=<key_id_hex_value>:<content_key_hex_value> \
--playready \
--widevine-header=#<Widevine_PSSH> \
--mpd-name=<name_of_manifest>.mpd \
<name_of_video_file>.mp4 \
<name_of_audio_file>.mp4
```

Packaging Content with FairPlay DRM

Below is an example of how we can send a request to Bento4 to encrypt our content with FairPlay DRM and package it:

```
mp4dash \
--hls \
--encryption-key=<key_id_hex>:<content_key_hex_value>:<iv_hex_value> \
--encryption-cenc-scheme=cbcs \
--fairplay-key-uri=<fairplay_laurl_value> \
--hls-master-playlist-name=<name_of_manifest>.m3u8 \
<name_of_video_file>.mp4 \
<name_of_audio_file>.mp4
```

Testing packaged content

Upload your packaged files and manifest to AWS S3 or an equivalent hosting service. You can now test your newly packaged content with our player on the VUDRM Admin site.

For more information, please view our documentation on how to Test Your Stream.

You can confirm a relevant license request has been made by checking the network tab of the browsers dev tools.

1.2.2 PLAYER INTEGRATIONS

Bitmovin

Bitmovin is an industry leading HTML5 video player.

The vuplay-bitmovin repository demonstrates at a lower level how to integrate VUDRM with Bitmovin.

Basic setup

```
var container = document.getElementById("my-player");
var vudrmToken = "<your-vudrm-token>";

var playerConfig = {
  key: "<your-bitmovin-player-key>"
};

var source = {
  title: "Getting Started with the Bitmovin Player",
  description:
    "Now you are ready to embed the Bitmovin Player into your own website :)",
  dash: "<your-mpeg-dash-stream-url>",
  hls: "<your-hls-stream-url>",
  poster: "https://bitmovin-a.akamaihd.net/content/MI201109210084_1/poster.jpg"
};
```

The source object above has a `drm` property, within this you can add the appropriate DRM configuration.

Widevine example

```
source.drm = {
  widevine: {
    LA_URL: "https://widevine-license.vudrm.tech/proxy",
    prepareMessage: function(keyMessage) {
      return JSON.stringify({
        token: vudrmToken,
        drm_info: Array.apply(null, new Uint8Array(keyMessage.message)),
        kid: "<your-content-key-id>"
      });
    }
  }
};
```

PlayReady example

```
source.drm = {
    playready: {
        LA_URL: "https://playready-license.vudrm.tech/rightsmanager.asmx?token=" +␣
↪encodeURIComponent(vudrmToken);
    }
}
```

FairPlay example

```
source.drm = {
  fairplay: {
    certificateURL: "<your-fairplay-cert>",
    LA_URL: "<fairplay-license-server-url>",
    certificateHeaders: {
      "x-vudrm-token": [vudrmToken]
    },
    headers: {
      "Content-Type": "application/json"
    },
    prepareMessage: function(keyMessageEvent, keySession) {
      return JSON.stringify({
        token: vudrmToken,
        contentId: keySession.contentId,
        payload: keyMessageEvent.messageBase64Encoded
      });
    },
    prepareContentId: function(rawContentId) {
      var tmp = rawContentId.split("/");
      return tmp[tmp.length - 1];
    },
    prepareCertificate: function(cert) {
      return new Uint8Array(cert);
    },
    prepareLicense: function(license) {
      return new Uint8Array(license);
    },
    licenseResponseType: "arraybuffer"
  }
};
```

dash.js

dash.js is an initiative of the DASH Industry Forum to establish a production quality framework for building video and audio players that play back MPEG-DASH content using client-side JavaScript libraries leveraging the Media Source Extensions API set as defined by the W3C.

The vuplay-dashjs repository demonstrates at a lower level how to integrate VUDRM with dash.js.

Basic setup

```
var streamUrl = "<your-stream-url>";
var vudrmToken = "<your-vudrm-token>";

var player = dashjs.MediaPlayer().create();
```

Widevine example

```
var overrideKeySystemWidevine = function() {
  return {
```

(continues on next page)

(continued from previous page)

```
    getInitData: function(cpData, kid) {
      this.kid = kid;
      if ("pssh" in cpData) {
        return BASE64.decodeArray(cpData.pssh.__text).buffer;
      }
      return null;
    },

    getLicenseRequestFromMessage: function(message) {
      var body = {
        token: vudrmToken,
        drm_info: Array.apply(null, new Uint8Array(message)),
        kid: this.kid
      };
      body = JSON.stringify(body);
      return body;
    }
  };
};

var overrideProtectionKeyController = function() {
  var parent = this.parent;

  return {
    getSupportedKeySystemsFromContentProtection: function(cps) {
      var cp, ks, ksIdx, cpIdx;
      var supportedKS = [];
      var keySystems = parent.getKeySystems();
      var cpsWithKeyId = cps.find(function(element) {
        return element.KID !== null;
      });

      if (cps) {
        for (ksIdx = 0; ksIdx < keySystems.length; ++ksIdx) {
          ks = keySystems[ksIdx];
          for (cpIdx = 0; cpIdx < cps.length; ++cpIdx) {
            cp = cps[cpIdx];
            if (cp.schemeIdUri.toLowerCase() === ks.schemeIdURI) {
              var initData = ks.getInitData(cp, cpsWithKeyId.KID);
              if (initData) {
                supportedKS.push({
                  ks: keySystems[ksIdx],
                  initData: initData
                });
              }
            }
          }
        }
      }
      return supportedKS;
    }
  };
};

var widevineLaUrl = "https://widevine-license.vudrm.tech/proxy";

player.setProtectionData({
```

(continues on next page)

(continued from previous page)

```
  "com.widevine.alpha": {
    serverURL: widevineLaUrl,
    httpRequestHeaders: {}
  }
});

player.attachSource(streamUrl);
```

PlayReady example

```
var playReadyLaUrl =
  "https://playready-license.vudrm.tech/rightsmanager.asmx?token=" +
  encodeURIComponent(vudrmToken);

player.setProtectionData({
  "com.microsoft.playready": {
    serverURL: playReadyLaUrl,
    httpRequestHeaders: {}
  }
});

player.attachSource(streamUrl);
```

No FairPlay support

dash.js does not currently support Fairplay.

JWPlayer

JWPlayer is an industry leading HTML5 video player.

The vuplay-jwplayer repository demonstrates at a lower level how to integrate VUDRM with JWPlayer.

Example of JWplayer Integration

This file can be found in the repo mentioned above here: https://github.com/Vualto/vuplay-jwplayer/blob/master/vuplay-jwplayer.js

```
(function() {
    // Set the mpeg-dash stream URL.
    var dashStreamURL = "<dash-stream-url>";
    // Set the hls stream URL.
    var hlsStreamURL = "<hls-stream-url>";

    // Set the URL to retrieve the fairplay certificate from.
    var fairplayCertURL = "<fairplay-cert-url>";

    // Please login to https://admin.vudrm.tech to generate a VUDRM token.
    var vudrmToken = "<your-vudrm-token>";
```

(continues on next page)

(continued from previous page)

```
    // setup jwplayer, passing the stream URLs and DRM configurations.
    jwplayer("vuplay-container").setup({
        "playlist": [{
            "sources": [{
                "file": dashStreamURL,
                "drm": {
                    "widevine": {
                        "url": "https://widevine-license.vudrm.tech/proxy",
                        "headers": [{
                            "name": "X-VUDRM-TOKEN",
                            "value": vudrmToken
                        }]
                    },
                    "playready": {
                        "url": "https://playready-license.vudrm.tech/rightsmanager.
↪asmx",
                        "headers": [{
                            "name": "X-VUDRM-TOKEN",
                            "value": vudrmToken
                        }]
                    }
                }
            },
            {
                "file": hlsStreamURL,
                "drm": {
                    "fairplay": {
                        "certificateUrl": fairplayCertURL,
                        "processSpcUrl": function (initData) {
                            return "https://" + initData.split("skd://").pop();
                        },
                        "licenseRequestHeaders": [
                            {
                                "name": "Content-type",
                                "value": "arraybuffer"
                            },
                            {
                                "name": "X-VUDRM-TOKEN",
                                "value": vudrmToken
                            }
                        ]
                    }
                }
            }]
        }]
    });
})();
```

Shaka

Shaka Player is a JavaScript library for adaptive video streaming. It plays DASH content without browser plugins using MediaSource Extensions and Encrypted Media Extensions.

The vuplay-shaka repository demonstrates at a lower level how to integrate VUDRM with Google's Shaka player.

Google's Shaka player can be used to playback MPEG-dash streams with Widevine and PlayReady encryption, or

HLS with Fairplay.

Example of Shaka Integration

This file can be found in the repo mentioned above here: https://github.com/Vualto/vuplay-shaka/blob/master/vuplay.js

```
(function () {
    // Set your mpeg dash stream url
    var mpegdashStreamUrl = "<mpeg-dash-stream-url>";

    // Set your HLS stream url
    var hlsStreamUrl = "<hls-stream-url>";

    // Set your fairplay certificate url
    var fairplayCertificateUrl = "<fairplay-certificate-url>";

    // Will get overridden with the one from the manifest but we have to set␣
↪something otherwise shaka will complain!
    var fairplayLicenseServerUrl = "https://fairplay-license.vudrm.tech/license";

    // Please login to https://admin.vudrm.tech to generate a vudrm token.
    var vudrmToken = "<vudrm-token>";

    // A bit of hacky way to detect Safari but will do for demo purposes...
    var isSafari = (navigator.userAgent.indexOf("Safari") != -1 && navigator.
↪userAgent.indexOf("Chrome") == -1);

    // Fetch the fairplay certificate used to generate the fairplay license request
    function getFairplayCertificate() {
        var certRequest = new XMLHttpRequest();
        certRequest.responseType = "arraybuffer";
        certRequest.open("GET", fairplayCertificateUrl, true);
        certRequest.onload = function (event) {
            if (event.target.status == 200) {
                loadPlayer(new Uint8Array(event.target.response));
            } else {
                var error = new Error("HTTP status: " + event.target.status + " when␣
↪getting Fairplay Certificate.");
                onError(error);
            }
        };
        certRequest.send();
    }

    // Returns a shaka player config for use with mpeg-dash streams
    function getNonSafariPlayerConfig() {
        return {
            drm: {
                servers: {
                    "com.widevine.alpha": "https://widevine-license.vudrm.tech/proxy",
                    "com.microsoft.playready": "https://playready-license.vudrm.tech/
↪rightsmanager.asmx"
                }
            }
        }
    }
```

(continues on next page)

(continued from previous page)

```
    // returns a shaka player config for use with HLS streams
    function getSafariPlayerConfig(fairplayCertificate) {
        return {
            drm: {
                servers: {
                    "com.apple.fps.1_0": fairplayLicenseServerUrl
                },
                advanced: {
                    "com.apple.fps.1_0": {
                        serverCertificate: fairplayCertificate
                    }
                },
                initDataTransform: function (initData, initDataType) {
                    if (initDataType == "skd") {
                        // Set the Fairplay license server URL with the one from the␣
↪HLS manifest
                        fairplayLicenseServerUrl = shaka.util.StringUtils.
↪fromBytesAutoDetect(initData);

                        // Create the initData for Fairplay
                        var contentId = fairplayLicenseServerUrl.split("/").pop();
                        var certificate = window.shakaPlayerInstance.drmInfo().
↪serverCertificate;
                        return shaka.util.FairPlayUtils.initDataTransform(initData,␣
↪contentId, certificate);
                    } else {
                        return initData;
                    }
                }
            }
        }
    }

    function loadPlayer(fairplayCertificate) {
        // setup the shaka player and attach an error event listener
        var video = document.getElementById("video");
        window.shakaPlayerInstance = new shaka.Player(video);
        window.shakaPlayerInstance.addEventListener("error", onErrorEvent);

        // configure the DRM license servers
        var playerConfig = isSafari ? getSafariPlayerConfig(fairplayCertificate) :␣
↪getNonSafariPlayerConfig();
        window.shakaPlayerInstance.configure(playerConfig);

        // Something special is needed for the widevine license request.
        window.shakaPlayerInstance
            .getNetworkingEngine()
            .registerRequestFilter(function (type, request) {
                // ignore requests that are not license requests.
                if (type != shaka.net.NetworkingEngine.RequestType.LICENSE)
                    return;

                // set the VUDRM token as a header on the license request
                request.headers["X-VUDRM-TOKEN"] = vudrmToken;

                // custom fairplay license request body required
                if (window.shakaPlayerInstance.drmInfo().keySystem == "com.apple.fps.
↪1_0") {
```

(continues on next page)

(continued from previous page)

```
                    request.headers["Content-Type"] = "ArrayBuffer"
                    request.uris = [fairplayLicenseServerUrl.replace("skd", "https")];
                }
            });

        // load the mpeg-dash or HLS stream into the shaka player
        window.shakaPlayerInstance
            .load(isSafari ? hlsStreamUrl : mpegdashStreamUrl)
            .then(function () {
                console.log("The stream has now been loaded!");
            })
            .catch(onError);
    }

    // Set polyfills required by shaka
    shaka.polyfill.installAll();
    // Check browser is supported and load the player.
    if (shaka.Player.isBrowserSupported()) {
        if (isSafari) {
            // Get the fairplay certificate, once the cert is retrieved then the␣
↪player will be loaded.
            getFairplayCertificate();
        } else {
            loadPlayer();
        }
    } else {
        console.error("This browser does not have the minimum set of APIs needed for␣
↪shaka!");
    }
})();

function onErrorEvent(event) {
    // Extract the shaka.util.Error object from the event.
    onError(event.detail);
}

function onError(error) {
    console.error("Error code", error.code, "object", error);
}
```

THEOplayer

THEOplayer is an industry leading HTML5 video player.

The vuplay-theoplayer repository demonstrates at a lower level how to integrate VUDRM with THEOplayer.

Basic setup

```
<div
  id="vuplay-container"
  class="video-js theoplayer-skin theo-seekbar-above-controls"
></div>
```

```
var streamUrl = "<your-stream-url>";
var vudrmToken = "<your-vudrm-token>";
var containerElement = document.getElementById("vuplay-container");

var player = new THEOplayer.Player(containerElement, {
  libraryLocation: "<your-theoplayerjs-scripts-path>",
  ui: {
    fluid: true
  }
});
```

Widevine example

```
player.source = {
  sources: [
    {
      src: streamUrl,
      type: "application/dash+xml",
      contentProtection: {
        integration: "vudrm",
        token: vudrmToken,
        widevine: {
          licenseAcquisitionURL: "https://widevine-license.vudrm.tech/proxy"
        }
      }
    }
  ]
};
```

PlayReady example

```
player.source = {
  sources: [
    {
      src: streamUrl,
      type: "application/dash+xml",
      contentProtection: {
        integration: "vudrm",
        token: vudrmToken,
        playready: {
          licenseAcquisitionURL:
            "https://playready-license.vudrm.tech/rightsmanager.asmx"
        }
      }
    }
  ]
};
```

FairPlay example

```
player.source = {
  sources: [
    {
      src: streamUrl,
      type: "application/x-mpegurl",
      contentProtection: {
        integration: "vudrm",
        token: vudrmToken
      }
    }
  ]
};
```

Video JS

Video JS is an industry leading HTML5 video player.

The vuplay-videojs repository demonstrates at a lower level how to integrate VUDRM with Video JS.

Example of Video JS Integration

You will need to load the following libraries:

```
    <script src="https://cdnjs.cloudflare.com/ajax/libs/video.js/7.10.2/video.min.js">
↪</script>
    <script src="https://unpkg.com/@videojs/http-streaming@2.3.0/dist/videojs-http-
↪streaming.js"></script>
    <script src="https://unpkg.com/videojs-contrib-eme@3.7.0/dist/videojs-contrib-eme.
↪js"></script>
```

The following code snippet can be found in our vuplay videojs repo.

