VUDRM Documentation

VUALTO

May 23, 2019

Contents

Here you will find a range of tools to guide you in using VUDRM services. If you have any questions or need assistance in using this platform, please contact docs@vualto.com.

CHAPTER 1

Contents:

1.1 ENCRYPTION KEY PROVISION

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

1.1.1 Key provider API

The 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://key-provider.drm.technology/<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.

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://key-provider.drm.technology/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/
↪0QzqN27rupHF3UYnYQ40V3ESE2PUIe8Cs8471QRHlruYlwtgBBmGme2ERZWFjrRFEeUt8SobQkCIZrzDEKaQ2bJMyOy1yMt8ok
↪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 key provider API.

You will need the `CLIENT_NAME`, `SHARED_SECRET` and the `KEY_PROVIDER_RESPONSE` value from the request to the Keyprovider 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]);
```

(continues on next page)

(continued from previous page)

```
            if (computedHash == splitKeyProviderResponse[1])
            {
                Console.WriteLine("key providers response hash passed validation");
                var decryptedKeyProviderResponse =␣
↪Decrypt(splitKeyProviderResponse[0], SharedSecret);
                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();
        }
    }
```

(continues on next page)

(continued from previous page)

```
}
```

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, "|")
```

(continues on next page)

(continued from previous page)

```
        encryptedMessage, hash := s[0], s[1]
        generatedHash := generateKeyProviderHash(SharedSecret, Client,␣
↪encryptedMessage)
        if generatedHash == hash {
                decryptedKeys := decrypt(encryptedMessage, SharedSecret)
                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"
```

(continues on next page)

(continued from previous page)

```
shared_secret = "SHARED_SECRET"
key_provider_response = "KEY_PROVIDER_RESPONSE"

print("Decrypting Key Provider Response...")
split_key_provider_response = key_provider_response.split("|")
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 Key Provider API. Each set has a different use case. For use with USP products *CENC* and *Fairplay* encryption keys are recommended. See the *Use with mp4split* section for more details on how to use `mp4split` with the Key Provider.

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 Key Provider API in order to retrieve `cenc` Keys.

```
curl -X GET https://key-provider.drm.technology/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":"fF5/kR95LyG+X+m8kZPDOw==",
    "key_id_hex":"7C5E7F911F792F21BE5FE9BC9193C33B",
    "content_key":"eI5JjujIo1Ek7lqO+3gg0A==",
    "content_key_hex":"788E498EE8C8A35124EE5A8EFB7820D0",
    "playready_key_iv":"cd340bf920a34bfb",
    "widevine_drm_specific_data":"CAESEJrENOhpDl58uLPtkqsm0/
↪kaBnZ1YWx0byIgN0M1RTdGOTExRjc5MkYyMUJFNUZFOUJDOTE5M0MzM0IqAkhEMgA=",
    "playready_laurl":"http://playready.drm.technology/rightsmanager.asmx",
    "widevine_laurl":"https://widevine-proxy.drm.technology/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.
- `widevine_drm_specific_data`: The Widevine PSSH box.
- `playready_laurl`: The PlayReady license server URL.
- `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 Key Provider API:

```
curl -X GET https://key-provider.drm.technology/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.drm.technology/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 Key Provider API:

```
curl -X GET https://key-provider.drm.technology/aes/<CLIENT>/<CONTENT_ID> -H 'API_
↪KEY: <API_KEY>'
```

An example decrypted response would be:

```
{
    "key_hex":"dd682004123622a99c2a2afcdad6217c",
    "key_url":"http://keyprovider.drm.technology: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 Key Provider API:

```
curl -X GET https://key-provider.drm.technology/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.drm.technology/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 Key Provider API:

```
curl -X GET https://key-provider.drm.technology/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-proxy.drm.technology/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.

Use with mp4split

The encryption keys provided by the Key Provider API are compatible with Unified Streaming Platform's products.

The recommended approach is to call the Key Provider API and 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 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 tokens should not be generated on the client side.

