PlayReady Notes Stage

Jules TIMMERMAN

S2 2022-2023

1 Overview

1.1 PlayReady

- PlayReady est une technologie développée par Microsoft pour les DRMs
- Différents serveurs (non fourni par PlayReady) :
  - *Packager* (pas fourni par PR) : Chiffre le contenu du service, génère le header associé et stocke le fichier ainsi que la paire de clés (KID pour identifier le contenu, Content Key pour déchiffrer) sur un serveur (Key Management System / Web Server).
  - *Key Management System* (idem) : stocke les paires KID / CK pour déchiffrer un contenu identifié par son KID.
- Une autre solution pour le packager est d'utiliser une génération aléatoire des CK avec une seed commune au serveur et d'utiliser le KID comme entrée. Cela permet de ne stocker que la seed et de recalculer la CK à partir du KID lorsque demandé.
- Différents serveurs pour les applications (fourni par PlayReady) :
  - *License Server* : chargé de donner des licenses aux utilisateurs c'est-à-dire les clés pour déchiffrer un contenu + les restrictions associées (par exemple, les dates limites etc...)
  - *Metering Server*
  - *Secure Stop Server*
  - *Secure Delete Server*
  - *Domain Server* : permet de grouper des appareils pour qu'ils puissent utiliser les mêmes licenses. Par exemple, permet l'embedding des licences dans le contenu, transfert de contenu d'un appareil à l'autre sans télécharger à nouveau.
- *Content Decryption Module (CDM)* : Client local qui s'occupe du déchiffrement (peut le faire avec ou sans hardware etc...), du respect des policies des licences.
- *License Store* : Stocke les licenses de manière chiffrées
- Souvent deux types de licences : Persistantes et Temporaires
- *Binding* : chiffrement d'une licence avec la clé d'une entité (machine / domain / ...)
- *License Nonce* : mis dans une demande de license pour éviter de juste renvoyer la même requête.

1.2 Security Levels

3 niveaux [1] :

- 150 : développement / tests. Aucune protection
- 2000 : Protection software (ou hardware)
- 3000 : Protection hardware (TEE). Contenu haut qualité

1. Security levels

1.3 EME

- Spécification du Web pour les protocoles DRMs
- Fournit une interface JS qui peut utiliser n'importe quel DRM
- Key System : couple CDM / License Server
- Fonctions appelées pour License Acquisition [2] :
  - `requestMediaKeySystemAccess` : Handler vers le DRM spécifié
  - `createMediaKeys` : Instantie le CDM (donne un objet `MediaKeys`)
  - `createSession` : Crée une session (permet d'isoler les origines) avec un session ID. Permet aussi de spécifier temporaire ou persistant (on recharge le session ID avec `load` plus tard)
  - `generateRequest` : crée une requête de License Acquisition. Dépend du Key System utilisé et est donc un message opaque (transférer ensuite du CDM au License Server)
  - `update` : Transmet la réponse du LS au CDM

1.4 License Acquisition

- Une licence permet de déchiffrer un contenu chiffré
- Contenu d'une licence :
  - Content Key : pour déchiffrer
  - Rights : les droits de la licence. Par exemple : Play
  - Conditions / Policies : Limitation de la licence. Par exemple : EndDate (ou custom cf XMR)
  - Données spécifiques à l'application
- Lors d'une requête de licence, le client fourni [3]
  - Client Certificate : Données sur le client, unique (mais peut changer donc ne pas utiliser pour identifier, privilégier les Platform IDs)
  - Content Header avec le/les KIDs
  - Données spécifiques à l'application
- Le transfert de licences peut se faire en HTTP ou HTTPS.
- Le protocole est basé sur le SOAP ($\sim$ XML) et peut être customisé.
- Après s'être authentifié auprès de l'Authentification Service (indépendant de PlayReady), on nous donne un Token que le client communique au License Server qui demande ensuite à l'AS si l'utilisateur est bien autorisé à accéder aux licences qu'il demande.

1.5 Client Certificate

- Identifie de manière unique les clients (mais ne pas utiliser car peut changer)
- Créé pendant la Client Initilization, ce qui peut arriver à l'usine, au boot, plus ou moins souvent... Par exemple, à distance : PlayReady Remote Provisioning
- Transmis lors de la demande de licence
- Contient :
  - Manufacturer name
  - Model name
  - Security Level
  - Version
  - Supported features
  - Unit Client ID

2. *Your DRM can watch you too : [...]* (Figure 1)
3. License and Policies

— Aussi [4]
  — Chaîne de certificats
  — Platform
  — Type
— cf `clientInfo.txt` des tests Bitmovin pour une liste
— *Remarque* : toujours pas de liste exacte en ayant déchiffré la requête

1.6 SOAP

— Protocole de base utiliser pour la communication avec le License Server
— Basé sur du XML
— Blocs de base :
  — *Enveloppe* : Contient les données et identifie le XML comme du SOAP
  — *Header*
  — *Body*
  — *Fault message* : optionnel, code d'erreur HTTP 500

1.7 Chiffrement

— Lors de la LA, HTTP ou HTTPS
— Protocoles [5] :
  — Contenu : AESCTR ou AESCBC (spécifié dans le PR Header)
  — Signature (protocole) / Chiffrement Content Keys : ECC
  — Signature (licence / transient keys / data) : AES OMAC1

1.8 CDMi

cf. [6]
— Interface entre EME et DRM
— Permettre l'implémentation des DRMs dans les navigateurs open-source

1.9 License Store

— Aussi appelé Hashed Data Store (HDS)
— Où les licences sont stockées
— Peut être persistent ou non
— Potentiellement plusieurs HDS, par exemple un par application, site...