```
const streamURL = "<your-stream-url>";

const token = "<your-vudrm-token>";

// playready
const playReadyLicenseServerURL =
    "https://playready-license.vudrm.tech/rightsmanager.asmx?token=" +␣
↪encodeURIComponent(token);

// widevine
const widevineLicenseServerURL =
    "https://widevine-license.vudrm.tech/proxy?token=" + encodeURIComponent(token);

// fairplay
const fairplayCertificateUri =
    "https://fairplay-license.vudrm.tech/certificate/<your-client-name>";

var fairplayLicenseUri;

var Utils = {
```

(continues on next page)

(continued from previous page)

```
    uint16ArrayToString: function(array) {
        var uint16Array = new Uint16Array(array.buffer);
        return String.fromCharCode.apply(String, uint16Array);
    }
};

(function () {
    var player = videojs("my-video", {
        autoplay: true,
    });

    // eme extension must be initialised before source is set
    player.eme();

    player.src({
        src: streamURL,
        type: (streamURL.endsWith('.mpd') ? "application/dash+xml" : "application/x-
↪mpegURL"),
        keySystems: {
            "com.apple.fps.1_0": {
                certificateUri: fairplayCertificateUri,
                getContentId: function (emeOptions, initData, eme) {
                    fairplayLicenseUri = "https://" + Utils.
↪uint16ArrayToString(initData).split("skd://").pop();
                    var contentId = fairplayLicenseUri.split("/").pop();
                    return contentId;
                },
                getLicense: function (emeOptions, contentId, keyMessage, callback) {
                    videojs.xhr({
                        url: fairplayLicenseUri,
                        method: 'POST',
                        responseType: 'arraybuffer',
                        body: keyMessage,
                        headers: {
                            'Content-type': 'application/octet-stream',
                            'x-vudrm-token': token
                        }
                    }, (err, response, responseBody) => {
                        if (err) {
                            callback(err)
                            return
                        }
                        callback(null, responseBody)
                    })
                },
            },
            "com.microsoft.playready": playReadyLicenseServerURL,
            "com.widevine.alpha": widevineLicenseServerURL,
        },
    });
})();
```

1.2.3 MOBILE SDKS

Android Widevine SDK

VUDRMWidevine is an Android Archive (AAR) which can be used during the media rendering pipeline to provide a DRM plugin to ExoPlayer 2.12.2 which will work with Vualto DRM workflow. VUDRMWidevine has been developed to specifically manage the session DRM, allowing complete asset and player management.

Current release: v0.3.5 (277)

Requirements

- Minimum SDK version is 19 (Android 4.4)
- Android Studio 4.0.1

Android Studio Integration

1. VUDRMWidevine is distributed from our maven repository, access to which is made possible by adding the following to the repositories closure of your apps top level build.gradle file:

```
allprojects {
        repositories {
                jcenter()
                maven {
                        url "https://maven.drm.technology/artifactory/vudrm-
↪widevine"
                        credentials {
                                username = "mavenUsername"
                                password = "mavenPassword"
                        }
                }
        }
}
```

2. Map the username and password to your credentials for authentication to the maven repository
3. Under your module's build.gradle dependencies add:

```
implementation 'com.vualto.vudrm:widevine:0.3.5'
```

4. Access to our KID plugin is enabled similarly using the same maven repository configuration, but with the url:

   https://maven.drm.technology/artifactory/kid-plugin

   and dependency:

```
implementation 'com.vualto.vudrm:kidplugin:0.3.5'
```

Example Usage

Instances of VUDRMWidevine are associated with specific assets. Offline assets can be tracked using Exoplayers OfflineLicenseHelper, DownloadService, and DownloadTracker classes. Two types of session are available, determined by the current session's VUDRM token policy, and the asset configuration.

For further information about VUDRM please contact us, or refer to our documentation.

Online and Offline Streaming Sessions

Exoplayer 12 introduces more 'under the hood' management of DRM assets, which simplifies the flow for online and offline streaming sessions. For an example of implementation we recommend referring to our multiple asset VUDRM Widevine demo application, which is directly adapted from Exoplayers demo application. To manage the required modification of license calls we have added a simple `VudrmHelper` class to the demo application.

Asset configurations should contain at least a stream `name`,`uri`,`drm_scheme`("widevine"), and a `drm_license_url`. Tokens may be presented using either Exoplayers license header `drm_key_request_properties`, or a URL encoded token may instead be added to the license server URI, eg: `"drm_license_uri": "https://widevine-license.staging.vudrm.tech/proxy?token=vualto-token-value"`.

1. To implement VUDRM Widevine, instantiate an AssetConfiguration using the fluent interface for both online and offline sessions. Minimally you need to provide the contents KID, DRM token, and license server URI.

```
assetConfiguration = new AssetConfiguration.Builder()
        .tokenWith(drmToken)
        .KIDWith(KID)
        .build();
```

2. Once you've built the object you can construct a plugin by instantiating a WidevineCallback object with the asset configuration.

```
MediaDrmCallback callback = new WidevineCallback (assetConfiguration);
```

3. Then you can pass it to ExoPlayer as the component required when creating the DefaultDrmSessionManager object.

```
return new DefaultDrmSessionManager.Builder()
        .setUuidAndExoMediaDrmProvider(C.WIDEVINE_UUID, FrameworkMediaDrm.DEFAULT_
↪PROVIDER)
          .setMultiSession(false)
        .build(mediaDrmCallback);
```

4. For offline sessions, Exoplayers DownloadTracker will create a `WidevineOfflineLicenseFetchTask` to acquire and download the license. After this the download will commence. When the download is complete, Exoplayer will manage the asset for playback.

References: http://google.github.io/ExoPlayer/doc/reference/

Error handling

DRM related errors will be bubbled up to ExoPlayer's event listener system, this should contain the license server's HTTP response code and a transaction ID in the exception if applicable.

Devices tested on:

Internally

- Xiaomi Redmi 5 Plus (8.1.0)
- Nexus 10 (5.1.1)
- Samsung S5 (6.0.1)
- Samsung Tab 10 A5 (6.0.1)
- HP 8 Tablet (4.4.2)
- Huawei Honor 9 (8.0.0)

Known Issues

- Exoplayer 12 for AndroidX introduced a requirement that the PSSH box be present in the main manifest. If the PSSH box is not present in the main manifest then Exoplayers `DefaultDrmSessionManager` will throw this error `MissingSchemeDataException: Media does not support uuid: edef8ba9-79d6-4ace-a3c8-27dcd51d21ed`.
- 32-bit devices displayed issues where a license expiry time (secs) is too large to be handled, it therefore returns 0 seconds remaining and considers the license expired. To work around this, always set the license expiry time in your VUDRM token policy. For further information about VUDRM please contact us, or refer to our documentation.

If you believe you have found any other issue, please contact us at support@vualto.com

Troubleshooting

- Most issues are content related. You can use our demo application to test your own content by updating the `media.exolist.json` with your configurations.
- Errors may also arise because the stream or asset configuration is not correct.
  - Tokens - You can easily eliminate token issues by beginning with an empty policy in the VUDRM token. Please ensure your token is formatted correctly and validates. For the avoidance of doubt, where tokens use dates, the dates should always be in the future. For further information about tokens please refer to our documentation.

If you are not able to play your content after checking it in our demo application please contact support@vualto.com with the demo application logs and the stream configuration used.

Release Notes

v0.3.5 (build 277) on 07/08/2019

- Update license server DNS

- Replaces instances of assetName and assetID with contentID to be consistent across DRM platform

v0.3.4 (build 272) on 22/05/2019

- Widevine Offline Implementation – Download asset and license, and play downloaded asset offline with saved license.

v0.3.3 (build 243) on 15/05/2019

- Build automation updates

v0.3.2 (build 233) on 17/04/2019

- Update dependencies
- Resolve deprecations
- Update to Android Studio 3.3.2 and SDK 28

v0.3.1 (build 216) on 21/02/2019

- Add Widevine provision callback

v0.3.0 (build 204) on 12/01/2018

- Add Widevine provision callback
- Bug fixes and improvements

v0.2.0 on 12/04/2017

- New major exoplanet release
- Bug fixes and improvements

v0.1.0 on 10/03/2017

- Initial release

iOS / tvOS FairPlay SDK

VUDRMFairPlaySDK is available for iOS and tvOS. This documentation describes the steps to integrate and use VUDRMFairPlaySDK on these platforms, and how to configure our demo applications.

Current release: iOS v1.0 (37) tvOS v1.0 (38)

- *Overview*
- *Requirements*
- *Xcode Integration*
- *Information about Application Transport Security (ATS)*
- *Preparation*
- *Example framework usage*
- *Limitations*
- *Known Issues*
- *Troubleshooting*
- *Release Notes*

Overview

VUDRMFairPlaySDK enables Apple's AVPlayer from AVFoundation to securely request licenses from Vualto's VUDRM cloud based DRM platform.

The deployment scenario will allow Online Playback / Download and Offline Playback using an associated rental or persist token.

Make use of Apple's AVPlayer and AVPlayerViewController to present users with the platform default skins, with DRM content, or create your own video player based on AVPlayer.

This SDK has a number of key benefits:

- Complete control over the entire AVPlayer lifecycle
- Numerous optimisations to enable faster playback
- Bitcode support

The SDK is fully supported in both Objective-C and Swift applications.

Multiple asset demo applications written in Swift, and based on Apple's FairPlay SDK example applications, are available on request. Please contact support@vualto.com to request access.

Requirements

- Minimum deployment target of iOS 10.0 and tvOS 9.0 or higher
- Xcode 12.4
- Swift 5.3
- Cocoapods

Xcode Integration

VUDRMFairPlaySDK is distributed using Cocoapods. Projects set up with Cocoapods use an Xcode workspace, so it is important that projects with Cocoapods are always opened using the workspace file instead of the project file. Our simple demo applications demonstrate the required set up for Cocoapods.

If you are integrating VUDRMFairPlaySDK into your own project, please ensure that the project has been set up correctly for Cocoapods. There are numerous online tutorials demonstrating how to do this. You can then replace or merge the `podfile` with the one in our demo application.

It is important that the latest version of Cocoapods is installed before the demo project or your project can be opened without error.

1. You can ensure that you have the latest version of Cocoapods by opening a new shell in the Terminal application and entering:

   ```
   sudo gem install cocoapods
   ```

2. To pull vudrmFairPlaySDK.framework in to the demo project or your project with Cocoapods, ensure that you quit Xcode if it is running, then open the Terminal application and navigate to the project directory, for example:

   ```
   cd /Users/username/Documents/Dev/vudrm-fairplay-demo
   ```

   Then enter:

   ```
   pod install
   ```

   or, where the pod has previously been installed:

```
pod update
```

If you have installed our earlier SDK's and receive this CocoaPod error: `[!] Unable to find a specification for 'vudrmFairPlaySDK'`

you may also need to enter the following:

```
pod install --repo-update
```

The project should now be ready to open and run in Xcode.

Please see below for more information on using the demo application.

If you are unable to use Cocoapods in your project, please contact support@vualto.com to discuss other options.

Information about application transport security (ATS)

iOS/tvOS 9 introduces Application Transport Security (ATS) which restricts the use of insecure HTTP. In order to permit playback of content over insecure HTTP exemptions need to be added into your application's `Info.plist`. For example to disable ATS for any connection you would add the following into your application's `Info.plist`:

```
<dict>
    <key>NSAppTransportSecurity</key>
    <dict>
        <key>NSAllowsArbitraryLoads</key> <true/>
    </dict>
</dict>
```

More information about ATS can be found in Apple's TechNote.

Preparation

Instances of VUDRMFairPlaySDK require some configuration which would normally be populated with an API. Our demo projects, which are based on Apple's FairPlay SDK example applications, are configured using the projects `Streams.plist` file.

To provide complete configuration for each `Stream` object you will require:

- An `skd://` URI entry in the `content_key_id_list`. The correct URI can be obtained by retrieving the manifest and parsing out the `URI` from the `EXT-X-SESSION-KEY` or `EXT-X-KEY`, this can be done in code (see our demo application for an example) or manually with curl in the Terminal app, e.g. enter: `curl https://eample.cloudfront.net/example-demo/examplecontent/examplecontent.ism/.m3u8`.
- The `playlist_url` to correctly prepared content.
- A `name` for the content. This will be the readable display name and is not used by the SDK. This may differ from the `content_id`.
- A `content_id` for the content, being the same content ID that the content was prepared with. This value will always be the last path component of the stream `content_key_id_list` or `skd://` URI entry. Our demo application shows how to retrieve and parse the `content_key_id_list`, however, the `content_id` always needs to be provided for use with offline assets.
- An `is_protected` boolean value which indicates that the `Stream` object is protected with DRM. This setting would not be needed in scenarios where all content is protected, as all assets can be set as protected by default in the source code.

- A `vudrm-token` which must be correctly generated for the type of instance you wish to create. Please see the *Tokens and Licensing* section for further information.
- An optional `renew_interval` value which represents lease renewal interval in seconds. This value is only used where token policy uses `duration_lease`.

For added convenience, VUDRMFairPlaySDK also bubbles up notifications of errors and progress during the licensing request. Errors can be intercepted as follows:

```
@objc func handleVudrmDidError(_ notification: Notification)
{
    if let data = notification.userInfo as? [String: String]
    {
        for (function, error) in data
        {
            print("vudrmFairPlaySDK - \(function) reported error \(error)!")
        }
    }
}
```

And, progress can be monitored using the following:

```
@objc func handleVudrmProgressUpdate(_ notification: Notification)
{
    if let data = notification.userInfo as? [String: String]
    {
        for (function, message) in data
        {
            print("vudrmFairPlaySDK - \(function) reported progress \(message).")
        }
    }
}
```

Example framework usage

We strongly recommend referring to our example iOS / tvOS multiple asset demo application.

Using either of Apples recommended implementations, being AssetResourceLoaderDelegate (AVAssetResourceLoader API) or ContentKeyDelegate (Contentkey API), VUDRMFairPlaySDK performs the two required licensing network tasks, being acquisition of an application certificate, followed by a FairPlay license either for online or offline use.

An application certificate is always required to initialise a request for a FairPlay license.

To invoke an instance of vudrmFairPlaySDK and call for an application certificate using either AssetResourceLoaderDelegate or ContentKeyDelegate, use:

```
self.drm = vudrmFairPlaySDK()
let applicationCertificate = try drm!.requestApplicationCertificate(token: self.
↪vuToken, contentID: contentID)
```

Using the application certificate, a call for a license may then be prepared on device using the same call but passing different streamingContentKeyRequestData options for online and offline requests. The resulting data is then passed back to the vudrmFairPlaySDK to request a license from the KSM for either online or offline use. The implementation of the next steps differ for each method of handling FairPlay requests, they are as follows:

AssetResourceLoaderDelegate:

Online Playback

```
let spcData = try resourceLoadingRequest.streamingContentKeyRequestData(forApp:␣
↪applicationCertificate!,contentIdentifier: assetIDData, options: nil)

let ckcData = try self.drm!.requestContentKeyFromKeySecurityModule(spcData: spcData,␣
↪token: vuToken, assetID: self.contentID, licenseURL: licenseUrl)

if ckcData != nil {
        resourceLoadingRequest.dataRequest?.respond(with: ckcData!)
} else {
        print("Failed to get CKC for the request object of the resource.")
        return
}
```

You should always set the contentType before calling finishLoading() on the resourceLoadingRequest to make sure you have a contentType that matches the key response.