1.2.1 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.

1.2.2 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.

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

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" }`.

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.

```
{
    "contentid":"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.

```
{
    "contentid":"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.

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

1.2.3 VUDRM token API

VUDRM tokens can be generated by making a request to the VUDRM token api: https://token-api.drm.technology/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.

The quotes in the DRM policy must be **escaped**.

For example:

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

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

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

1.3 SPEKE KEY PROVIDER API

These are the docs for the speke key provider, which has a speke endpoint for use with AWS.

1.3.1 Use with AWS Media Convert (VOD)

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 "https://speke-keyprovider-staging.drm.technology/vualto-demo/speke"

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 "https://speke-keyprovider-staging.drm.technology/vualto-demo/speke"

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 “https://speke-keyprovider-staging.drm.technology/vualto-demo/speke”
- 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://cdn.vuplay.co.uk/ibc-demo/index.html” in a new Google Chrome tab and select custom stream.
- Press the “Edit” button next to the player key box and type “arconics-staging|9116e9b8-5bd2-4ab1-a282-085ce379f8a6”
- Return to your S3 tab and copy the link from the bottom of the page and paste it into “Custom stream url” box, it should look something like “https://s3-eu-west-1.amazonaws.com/YOUR-BUCKET/YOUR-FOLDER-NAME/manifest.mpd”
- Generate a VUDRM token in the admin for your client and paste it into the “Custom VUDRM Token”
- Press “Load player”

Playready

- Open “https://cdn.vuplay.co.uk/ibc-demo/index.html” in a new Microsoft Edge tab and select custom stream.
- Return to your S3 tab and copy the link from the bottom of the page and paste it into “Custom stream url” box, it should look something like “https://s3-eu-west-1.amazonaws.com/YOUR-BUCKET/YOUR-FOLDER-NAME/manifest.mpd”
- Generate a VUDRM token in the admin for the your client and paste it into the “Custom VUDRM Token”
- Press “Load player”

1.3.2 Use with AWS Media Packager (LIVE)

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 "https://speke-keyprovider-staging.drm.technology/client-name/speke" where client-name is the name of the client.

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://cdn.vuplay.co.uk/ibc-demo/index.html" in a new Google Chrome tab and select custom stream.
- Press the "Edit" button next to the player key box and type "arconics-staging|9116e9b8-5bd2-4ab1-a282-085ce379f8a6"
- Paste in the endpoints URL for the "Custom stream url"
- Generate a VUDRM token in the admin for your client and paste it into the "Custom VUDRM Token"
- Press "Load player"

Playready

- Open "https://cdn.vuplay.co.uk/ibc-demo/index.html" in a new Microsoft Edge tab and select custom stream.
- Paste in the endpoints URL for the "Custom stream url"
- Generate a VUDRM token in the admin for your client and paste it into the "Custom VUDRM Token"
- Press "Load player"

1.3.3 Endpoints

Staging url: https://speke-keyprovider-staging.drm.technology/

1.3.4 SPEKE

To retrieve drm information formatted to the CPIX standard POST a request, formatting below, to **https://speke-keyprovider-staging.drm.technology/client-name/speke**, where "client-name" is the name of the client.

Please note when requesting a XML response Fairplay must be requested seperately to Widevine and Playready.

Examples

Requests

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 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>
```

(continues on next page)

(continued from previous page)

```
            </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>playreadyProtectionHeader</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">
    <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>
```

1.4 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.

1.4.1 1. Make a Request

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

`https://geo.drm.technology/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.

1.4.2 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,
        "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.drm.technology/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.drm.technology/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.drm.technology/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.drm.technology/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.drm.technology/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.drm.technology/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;
}
```

1.5 INTEGRATIONS

1.5.1 MOBILE

VUDRMWidevine SDK Documentation

VUDRMWidevine is an Android Archive (AAR) which can be used during the media rendering pipeline to provide a DRM plugin to ExoPlayer 2.9.6 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: v0.3.4 (272)