1.10 Licence

Dans une réponse de LA :
— Base64 puis au format Extensible Media Rights (XMR)

4. Test Client Info
5. Content Encryption
6. CDMi Specification

2 Reverse

2.1 Test Bitmovin

- Test fait sur Edge
- Utilisation du Key System `com.microsoft.playready`
- `listernersAdded_ : true` dans l'objet MediaKeys ? ?
- `isTrusted` : attribut de l'évènement pour savoir si il a été initié par le navigateur ou pas

Résultats :

- cf `clientInfo.txt`
- Plusieurs essais de `RequestMediaKeySystemAccess` (3 blocs)
- Demande de licence avec des données chiffrées
- Le contenu de la demande est chiffré en AES CBC
- La clé symétrique (pour la demande) est chiffrée par ECC256, KeyName : `WMRMServer`
- Licence : Base64 puis XMR
- Dans `audio/videoCapabilities` : `encryptionScheme : null`
- `distinctiveIdentifier : not-allowed` dans les promesses
- `pssh` dans le header de LA

2.2 Bitmovin XMR (mieux plus bas)

Comparaison de licences dans plusieurs cas cf `testXMR.txt`

- Remarque : lorsque persistant, plus de choses dans la réponse de LA

2.3 License Comparison

Comparaison de licenses (deux vidéos différentes)

- Même début du message
- On voit les policies au début en clair
- Souvent, la fin d'un champ de policies est marqué par un null byte (0x00)
- Schéma général :
  - Segment qui annonce les données (avec un ID à la fin ?)
  - Données
  - Null Byte
- Au début, on voit le Nonce de la license (que l'on a pu observer lors de la requête de licence)
- Vers la fin, champs de 64 byte (512 bits) dépendant du client, pas chiffré (constant)
  - Sûrement un hash de Client Info
  - Semble pas être le Digest / SignatureKey
  - Unique par origine, ne survit pas au reset.
- A la fin, champs 128bits
  - Signature de la license avec AES OMAC1 : vérifier l'intégrité sans reparler au serveur
  - Taille de 128 bits semble coller avec la block-cipher de AES OMAC1
- Content Key (sûrement au milieu)
  - Taille : 1024 bits
  - 1 chiffrement ECC ElGamal 256 : 512 bits (2 éléments de la courbe)
  - Il semblerait donc qu'on chiffre 2 blocs
- Dans AES CBC, champ supplémentaire à la fin
- Pistes :

- License Chaining : il faut réussir à mettre l'information quelque part (bound à quelque chose)
- Alg ID
- Rights (ie Play Right)

2.4 PoC Test Server Client Info

- Avec l'option `msg:clientinfo`, le serveur sert de reflector et renvoie le Client Certificate
- Ce Client Certificate est usuellement chiffré (cf AES + ECC) lors de la License Acquisition
- Idée : Faire une requête "en secret" au serveur de test pour déchiffrer le contenu de la requête
- Intéressant si le contenu du Client Certificate est unique / stable (par exemple, l'Issuer Key)
- Dans la spécification EME : "*message MUST NOT contain Distinctive Permanent Identifier(s), even in an encrypted form. message MUST NOT contain Distinctive Identifier(s), even in an encrypted form, if the MediaKeySession object's use distinctive identifier value is false.*" [7]
- Test [8]
  - Aucune information stockée sur le serveur (il faudrait demander à l'utilisateur s'il souhaite participer)
  - Issuer Key
    - Semble avoir de la stabilité (même résultat entre la 26/05 et 08/06)
    - Unicité liée au ModelNumber
    - ModelNumber lié à la mise à jour
  - Client Time : -2h (décalage sur GMT mais basé uniquement l'heure du PC)
  - Cert 0 Digest : varie quand même alors que toutes les informations sont identiques (mais stable avec reboot et refresh)
  - Public Signing Key : Stable sur plusieurs jour
  - PSK et Digest sont uniques par origin (test avec localhost et perso)
  - Edge : clear des données Media Foundation change les valeurs
  - Ne semble pas forcément être un Distinctive Identifier puisque facilement clearable / ne dépend pas forcément d'un Distinctive Permanent Identifier (puisque ça change ?). Si cf le passage sur l'individualization

2.5 myCanal

- Beaucoup de configurations PlayReady (avec plein de Key System)
- Utilisation de `com.microsoft.playready.recommandation.3000`
- `distinctiveIdentifier : required`
- `GenerateRequest`
  - Custom Attributes : `encryptionref`
- `Update`
  - Multiple licenses at the same time

2.6 Media Foundation EME Sample

Étude du code [9] :