```
resourceLoadingRequest.contentInformationRequest?.contentType =␣
↪AVStreamingKeyDeliveryContentKeyType
resourceLoadingRequest.finishLoading()
```

Download and Offline Playback:

```
let spcData = try resourceLoadingRequest.streamingContentKeyRequestData(forApp:␣
↪applicationCertificate!, contentIdentifier: assetIDData, options:␣
↪[AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey:␣
↪true])

let ckcData = try self.drm!.requestContentKeyFromKeySecurityModule(spcData: spcData,␣
↪token: self.vuToken, assetID: self.contentID, licenseURL: self.licenseUrl, renewal:␣
↪self.renewalInterval)

let persistentKey = try resourceLoadingRequest.
↪persistentContentKey(fromKeyVendorResponse: ckcData!, options: nil)
```

You can now write the persistent content key to disk:

```
try writePersistableContentKey(contentKey: persistentKey, withContentKeyIdentifier:␣
↪contentID)
```

and then provide the content key response to make protected content available for processing before calling finishLoading() on the resourceLoadingRequest.

```
resourceLoadingRequest.dataRequest?.respond(with: persistentKey)
resourceLoadingRequest.finishLoading()
```

The call for an offline license should be made upon download of the asset.

ContentKeyDelegate:

Online Playback

```
let completionHandler = { [weak self] (spcData: Data?, error: Error?) in
        guard let strongSelf = self else { return }
                 if let error = error {
                        keyRequest.processContentKeyResponseError(error)
                                return
                }
        guard let spcData = spcData else { return }
        do {
                guard let ckcData = try strongSelf.drm!.
↪requestContentKeyFromKeySecurityModule(spcData: spcData, token: vuToken, assetID:␣
↪self!.contentID, licenseURL: licenseUrl, renewal: self.renewalInterval) else {␣
↪return }
                let keyResponse =␣
↪AVContentKeyResponse(fairPlayStreamingKeyResponseData: ckcData)
                keyRequest.processContentKeyResponse(keyResponse)
                        } catch {
                keyRequest.processContentKeyResponseError(error)
                        }
                }
        keyRequest.makeStreamingContentKeyRequestData(forApp: applicationCertificate!,
↪ contentIdentifier: assetIDData, options: [AVContentKeyRequestProtocolVersionsKey:␣
↪[1]], completionHandler: completionHandler)
```

Download and Offline Playback:

```
let ckcData = try self!.drm!.requestContentKeyFromKeySecurityModule(spcData: spcData,␣
↪token: self!.vuToken, assetID: self!.contentID, licenseURL: self!.licenseUrl,␣
↪renewal: self.renewalInterval)

let persistentKey = try keyRequest.persistableContentKey(fromKeyVendorResponse:␣
↪ckcData!, options: nil)
```

You can now write the persistent content key to disk:

```
try strongSelf.writePersistableContentKey(contentKey: persistentKey,␣
↪withContentKeyIdentifier: self!.contentID)
```

Finally, provide the content key response to make protected content available for processing.

```
let keyResponse = AVContentKeyResponse(fairPlayStreamingKeyResponseData:␣
↪persistentKey)
keyRequest.processContentKeyResponse(keyResponse)
```

Tokens and Licensing:

Requests for playback or download will result in a license request for the content from the license server, based on the type of token presented for each VUDRMFairPlaySDK request. The tokens may be FairPlay Rental, FairPlay Lease, or FairPlay Persist.

- FairPlay Rental token license requests will be provided a streaming content key, which may be used for online streaming only.

- FairPlay Lease token license requests will be provided a streaming content key, which may be used for online streaming only, and may be renewed periodically.
- FairPlay Persist token license requests will be provided a persistent content key, which may be written to device and subsequently used for both online streaming, and offline playback of downloaded content.

In iOS, when a download of an asset has been completed with a persist token policy, further playback requests will use the downloaded asset and associated license (or content key), even when online.

Example VUDRM token type templates available are:

- Fairplay Rental `{"type": "r","duration_rental": 3600}`
  - Fairplay Rental tokens may only be used to stream online content.
- Fairplay Lease `{"type": "l","duration_lease": 3600}`
  - Fairplay Lease tokens may only be used to stream online content. The token duration should always be greater than `360` seconds and should also exceed the streams `renew_interval` period. The streams `renew_interval` should always exceed `300` seconds. We recommend a difference of at least `60` seconds between the token duration and `renew_interval` to allow adequate time for the license to be retrieved and processed by the OS. In all other circumstances the `renew_interval` period should be set to `0` or not included in the stream configuration. Lease renewals will fail if the `renew_interval` is set to any value below `300` seconds, except for `0`. This is to prevent license server overload and unexpected overheads. Please refer to our demo application for an example, and do not hesitate to contact us to discuss the use of Fairplay Lease.
- Fairplay Persist `{"type": "p","duration_persist": 3600}`
- Fairplay Persist tokens may be used to stream online content and play offline (downloaded) content.

For further information about VUDRM please contact us, or refer to our documentation.

Note:

The content ID should be unique to each asset, as it is used along with the `playlist_url` to create a path to, and identify, offline assets and their associated content keys. The tvOS platform does not support downloading or offline playback.

There are significant limitations using AirPlay to stream any content to an Apple TV that has been downloaded to the users device. Please see the *Limitations* section for further information.

Demo Applications

We have developed demo applications based on Apples Fairplay SDK examples, using `AssetResourceLoaderDelegate` for iOS 10 and above, and the newer `ContentKeyDelegate` for iOS 11.2 and above.

The demo applications target both iOS and tvOS platforms using shared source code. Each target platform references its own version of the framework. The lowest version of iOS supported in the demo applications is iOS 11.2, however it is possible to add the required support for iOS 10 and above.

With the VUDRMFairPlaySDK framework installed, both the example demo applications read entries in the `Streams.plist` file to configure the content and DRM. You can edit the `Streams.plist` file values, and/or add your own valid values, which should include a `content_id`, `playlist_url`, `renew_interval`, and `vudrm_token`(see *Preparation* above).

- The `content_key_id_list` entry in the `Streams.plist` is retrieved in the demo by parsing the manifest using the `getContentKeyIDList` function in the `AssetListTableViewController`.

- The `name` value is the display name, which may differ from the `content_id`.
- The `is_protected` boolean value indicates whether the content is protected with DRM or not. This value can be overridden in source if all content is protected.
- The optional `renew_interval` value represents the lease renewal interval in seconds. This value is only used where token policy uses `duration_lease` and in all other cases `renew_interval` should either be set to `0` or not used in the configuration. Where `renew_interval` is used, it must always be a value above `300` seconds to prevent license server overload and/or unexpected overheads. Requests will fail for a renewed license where the value used is below `300`, except for `0`. There should be a difference of at least `60` seconds between the token duration and `renew_interval` to allow adequate time for the license to be retrieved and processed by the OS.

In both target platforms, the `AssetResourceLoaderDelegate` or `ContentKeyDelegate` class handles license requests for online streaming.

In iOS only the `AssetResourceLoaderDelegate+Persistable` or `ContentKeyDelegate+Persistable` extension handles license requests for iOS offline playback.

The call to make a request for an online license from the framework is made by iOS when a protected asset is requested for playback in the `override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)` function of the `AssetListTableViewController`. This corresponds to the specific stream, and an `Asset` object (with associated `AVURLAsset` is initialised for the stream by the returned table cell.

In iOS, the call used to make a request for an offline license from the framework in the AssetResourceLoaderDelegate demo is `ContentKeyManager.shared.assetResourceLoaderDelegate.requestPersistableContentKeys(forAsset: asset)`, or in the ContentKeyDelegate demo `ContentKeyManager.shared.contentKeyDelegate.requestPersistableContentKeys(forAsset: asset)` and this call is in the: `override func tableView(_ tableView: UITableView, accessoryButtonTappedForRowWith indexPath: IndexPath)` function of the `AssetListTableViewController`. This corresponds to the specific stream Download button.

In iOS, a call is made upon launch by the `AppDelegate` class to the `AssetPersistenceManager` class to restore any persisted content. The call `AssetPersistenceManager.sharedManager.restorePersistenceManager()` asks the `AssetPersistenceManager` class to check any already mapped asset against any outstanding `AVAssetDownloadTask`.

Limitations

- The SDK does not support AirPlay of protected content after a persist content key has been persisted (content downloaded to device). Apple TV is never able to play protected offline/downloaded content with a persist content key because the key cannot be written to the playback device. Attempting to do so may result in unexpected behaviours, crashes referring to the content key type, or the content may not load. Protected content may be transmitted with AirPlay using a streaming content key whenever the Apple TV has a network connection.
- The SDK does not support playback of protected content when running in the iOS Simulator. Attempting to do so will result in an invalid `KEYFORMAT` error and the content will not load.

Known Issues

- If you believe you have found any issues, please contact us at support@vualto.com.

Troubleshooting

- Most issues are content related. You can use our demo application to test your own content by updating it with your `Stream` configurations.
- Errors may also arise because the `Stream` configuration is not correct.
  - Content ID - Please ensure that the `content_id` corresponds to that which was used when preparing your content. The `content_id` should always be unique for each `Stream` instance.
  - Tokens - You can easily eliminate token issues by beginning with a default FairPlay token policy. Please ensure your token is formatted correctly and validates. For the avoidance of doubt, where tokens use dates, the dates should always be in the future. For further information about tokens please refer to our token documentation.

If you are not able to play your content after checking it in our demo application please contact support@vualto.com with the demo application logs and the stream configuration used.

Release notes (iOS / tvOS)

v1.0 (build 37/38) on 20/09/2021

- Added short renewal prevention limiting license renewals to periods exceeding 300 seconds, and added SDK license renewal initialiser.

v1.0 (build 22/23) on 19/03/2021

- Initial Release

Legacy

iOS / tvOS FairPlay SDK

The VUDRMFairPlay SDK is available for iOS and tvOS. This documentation describes the steps to integrate and use the VUDRMFairPlay SDK on these platforms, and how to configure our demo application.

Current release: iOS v1.0 (254) tvOS v1.0 (255)

- *Overview*
- *Requirements*
- *Xcode Integration*
- *Information about Application Transport Security (ATS)*
- *Preparation*
- *Example framework usage*
- *Limitations*
- *Known Issues*
- *Troubleshooting*
- *Release Notes*

Overview

VUDRMFairPlay SDK enables Apple's AVPlayer from AVFoundation to securely request licenses from Vualto's VUDRM cloud based DRM platform using an extended instance of AssetResourceLoaderDelegate.

The deployment scenario will allow Online Playback / Download and Offline Playback using an associated rental or persist token.

Make use of Apple's AVPlayer and AVPlayerViewController to present users with the platform default skins, with DRM content. Or create your own video player based on AVPlayer.

This SDK has a number of key benefits:

- Complete control over the entire AVPlayer lifecycle
- Numerous optimisations to enable faster playback
- Bitcode support

The SDK is fully supported in both Objective-C and Swift applications.

A multiple asset demo application written in Swift, and based on Apple's FairPlay SDK example application, is available on request. Please contact support@vualto.com to request access.

Requirements

- Minimum deployment target of iOS 12.0 and tvOS 12.0 or higher
- Xcode 12.2
- Swift 5.3
- Cocoapods

Xcode Integration

The VUDRMFairPlay SDK is distributed using Cocoapods. Projects set up with Cocoapods use an Xcode workspace, so it is important that projects with Cocoapods are always opened using the workspace file instead of the project file. Our simple demo application demonstrates the required set up for Cocoapods.

If you are integrating the VUDRMFairPlay SDK into your own project, please ensure that the project has been set up correctly for Cocoapods. There are numerous online tutorials demonstrating how to do this. You can then replace or merge the `podfile` with the one in our demo application.

It is important that the latest version of Cocoapods is installed before the demo project or your project can be opened without error.

1. You can ensure that you have the latest version of Cocoapods by opening a new shell in the Terminal application and entering:

   ```
   sudo gem install cocoapods
   ```

2. To pull vudrmFairPlay.framework in to the demo project or your project with Cocoapods, ensure that you quit Xcode if it is running, then open the Terminal application and navigate to the project directory, for example:

   ```
   cd /Users/username/Documents/Dev/vudrm-fairplay-demo
   ```

   Then enter:

   ```
   pod install
   ```

   or, where the pod has previously been installed:

```
pod update
```

The project should now be ready to open and run in Xcode.

Please see below for more information on using the demo application.

If you are unable to use Cocoapods in your project, please contact support@vualto.com to discuss other options.

Information about application transport security (ATS)

iOS/tvOS 9 introduces Application Transport Security (ATS) which restricts the use of insecure HTTP. In order to permit playback of content over insecure HTTP exemptions need to be added into your application's Info.plist. For example to disable ATS for any connection you would add the following into your application's Info.plist:

```
<dict>
    <key>NSAppTransportSecurity</key>
    <dict>
        <key>NSAllowsArbitraryLoads</key> <true/>
    </dict>
</dict>
```

More information about ATS can be found in Apple's TechNote: https://developer.apple.com/library/prerelease/ios/technotes/App-Transport-Security-Technote/

Preparation

The demo project is based on Apple's FairPlay SDK example application. This is configured using the projects `Streams.plist` file.

To provide complete configuration for each `Stream` object in the `Streams.plist`you will require:

- An `skd://` URI entry in the `content_key_id_list`. The correct URI can be obtained by retrieving the manifest and parsing out the `URI` from the `EXT-X-SESSION-KEY` or `EXT-X-KEY`, this can be done in code (see our demo application for an example) or manually with curl in the Terminal app, e.g. enter: `curl https://eample.cloudfront.net/example-demo/examplecontent/examplecontent.ism/.m3u8`.
- The `playlist_url` to correctly prepared content.
- A `name` for the content. This will be the readable display name and is not used by the SDK. This may differ from the `content_id`.
- A `content_id` for the content, being the same content ID that the content was prpeared with. This value will always be the last path component of the stream `content_key_id_list` or `skd://` URI entry. Our demo application shows how to retrieve and parse the `content_key_id_list`, however, the `content_id` always needs to be provided for use with offline assets.
- An `is_protected` boolean value which indicates that the `Stream` object is protected with DRM. This setting would not be needed in scenarios where all content is protected, as all assets can be set as protected by default in the source code.
- A `vudrm-token` which must be correctly generated for the type of instance you wish to create. Example VUDRM token type templates available are:
  - Fairplay Rental `{"type": "r","duration_rental": 3600}`
    - Fairplay Rental tokens may only be used to stream online content to devices running iOS 12 or higher.
  - Fairplay Persist `{"type": "p","duration_persist": 3600}`

* Fairplay Persist tokens may be used to stream online content and play offline (downloaded) content to devices running iOS 12 or higher.

For further information about VUDRM please contact us, or refer to our documentation: https://docs.vualto.com/projects/vudrm/en/latest/VUDRM-token.html

For added convenience, VUDRMFairPlay also bubbles up notifications of errors and progress during the licensing request.

Errors can be intercepted as follows:

```
@objc func handleVudrmDidError(_ notification: Notification)
{
    if let data = notification.userInfo as? [String: String]
    {
        for (function, error) in data
        {
            print("vudrmFairPlay - \(function) reported error \(error)!")
        }
    }
}
```

And, progress can be monitored using the following:

```
@objc func handleVudrmProgressUpdate(_ notification: Notification)
{
    if let data = notification.userInfo as? [String: String]
    {
        for (function, message) in data
        {
            print("vudrmFairPlay - \(function) reported progress \(message).")
        }
    }
}
```

Example framework usage

We strongly recommend referring to our iOS / tvOS example multiple asset demo application which uses a `ContentKeyManager`class to initialise an instance of VUDRMFairPlay for each stream, it then attaches the instance to an `AssetResourceLoaderDelegate` which is added to the `AssetResourceLoaderDelegateQueue`. Used with an `AssetPersistenceManager` class, the `ContentKeyManager` can also correctly handle persistence restoration with the `updateResourceLoaderDelegate` function.

The ContentKeyManager class should look like this:

```
import AVFoundation
import vudrmFairPlay

class ContentKeyManager {

/// MARK: Types.