- Requirements
- Android Studio Integration
- Example Usage
- Error handling
- Devices tested on:
- Known Issues
- Release Notes

Requirements

- Minimum SDK version is 19 (Android 4.4)
- Android Studio 3.3.2

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.        Change the username and password to your credentials for authentication to␣
↪the maven repository

3.        Under your module’s build.gradle dependencies add:

        ```java
    implementation 'com.vualto.vudrm:widevine:0.3.4'
        ```

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:0.3.4'```

Example Usage

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

(continues on next page)
```

(continued from previous page)

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

### Online Streaming Sessions

1.        Instantiate an AssetConfiguration using the fluent interface, minimally you need to provide the contents KID and DRM token, the default license server in this case will be '<https://widevine-proxy.drm.technology/proxy>'.

        The default license server can also be overridden using the API call 'licenceUrlWith(<Your license URL>)'. An Asset ID is required only for offline streaming sessions storage and management.

        ```java
        try {
            assetConfiguration = new AssetConfiguration.Builder()
            .tokenWith(drmToken)
            .KIDWith(KID)
            .build();
                } catch (Exception e) {
                    // Handle exception
                }
        ```

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

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

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

        ```java
try {
            return new DefaultDrmSessionManager<>(vudrm.widevineDRMSchemeUUID,
            FrameworkMediaDrm.newInstance(vudrm.widevineDRMSchemeUUID),
            callback,
            null,
            new Handler(),
            null);
        } catch (UnsupportedDrmException e) {
            e.printStackTrace();
            return null;
        }
````

Offline Streaming Sessions

Offline sessions require a method of management that will enable the storage and recall of offline assets, with an associated content URL, and the contents offline license.

The management system will ensure that when offline the offline asset will not need to be loaded from the file path where it was stored on device, instead the system will call the original online URL, and the stored offline asset will load with the correct license. As demonstrated in our example app, Exoplayer offers us classes that act as such a
````

management system.

It is important to consider that while online, VUDRMWidevine handles generating the Widevine license only for the purposes of online playback or acquisition of an offline license. Offline licenses are persisted as standard Widevine licenses which means that the asset configuration (using `OfflineAssetConfiguration`) required for offline playback is constructed with minor differences to the online configuration (using `AssetConfiguration`).

1. Initially identical to online sessions, start by instantiating an (online) AssetConfiguration using the fluent interface. This configuration is used to acquire the offline license for later use when offline.

```
try {
        assetConfiguration = new AssetConfiguration.Builder()
        .tokenWith(drmToken)
        .KIDWith(KID)
        .build();
} catch (Exception e) {
        // Handle exception
}
```

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


                WidevineCallback callback = new WidevineCallback(assetConfiguration);

3.        When creating an ExoPlayer OfflineLicenseHelper you can pass this callback␣
↪as the component required when creating the DefaultDrmSessionManager
↪<FrameworkMediaCrypto> object.

        The OfflineLicenseHelper requires DrmInfo as ```drmInitData``` which can be␣
↪collated using ExoPlayers DashUtil class.

4.        An offline license can then be acquired using Exoplayers␣
↪OfflineLicenseHelper call:

        ```
byte[] offlineLicenseKeySetId = mOfflineLicenseHelper.downloadLicense(drmInitData);
```

```
The ```offlineLicenseKeySetId``` is the key binder between the offline asset and the␣
↪associated offline license.
```

1. Depending on the chosen set up, when the license has been acquired, the user may then download the asset to their device by calling the Exoplayer DownloadTracker with the content URL. This call will automatically create an assoicated Exoplayer DownloadAction and Exoplayer DownloadService for the given asset. Use these classes with the Exoplayer DownloadTracker to manage the assets and their current status. The DownloadTracker will detect whether the asset for the URL already exists.

