SECURITY RESEARCH

IDEAS FOR MICROSOFT PLAYREADY SECURITY IMPROVEMENTS

Last update: 30-01-2023

SECURITY RESEARCH

DISCLAIMER

INFORMATION PROVIDED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW NEITHER ADAM GOWDIAK SECURITY REASEARCH, ITS LICENSORS OR AFFILIATES, NOR THE COPYRIGHT HOLDERS MAKE ANY REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR THAT THE INFORMATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS, OR OTHER RIGHTS. THERE IS NO WARRANTY BY ADAM GOWDIAK SECURITY REASEARCH OR BY ANY OTHER PARTY THAT THE INFORMATION CONTAINED IN THE THIS DOCUMENT WILL MEET YOUR REQUIREMENTS OR THAT IT WILL BE ERROR-FREE.

YOU ASSUME ALL RESPONSIBILITY AND RISK FOR THE SELECTION AND USE OF THE INFORMATION TO ACHIEVE YOUR INTENDED RESULTS AND FOR THE INSTALLATION, USE, AND RESULTS OBTAINED FROM IT.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL ADAM GOWDIAK SECURITY REASEARCH, ITS EMPLOYEES OR LICENSORS OR AFFILIATES BE LIABLE FOR ANY LOST PROFITS, REVENUE, SALES, DATA, OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, PROPERTY DAMAGE, PERSONAL INJURY, INTERRUPTION OF BUSINESS, LOSS OF BUSINESS INFORMATION, OR FOR ANY SPECIAL, DIRECT, INDIRECT, INCIDENTAL, ECONOMIC, COVER, PUNITIVE, SPECIAL, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND WHETHER ARISING UNDER CONTRACT, TORT, NEGLIGENCE, OR OTHER THEORY OF LIABILITY ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION CONTAINED IN THIS DOCUMENT, EVEN IF ADAM GOWDIAK SECURITY REASEARCH OR ITS LICENSORS OR AFFILIATES ARE ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS.

INTRODUCTION

Microsoft PlayReady certificate may be generated in a dynamic manner for a client device with the use of a group cert (common for same device models).

This implicates the risk of secrets theft (PlayReady group cert and key) from one device only such as the demonstrated in CANAL+ environment [1]. This also implicates no knowledge of the subscriber that makes use of the stolen secrets and that relies on a fake identity for malicious purposes.

All of the above makes attribution harder too as the attacker can easily spoof the identity of other innocent users (STB serial and MAC sufficient for that - these data can be sometimes acquired from a store, where CANAL+ STBs are sold).

This documents presents two brief ideas of which goal is to make impersonation of PlayReady client devices harder to accomplish. It also describes an idea to provide support for license server "synchronization" / CDN authentication through the concept of a delegation token.

While the presented ideas are quite generic and have not been verified in practice, we still believe they constitute a potentially interesting starting point for exploration.

Some PlayReady environments offer hardware features. In our opinion, these should be always taken into account as breaking hardware is usually harder to accomplish when compared to software means. Our experiences indicate this is not the case (environment of CANAL+ STi7111 based STB devices).

The reason for it could be either:

- an old / outdated PlayReady SW in CANAL+ boxes,
- no use of STi7111 hardware security features by PlayReady at all.

Additionally, the base PlayReady technology (PlayReady SDK) doesn't support authentication and authorization. We believe a minimal support for these should be an integral part of the technology though. It could help:

- avoid situation where authentication is implemented improperly (such as demonstrated in CANAL+ environment),
- provide a primitive for transfer of authentication state to other services or interfaces (such as CDN), so that access to content would correspond to the client and its license terms.

The content of PlayReady Server Agreements [2] indicate that licensees cannot proceed with custom changes to PlayReady protocol, the licensing mechanism, etc.

In some way this leaves them at the mercy of the vendor to implement additional security features specific to their environments.

PlayReady content protection implemented in software[1] and on a client side has little chances of a "survival" (understood as a state of not being successfully reverse engineered and compromised). In that context, this is vendor's responsibility to constantly increase the bar and with the use of all available technological means.

#1 HARDWARE BINDING OF A PLAYREADY CERTIFICATE

All CANAL+ set-top-boxes contain the functionality to pair the STB with a smart card. Similar idea could be used for PlayReady certificates as they could be bound to the STB too.