/// The singleton for `ContentKeyManager`
static let shared: ContentKeyManager = ContentKeyManager()
// MARK: Properties.

```

(continues on next page)

(continued from previous page)

```
/**
 The instance of `AssetResourceLoaderDelegate` which conforms to␣
↪`AVAssetResourceLoaderDelegate` and is used to respond to content key requests
 from `AVAssetResourceLoader`.
 */
let assetResourceLoaderDelegate: vudrmFairPlay

/// The DispatchQueue to use for delegate callbacks.
let assetResourceLoaderDelegateQueue = DispatchQueue(label: "com.vualto.vurdrm.
↪fairplay.AssetResourceLoaderDelegateQueue")

var drm: vudrmFairPlay?

/// MARK: Initialization.

private init() {
    var token = ""
    var contentID = ""
    for stream in StreamListManager.shared.streams {
        if stream.vudrmToken != nil {
            token = stream.vudrmToken!
            contentID = stream.name
            let asset = AVURLAsset(url: URL(string: stream.playlistURL)!)
            self.drm = vudrmFairPlay(asset: asset, contentID: contentID, token:␣
↪token)!
        }
    }
    assetResourceLoaderDelegate = self.drm!
}

func updateResourceLoaderDelegate(forAsset asset: AVURLAsset) {
    asset.resourceLoader.setDelegate(self.drm, queue:␣
↪assetResourceLoaderDelegateQueue)
    }
}
```

Online Playback / Download and Offline Playback:

The tvOS platform does not support downloading or offline playback.

Requests for playback or download will result in a license request for the content from the license server, based on the type of token presented for each instance of VUDRMFairPlay. The tokens may be FairPlay Rental or FairPlay Persist. FairPlay Rental token license requests will be provided a streaming content key, which may be used for online streaming only. FairPlay Persist token license requests will be provided a persistent content key, which may be used for both online streaming, and offline playback of downloaded content.

In iOS, when a download of an asset has been completed with a persist token policy, further playback requests will use the downloaded asset and associated content key, even when online.

There are significant limitations using AirPlay to stream any content to an Apple TV that has been downloaded to the users device. Please see the *Limitations* section for further information.

To initialise your assets DRM instances, your application should create a DRM instance for each entry in the `Streams.plist` file (or comparable configuration) using a `ContentKeyManager` class. How to do this is demonstrated in our demo application. The `ContentKeyManager` attaches instances of VUDRMFairPlay to an `AssetResourceLoaderDelegate` for each asset using the asset URL, content ID, and valid token. The content

ID should be unique to each asset, as it is used along with the `playlist_url` to create a path to, and identify, offline assets and their associated content keys.

When using the same implementation as our demo application, iOS requests for licences for offfine content would reference the framework instance using the`ContentKeyManager`by calling:`ContentKeyManager.shared.assetResourceLoaderDelegate.doRequestPersistableContentKeys(contentKeyIDList: asset.stream.contentKeyIDList!, streamName: asset.stream.contentID, vudrmToken: asset.stream.vudrmToken!, avUrlAsset: avasset)`

DEMO

The demo application that we use targets both iOS and tvOS platforms using shared source code. Each target platform references its own version of the framework.

With the VUDRMFairPlay framework installed, the example demo application reads entries in the `Streams.plist` file to configure the content and DRM. You can edit the `Streams.plist` file values, and/or add your own valid values, which should include a `content_id`, `playlist_url`, and `vudrm_token`(see *Preparation* above).

- The `content_key_id_list` entry in the `Streams.plist` is retrieved in the demo by parsing the manifest using the `getContentKeyIDList` function in the `AssetListTableViewController`.
- The `name` value is the display name, which may differ from the `content_id`.
- The `is_protected` boolean value indicates whether the content is protected with DRM or not. This value can be overridden in source if all content is protected.

`Stream` class objects are mapped to `Asset` class objects to initialise AVURLAssets using the `ContentKeyManager` class.

In iOS, a call is made upon launch by the `AppDelegate` class to the `AssetPersistenceManager` class to restore any persisted content. The call `AssetPersistenceManager.sharedManager.restorePersistenceManager()` asks the `AssetPersistenceManager` class to check any already mapped asset against any outstanding `AVAssetDownloadTask`.

In both target platforms, the call to initialise content from the `Streams.plist` is then performed by the `ContentKeyManager` class.

In iOS, the call to request an offline license from the framework is: `ContentKeyManager.shared.assetResourceLoaderDelegate.doRequestPersistableContentKeys(contentKeyIDList: asset.stream.contentKeyIDList!, streamName: asset.stream.contentID, vudrmToken: asset.stream.vudrmToken!, avUrlAsset: avasset)` and is in the: `override func tableView(_ tableView: UITableView, accessoryButtonTappedForRowWith indexPath: IndexPath)` function of the `AssetListTableViewController`. This corresponds to the specific stream Download button.

Limitations

- The SDK does not support AirPlay of protected content after a persist content key has been persisted (content downloaded to device). Apple TV is never able to play protected offline/downloaded content with a persist content key. Attempting to do so may result in unexpected behaviours, crashes referring to the content key type, or the content may not load. Protected content may be transmitted with AirPlay before downloading content (and the content key has been persisted on the users device). Protected content may be transmitted with AirPlay using a streaming content key whenever the Apple TV has a network connection.
- The SDK does not support playback of protected content when running in the iOS Simulator. Attempting to do so will result in an invalid `KEYFORMAT` error and the content will not load.

Known Issues

- If you believe you have found any issues, please contact us at support@vualto.com

Troubleshooting

- Most issues are content related. You can use our demo application to test your own content by updating it with your `Stream` configurations.
- Errors may also arise because the `Stream` configuration is not correct.
  - Content ID - Please ensure that the `content_id` corresponds to that which was used when preparing your content. The `content_id` should always be unique for each `Stream` instance.
  - Tokens - You can easily eliminate token issues by beginning with a default FairPlay token policy. Please ensure your token is formatted correctly and validates. For the avoidance of doubt, where tokens use dates, the dates should always be in the future. For further information about tokens please refer to our token documentation.

If you are not able to play your content after checking it in our demo application please contact support@vualto.com with the demo application logs and the stream configuration used.

Release notes (iOS / tvOS)

v1.0 (build 254/255) on 04/12/2020

- Update to Xcode 12.2, Swift 5.3.1.
- Minor improvements.

v1.0 (build 244/212) on 07/10/2020

- Fix issue with iOS simulators.
- Merge iOS and tvOS documentation.

v1.0 (build 211) on 02/10/2020

- Update to Xcode 12, iOS 14, Swift 5.3.

v1.0 (build 182) on 09/07/2020

- Added error and progress notifications.
- Improved error handling.

v1.0 (build 161) on 10/06/2020

- SDK redeveloped to:
  - Resolve issue with persistence restoration after application cache clearance.

  - Enable demonstration of multiple asset DRM based on Apple's HLS Catalog example application.
  - Improve flow and handling using Apple's latest FairPlay SDK methods.

v0.0.1 (build 146) on 07/04/2020

- Added retrieval of license server URI from manifest
- Update required for XCode 11.4

v0.0.1 (build 131) on 29/01/2020

- Cocoapod distribution integration

v0.0.1 (build 105) on 24/09/2019

- Update required for XCode 11 / iOS 13

v0.0.1 (build 93) on 29/07/2019

- Update license server DNS
- Replaces instances of assetName and assetID with contentID to be consistent across DRM platform

v0.0.1 (build 85) on 27/03/2019

- Update to Swift 5
- Bug fixes and improvements

v0.0.1 (build 74) on 21/03/2019

- Fix streaming content key handling
- Bug fixes and improvements

v0.0.1 (build 70) on 15/01/2019

- Update Persistable Offline framework to Swift 4.2

v0.0.1 (build 65) on 14/12/2018

- Persistable Offline Release

v0.0.1 (build 58) on 05/11/2018

- Update framework to Swift 4.2

v0.0.1 (build 44) on 09/04/2018

- Bug fixes and improvements

v0.0.1 (build 15) on 05/04/2018

- Update framework to Swift 4.1

v0.0.1 (build 11) on 27/03/2018

-Bug fixes and improvements

v0.0.1 (build 5) on 15/11/2017

- Initial Release

1.2.4 SMART TV'S AND CASTING

LG

LG TV's use the WebOS SDK to provide applications to their Smart TV platforms.

Basic setup

WebOS applications are HTML5, Javascript applications which incorporate a WebOS Javascript library which allows access to TV functionality.

Being a HTML5, Javascript friendly environment there are two options for you when writing an application and integrating with VUDRM.

- Use one of our partner players with existing integration (Bitmovin, JWPlayer, THEOplayer).
- Use the inbuilt `luna` services with a `video` element.

Mandatory stream considerations

It is not possible to use the `mpd.inline_drm` setting in the playout profiles for LG TV streams, this will result in playback issues.

Recommendations

Below are a list of choices we recommend based on our experience using the LG WebOS platform.

Player choice

Using a partner player will give your viewers and your dev team the best experience at getting your application up and running as a robust understanding of PlayReady or Widevine is required.

Links to the documentation for integrating VUDRM with our partner players can be found below:

- Bitmovin
- JWPlayer
- THEOplayer

If you choose to use the luna services please follow LG's own documentation describing how to play DRM content.

PlayReady XML message retrieval

Within the LG documentation you are shown a snippet of XML you are to provide to the luna DRM service.

```
function getPlayReadyMessage(manifest, token) {

        return '<?xml version="1.0" encoding="utf-8"?>'
        + '<PlayReadyInitiator xmlns= "http://schemas.microsoft.com/DRM/2007/03/
↪protocols/">'
        + '<SetCustomData>'
        + '<CustomData>' + token + '</CustomData>'
        + '</SetCustomData>'
        + '</PlayReadyInitiator>';
    });
}

let XMLMessages = getPlayReadyMessage(manifestString, "VUDRM TOKEN");
```

Tizen

Samsung TV's use the Tizen SDK extension for providing applications to their Smart TV platforms.

Basic setup

Samsung provide extensive documentation showing how to get a development environment up and running, including this quick start guide.

DRM support

PlayReady DRM is currently the only DRM supported by Samsung.

Player Partners support

All of our player partners provide mechanisms to make use of VUDRM, and as Tizen only supports PlayReady, the documentation for each can be followed to implement.

Native support

Tizen has an inbuilt player within the SDK which can be used to provide encrypted playback utilising PlayReady DRM.

For further information on Samsungs PlayReady support please visit the Tizen PlayReady documentation.

Implementing a PlayReady solution can be done with the following demo repo.

```
// create listeners for your player
const listeners = {};
const source = "<YOUR DASH STREAM>";
const drmParam = {
    DeleteLicenseAfterUse: true,
    LicenseServer: "https://playready-license.vudrm.tech/rightsmanager.asmx",
    CustomData: "<YOUR_VUDRM_TOKEN>"
};

webapis.avplay.open(source);
webapis.avplay.setDisplayRect(0, 0, 1920, 1080);
webapis.avplay.setListener(listeners);
webapis.avplay.setDrm("PLAYREADY", "SetProperties", JSON.stringify(drmParam));
webapis.avplay.prepareAsync(function () {
    // do something
});
```

Tizen legacy

Samsung TV's use the Tizen SDK extension for providing applications to their Smart TV platforms.

Basic setup

Samsung provide extensive documentation showing how to get a development environment up and running, including this quick start guide.

DRM support

PlayReady DRM is currently the only DRM supported by Samsung.

Player Partners support

All of our player partners provide mechanisms to make use of VUDRM, and as Tizen only supports PlayReady, the documentation for each can be followed to implement.

Native support

Tizen has an inbuilt player within the SDK which can be used to provide encrypted playback utilising PlayReady DRM.

For further information on Samsungs PlayReady support please visit the Tizen PlayReady documentation.

Implementing a PlayReady solution can be done with the following code sample.

For Samsungs full documentation on PlayReady support please visit the Tizen PlayRead documentation.

```
avplayObj.show();
avplayObj.open(
    url,
    {
        drm : {
            type : "Playready",
            company : 'Microsoft Corporation',
            deviceID : '1'
        }
    }
);
avplayObj.play(
    self.playSuccess,
    function (error) {
        console.error(error.message);
    },
    0
);

let customData = "<YOUR VUDRM TOKEN>"
avplayObj.setPlayerProperty(3, customData, customData.length);

let laUrl ="https://playready-license.vudrm.tech/rightsmanager.asmx";
avplayObj.setPlayerProperty(4, laUrl, laUrl.length);

SefPluginPlayReady = document.getElementById('PluginSefPlayReady');
SefPluginPlayReady.Open("PlayReadyDrm",  "1.000",  "PlayReadyDrm");
SefPluginPlayReady.Execute("ProcessInitiatorsFromUrl",  laUrl);
SefPluginPlayReady.Close();
```

Chromecast

Chromecast is a casting device developed by Google, to enable streaming from web apps upon large screen display devices.

Google have extensive documentation about their SDK. If you have never developed a Chromecast application before we would strongly advise you start here.

Basic setup

In order to stream DRM protected content on a Chromecast using Googles' web SDK you will need to create a Custom Receiver app. The app will utilise two parts a `Sender` and a `Receiver`.

Typically the `Sender` would reside either as part of the web page or your chosen player.

The `Receiver` is an application that you host and register with Google.

Below is an example of basic `Sender` & `Receiver` side code using the Google SDK.

Sender

Setup of the load function within your `sender` within your site or player.

```
load(url, token, laUrl) {
    if (player.isConnected) { return };
    let mediaInfo = new chrome.cast.media.MediaInfo(url);
    mediaInfo.metadata = new chrome.cast.media.GenericMediaMetadata();
    mediaInfo.metadata.metadataType = chrome.cast.media.MetadataType.GENERIC;
    mediaInfo.metadata.title = 'VUDRM Demo';
    mediaInfo.customData = { laUrl, token };

    let request = new chrome.cast.media.LoadRequest(mediaInfo);
    request.autoplay = true;
    request.currentTime = 0;
    cast.framework.CastContext.getInstance().getCurrentSession().loadMedia(request).
↪catch((error) => {
        console.error(error);
    });
}
```

Receiver

Setup of the host within your `receiver` application.

```
let videoElement = document.getElementById("your-video-element");
if (event.data.media && event.data.media.contentId){
    let url = event.data.media.contentId;
    let initStart = event.data.media.currentTime || 0;
    let autoplay = !!event.data.autoplay;
    let customData = event.data.media.customData || {};

    let host = new cast.player.api.Host({mediaElement:videoElement, url});
    let protocol = cast.player.api.CreateDashStreamingProtocol(host);
    let player = new cast.player.api.Player(host);
    player.load(protocol, initStart);
    host.onError = (errorCode) => {
        console.error('Fatal Error ' + errorCode);
        if (player) {
            player.unload();
            player = null;
        }
    }
```

MPEG-DASH overrides

```
host.protectionSystem = cast.player.api.ContentProtection.WIDEVINE;
host.licenseUrl = 'https://widevine-license.vudrm.tech/proxy';

host.processManifest = (manifest) => {
    let parser = new DOMParser();
    let xmlDoc = parser.parseFromString(manifest, 'text/xml');
    let el = xmlDoc.querySelector('ContentProtection');
    let kid = el ? el.getAttribute('cenc:default_KID') : '';
    return manifest;
};
```

(continues on next page)

(continued from previous page)

```
host.updateLicenseRequestInfo = (requestInfo) => {
    // customData is the message object that was sent to the receiver.
    let body = JSON.stringify({
        token: customData.token,
        drm_info: Array.apply(null, requestInfo.content),
        kid: kid
    });

    let buffer = [];

    for (let i=0; i<body.length; i++) {
        buffer.push(body.charCodeAt(i));
    }

    requestInfo.headers['Content-Type'] = 'application/json';
    requestInfo.content = new Uint8Array(buffer);
}

protocol = cast.player.api.CreateDashStreamingProtocol(host);
```

Roku

The Roku OS was purpose-built for streaming and runs across all Roku devices, including streaming players and Roku TVs. On the Roku platform the applications that stream your media are called channels.

Basic setup

Follow the getting started instructions laid out in the Roku documentation to set up your environment. Roku uses it's own scripting language called BrightScript to set up channels/apps.

We recommend using DASH with PlayReady on Roku devices.

The documentation for which can be found in the content protection.