   When the download is complete, offline playback can then be achieved by calling the asset using a VUDRM OfflineAssetConfiguration and a standard Widevine DefaultDrmSessionManager object using an HttpMediaDrmCallback that can be passed to the player.

```
```

try { OfflineAssetConfiguration assetConfiguration = new OfflineAssetConfiguration.Builder() .kidProviderWith(new HttpKidSource(new URL(streamUrl))) .build(); } catch (Exception e) { // Handle exception }
```

```
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

- 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:
<https://docs.vualto.com/projects/vudrm/en/latest/VUDRM-token.html>

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

## Release Notes
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
```

(continues on next page)

(continued from previous page)

```
- New major exoplanet release
- Bug fixes and improvements

v0.1.0 on 10/03/2017

- Initial release
```

iOS FairPlay SDK

The VUDRMFairPlay SDK is available for iOS. This documentation describes the steps to integrate and use the VUDRMFairPlay SDK on this platform.

Current release: v0.0.1 (85)

- Overview
- Requirements
- Xcode integration
- Information about Application Transport Security (ATS)
- Example framework usage
- Limitations
- Known Issues
- 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.

Three deployment scenarios exist, being Online Playback / Download and Offline Playback using an associated rental or persist token and two legacy methods which are restricted to Online Playback and use an associated rental 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.

As an evolution of the the hugely successful VUPLAY SDK, this SDK has a number of key benefits:

- Complete control over the entire AVPlayer lifecycle
- Numerous optimisations to enable faster playback
- Bitcode support
- Support for building/linking against the simulators

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

A demo application written in Swift is available on request. Please contact support@vualto.com to request a download of the SDK.

Requirements

- Minimum iOS deployment target of 9.x or higher
- Xcode 10.2
- Swift 5

Xcode Integration

1. Copy the vudrmFairPlay.framework into your application's project directory
2. Navigate to your target and add vudrmFairPlay.framework to your 'Embedded Binaries'. You should see that vudrmFairPlay.framework also now appears under 'Linked Frameworks and Libraries':

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

Before you can use the demo project or framework in your own project, for each instance you will require a URL to correctly prepared content, an asset ID for the content, and 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 9.x 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 10.x 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

Example framework usage

- Import the VUDRMFairPlay module into your view controller:

  ```
  import vudrmFairPlay
  ```

- Initialise an instance of VUDRMFairPlay:

Online Playback / Download and Offline Playback:

This method will request a license for the content from the license server based on the type of token presented for each instance of VUDRM FairPlay. 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 provide a persistent content key, which may be used for both online streaming, and offline playback of downloaded content.

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 that has been downloaded to the users device to an Apple TV. Please see the Limitations section for further information.

To initialise an instance, create an AVURL asset with your asset URL, and pass that in to the initialiser with the asset ID and a valid token.

Our example uses a single dynamically assigned instance. It would be expected that your application will manage the applications offline assets, including their ID's, locations and download status. This method uses AVAssetDownloadURLSession with a URLSessionConfiguration to create an AVAssetDownloadTask for each configured asset. The asset ID should be unique to each asset, as it is used to create a path to, and identify, offline assets and their associated content keys.

The current example initialises an AVAssetDownloadURLSession with a URLSessionConfiguration upon loading the ViewController:

```
backgroundConfiguration = URLSessionConfiguration.background(withIdentifier:
↪"AssetDownloadConfigurationIdentifier")
assetUrlSession = AVAssetDownloadURLSession(configuration: backgroundConfiguration!,
assetDownloadDelegate: self, delegateQueue: .main)
```

The asset configuration is then assigned upon use of the download or play buttons. To play an asset we create the assetDownloadTask and then assign the associated urlAsset to a player:

```
let asset = AVURLAsset(url: assetURL!)
self.drm = vudrmFairPlay(asset: asset, assetName: assetName, token: token)
assetDownloadTask = assetUrlSession.makeAssetDownloadTask(asset: asset, assetTitle:␣
↪assetName,
assetArtworkData: nil, options: nil)!
assetDownloadTask.taskDescription = assetName

let playerItem = AVPlayerItem(asset: assetDownloadTask.urlAsset)
let player = AVPlayer(playerItem: playerItem)
let playerViewController = AVPlayerViewController()
playerViewController.player = player
present(playerViewController, animated: true) {
player.play()
}
```

And to download, instead of assigning the asset to a player, call:

```
`assetDownloadTask.resume()`
```

To cancel a download, call:

```
`assetDownloadTask.cancel()`
```

Legacy methods:

These are the originally provided methods which are for online playback use only, and are provided for ease of upgrade for legacy users. These methods now also require an asset ID in the initialiser to be compatible with the new framework.

- Legacy method 1: Pass in a reference to your AVPlayer instance, the token to be used, and the asset ID when retrieving a license.

```
let player = AVPlayer(url: assetURL!)
self.drm = vudrmFairPlay(player: player, token: token, assetName: assetName)
```

- Legacy method 2: Pass in a reference to your stream URL and the token to be used when retrieving a license. An AVAsset will be created by VUDRMFairPlay and this can be used to create your player.

```
self.drm = vudrmFairPlay(url:assetURL! as NSURL, token: token, assetName: assetName)
let playerItem = AVPlayerItem(asset: (drm?.asset)!)
let player = AVPlayer(playerItem: playerItem)
```

Finally, present your player:

```
self.player = player
let playerController = AVPlayerViewController()
playerController.delegate = self as? AVPlayerViewControllerDelegate
self.present(playerController, animated: true, completion: nil)
playerController.view.frame = self.view.frame
playerController.player = self.player player.play()
```

DEMO

With the VUDRMFairPlay framework installed, the example demo application can be initialised either via source code in the IDE, or using field entry on device.

- To initialise via code in IDE, before running, simply replace the empty ViewController.swift values with your own valid values.

```
private var assetURL = URL(string: "")
private var token = ""
private var assetName = ""
```

- To initialise via view controller on device, populate the URL, Token, and Asset Name fields with valid values. For ease of use, pasting into the URL field in the form URL + space + Token will auto populate both fields correctly. At runtime, any values in the URL and Token fields will override any hard coded values.

Limitations

- The current example application demonstrates a single instance of offline playback using an AVAssetDownloadURLSession with a URLSessionConfiguration to create an AVAssetDownloadTask for that asset. The example is not able to handle multiple assets. Users are expected to develop a method of mapping instances to assets within their application which will avoid multiple downloads of the same asset and will enable management of

assets. The example includes legacy methods for online playback only. The use of these methods in conjunction with the default method may cause unexpected results and is not recommended.

- 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

- There is a known issue in iOS 11 where audio may not be correctly routed by the OS. This issue can be resolved by adding the following to `viewDidLoad` in your initial view controller:

```
do {
        if #available(iOS 10.0, *) {
                try AVAudioSession.sharedInstance().setCategory(.playback, mode: .
↪default, options: [])
        } else {
                // Fallback on earlier versions
    }
}
catch {
        print("Setting category to AVAudioSessionCategoryPlayback failed.")
}
```

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

Release notes

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

tvOS FairPlay SDK

The VUDRMFairPlay SDK is available for tvOS. This documentation describes the steps to integrate and use the VUDRMFairPlay SDK on this platform.

Current release: v0.0.1 (10)

- Overview
- Requirements
- Xcode integration
- Information about Application Transport Security (ATS)
- Example framework usage
- Limitations
- Known Issues
- 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.

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.

As an evolution of the the hugely successful VUPLAY SDK, this SDK has a number of key benefits:

- Complete control over the entire AVPlayer lifecycle
- Numerous optimisations to enable faster playback
- Bitcode support
- Support for building/linking against the simulators

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

A demo application written in Swift is available on request. Please contact support@vualto.com to request a download of the SDK.

Requirements

- Minimum tvOS deployment target of 11.x or higher
- Xcode 10.2
- Swift 5

Xcode Integration

1. Copy the vudrmFairPlay.framework into your application's project directory
2. Navigate to your target and add vudrmFairPlay.framework to your 'Embedded Binaries'. You should see that vudrmFairPlay.framework also now appears under 'Linked Frameworks and Libraries':

Information about application transport security (ATS)

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

Before you can use the demo project or framework in your own project, for each instance you will require a URL to correctly prepared content and a VUDRM token which must be correctly generated for the type of instance you wish to create. An example VUDRM token type template available for use with tvOS is:

- Fairplay Rental `{"type": "r","duration_rental": 3600}`

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

Example framework usage

- Import the VUDRMFairPlay module into your view controller:

  ```
  import vudrmFairPlay
  ```

- Initialise an instance of VUDRMFairPlay:

Pass in a reference to your stream URL and the token to be used when retrieving a license. An AVAsset will be created by VUDRMFairPlay and this can be used to create your player.

```
self.drm = vudrmFairPlay(url:assetURL! as NSURL, token: token)
let playerItem = AVPlayerItem(asset: (drm?.asset)!)
let player = AVPlayer(playerItem: playerItem)
```

Finally, present your player:

```
self.player = player
let playerController = AVPlayerViewController()
playerController.delegate = self as? AVPlayerViewControllerDelegate
self.present(playerController, animated: true, completion: nil)
playerController.view.frame = self.view.frame
playerController.player = self.player player.play()
```

DEMO

With the VUDRMFairPlay framework installed, the example demo application can be initialised either via source code in the IDE, or using field entry on device.

- To initialise via code in IDE, before running, simply replace the empty ViewController.swift values with your own valid values.

```
private var assetURL = URL(string: "")
private var token = ""
```

- To initialise via view controller on device, populate the URL and Token fields with valid values.

Limitations

- The SDK does not support playback of protected content when running in the tvOS Simulator. Attempting to do so will result in an invalid `KEYFORMAT` error and the content will not load.

Known Issues

- There are no known issues at this time
- If you believe you have found any issues, please contact us at support@vualto.com

Release notes

v0.0.1 (build 10) on 08/04/2019

- Initial release

1.5.2 PLAYERS

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

(continues on next page)

(continued from previous page)

```
    }
  }
};
```

PlayReady example

```
source.drm = {
    playready: {
        LA_URL: "https://playready-license.drm.technology/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 {
    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({
```

(continues on next page)

(continued from previous page)

```
                    ks: keySystems[ksIdx],
                    initData: initData
                  });
                }
              }
            }
          }
        }
        return supportedKS;
      }
    };
};

var widevineLaUrl = "https://widevine-proxy.drm.technology/proxy";

player.setProtectionData({
  "com.widevine.alpha": {
    serverURL: widevineLaUrl,
    httpRequestHeaders: {}
  }
});

player.attachSource(streamUrl);
```

PlayReady example

```
var playReadyLaUrl =
  "https://playready-license.drm.technology/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.

Basic setup

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

// where `player` is the id of a div
jwplayer("player").setup({
  playlist: [
    {
      sources: [
        {
          file: mpegDashStreamUrl,
          drm: {
            playready: playReadyDrmConfig,
            widevine: widevineDrmConfig
          }
        },
        {
          file: hlsStreamUrl,
          drm: {
            fairplay: fairplayDrmConfig
          }
        }
      ]
    }
  ]
});
```

Each `source` object within the `sources` array, has a `drm` property where a drm type specific configuration object can be passed. Examples of each type of configuration are provided below.

Widevine example

```
var widevineDrmConfig = {
  url: "https://widevine-proxy.drm.technology/proxy",
  licenseRequestFilter: function(request, drmInfo) {
    var keyId = drmInfo.keyIds[0].toUpperCase();
    var body = JSON.stringify({
      token: vudrmToken,
      drm_info: Array.apply(null, new Uint8Array(request.body)),
      kid: keyId
    });
    request.body = body;
    request.headers["Content-Type"] = "application/json";
  }
};
```

PlayReady example

```
var playReadyDrmConfig = {
  url:
    "https://playready-license.drm.technology/rightsmanager.asmx?token=" +
    encodeURIComponent(vudrmToken)
};
```

FairPlay example