For STB devices relying on STi7111 SoC, the pairing function relies on a HW secret unique to the chip (SCK key) to establish CWPK (Control Words Pairing Key). It has the following cryptographic formula:

$$CWPK = TDES_{dec}(CWPK_{enc}, SCK)$$

This pairing key can be later used for decrypting CW (Control Word) keys[2]:

$$CW = TDES_{dec}(CW_{enc}, CWPK)$$

While security of STi711 chipset pairing has been shown to be insecure (Conax CAS control words could be extracted from the chip as depicted in [3] and [4]), SCK key compromise hasn't been demonstrated so far[3]. In that context and regardless of the vulnerabilities present in STi7111 chipsets, SCK key is worth considering as a feature that could help in the verification of the box identity (client device).

BASE PRIMITIVES

STi7111 chipset provides hardware support for AES operations in CBC mode with the use of SCK key too:

$$OUTPUT_{enc} = AES_{cbc-enc}(INPUT, SCK, IV)$$

[1] such as client for PlayReady SL2000 level
[2] for Conax CAS, CW correspond to the keys changed each 10s and used to encrypt SAT TV signal.
[3] compromise of SCK key would be devastating for the security of a pairing function.

$$INPUT = AES_{cbc-dec}(OUTPUT_{enc}, SCK, IV)$$

The above functionality is used to implement FLASH / root file system security (among others).

The leaf of a PlayReady certificate is composed of the following fields:

```
### CERT
  - random
    0000:  be e2 7c bf 64 aa c0 c9 4c d6 0f f2 8a 05 e1 b4  ..|.d...L.......
  - seclevel 2000
  - uniqueid
    0000:  2c 24 1b 10 43 99 e0 33 30 41 34 30 37 32 44 38  ,$..C..30A4072D8
  - pubkey_sign
    0000:  1f 34 53 e7 68 e2 4f 0a 86 3c 02 60 2d 17 e4 24  .4S.h.O..<..-..$
    0010:  cf 12 8f 96 6b 6d 29 a7 be 71 f1 14 89 e5 78 ad  ....km)..q....x.
    0020:  77 37 5e fe ba e9 93 f1 bb f3 79 b2 7f 1e 08 00  w7^.......y.....
    0030:  86 fd 1b 30 e4 cf 33 36 82 cc a2 e2 b8 ef 9c a1  ...0..36........
  - pubkey_enc
    0000:  d2 96 26 1e 5f 0b f2 4c c0 73 4d 76 c8 10 ac ef  ..&._..L.sMv....
    0010:  7b 49 2c 16 04 07 8a 51 9c 6f 54 58 6c bf be df  {I,....Q.oTXl...
    0020:  ea a2 6c f7 29 cc 0f 74 75 d2 20 bd cf 64 fa 69  ..l.)..tu....d.i
    0030:  fa 9e f2 80 c5 66 f0 83 c3 a0 80 d5 70 df ca 0d  .....f......p...
  - digest
    0000:  ad 80 a5 d2 03 9d de e5 5e ea 4a 58 5a c9 e3 c9  ........^.JXZ...
    0010:  9f 7a df c6 d9 22 47 cf 9f 45 ae a9 3c 62 6d e2  .z..."G..E..<bm.
  - signature
    0000:  2b be 7d ba 8a 71 bb 61 1d 58 29 fc 39 71 a7 9b  +.}..q.a.X).9q..
    0010:  fe b9 a5 0e a4 af 61 6c 24 14 21 17 70 19 e5 d2  ......al$.!.p...
    0020:  8b 0f 5c 9f 9f 06 ab 48 e4 0a e2 3b 2b 79 59 66  .......H...;+yYf
    0030:  8c a0 bb 98 41 92 57 55 d5 51 2e c9 2c f5 32 ff  ....A.WU.Q..,.2.
  - signkey
    0000:  2b be 7d ba 8a 71 bb 61 1d 58 29 fc 39 71 a7 9b  +.}..q.a.X).9q..
    0010:  fe b9 a5 0e a4 af 61 6c 24 14 21 17 70 19 e5 d2  ......al$.!.p...
    0020:  f8 36 8d d6 69 a2 12 fb 74 cf 56 48 12 03 6f e3  .6..i...t.VH..o.
    0030:  f3 87 7f 05 61 cf 10 fa c2 83 db 43 0d 88 98 37  ....a......C...7
```

Each leaf certificate contains ECC signature of the cert data conducted with the use of the signature cert (group cert in this case). This signature has a fixed size (0x40 bytes or 512 bits).