```
customData = { "<VUDRM-TOKEN>" };
contentNode = createObject("roSGNode", "contentNode")
contentNode.streamFormat = "dash"
contentNode.url = "<MPEG-DASH-URL>"
contentNode.encodingType = "PlayReadyLicenseAcquisitionAndChallenge"
contentNode.encodingKey = "PlayReadyLicenseServerUrl" + "%%%" + customData
```

Testing your stream

You can also test your streams outside of the Roku app using the Stream testing tool.

- Add your device `IP` and `Model` to the device manager.
- Ensure that `Mode` is set to `Video`.
- Ensure that `Format` is set to either `Auto` or `DASH`.
- Within the DRM section choose `encoding type : PlayReady: LA And Challenge`.
- Put your newly generated VUDRM-token in to the custom data input box.

- In `License server Url(optional)` put `https://playready-license.vudrm.tech/rightsmanager.asmx`.
- Click `Play`.

1.2.5 ENCRYPTION KEY PROVISION

The VUDRM key provision service exposes the generation of DRM encryption. It provides two secure REST APIs allowing you to retrieve encryption keys for all DRM types in order to encrypt your content.

CPIX Key Provider API

We recommend using our CPIX API to fetch VUDRM encryption keys in CPIX XML document format. Compatible with Unified Streaming (version 1.9.3 and above). This API can also be used to return the keys from the CPIX document in JSON format; for more information on this please see the *JSON keys* section. All requests made to this API will require your API key set as the header `API-KEY`. If you do not know or have not been given your API key please contact support@vualto.com.

> Replace `<api-key>`, `<client-name>`, and `<content-id>` with appropiate values.
>
> `<content-id>` may only contain alphanumeric characters, underscores, and hyphens.
>
> Available DRM systems [`fairplay`, `playready`, `widevine`].

Endpoints

Default

Get a CPIX 2.1 document with all DRM systems `<client-name>` is entitled to use.

Optional query params:

- `drm` : comma separated list of DRM systems to be included in the response.

Request

```
curl -XGET -H "API-KEY: <api-key>" \
"https://cpix.vudrm.tech/v1/cpix/<client-name>/<content-id>"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc" xmlns:xsi=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:cpix="urn:dashif:org:cpix"␣
↪xsi:schemaLocation="urn:dashif:org:cpix cpix.xsd">
  <cpix:ContentKeyList>
    <cpix:ContentKey kid="11111111-1111-1111-1111-111111111111" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
      <cpix:Data>
        <pskc:Secret>
          <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
        </pskc:Secret>
      </cpix:Data>
```

(continues on next page)

(continued from previous page)

```
    </cpix:ContentKey>
  </cpix:ContentKeyList>
  <cpix:DRMSystemList>
    <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
      <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
      <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
      <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
      <cpix:HLSSignalingData>YmFzZTY0ZW5jb2RlZAo=</cpix:HLSSignalingData>
    </cpix:DRMSystem>
  </cpix:DRMSystemList>
</cpix:CPIX>
```

Decrypt

Get a CPIX 2.1 document with only a single content key.

Optional query params:

- `drm` : comma separated list of DRM systems to be included in the response.

Request

```
curl -XGET -H "API-KEY: <api-key>" \
"https://cpix.vudrm.tech/v1/cpix/<client-name>/<content-id>/decrypt"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc" xmlns:xsi=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:cpix="urn:dashif:org:cpix"␣
↪xsi:schemaLocation="urn:dashif:org:cpix cpix.xsd">
  <cpix:ContentKeyList>
    <cpix:ContentKey kid="11111111-1111-1111-1111-111111111111" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
      <cpix:Data>
        <pskc:Secret>
          <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
        </pskc:Secret>
      </cpix:Data>
    </cpix:ContentKey>
  </cpix:ContentKeyList>
</cpix:CPIX>
```

Multikey

Get a CPIX 2.1 document that uses separate keys for video and audio tracks.

Optional query params:

- `drm` : comma separated list of DRM systems to be included in the response.

Request

```
curl -XGET -H "API-KEY: <api-key>" \
"https://cpix.vudrm.tech/v1/cpix/<client-name>/<content-id>/multikey"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc" xmlns:xsi=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:cpix="urn:dashif:org:cpix"␣
↪xsi:schemaLocation="urn:dashif:org:cpix cpix.xsd">
  <cpix:ContentKeyList>
    <cpix:ContentKey kid="11111111-1111-1111-1111-111111111111" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
      <cpix:Data>
        <pskc:Secret>
          <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
        </pskc:Secret>
      </cpix:Data>
    </cpix:ContentKey>
    <cpix:ContentKey kid="22222222-2222-2222-22222-22222222222" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
      <cpix:Data>
        <pskc:Secret>
          <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
        </pskc:Secret>
      </cpix:Data>
    </cpix:ContentKey>
  </cpix:ContentKeyList>
  <cpix:DRMSystemList>
    <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
      <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
      <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
      <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
      <cpix:HLSSignalingData playlist="master">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
      <cpix:HLSSignalingData playlist="media">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
```

(continues on next page)

(continued from previous page)

```
      <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
      <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
      <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
      <cpix:HLSSignalingData playlist="master">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
      <cpix:HLSSignalingData playlist="media">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
    </cpix:DRMSystem>
  </cpix:DRMSystemList>
  <cpix:ContentKeyUsageRuleList>
    <cpix:ContentKeyUsageRule kid="11111111-1111-1111-1111-111111111111">
      <cpix:VideoFilter></cpix:VideoFilter>
    </cpix:ContentKeyUsageRule>
    <cpix:ContentKeyUsageRule kid="22222222-2222-2222-22222-22222222222">
      <cpix:AudioFilter></cpix:AudioFilter>
    </cpix:ContentKeyUsageRule>
  </cpix:ContentKeyUsageRuleList>
</cpix:CPIX>
```

Multikey with audio clear

Get a CPIX 2.1 document that uses only 1 content key for video; audio tracks are left in the clear.

Optional query params:

- `drm` : comma separated list of DRM systems to be included in the response.

Request

```
curl -XGET -H "API-KEY: <api-key>" \
"https://cpix.vudrm.tech/v1/cpix/<client-name>/<content-id>/multikey/audioclear"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc" xmlns:xsi=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:cpix="urn:dashif:org:cpix"␣
↪xsi:schemaLocation="urn:dashif:org:cpix cpix.xsd">
  <cpix:ContentKeyList>
    <cpix:ContentKey kid="11111111-1111-1111-1111-111111111111" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
      <cpix:Data>
        <pskc:Secret>
          <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
        </pskc:Secret>
      </cpix:Data>
    </cpix:ContentKey>
```

(continues on next page)

(continued from previous page)

```
    <cpix:ContentKey kid="22222222-2222-2222-22222-22222222222">
    </cpix:ContentKey>
  </cpix:ContentKeyList>
  <cpix:DRMSystemList>
    <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
      <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
      <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
      <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
      <cpix:HLSSignalingData playlist="master">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
      <cpix:HLSSignalingData playlist="media">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
      <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
      <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
      <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
      <cpix:HLSSignalingData playlist="master">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
      <cpix:HLSSignalingData playlist="media">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
    </cpix:DRMSystem>
  </cpix:DRMSystemList>
  <cpix:ContentKeyUsageRuleList>
    <cpix:ContentKeyUsageRule kid="11111111-1111-1111-1111-111111111111">
      <cpix:VideoFilter></cpix:VideoFilter>
    </cpix:ContentKeyUsageRule>
    <cpix:ContentKeyUsageRule kid="22222222-2222-2222-22222-22222222222">
      <cpix:AudioFilter></cpix:AudioFilter>
    </cpix:ContentKeyUsageRule>
  </cpix:ContentKeyUsageRuleList>
</cpix:CPIX>
```

Multikey with seperate bitrates based key

Get a CPIX 2.1 document that uses different content keys per bitrate and one for audio. If provided with a single bitrate the document will use one key for all bitrates up to the given bitrate, and one key for the given bitrate up. If provided with two bitrates the document will use one key for all bitrates up to the lowest given bitrate, one key for all bitrates between the lowest and highest bitrates, and one key for the highest given bitrate up.

Query params:

- `bitrates` : a single bitrate, or two seperated by a comma.

Optional query params:

- `drm` : comma separated list of DRM systems to be included in the response.

Request

```
curl -XGET -H "API-KEY: <api-key>" \
"https://cpix.vudrm.tech/v1/cpix/<client-name>/<content-id>/multikey?bitrates=8"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc" xmlns:xsi=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:cpix="urn:dashif:org:cpix" xmlns:speke=
↪"urn:aws:amazon:com:speke" xsi:schemaLocation="urn:dashif:org:cpix cpix.xsd">
    <cpix:ContentKeyList>
        <cpix:ContentKey kid="11111111-1111-1111-1111-111111111111" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
        <cpix:ContentKey kid="22222222-2222-2222-22222-22222222222" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
        <cpix:ContentKey kid="33333333-3333-3333-33333-33333333333" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
    </cpix:ContentKeyList>
    <cpix:DRMSystemList>
        <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
```

(continues on next page)

(continued from previous page)

```
            <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
            <cpix:HLSSignalingData playlist="master">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
            <cpix:HLSSignalingData playlist="media">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
            <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
            <cpix:HLSSignalingData playlist="master">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
            <cpix:HLSSignalingData playlist="media">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="33333333-3333-3333-33333-33333333333" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="33333333-3333-3333-33333-33333333333" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="33333333-3333-3333-33333-33333333333" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
            <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
            <cpix:HLSSignalingData playlist="master">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
            <cpix:HLSSignalingData playlist="media">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
        </cpix:DRMSystem>
    </cpix:DRMSystemList>
    <cpix:ContentKeyPeriodList></cpix:ContentKeyPeriodList>
    <cpix:ContentKeyUsageRuleList>
        <cpix:ContentKeyUsageRule kid="11111111-1111-1111-1111-111111111111">
            <cpix:BitrateFilter maxBitrate="8"></cpix:BitrateFilter>
        </cpix:ContentKeyUsageRule>
        <cpix:ContentKeyUsageRule kid="22222222-2222-2222-22222-22222222222">
            <cpix:BitrateFilter minBitrate="8"></cpix:BitrateFilter>
        </cpix:ContentKeyUsageRule>
        <cpix:ContentKeyUsageRule kid="33333333-3333-3333-33333-33333333333">
            <cpix:AudioFilter></cpix:AudioFilter>
        </cpix:ContentKeyUsageRule>
    </cpix:ContentKeyUsageRuleList>
</cpix:CPIX>
```

Multikey with seperate video quality based keys

Get a CPIX 2.1 document that uses different content keys per video qualty and one for audio. If provided with multiple video qualities the response will contain a encryption key for each video quality and one encrypt key for everything else.

Query params:

- `videoTracks` : a list of video qualities seperated by a comma, can only contain `SD`, `HD`, or `UHD`.

Optional query params:

- `drm` : comma separated list of DRM systems to be included in the response.

Request

```
curl -XGET -H "API-KEY: <api-key>" \
"https://cpix.vudrm.tech/v1/cpix/<client-name>/<content-id>/multikey?videoTracks=SD,HD
↪"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc" xmlns:xsi=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:cpix="urn:dashif:org:cpix" xmlns:speke=
↪"urn:aws:amazon:com:speke" xsi:schemaLocation="urn:dashif:org:cpix cpix.xsd">
    <cpix:ContentKeyList>
        <cpix:ContentKey kid="11111111-1111-1111-1111-111111111111" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
        <cpix:ContentKey kid="22222222-2222-2222-22222-22222222222" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
        <cpix:ContentKey kid="33333333-3333-3333-33333-33333333333" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
        <cpix:ContentKey kid="44444444-4444-4444-44444-44444444444" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
            <cpix:Data>
```

(continues on next page)

(continued from previous page)

```
                <pskc:Secret>
                    <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
    </cpix:ContentKeyList>
    <cpix:DRMSystemList>
        <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
            <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
            <cpix:HLSSignalingData playlist="master">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
            <cpix:HLSSignalingData playlist="media">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
            <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
            <cpix:HLSSignalingData playlist="master">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
            <cpix:HLSSignalingData playlist="media">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="33333333-3333-3333-33333-33333333333" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="33333333-3333-3333-33333-33333333333" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="33333333-3333-3333-33333-33333333333" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
            <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
            <cpix:HLSSignalingData playlist="master">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
            <cpix:HLSSignalingData playlist="media">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="44444444-4444-4444-44444-44444444444" systemId="9a04f079-
↪9840-4286-ab92-e65be0885f95">
```

(continues on next page)

(continued from previous page)

```
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="44444444-4444-4444-44444-44444444444" systemId="edef8ba9-
↪79d6-4ace-a3c8-27dcd51d21ed">
            <cpix:PSSH>YmFzZTY0ZW5jb2RlZAo=</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="44444444-4444-4444-44444-44444444444" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
            <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
            <cpix:HLSSignalingData playlist="master">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
            <cpix:HLSSignalingData playlist="media">YmFzZTY0ZW5jb2RlZAo=</
↪cpix:HLSSignalingData>
        </cpix:DRMSystem>
    </cpix:DRMSystemList>
    <cpix:ContentKeyPeriodList></cpix:ContentKeyPeriodList>
    <cpix:ContentKeyUsageRuleList>
        <cpix:ContentKeyUsageRule kid="11111111-1111-1111-1111-111111111111">
            <cpix:BitrateFilter maxPixels="409920"></cpix:BitrateFilter>
        </cpix:ContentKeyUsageRule>
        <cpix:ContentKeyUsageRule kid="22222222-2222-2222-22222-22222222222">
            <cpix:BitrateFilter minPixels="409921" maxPixels="2073600"></
↪cpix:BitrateFilter>
        </cpix:ContentKeyUsageRule>
        <cpix:ContentKeyUsageRule kid="33333333-3333-3333-33333-33333333333">
            <cpix:VideoFilter minPixels="2073601"></cpix:VideoFilter>
        </cpix:ContentKeyUsageRule>
        <cpix:ContentKeyUsageRule kid="44444444-4444-4444-44444-44444444444">
            <cpix:AudioFilter></cpix:AudioFilter>
        </cpix:ContentKeyUsageRule>
    </cpix:ContentKeyUsageRuleList>
</cpix:CPIX>
```

Key Rotation

Get a CPIX 2.1 document with Fairplay keys at every given interval between two given times; key rotation is only supported by HLS with Fairplay or AES.

USP support

Key rotation works with USP versions `1.10.12` and `1.10.18`. If you wish to use version `1.9.5` the `explicitIV` needs to be removed from each content key before the CPIX document is used. This can be done with the following command if the CPIX document has been stored in a file name `keys.cpix`.