7. EME Specification
8. PoC
9. MediaFoundation Git

— But : comprendre le fonctionnement du DRM PlayReady sur la machine sur un projet plus petit que Edge (pour localiser le DLL etc... avec une approche dynamique)
— Étudier les mêmes requêtes que dans EME et comparer (`generateRequest`), trouver le parsing de licence
— Remarque : ce ne sont que des interfaces, on peut l'utiliser avec n'importe quel CDM qui implémente le tout
— `IMFContentDecryptionModuleFactory`
  — `CreateContentDecryptionModuleAccess` : basé sur EME (`keySystem` en str). Donne accès à une instance de `IMFContentDecryptionModuleAccess`
  ∼ `requestMediaKeySystem`
  — Point de départ de tout. Créer avec la magie CoCreateInstance (cf EmeFactory qui crée une classFactory pour avoir ensuite un CDMFactory)
— `IMFContentDecryptionModuleAccess`
  — Basé sur EME `MediaKeySystemAccess`
  — `CreateContentDecryptionModule` : Construit le CDM. Renvoit une instance de `IMFContentDecryptionMod`
  ∼ `createMediaKeys`
— `IMFContentDecryptionModule`
  — `CreateSession` : Renvoit une instance `IMFContentDecryptionModuleSession`
  ∼ `createSession`
— `IMFContentDecryptionModuleSession` ∼ `MediaKeySession`
  — `onmessage` callback (`App::OnEMEKeyMessage`)
  — `generateRequest`
— `IMFContentProtectionManager`
  — Implémenté par `MediaEngineProtectionManager`
  — `BeginEnableContent` : appelé automatiquement lorsque nécessaire pour la lecture. Dispatch ensuite au CDM
  — Semble ne jamais être appelé ? ? ?
  — Un Content Enabler spécifie le type d'action à faire
— `IMFContentEnabler`
  — Une étape qui doit être fait pour déchiffrer, par exemple demande de licence
— DLL : `Windows.MediaProtection.PlayReady.dll` dans System32
— Reverse avec Ghidra + debugger VS
  — On utilise le .exe pour obtenir certains Data Types pour le DLL
  — On peut typer petit à petit les fonctions grâce à la doc
  — Problème : le DLL semble obfusquer et le code se construit au chargement : on dump le DLL de la mémoire avec WinDBG et on analyse plutôt ça
  — Problème : impossible à débugger ? Certaine fonction (qui je pense sont appelées) ne break pas + le code change en fonction de s'il y a un breakpoint ou pas (cf expérience avec `tempParseLicense` quand on met le breakpoint avant ou après `Update`)
— Ce DLL semble être lié à la documentation C# [10]
— Autre Sample encore plus minimaliste (pas trouvé le DLL) [11]

2.7 Only PlayReady

— Objectif : lire une autre vidéo (un autre stream) que celle de test de C++
— Un peu difficile de travailler avec des streams DASH : on télécharge tous les segments / init et on concatène pour travailler avec un seul fichier

10. Documentation UWP
11. https ://github.com/microsoft/Windows-universal-samples/tree/main/Samples/PlayReady

— On doit trouver le Header PlayReady : soit dans le mpd, soit dans le fichier directement (plus de précisions [12])
— Dans WinRT (C++), on utilise les fonctions UWP : `Windows.Media.Protection.PlayReady`
— Avec UWP :
  — On créer un MediaProtectionManager [13] (peu importe le DRM) où l'on donne comme propriété le bon GUID de DRM
  — On setup les callbacks, notamment `ServiceRequested`.
  — Pour chaque DRM, les events devront être converti en spécifique [14]
  — Dans l'app C#, ils connectent avec un `MediaElement` le playback

3 Questions / Pistes / TODO

✓ "Client stack" ? (cf Initialization) : sûrement juste : "maintenant ça marche"
✓ Contenu exact du Client Certificate
— Problème de privacy si certificat transmis en HTTP
✓ Creuser les chiffrements possibles
— Suite des clés de chiffrement
— CDMi [15]
— Documentations supplémentaires [16]
✓ Réussir le test avec clientInfo et le testServer...
— Provisioning (récupération du Client Certficate)
✓ Plugin dans WideXtractor pour les appels EME
✓ Bitmovin pour tester les DRMs
— KeyName : `WMRMServer`
— Stockage persistant des données
✓ Renouvellement licence ? (pas de Message Event avec realtime sur Bitmovin (juste une erreur `MEDIA _ERR_DECODE` de Media Foundation), pas de policy partiulière non plus)
— Comparer des licences pour essayer de comprendre XMR
✓ UWP PlayReady [17]
✓ Media Foundation (UWP avec EME sample) [18]
✓ Transaction ID (pour l'acknowledgement a priori) : lien avec metering
— Creuser persistance (cf. plus de choses dans la réponses)
✓ Cookie `ARRAffinity` : lié à Azure
— Secure Stop / Secure Delete en profondeur

12. DASH PlayReady specification
13. MediaProtectionManager
14. PlayReady Service Events
15. CDMi Specification
16. Documents
17. UWP PlayReady ; API
18. MediaFoundation Git