GENERIC IDEA

PlayReady certificate could be bound with the STB device through the cryptographic pairing function SIG_{mac}defined as following[4]:

$$SIG_{mac} = SHA_{512}(AES_{cbc}(SHA_{512}(ECC_{signature}), SCK, iv))$$

[4] device certificate pairing relying on SCK HW key was initially described in the README.md file accompanying Microsoft Play Ready research material [1]

The idea is to create CBC-MAC / HMAC like [5][6] Message Authentication Code of a PlayReady leaf certificate with the use of SCK secret key unique to the target device.

While ECC signature verifies the integrity of the PlayReady certificate content, the goal of the signature MAC is to make sure the certificate was created by a legitimate device.

The resulting SIG_{mac} value could be made part of the PlayReady leaf certificate through the implementation of an additional *BCert* attribute. Such an additional attribute could be ignored for implementations with no support for the pairing.

The pros

- Server can verify that a given certificate has been generated by a target device (server verifies a pairing of SCK key with PlayReady certificate, the identity embedded in the cert is used for that)
- Valid certificate carries attribution information (target STB can be easily identified and blacklisted)

The cons

- Certificate theft and use on a desktop still possible
- Access to SCK keys (or network APIs dedicated to SIG_{mac} values verification) at server side (the need to integrate with CAS solution or provide interface for such an integration)

#2 HARDWARE BINDING OF A PLAYREADY LICENSE REQUEST

In order to eliminate the risk related to the use of a stolen PlayReady certificate (such as the one that has been bound to the hardware), the pairing function could be used for the license request too.

PlayReady license request data is issued to the license server as SOAP XML data:

```
<?xml version="1.0" encoding="utf-8"?>
 <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

  <soap:Body>
   <AcquireLicense
    xmlns="http://schemas.microsoft.com/DRM/2007/03/protocols">
    <challenge>
     <Challenge
      xmlns="http://schemas.microsoft.com/DRM/2007/03/protocols/messages">
```

```
<LA xmlns="http://schemas.microsoft.com/DRM/2007/03/protocols"
Id="SignedData" xml:space="preserve">
<Version>1</Version>
 <ContentHeader>
  <WRMHEADER
   xmlns="http://schemas.microsoft.com/DRM/2007/03/PlayReadyHeader"
   version="4.0.0.0">
   <DATA>
   ...
```

The integrity and authenticity of the license request is protected with the use of a cryptographic signature (`<SignatureValue>` XML tag).

This signature could be also bound to the hardware with the use of a pairing function $LICSIG_{mac}$similar to the one defined previously.

In order to eliminate the risk of replay attacks, the SIG_{mac} calculation should rely on a time too[5]. The time should correspond to the time of issuing a license request:

$$LICSIG_{mac} = SHA_{512}(AES_{cbc}(SHA_{512}(ECC_{signature}|TIME),SCK,TIME))$$

The resulting $LICSIG_{mac}$ value along *TIME* could be made part of the PlayReady license request (through the implementation of additional XML attributes). Again, such additional attributes could be ignored for implementations with no support for the pairing.

The pros

- Server can verify that a given license request has been generated by a target device (server verifies a pairing of SCK key with the license request, the server also verifies whether the request time is valid)
- The attacker needs constant access to STB (theft of a PlayReady certificate is not sufficient)
- Valid license requests reveal attacker's identity (target STB can be easily identified and blacklisted), this helps in tracking and anomalies detection

The cons

- Access to SCK keys (or network APIs dedicated to $LICSIG_{mac}$ values verification) at server side (the need to integrate with CAS solution or provide interface for such an integration)

[5] the time obtained from secure clock service could be used for that.

#3 ADDITIONAL AUTHENTICATION / DELEGATION TOKEN

PlayReady might provide support for additional authentication token to be returned upon successful verification of the license request.