```
sed -i -E 's/ explicitIV=.*"//' keys.cpix
```

Query params:

- `start` : start time in `yyyy-MM-ddThh-mm-ssZ` format. E.g. 1970-01-01T00:00:00Z
- `end` : end time in `yyyy-MM-ddThh-mm-ssZ` format. E.g. 1970-01-01T01:00:00Z
- `interval` : interval each key should last in seconds.

Optional query params:

- `drm` : comma separated list of DRM systems to be included in the response. **Can only be `fairplay` or `aes`**

Request

```
curl -XGET -H "API-KEY: <api-key>" \
"https://cpix-beta.eu-central-1.vudrm.tech/v1/cpix/<client-name>/<content-id>/rotate?
↪start=<start-time>&end=<end-time>&interval=<interval>"
```

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX xmlns:pskc="urn:ietf:params:xml:ns:keyprov:pskc" xmlns:xsi=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:cpix="urn:dashif:org:cpix" xmlns:speke=
↪"urn:aws:amazon:com:speke" xsi:schemaLocation="urn:dashif:org:cpix cpix.xsd">
  <cpix:ContentKeyList>
    <cpix:ContentKey kid="11111111-1111-1111-1111-111111111111" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
      <cpix:Data>
        <pskc:Secret>
          <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
        </pskc:Secret>
      </cpix:Data>
    </cpix:ContentKey>
    <cpix:ContentKey kid="22222222-2222-2222-22222-22222222222" explicitIV=
↪"YjIxZmRlNzI5MDUyMDEzYw==">
      <cpix:Data>
        <pskc:Secret>
          <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
        </pskc:Secret>
      </cpix:Data>
    </cpix:ContentKey>
    <cpix:ContentKey kid="33333333-3333-3333-3333-333333333333" explicitIV=
↪"YmFzZTY0ZW5jb2RlZAo=">
      <cpix:Data>
        <pskc:Secret>
          <pskc:PlainValue>YmFzZTY0ZW5jb2RlZAo=</pskc:PlainValue>
        </pskc:Secret>
      </cpix:Data>
    </cpix:ContentKey>
  </cpix:ContentKeyList>
  <cpix:DRMSystemList>
    <cpix:DRMSystem kid="11111111-1111-1111-1111-111111111111" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
      <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
      <cpix:HLSSignalingData>YmFzZTY0ZW5jb2RlZAo=</cpix:HLSSignalingData>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="22222222-2222-2222-22222-22222222222" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
      <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
      <cpix:HLSSignalingData>YmFzZTY0ZW5jb2RlZAo=</cpix:HLSSignalingData>
    </cpix:DRMSystem>
    <cpix:DRMSystem kid="33333333-3333-3333-3333-333333333333" systemId="94ce86fb-
↪07ff-4f43-adb8-93d2fa968ca2">
      <cpix:URIExtXKey>YmFzZTY0ZW5jb2RlZAo=</cpix:URIExtXKey>
      <cpix:HLSSignalingData>YmFzZTY0ZW5jb2RlZAo=</cpix:HLSSignalingData>
```

(continues on next page)

(continued from previous page)

```
    </cpix:DRMSystem>
  </cpix:DRMSystemList>
  <cpix:ContentKeyPeriodList>
    <cpix:ContentKeyPeriod id="period_0" start="1970-01-01T00:00:00Z" end="1970-01-
↪01T00:30:00Z"></cpix:ContentKeyPeriod>
    <cpix:ContentKeyPeriod id="period_1" start="1970-01-01T00:30:00Z" end="1970-01-
↪01T01:00:00Z"></cpix:ContentKeyPeriod>
    <cpix:ContentKeyPeriod id="period_2" start="1970-01-01T01:00:00Z" end="1970-01-
↪01T01:30:00Z"></cpix:ContentKeyPeriod>
  </cpix:ContentKeyPeriodList>
  <cpix:ContentKeyUsageRuleList>
    <cpix:ContentKeyUsageRule kid="11111111-1111-1111-1111-111111111111">
      <cpix:KeyPeriodFilter periodId="period_0"></cpix:KeyPeriodFilter>
    </cpix:ContentKeyUsageRule>
    <cpix:ContentKeyUsageRule kid="22222222-2222-2222-22222-22222222222">
      <cpix:KeyPeriodFilter periodId="period_1"></cpix:KeyPeriodFilter>
    </cpix:ContentKeyUsageRule>
    <cpix:ContentKeyUsageRule kid="33333333-3333-3333-3333-333333333333">
      <cpix:KeyPeriodFilter periodId="period_2"></cpix:KeyPeriodFilter>
    </cpix:ContentKeyUsageRule>
  </cpix:ContentKeyUsageRuleList>
</cpix:CPIX>
```

JSON Keys

Get encryption keys and license URLs for a given `<content-id>` as JSON.

Optional query params:

- `drm` : comma separated list of DRM systems to be included in the response.

Request

```
curl -XGET -H "API-KEY: <api-key>" \
"https://cpix.vudrm.tech/v1/keys/<client-name>/<content-id>"
```

```
{
  "key_id_hex": "76616c7565686578666f726d61746564",
  "content_key_hex": "76616c7565686578666f726d61746564",
  "iv_hex": "76616c7565686578666f726d61746564",
  "playready_pssh_data": "YmFzZTY0LXBsYXlyZWFkeS1oZWFkZXItb2JqZWN0Cg==",
  "playready_system_id": "9a04f079-9840-4286-ab92-e65be0885f95",
  "widevine_drm_specific_data": "YmFzZTY0LXdpZGV2aW5lLXBzc2gtZGF0YQo=",
  "widevine_system_id": "edef8ba9-79d6-4ace-a3c8-27dcd51d21ed",
  "fairplay_system_id": "94ce86fb-07ff-4f43-adb8-93d2fa968ca2",
  "fairplay_laurl": "skd://fairplay-license.vudrm.tech/license/<content-id>",
}
```

The values are:

- `key_id_hex`: Unique ID for the encryption. Base16 in Little Endian. Used by PlayReady and Widevine.

- `content_key_hex`: 128bit encryption key in base16 in Little Endian. Used by Fairplay, PlayReady, and Widevine.
- `iv_hex`: The encryption key. Used by Fairplay.
- `playready_pssh_data`: Base 64 encoded pssh box containing the PlayReady header.
- `playready_system_id`: The DASH protection system-specific identifier for PlayReady.
- `widevine_drm_specific_data`: Base 64 encoded pssh box containing the content id.
- `widevine_system_id`: The DASH protection system-specific identifier for Widevine.
- `fairplay_system_id`: The DASH protection system-specific identifier for Fairplay.
- `fairplay_laurl`: The Fairplay license url.

SPEKE Key Provider API

These are the docs for the speke key provider, which has a speke endpoint for use with AWS. To learn how to use our SPEKE API with AWS' Media Services please read our guide.

To retrieve drm information formatted to the AWS SPEKE standard send a POST request, formatting below, to **https://speke.vudrm.tech//client-name/speke**, where "client-name" is the name of the client.

Examples

Requests

Headers

```
Content-Type: application/xml
API-KEY: <api-key>
```

Widevine and Playready

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX id="someContentId" xmlns:cpix="urn:dashif:org:cpix" xmlns:pskc=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:speke="urn:aws:amazon:com:speke">
   <cpix:ContentKeyList>
      <cpix:ContentKey kid="" explicitIV=""/>
   </cpix:ContentKeyList>
   <cpix:DRMSystemList>
      <!-- playready -->
      <cpix:DRMSystem kid="" systemId="9a04f079-9840-4286-ab92-e65be0885f95">
      </cpix:DRMSystem>
      <!-- widevine -->
      <cpix:DRMSystem kid="" systemId="edef8ba9-79d6-4ace-a3c8-27dcd51d21ed">
      </cpix:DRMSystem>
   </cpix:DRMSystemList>
</cpix:CPIX>
```

Fairplay

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX id="someContentId" xmlns:cpix="urn:dashif:org:cpix" xmlns:pskc=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:speke="urn:aws:amazon:com:speke">
   <cpix:ContentKeyList>
      <cpix:ContentKey kid="" explicitIV=""/>
   </cpix:ContentKeyList>
   <cpix:DRMSystemList>
      <!-- fairplay -->
      <cpix:DRMSystem kid="" systemId="94ce86fb-07ff-4f43-adb8-93d2fa968ca2">
      </cpix:DRMSystem>
   </cpix:DRMSystemList>
</cpix:CPIX>
```

Responses

Headers

```
Content-Type: application/xml
```

Widevine and Playready

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX id="someContentId" xmlns:cpix="urn:dashif:org:cpix" xmlns:pskc=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:speke="urn:aws:amazon:com:speke">
    <cpix:ContentKeyList>
        <cpix:ContentKey explicitIV="" kid="someKid">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:PlainValue>playreadyContentKey</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
    </cpix:ContentKeyList>
    <cpix:DRMSystemList>
        <cpix:DRMSystem kid="someKid" systemId="9a04f079-9840-4286-ab92-e65be0885f95">
            <speke:ProtectionHeader>playreadyProtectionHeader</speke:ProtectionHeader>
            <cpix:PSSH>playreadyPSSHBox</cpix:PSSH>
        </cpix:DRMSystem>
        <cpix:DRMSystem kid="someKid" systemId="edef8ba9-79d6-4ace-a3c8-27dcd51d21ed">
            <PSSH>widevinePSSHBox</PSSH>
        </cpix:DRMSystem>
    </cpix:DRMSystemList>
</cpix:CPIX>
```

Fairplay

```
<?xml version="1.0" encoding="UTF-8"?>
<cpix:CPIX id="someContentId" xmlns:cpix="urn:dashif:org:cpix" xmlns:pskc=
↪"urn:ietf:params:xml:ns:keyprov:pskc" xmlns:speke="urn:aws:amazon:com:speke">
```

(continues on next page)

(continued from previous page)

```
    <cpix:ContentKeyList>
        <cpix:ContentKey explicitIV="fairplayIvHex-base64-encoded" kid="someKid">
            <cpix:Data>
                <pskc:Secret>
                    <pskc:PlainValue>fairplayKeyHex-base64-encoded</pskc:PlainValue>
                </pskc:Secret>
            </cpix:Data>
        </cpix:ContentKey>
    </cpix:ContentKeyList>
    <cpix:DRMSystemList>
        <cpix:DRMSystem kid="someKid" systemId="94ce86fb-07ff-4f43-adb8-93d2fa968ca2">
            <URIExtXKey>fairplayLaurl-base64-encoded</URIExtXKey>
            <KeyFormat>fairplayKeyFormat-base64-encoded</KeyFormat>
            <KeyFormatVersions>fairplayKeyFormatVersion-base64-encoded</
↪KeyFormatVersions>
        </cpix:DRMSystem>
    </cpix:DRMSystemList>
</cpix:CPIX>
```

Legacy

Legacy JSON Key provider API

The Legacy JSON Key Provider API will provide all the information required in order to encrypt various types of content. In order to be as flexible as possible it will return encryption keys in Base16 and Base64 as some systems require different values.

The following sections describe the requests and responses in order to retrieve the keys. The most common scenario is to request CENC and Fairplay encryption keys.

Request

A `GET` should be made to the following URL:

```
https://keyprovider.vudrm.tech/<DRM_TYPE>/<CLIENT_NAME>/<CONTENT_ID>
```

The breakdown of this URL is:

- `DRM_TYPE`: The type of encryption keys being requested. Possible values are `cenc`, `fairplay`, `widevine`, `playready`, and `aes`. See *Using the encryption keys* for more information.
- `CLIENT_NAME`: The account name. Plese contact support@vualto.com if you do not have an account name.
- `CONTENT_ID`: A unique content identifier. This value will always generate the same encryption keys. May only contain alphanumeric characters, underscores, and hyphens.

In order to provide the keys securely, an API key is required in the header of the request. Please contact support@vualto.com if you do not have an API key.

Below is an example curl request for `cenc` encryption keys using the client `vualto` and with the `CONTENT_ID` of `test`.

```
curl -X "GET" "https://keyprovider.vudrm.tech/cenc/vualto/test" -H "API_KEY: <API_KEY>
↪"
```

Response

In order to provide the keys securely, the DRM encryption keys are encrypted in the response.

The format of the response will be:

```
{
    "key":"r8tXpf8wSSHKVnW1fz+MgZY3BTnTO+/
↪IGFglt9oUwtKWj8eyguSLd0bzOhcn4DMF4JWOAbhbKopJE/cZWPqIfl3RCIwl46UP57/m7870+z3NWxh/
↪JSvrfWBq4SGmkykfDLKjyLBqb5F6dDlUB+flcfZMcQh6FGk5GNGEniXliB/kyCrVDiKdwAw3hft8rT4/
↪itFQDMRKkhJf1Fh67AZ1LyzOI3CCY/oO4w/XF/
↪XMAJ1z92tAKULI+cPMFaXT3I77N3iaQoYN78mJkKgAr71Tlf91+ASB8jqOCHD6nNgm6nGxZlsTVK5wxdhT4zbMJq4xb7daSy7xM
↪1eXuIh0ZhWicKJqbWHKIeifZWFras/
↪0QzqN27rupHF3UYnYQ40V3ESE2PUIe8Cs8471QRHlruYlwtgBBmGme2ERZWFjrRFEeUt8SobQkCIZrzDEKaQ2bJMyOy1yMt8okl
↪StdpTKCn1yxr3KoyxGAk3L|166c43d4023880a99691fd9679e6ad785305f205"
}
```

The value of `key` has two components separated by a pipe (ASCII code 124):

- Encrypted blob containing the DRM encryption keys.
- Hash used in a checksum.

Decrypting the response

This section contains examples of how to decrypt the response from the Legacy JSON Key Provider API.

You will need the `CLIENT_NAME`, `SHARED_SECRET` and the `KEY_PROVIDER_RESPONSE` value from the request to the Legacy JSON Key Provider API.

C#

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;

namespace Vudrm.EncryptionExamples
{
    class Program
    {
        private const string Client = "CLIENT_NAME";
        private const string SharedSecret = "SHARED_SECRET";
        private const string KeyProviderResponse = "KEY_PROVIDER_RESPONSE";

        static void Main(string[] args)
        {
            Console.WriteLine("Decrypting keyprovider API response...");
            var splitKeyProviderResponse = KeyProviderResponse.Split('|');
            var computedHash = ComputeHash(SharedSecret, Client,␣
↪splitKeyProviderResponse[0]);
            if (computedHash == splitKeyProviderResponse[1])
            {
                Console.WriteLine("key providers response hash passed validation");
                var decryptedKeyProviderResponse =␣
↪Decrypt(splitKeyProviderResponse[0], SharedSecret);
```

(continues on next page)

(continued from previous page)

```
                Console.WriteLine("Decrypted key provider response: " +␣
↪decryptedKeyProviderResponse);
            }
            else
            {
                Console.WriteLine("Key provider response hash did not validate");
            }
            Console.ReadLine();
        }

        private string Decrypt(string policy, string sharedSecret)
        {
            try
            {
                var encrypted = Convert.FromBase64String(policy);
                var key = Encoding.ASCII.GetBytes(sharedSecret.Substring(0, 16));
                var rj = new RijndaelManaged
                {
                    Mode = CipherMode.ECB,
                    BlockSize = 128,
                    Key = key
                };
                var decryptor = rj.CreateDecryptor(key, null);
                var ms = new MemoryStream(encrypted);
                var cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read);
                var plain = new byte[encrypted.Length];
                var decryptcount = cs.Read(plain, 0, plain.Length);
                ms.Close();
                cs.Close();
                return (Encoding.UTF8.GetString(plain, 0, decryptcount));
            }
            catch
            {
                return policy;
            }
        }

        private string ComputeHash(string sharedSecret, string client, string␣
↪encryptedMessage)
        {
            var preComputed = sharedSecret + client + encryptedMessage;
            var algorithm = new SHA1Managed();
            var bytesOf = Encoding.UTF8.GetBytes(preComputed);
            var hashOf = algorithm.ComputeHash(bytesOf);
            var result = new StringBuilder();
            foreach (var b in hashOf)
            {
                result.Append(b.ToString("x2").ToLower());
            }
            return result.ToString();
        }
    }
}
```

Ruby

```ruby
require 'base64'
require 'crypt/rijndael'
require 'digest'
require 'openssl'

client = "CLIENT_NAME"
shared_secret = "SHARED_SECRET"

encrypted_keys = "KEY_PROVIDER_RESPONSE"

message, message_hash = encrypted_keys.split('|')

pre_computed = client + message
computed_hash = Digest::SHA1.new.hexdigest(shared_secret + pre_computed)

if computed_hash == message_hash
  decoded_message = Base64.strict_decode64(message)
  decipher = OpenSSL::Cipher::Cipher.new('AES-128-ECB').decrypt
  decipher.key = shared_secret[0..15]
  unencrypted_keys = decipher.update(decoded_message) + decipher.final
else
  raise "Key provider response hash did not validate"
end
```