```
var fairplayDrmConfig = {
  certificateUrl: fairplayCertUrl,
  processSpcUrl: "https://fairplay-license.drm.technology/license",
  extractContentId: extractContentId,
  licenseRequestHeaders: [
    {
      name: "Content-Type",
      value: "application/json"
    }
  ],
  licenseResponseType: "arraybuffer",
  licenseRequestMessage: createLicenseRequestMessage
};

var extractContentId = function(initData) {
  var laurlAsArray = initData.split("/");
  contentId = laurlAsArray[laurlAsArray.length - 1];
  return contentId;
};

var createLicenseRequestMessage = function(keyMessage) {
  var body = {
    token: vudrmToken,
    contentId: contentId,
    payload: base64EncodeUint8Array(keyMessage)
  };
  return JSON.stringify(body);
};

var base64EncodeUint8Array = function(input) {
  var keyStr =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";
  var output = "";
  var chr1, chr2, chr3, enc1, enc2, enc3, enc4;
  var i = 0;

  while (i < input.length) {
    chr1 = input[i++];
    chr2 = i < input.length ? input[i++] : Number.NaN;
    chr3 = i < input.length ? input[i++] : Number.NaN;

    enc1 = chr1 >> 2;
    enc2 = ((chr1 & 3) << 4) | (chr2 >> 4);
    enc3 = ((chr2 & 15) << 2) | (chr3 >> 6);
    enc4 = chr3 & 63;

    if (isNaN(chr2)) {
      enc3 = enc4 = 64;
    } else if (isNaN(chr3)) {
      enc4 = 64;
    }
    output +=
      keyStr.charAt(enc1) +
      keyStr.charAt(enc2) +
      keyStr.charAt(enc3) +
```

(continues on next page)

(continued from previous page)

```
      keyStr.charAt(enc4);
  }
  return output;
};
```

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 Googles' Shaka player.

Basic setup

```
<div id="video"></div>
```

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

shaka.polyfill.installAll();
if (shaka.Player.isBrowserSupported()) {
  var video = document.getElementById("video");
  shakaPlayer = new shaka.Player(video);
  shakaPlayer.addEventListener("error", onErrorEvent);
}
```

Widevine example

```
shakaPlayer.configure({
  drm: {
    servers: {
      "com.widevine.alpha": "https://widevine-proxy.drm.technology/proxy"
    }
  }
});

shakaPlayer
  .getNetworkingEngine()
  .registerRequestFilter(function(type, request) {
    if (type != shaka.net.NetworkingEngine.RequestType.LICENSE) return;

    var selectedDrmInfo = shakaPlayer.drmInfo();
    if (selectedDrmInfo.keySystem !== "com.widevine.alpha") {
      return;
    }

    var keyId = selectedDrmInfo.keyIds[0].toUpperCase();

    var body = {
      token: vudrmToken,
```

(continues on next page)

(continued from previous page)

```
      drm_info: Array.apply(null, new Uint8Array(request.body)),
      kid: keyId
    };
    body = JSON.stringify(body);
    request.body = body;
    request.headers["Content-Type"] = "application/json";
  });

shakaPlayer
  .load(streamUrl)
  .then(function() {
    console.log("The stream has now been loaded!");
  })
  .catch(onError);
```

PlayReady example

```
var playReadyLaURL =
  "https://playready-license.drm.technology/rightsmanager.asmx?token=" +
  encodeURIComponent(vudrmToken);
shakaPlayer.configure({
  drm: {
    servers: {
      "com.microsoft.playready": playReadyLaURL
    }
  }
});

shakaPlayer
  .load(streamUrl)
  .then(function() {
    console.log("The stream has now been loaded!");
  })
  .catch(onError);
```

No FairPlay support

Shaka does not currently support FairPlay.

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-proxy.drm.technology/proxy"
        }
      }
    }
  ]
};
```

PlayReady example

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

FairPlay example

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