By default, PlayReady license response contains data that is assumed to be consumed by the license requestor only (a client):

```
LICENSE
  CUSTOM DATA
    UserToken:       135abf9b-7006-46f2-9b3f-52f5d79f361a-stb
    BrandGuid:       448ab54c-d127-45f6-b651-4c59aee2f431
    LicenseType:     NonPersistent
    BeginDate:       2022-06-27T10:29:41.2331761
    ExpirationDate:  2022-06-29T10:29:41.2331761
    ErrorCode:       0
    TransactionId:   auto:56
  XMR LICENSE
   version: 3
     attr: 0001 OuterContainer
       attr: 0036 Unknown
       data
         0000:  00 00 00 39 00 00 00 18 d8 27 66 78 a6 c2 be 44  ...9.....'fx...D
         0010:  8f 88 08 ae 25 5b 01 a7                          ....%[..
       attr: 0002 GlobalContainer
         attr: 000d Unknown
         data
           0000:  00 01                                            ..
         attr: 0032 DWORD_Versioned
         data
           0000:  00 00 00 40                                      ...@
         SecurityLevel
           level: SL2000
       attr: 0009 KeyMaterialContainer
         ContentKey
           key_id
             0000:  04 25 01 44 01 78 12 4f a5 51 56 ca f9 7a 2f f5
           v1:           1
           v2:           3
           enc_data_len: 0080
           enc_data
             0000:  4d 6e 63 6a 80 5d ea a2 20 e4 5f dc 1b 3a b8 07
             0010:  84 c5 d5 0c 91 37 69 6e 94 71 b6 0c 1a 20 f0 de
             0020:  24 79 38 8a 04 b0 02 e8 d2 4c fb 19 4d 24 b5 7e
             0030:  e5 08 12 f3 28 46 76 82 43 13 34 20 d3 01 15 61
             0040:  03 04 71 a9 19 a5 98 c3 43 67 42 1e 5a 50 5e 8c
             0050:  ac bf b4 c6 af b6 6d 58 7a c9 7a 3d 41 a2 d0 cb
             0060:  dd 76 04 fd b5 02 2c 07 11 65 d6 53 0f 03 5c 66
             0070:  e2 45 09 eb 0b 49 e2 db 99 9a d7 44 ce aa 8e e5
         attr: 002a ECCDeviceKey
         data
           0000:  00 01 00 40 d2 96 26 1e 5f 0b f2 4c c0 73 4d 76  ...@..&._..L.sMv
           0010:  c8 10 ac ef 7b 49 2c 16 04 07 8a 51 9c 6f 54 58  ....{I,....Q.oTX
```

```
    0020:  6c bf be df ea a2 6c f7 29 cc 0f 74 75 d2 20 bd  l.....l.)..tu...
    0030:  cf 64 fa 69 fa 9e f2 80 c5 66 f0 83 c3 a0 80 d5  .d.i.....f......
    0040:  70 df ca 0d                                      p...
attr: 000b Signature
data
  0000:  00 01 00 10 46 f5 e0 76 c1 87 f6 89 8d a7 cd e7  ....F..v........
  0010:  be ad a6 64                                      ...d
```

License data primarily carries information about license terms and content key. As such, it does not seem to be suitable for other uses[6] such as verification of access to CDN content. Such an access should still be a subject of a verification and monitoring for several reasons:

- to prohibit unauthorized access to CDN, such an access could facilitate CDN copy[7]
- to detect malicious patterns (massive downloads, access to excessive number of assets, parallel access to content from devices that display one content at a time, etc.).

It would be the best to make use of the hardware pairing function idea depicted as #1 and #2 for CDN requests too. Taking into account the requirements for the check (HW support, access to SCK keys or network APIs dedicated to SIG_{mac} values verification) and the fact that CDN services might be outsourced to a 3rd party (CANAL+ case), such an implementation might not be possible in practice.

The license response could however embed additional cryptographic token ($DELEGATION_{token}$) signed by the license server, which could act as a proof that a license to given content had been successfully issued for a client (that a client is both legitimate and authorized to access given content). Such a delegation token could accompany requests to other services comprising content provider infrastructure (i.e. CDN).

$DELEGATION_{token}$ could embed base information about both the client and the license that was granted to it. This includes, but is not limited to the following data ($TOKEN_{data}$):

- client identity (UUID preferred than STB SERIAL and MAC addresses)
- content ID (UUID form preferred than URL[8])
- random data
- license number (the number of licenses issued to the client)
- token validity time (directly corresponding to license period for content ID)

[6] or pure transfer.

[7] a content in encrypted form should still be perceived as an asset requiring protection, CDN may serve content encrypted with the use of static keys (CANAL+ case), providing unlimited access to download such a content gives the advantage to the attacker (attackers just needs the key to "unlock" the content).

[8] UUID can hide the map to real content URL at CDN level.