GO

```go
package main

import (
        "crypto/aes"
        "crypto/sha1"
        "encoding/base64"
        "errors"
        "fmt"
        "strings"
)

var (
        Client = "CLIENT"
        SharedSecret = "SHARED_SECRET"
        KeyProviderResponse = "KEY_PROVIDER_RESPONSE"
        PKCS5 = &pkcs5{}
)

type pkcs5 struct{}

func main() {
        s := strings.Split(KeyProviderResponse, "|")
        encryptedMessage, hash := s[0], s[1]
        generatedHash := generateKeyProviderHash(SharedSecret, Client,␣
↪encryptedMessage)
        if generatedHash == hash {
                decryptedKeys := decrypt(encryptedMessage, SharedSecret)
```

(continues on next page)

(continued from previous page)

```
                fmt.Printf(decryptedKeys)
        } else {
                fmt.Printf("hash failed validation")
        }
}

func generateKeyProviderHash(sharedSecret string, client string, encryptedMessage␣
↪string) string {
        v := fmt.Sprintf("%s%s%s", sharedSecret, client, encryptedMessage)
        hasher := sha1.New()
        hasher.Write([]byte(v))
        return fmt.Sprintf("%x", hasher.Sum(nil))
}

func decrypt(encryptedData string, sharedSecret string) string {
        source, _ := base64.StdEncoding.DecodeString(encryptedData)
        key := []byte(sharedSecret)[0:16]

        cipher, _ := aes.NewCipher([]byte(key))
        buffer := make([]byte, len(source))
        size := 16

        for bs, be := 0, size; bs < len(source); bs, be = bs+size, be+size {
                cipher.Decrypt(buffer[bs:be], source[bs:be])
        }

        plainBytes, _ := PKCS5.unPadding(buffer, 16)
        return string(plainBytes)
}

func (p *pkcs5) unPadding(src []byte, blockSize int) ([]byte, error) {
        srcLen := len(src)
        paddingLen := int(src[srcLen-1])
        if paddingLen >= srcLen || paddingLen > blockSize {
                return nil, errors.New("padding size error")
        }
        return src[:srcLen-paddingLen], nil
}
```

Python

```
import base64
import binascii
import json
import hashlib

from Crypto.Cipher import AES

client = "CLIENT"
shared_secret = "SHARED_SECRET"
key_provider_response = "KEY_PROVIDER_RESPONSE"

print("Decrypting Key Provider Response...")
split_key_provider_response = key_provider_response.split("|")
```

(continues on next page)

(continued from previous page)

```
pre_computed = (shared_secret + client + split_key_provider_response[0]).encode("utf-8
↪")
computed_hash = hashlib.sha1(pre_computed).hexdigest()
if computed_hash == split_key_provider_response[1]:
    key = shared_secret[0:16]
    decoded_text = base64.b64decode(split_key_provider_response[0])
    aes = AES.new(key.encode("utf8"), AES.MODE_ECB)
    padded_text = aes.decrypt(decoded_text)
    unpadded_text = padded_text[:-padded_text[-1]]
    decrypted_keys = json.loads(unpadded_text)
    print("Decrypted keys: " + json.dumps(decrypted_keys))
else:
    print("Hash validation failed for key provider response!")
```

Using the encryption keys

The next section explains the various encryption keys provided by the Legacy JSON Key Provider API. Each set has a different use case. For use with USP products *CENC* and *Fairplay* encryption keys are recommended. See the Unified Streaming Integration section below for more details on how to use `mp4split` with the Legacy JSON Key Provider API.

This section uses the Legacy JSON Key Provider API but the JSON keys retrieved from the CPIX key provider can be used in the same way.

CENC

CENC is the Common Encryption Scheme and it standardises encryption keys between different DRM systems. This allows a single set of encryption keys to be use to encrypt a single file using different DRM systems. VUDRM supports Widevine and PlayReady when generating CENC encryption keys.

Make the following request to the Legacy JSON Key Provider API in order to retrieve `cenc` Keys.

```
curl -X GET https://keyprovider.vudrm.tech/cenc/<CLIENT>/<CONTENT_ID> -H 'API_KEY:
↪<API_KEY>'
```

The keys returned in this response can be used for PlayReady and Widevine scenarios. They allow a single piece of content to be used across multiple devices and browsers.

An example decrypted response would be:

```
{
    "key_id_big": "qMTumUz5drgbusWY+fi5LA==",
    "key_id_hex": "A8C4EE994CF976B81BBAC598F9F8B92C",
    "content_key": "ETweRxyDIuDqAadsWdx0HQ==",
    "content_key_hex": "113C1E471C8322E0EA01A76C59DC741D",
    "playready_key_iv": "dd80b66b2fe44be4",
    "playready_laurl": "https://playready-license.vudrm.tech/rightsmanager.asmx",
    "playready_checksum": "j+7cluk2J58=",
    "widevine_drm_specific_data": "Ig1zb21lY29udGVudGlkSOPclZsG",
    "widevine_laurl": "https://widevine-license.vudrm.tech/proxy"
}
```

The values are:

- `key_id_big`: Unique ID for the encryption. Base64 in Big Endian.

- `key_id_hex`: Unique ID for the encryption. Base16 in Little Endian. This one should be used with `mp4split`.
- `content_key`: 128bit encryption key in base64 in Big Endian.
- `content_key_hex`: 128bit encryption key in base16 in Little Endian. This one should be used with `mp4split`.
- `playready_key_iv`: Additional random value to strengthen the PlayReady encryption.
- `playready_laurl`: The PlayReady license server URL.
- `playready_checksum`: A PlayReady specific security value required by some older PlayReady solutions and *Wowza*.
- `widevine_drm_specific_data`: The Widevine PSSH box.
- `widevine_laurl`: The Widevine license server URL.

Fairplay

Fairplay is Apple's DRM system and is commonly used in conjunction with CENC encryption to provide support to the widest amount of devices possible.

Make the following request to retrieve `fairplay` keys from the Legacy JSON Key Provider API:

```
curl -X GET https://keyprovider.vudrm.tech/fairplay/<CLIENT>/<CONTENT_ID> -H 'API_
↪KEY: <API_KEY>'
```

An example decrypted response would be:

```
{
    "key_hex": "457A8E3300DE6D549A95037F1C7ADEB1",
    "iv_hex": "EB86EAFBD487391383E8FFF957561B0C",
    "laurl": "skd://fairplay-license.vudrm.tech/license/somecontentid"
}
```

The values are:

- `key_hex`: Unique ID for the encryption.
- `iv_hex`: The encryption key.
- `laurl`: The Fairplay license server URL. At the point of the request to the license server the `skd` protocol should be replaced with `https`.

HLS AES encryption

HLS AES-128 is the Advanced Encryption Standard using a 128 bit key, Cipher Block Chaining (CBC) and PKCS7 padding.

Make the following request to retrieve HLS AES encryption keys from the Legacy JSON Key Provider API:

```
curl -X GET https://keyprovider.vudrm.tech/aes/<CLIENT>/<CONTENT_ID> -H 'API_KEY:
↪<API_KEY>'
```

An example decrypted response would be:

```
{
    "key_hex":"dd682004123622a99c2a2afcdad6217c",
    "key_url":"http://keyprovider.vudrm.tech:9293/aes/getkey/vualto/somecontentid"
}
```

The values are:

- `key_hex`: Base16 AES Content key
- `key_url`: URL to the AES Key Server

PlayReady

PlayReady is Microsoft's DRM system. Only use these keys directly to target PlayReady enabled devices. The use of CENC keys is preferred.

Make the following request to retrieve `playready` keys from the Legacy JSON Key Provider API:

```
curl -X GET https://keyprovider.vudrm.tech/playready/<CLIENT>/<CONTENT_ID> -H 'API_
↪KEY: <API_KEY>'
```

An example decrypted response would be:

```
{
    "key_id":"kX9efHkfIS++X+m8kZPDOw==",
    "key_id_guid":"7c5e7f91-1f79-2f21-be5f-e9bc9193c33b",
    "key_id_uuid":"917f5e7c791f-2f21-be5fe9bc9193c33b",
    "key_id_hex":"917F5E7C791F212FBE5FE9BC9193C33B",
    "content_key":"eI5JjujIo1Ek7lqO+3gg0A==",
    "content_key_hex":"788E498EE8C8A35124EE5A8EFB7820D0",
    "laurl":"http://vualto.playready-license.vudrm.tech/rightsmanager.asmx",
    "service_id":"gwICI8yfIUGf4R/5qOWuqg==",
    "service_id_guid":"23020283-9fcc-4121-9fe11ff9a8e5aeaa",
    "key_iv":"07261ee6ee074f1d",
    "checksum":"wf0goCGB2O4="
}
```

The values are:

- `key_id`: Unique ID for the encryption. Base64 in Big Endian.
- `key_id_guid`: Unique ID for the encryption. GUID in Big Endian.
- `key_id_uuid`: Unique ID for the encryption. UUID in Little Endian.
- `key_id_hex`: Unique ID for the encryption. Base16 in Little Endean. This one should be used with `mp4split`.
- `content_key`: 128bit encryption key in base64.
- `content_key_hex`: 128bit encryption key in base16. This one should be used with `mp4split`.
- `laurl`: The PlayReady license server URL.
- `service_id`: Unique VUDRM service ID for Vualto in Base64.
- `service_id_guid`: Unique VUDRM service ID for Vualto as a GUID.
- `key_iv`: Additional random value to strengthen the PlayReady encryption.
- `checksum`: Extra security value required by some older PlayReady solutions.

Widevine

Widevine is Google's DRM system. Use these keys to target Widevine enabled devices directly. The use of CENC keys is preferred.

Make the following request to retrieve `widevine` keys from the Legacy JSON Key Provider API:

```
curl -X GET https://keyprovider.vudrm.tech/widevine/<CLIENT>/<CONTENT_ID> -H 'API_
↪KEY: <API_KEY>'
```

An example decrypted response would be:

```
{
    "key_id":"NxB8uJFHVSuaO5BjiMzuEQ==",
    "key_id_hex":"37107CB89147552B9A3B906388CCEE11",
    "content_key":"Unyx1s3fMFUedA688fCNxw==",
    "content_key_hex":"527CB1D6CDDF30551E740EBCF1F08DC7",
    "laurl":"https://widevine-license.vudrm.tech/proxy",
    "drm_specific_data":"CAESEDcQfLiRR1UrmjuQY4jM7hEaBnZ1YWx0byIFdGVzdDEqAkhEMgA="
}
```

The values are:

- `key_id`: Unique ID for the encryption. Base64 in Little Endian.
- `key_id_hex`: Unique ID for the encryption. Base16 in Little Endian. This one should be used with `mp4split`.
- `content_key`: 128bit encryption key in base64.
- `content_key_hex`: 128bit encryption key in base16. This one should be used with `mp4split`.
- `laurl`: The Widevine license server URL.
- `drm_specific_data`: The Widevine PSSH box.

Unified Streaming Integration

The encryption keys provided by the Legacy JSON Key Provider APIs are compatible with Unified Streaming Platform's mp4split product.

The recommended approach is to call the CPIX Key Provider API to retrieve CENC and Fairplay Keys. Once the encryption keys have been retrieved they can be used with mp4split to generate an ism:

```
mp4split --license-key=$LICENSE_KEY -o $ISM \
 --iss.key=${key_id_hex}:${content_key_hex} \
 --iss.key_iv=${playready_key_iv} \
 --iss.license_server_url=${playready_laurl} \
 --widevine.key=${key_id_hex}:${content_key_hex} \
 --widevine.license_server_url=${widevine_laurl} \
 --widevine.drm_specific_data=${widevine_drm_specific_data} \
 --hls.client_manifest_version=4 \
 --hls.key=:${key_hex} \
 --hls.key_iv=${iv_hex} \
 --hls.license_server_url=${laurl} \
 --hls.playout=sample_aes_streamingkeydelivery
```

The HLS values come from the Fairplay encryption keys, all other values are from the CENC keys.

1.2.6 VUDRM TOKEN

The VUDRM token has two purposes: authentication and the delivery of the DRM policy to the license server. It also represents a signed authorisation on the client's behalf for Vualto to issue a DRM license to the holder of the token, issuing a VUDRM token to a player will grant that player access to the DRM-protected content.

Due to the first purpose VUDRM tokens have a limited lifetime and are designed to be single use. Please contact support@vualto.com if you do not know what the VUDRM token's TTL is for your account.

The second purpose allows the user's playback rights for individual pieces of content to be set dynamically. A single user may be granted different rights on a single piece of content depending on business requirements.

VUDRM tokens should be generated using the *VUDRM token API*. The request to the *VUDRM token API* should be made from a server side application and the VUDRM token should then be delivered to the client side for use by a player in a license request.

VUDRM token structure

```
test-client|2018-26-11T11:01:04Z|cO9H1B2VKw0WyyNTOfoHEw==|05aff2dd06f4c52291b7a032c815ce8f5
```

The VUDRM token is comprised of four components each separated by the pipe character (ASCII code 124):

- The client name.
- The time the token was generated in an ISO8601 format (yyyy-MM-ddThh:mm:ssZ).
- The encrypted DRM policy.
- A signed hash.

DRM policy

The DRM policy is included in the VUDRM token as the encrypted third component. When using multiple DRM providers the parameters that are applicable to `ALL` will work across all DRM technologies. Parameters specific to one DRM provider can be used but will be ignored if not relevant. For example, a VUDRM token can be created that pertains to both Widevine and PlayReady. Any PlayReady specific parameters will only be applied when a PlayReady licence is required and will be ignored for Widevine.

This table is not an exhaustive list, for example it does not include advanced PlayReady settings. If you require the use of more advanced settings please contact support@vualto.com

The `polbegin` and `polend` settings use the timezone set at the account level. Please contact support@vualto.com to confirm the timezone for your account.

Default values

*When these values have not been set in the policy, as long as no other values would cause playback to stop, content will play indefinitely.

There are also limitations depending on environments that are not explained in the table. Please refer to the following sections for more detail:

Match content id

If `content_id` has been set and `match_content_id` has been set to `true`, when a license request is made the content id in the license request and the content id in the VUDRM token will be compared. If they **are** the same a

license will be served as normal; if they are **not** the same the license request will be denied. If `match_content_id` has been set to `false` **or** `content_id` has not been specified then this comparision will not occur.

GEO whitelisting

If `geo_whitelist` has been added to the DRM policy, when a license is requested the GEO location of the client's ip will be checked. If the country is in the whitelist the license request will be successful. If the country is not in the whitelist the request will be denied. E.g. if the DRM policy is `{ "geo_whitelist":[ "gbr", "deu" ] }` only users from the UK and Germany will be able to make successful license requests.

DRM Session in policy

It is possible to set in the policy used by VUDRM token information about the current DRM session. With this information we will make a request to make to an API of your choosing with an ID and any needed headers. You can set the request to be either a GET request or a POST request. In the case of a GET request we would make a GET request to the URL you specify with a query string parameter called `sessionId` set to ID the passed in the policy. E.g. a GET request to `https://some-url.com/valid?sessionId=abc123`. In the case of a POST request we would make a POST request to the URL you specify with the body of the request being the `sessionId` in JSON. E.g. a POST request to `https://some-url.com/valid` with the body being `{"sessionId":"abc123"}`. The result of the request we make will dictate whether or not a license is returned. If the result **is** a `200` we will serve a valid license back. If the result is **not** a `200` we will not return a valid license.

There are four parts to the session information:

The structure of the session information in a policy should be as follows:

```
{
    ... other policy values ...
    "session": {
        "id": "...",
        "url": "...",
        "method": "...",
        "headers": { "headerKey": "headerValue", ... }
    }
}
```

Example policy with session information

```
{
    "polbegin": "02-04-2019 12:00:00",
    "polend": "02-04-2019 17:00:00",
    "session": {
        "id": "someID",
        "url": "https://www.some-url.com/valid",
        "method": "GET",
        "headers": { "myHeader": "someValue", "myOtherHeader": "someOtherValue" }
    }
}
```

Default DRM policy

It is possible to specify a default DRM policy that will be used when we receive a VUDRM token. For example if you wish for all licenses requested to default to caching the license, the default DRM policy would be `{ "liccache":"yes" }`, meaning a VUDRM token with the policy `{ "polend":"DD-MM-YYYY HH:mm:ss" }` would be the same as `{ "liccache":"yes", "polend":"DD-MM-YYYY HH:mm:ss" }`. The values in the default policy can be overridden by simply putting them in the policy you pass in the VUDRM token. For example if the default DRM policy was `{ "liccache":"yes" }` but you wanted a license to not be cached, the policy in the VUDRM token would be `{ "liccache":"no" }`. Any of the values listed above can be be set in the default DRM policy.

If you wish to utilise this feature please contact support@vualto.com.

PlayReady DRM policy

PlayReady has an extensive list of policy options described here. The majority of these options are supported via the VUDRM token's policy. Please contact support@vualto.com for more details.

Fairplay DRM policy

Fairplay has three types of license; `rental`, `lease`, and `persist`.

Fairplay licenses can only be persisted past the user session by an offline enabled iOS application. When using Safari a session will be preserved until the browser tab is closed.

Fairplay does not allow playback over non-HDCP connections.

Fairplay rental DRM policy

You can specify a `rental` license by setting the `type` key to `r` in the DRM policy.

You can set the expiry of the license using the `duration_rental` key or the `polend` key. If both `duration_rental` and `polend` are set in a policy then `duration_rental` will be used.

When using the `type` of `rental` playback will continue after license expiry until the encryption keys change.

Fairplay lease DRM policy

You can specify a `lease` license by setting the `type` key to `l` in the DRM policy.

You can set the expiry of the license using the `duration_lease` key or the `polend` key. If both `duration_lease` and `polend` are set in a policy then `duration_lease` will be used.

Using the `type` of `lease` will cause playback to stop at the license expiry. Normally a player will make a new license request at this point, please ensure the VUDRM token is updated before further license requests are made. Using an expired VUDRM token will cause the license request to fail.

Fairplay persist DRM policy

A `persist` license is used for offline playback.

Setting `liccache` to `yes` or the `type` to `p` will generate a persisted license. Setting `liccache` to `yes` will override any other `type` settings.

Setting `type` to `p` will override setting `liccache` to `no`.

You can set the expiry of the license using the `duration_persist` key or the `polend` key. If both `duration_persist` and `polend` are set in a policy then `duration_persist` will be used.

Using the `type` of `persist` will cause playback to stop at the license expiry and the license will be removed from the device.

Widevine DRM policy

Widevine licenses can only be persisted past the user session by an Android application. When using Chrome a session will be preserved until the browser tab is closed.

DRM policy examples

The following sections display some example polices. These polices apply to all DRM providers.

Rental

This policy is designed for a rental business model. The license generated from this policy will expire at the time set by the `polend` value and will allow immediate playback.

```
{
    "content_id":"filename",
    "polend":"DD-MM-YYYY HH:mm:ss",
    "type":"r"
}
```

Subscription

This policy is designed for a subscription business model. The license generated from this policy will not allow playback until `polbegin` is reached and will expire when `polend` is reached.

```
{
    "content_id":"filename",
    "polbegin":"DD-MM-YYYY HH:mm:ss",
    "polend":"DD-MM-YYYY HH:mm:ss",
    "type":"l"
}
```

Offline playback license

This policy is designed for an offline playback scenario. The license generated from this policy will allow playback immediately and the license will expire when the `polend` value is reached. The license will be cached until the `polend` value is reached.

```
{
    "content_id":"filename",
    "polend":"DD-MM-YYYY HH:mm:ss",
    "liccache":"yes"
}
```

VUDRM token API

VUDRM tokens can be generated by making a request to the VUDRM token api: https://token.vudrm.tech/generate

The request to the VUDRM token API needs to be a POST and requires an `API_KEY` header with your account API key as the value. Please contact support@vualto.com if you do not have this.

The body of the POST request comprises of the account name and the DRM policy.

For example:

```
{
    "client": "YOUR_NAME",
    "policy": {
        "content_id": "filename",
        "polend": "26-11-2018 16:48:59"
    }
}
```

An example request to the VUDRM token API in curl is:

```
curl -X POST \
  https://token.vudrm.tech/generate \
  -H 'API_KEY: <your-api-key>' \
  -d '{"client": "<client>","policy": {"content_id":"<content-id>","polend":"<pol-end>
↪","liccache":"no"}}'
```

1.2.7 GEO LOCATION API

This API can be used to retrieve geographical location information about IPv4 or IPv6 addresses.

This service requires an API Key and client name. Please contact support@vualto.com if you do not have this information.

1. Make a Request

A `GET` request should be made to the following URL:

`https://geo-location.vudrm.tech/ip_lookup/<client>/<ip_address>`

Replace `<client>` with your client name and `<ip_address>` with the IPv4 or IPv6 address.

The `X_AUTH_KEY` header should be set and the value should be the Geo Location API Key.

This is not the same as your VUDRM token API key.

2. The Response

The response will be in JSON and will look like this:

```
{
    "ip": "188.39.163.11",
    "geo": {
        "timezone_name": "europe/london",
        "country": "gbr",
        "country_code": 826,
```

(continues on next page)

(continued from previous page)


```
        "country_confidence_percentage": 99,
        "two_letter_country_code": "uk",
        "continent_code": 5,
        "region": "ply",
        "region_code": 25486,
        "region_confidence_percentage": null,
        "area_codes": null,
        "metro_code": 826046,
        "postal_code": "pl1 1ab",
        "postal_confidence_percentage": 30,
        "city": "plymouth",
        "city_code": 12140,
        "city_confidence_percentage": 60,
        "longitude": 0,
        "latitude": 50,
        "gmt_offset": "+0",
        "in_dst": false,
        "connection_speed": "xdsl"
    },
    "proxy": {
        "identification": null,
        "type": null
    }
}
```


The key `ip` details the requested IP address. The following sections explain the other sections in the returned JSON.

geo

NB:

- Confidence values range from 0 to 100, with 0 representing least confidence in data sources and 100 representing total confidence in data sources.
- All values are nullable.

proxy

Possible values for `proxy.identification`:

Possible values for `proxy.type`:

Example Requests

CURL

```
curl -X GET \
https://geo-location.vudrm.tech/ip_lookup/vualto-demo/188.39.163.11 \
-H 'X_AUTH_KEY: <geo-location-api-key>'
```

GO

```
package main

import (
        "fmt"
        "net/http"
        "io/ioutil"
)

func main() {

        url := "https://geo-location.vudrm.tech/ip_lookup/vualto-demo/188.39.163.11"

        req, _ := http.NewRequest("GET", url, nil)

        req.Header.Add("X_AUTH_KEY", "<geo-location-api-key>")

        res, _ := http.DefaultClient.Do(req)

        defer res.Body.Close()
        body, _ := ioutil.ReadAll(res.Body)

        fmt.Println(res)
        fmt.Println(string(body))

}
```

Ruby (Net:HTTP)

```
require 'uri'
require 'net/http'

url = URI("https://geo-location.vudrm.tech/ip_lookup/vualto-demo/188.39.163.11")

http = Net::HTTP.new(url.host, url.port)

request = Net::HTTP::Get.new(url)
request["X_AUTH_KEY"] = '<geo-location-api-key>'

response = http.request(request)
puts response.read_body
```

C# (RestSharp)

```
var client = new RestClient("https://geo-location.vudrm.tech/ip_lookup/vualto-demo/
↪188.39.163.11");
var request = new RestRequest(Method.GET);
request.AddHeader("X_AUTH_KEY", "<geo-location-api-key>");
IRestResponse response = client.Execute(request);
```

Python Requests

```
import requests

url = "https://geo-location.vudrm.tech/ip_lookup/vualto-demo/188.39.163.11"

headers = {
    'X_AUTH_KEY': "<geo-location-api-key>",
    }

response = requests.request("GET", url, headers=headers)

print(response.text)
```

PHP HttpRequest

```
<?php

$request = new HttpRequest();
$request->setUrl('https://geo-location.vudrm.tech/ip_lookup/vualto-demo/188.39.163.11
↪');
$request->setMethod(HTTP_METH_GET);

$request->setHeaders(array(
  'X_AUTH_KEY' => '<geo-location-api-key>'
));

try {
  $response = $request->send();

  echo $response->getBody();
} catch (HttpException $ex) {
  echo $ex;
}
```

1.2.8 VUDRM SUPPORT MATRIX

See bottom of page for caveats.

- Widevine mandates all browser CDM implementations to stay current with Chrome stable releases to ensure that the latest updates are applied. Older versions of Chrome, Firefox, and Opera may not be able to use DRM. See Widevine's deprecation schedule.
- VUDRM uses Widevine Modular by default (Widevine Classic is not supported).
- Some Smart TV Alliance models may not support Widevine.
- In Linux based systems, Widevine may not be supported on some versions of Chrome and Firefox.

1.3 RELEASE NOTES

- *December 2021*

1.3.1 December 2021

- Internal improvements to our Config API. Added functionality to integrate with JW Player systems.
- Internal improvements to our CPIX API.

1.3.2 November 2021

- Improvements to VUDRM Admin. Video.js and Dash.js added as an option for the public player.
- Improvements to our PlayReady license server. Added support for Token v2.
- Bug fixes on our FairPlay license server. Fixed an iOS issue when downloading content.
- Internal improvements to our Stats API.
- Internal improvements to our CPIX API.
- Internal improvements to our KP API.

1.3.3 October 2021

- Improvements to our FairPlay, Widevine, and KP API systems to be compatible with Token v2.
- Improvements to VUDRM Admin. HLS JS can now be enabled in the public player.
- Internal improvements to our Stats API.
- Internal improvements to our KP API.

1.3.4 September 2021

- Bug fixes on our CPIX API. Now returns the correct encryption schemes.
- Fixed issues with our PlayReady CircleCI tests.

1.3.5 August 2021

- Internal improvements to the Widevine license server. Added license renewal support.
- Internal improvements for our CPIX logging.

1.3.6 July 2021

- Updates to O/S on most services.
- Multikey support for all DRM providers added to CPIX API. Our documentation has been updated to reflect this.
- VUDRM Admin user guide has now been added to our public documentation.
- Various improvements to VUDRM Admin, including a new public demo page.
- Internal improvements for our Stats API logging.
- Beta release for VUDRM Token v2.

1.3.7 June 2021

- New VUDRM cluster based in Milan has been released and is serving traffic for the duration of the Euro 2020 Championship.
- Various improvements to VUDRM Admin, including the ability for the client to edit default VUDRM Token policy values.
- Various internal improvements to KP-API, Legacy KeyProvider API, Token API, FairPlay license server, GeoLocation API, Widevine license server and CPIX API. Updated builds and services used.

1.3.8 May 2021

- Various improvements to VUDRM Admin, including the integration of the JW Player client for testing purposes and Health page amends.

1.3.9 April 2021

- Improvements for our Widevine logging. Logs that show the request start and completion now appear in Kibana.
- Bug fixes in the VUDRM Admin site, including a client cloning issue displaying incorrect errors and the user sometimes being redirected to the Login page incorrectly.

1.3.10 March 2021

- Improvements to the AES license server. Now responds with a 4xx status code when the call to KP API fails.
- Improvements have been made to our Documentation. All documentation now follows a unified template throughout.
- Bug fix in the VUDRM Admin site where the Configuration page would sometimes break the service for certain clients.
- Improvements to the Widevine license server. Now forwards the Referer to the Stats API.
- Internal improvements to Kibana. Referer and License URL are displayed in Kibana.

1.3.11 February 2021

- Improvements to our cluster infrastructure. Pods will automatically recycle when they are 14 days old.
- Internal improvements to the KP API and Widevine license server. Settings for each can ammended via a VUDRM Token.
- Internal improvements to the Config API. Failure to connect to the Database will now respond with a 5xx status code.

1.3.12 January 2021

- Internal improvements to the ElasticSearch monitoring.
- Improvements to the Widevine license server - service has been re-written in Go.
- Various improvements to VUDRM Admin, including the ability to create a video for a client and JSON logging.

1.3.13 December 2020

- Elasticsearch migration to improve reliablity.
- Various improvements to VUDRM Admin, including a new status page, a HTTPS redirect, and speed improvements to the dashboard.

1.3.14 November 2020

- FairPlay license server now accepts a more standard request.
- Improved internal logging.
- All license requests can now accept a VUDRM token in the request body, header, or as a query string parameter.
- VPN's and TOR networks can now be blocked by setting `block_vpn_and_tor` in a VUDRM token's policy.
- Widevine license server now returns `x-request-id` instead of `vualto-transaction-id` bringing it inline with web standards.

1.3.15 September 2020

- Improvements to license server responses to aid in issue detection.
- New VUDRM cluster based in Paris has been released and is serving traffic.
- Improvement to internal apis to reduce length of license requests.
- `match_content_id` in token policy now support content encrypted with multiple keys from our CPIX API.

1.3.16 July 2020

- New VUDRM cluster based in Oregon has been released and is serving traffic.
- Improved internal logging.
- Improved stats recorded.
- NGINX VOD Module support added to CPIX API.
- Harmonic support added to CPIX API.
- GEO based restrictions in can now be set in VUDRM token policy.

1.3.17 May 2020

- New VUDRM cluster based in Bahrain has been released and is serving traffic.
- New endpoint for VUDRM CPIX API has been release to work with USP version 1.9.5.
- Infrastructure update to ensure latest security patches e.t.c are running on all VUDRM clusters.

1.4 SUPPORT

1.4.1 How to contact us

For all non-critical support requests please email support@vualto.com. If you are a registered user you can contact support via the help centre.

For all critical issues please contact support via one of the numbers below.

- +44 (0)800 0314391
- +44 (0)1752 916051
- (800) 857 1808 (toll-free US number)

When contacting our support about a DRM issue please include as much relevant info as possible, this should include but not be limited to the *x-request-id* for failing requests, the VUDRM token used, the VUDRM service, the DRM type/s, relevant content URL/s, response codes of the failed request, type of device/s the issue is occurring on, and browser type. If you can not get any of this information please state in you initial message that you are unable to provide this otherwise our first response will be asking for this information.

X-Request-ID

All requests made to and from VUDRM services will have a header called `x-request-id`. This header allows us to track a request through our backend; if you are getting unexpected errors from our services giving us the value of this header will allow us to efficiently track any errors that occurred while processing your request.

Vualto-Transaction-ID

On some requests you may find a header called `vualto-transaction-id`; this header was originally used for the same purpose as `x-request-id` header. In order to be more in line with industry standards, we are currently in the process of removing the `vualto-transaction-id` and replacing it with `x-request-id`. Please ensure that you use `x-request-id` contact support when contacting support and in any internal logging you perform.

Fairplay onboarding support

In order to configure Fairplay for use with VUDRM we need the following:

- Private key: This is normally named 'privatekey.pem' and is encrypted with a password.
- Password for private key, so that the private key can be decrypted.
- An 'Application Secret key' aka the ASk. This is normally named 'Ask.txt'.
- FairPlay certificate. This is normally named 'fairplay.cer'.

You can request these from Apple by visiting their FairPlay streaming page and clicking on "Request FPS Deployment Package".