- public part of $DELEGSIGN_{key}$ (to identify the key used at the server side for the purpose of $DELEGATION_{token}$ verification)

ECC signature of the above (conducted with the use of a content provider specific $DELEGSIGN_{key}$ server key dedicated solely for the purpose of issuing and verification of delegation tokens) should be appended to the end of a $DELEGATION_{token}$. Such data could constitute the base for any request issued to content provider service requiring verification of license rights to content ($REQUEST_{envelope}$).

$REQUEST_{envelope}$ could be BASE64 encoded and provided to CDN service as HTTP cookie[9].

CDN service could proceed with the verification of the request for content by verifying the data and authenticity of the received $DELEGATION_{token}$ (signature verification and checking that access to content is done within the period granted by the license server in particular).

Upon successful verification and for speed purposes[10], CDN service could optionally issue a short-lived, browser session based crypto token for use in consecutive requests (following the initial one).

ADDITIONAL CLIENT SIGNING

Due to the fact that CDN network could rely on plaintext HTTP protocol only (again, CANAL+ case), sending $DELEGATION_{token}$ over the plaintext communication channel creates a risk that it could be sniffed from the network and used by a 3rd party (reuse / replay attack).

This could be mitigated by additionally signing the $REQUEST_{envelope}$ with a temporary (generated and valid with given delegation token only) ECC key ($REQSIGN_{key}$).

A unique $REQSIGN_{key}$ could be provided[11] by the license server as part of the license response (it could be an integral part of a PlayReady license issued to the client).

The $TOKEN_{data}$ should be modified to reflect this change as well by including the public part of the $REQSIGN_{key}$ in it:

[9] maximum size of a HTTP cookie is 4096 bytes for most web browsers, this should provide sufficient space to accommodate request envelope data.
[10] to avoid license tokens verification at the time of each movie fragment request.
[11] generated at the time of issuing each new license and as such valid in the context of given license only.

- client identity
- content ID
- random data
- license number
- token validity time
- public part of $DELEGSIGN_{key}$
- public part of $REQSIGN_{key}$

The reason for this is to provide credibility (through signature) of the $REQSIGN_{key}$, which can be used by PlayReady client to sign arbitrary $REQUEST_{envelope}$ data.

The signing should take place at the time of requesting data from a target service such as CDN. The $REQUEST_{envelope}$ data should be extended too and beside signed $DELEGATION_{token}$, it might include information about the following:

- envelope id (corresponding to request id, incremented each time $REQUEST_{envelope}$ is generated for given $DELEGATION_{token}$)
- envelope validity time (issue / signing and expiration time) – contrary to license period, this should correspond to shorter time range (validity of the request should be estimated upon CDN response time, but it should not be longer than 30 sec), please note that shorter times might implicate the need for clock synchronization
- any other data specific to the client and required by target service.

CDN service should internally hold information corresponding to the last successfully verified $REQUEST_{envelope}$ received from given client (as session based data). Such an information should be held for the time corresponding to the validity of the request.

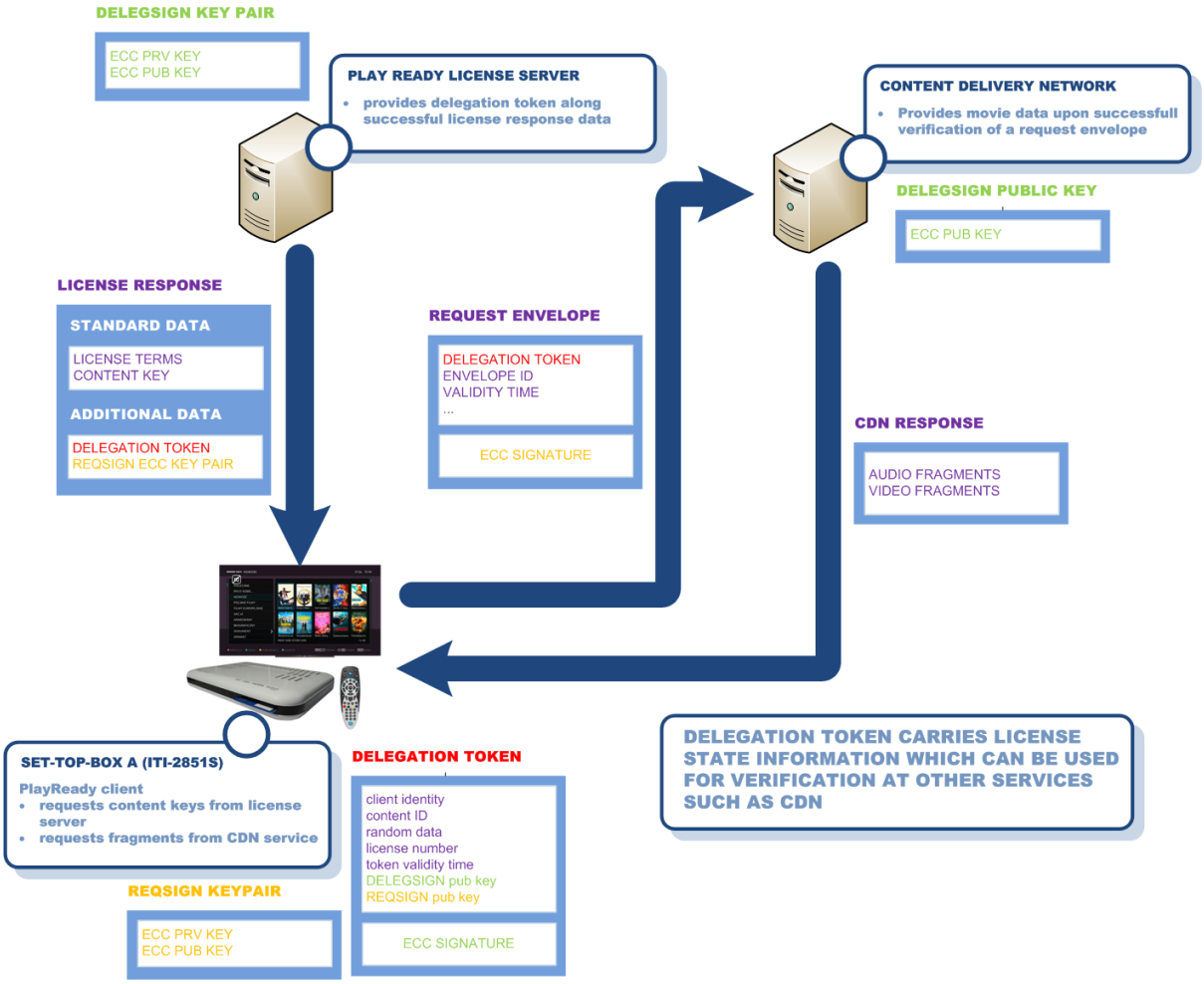
CDN should also allow one session for a given client[12], which implicates the need to keep track of a single tuple $(content\ id, envelope\ id)$ per client (updated upon access to new content or cleared at the time of a browser session termination).

CDN should reject (and log) any request corresponding to same content id with envelope id value less or equal to the one held internally. Reception of such a request would correspond to the duplicate request (such as sniffed one). Only STB has the means to generate legitimate CDN requests (those signed with $REQSIGN_{key}$

[12] the assumption is that a client corresponds to one hardware device (STB), there is no way for the client to watch two different movies (request two different content ids from CDN) at the same time (in parallel)

of which private part is available on a client device only and with envelope id value being increased for each new request).

An illustration of the described mechanism of an additional authentication token and relationships between various data is shown on Pic 1.



Pic 1 Additional PlayReady authentication token and its use for CDN access verification.

SHARED CRYPTO TOKEN FOR CDN ACCESS

Verification of a $DELEGATION_{token}$ at the time of each request to CDN content might not be efficient from a point of view of CDN response time (A/V playback).

The $REQSIGN_{key}$ embedded in the $REQUEST_{envelope}$ could be however used to deliver a shared secret from the server to PlayReady client in a secure manner. More specifically, public key of $REQSIGN_{key}$ could be used to encrypt such a value.

Security of CDN access used in CANAL+ environment relies on a shared secret too. The problem is that this secret is same for all STB devices and VOD collections and

its knowledge is sufficient to access any CDN content (no authentication / sync conducted with respect to license server, access irrespective of license terms).

An idea of a delegation token accompanied by a dedicated ECC key pair ($REQSIGN_{key}$) can be extended to implement secure and fast access to CDN content though (or more generally, to any service of a 3rd party provider that needs to verify client's license status for a given asset).

A browser session based $shared\_secret$ could be generated in a random fashion and issued as a response to a successful verification of the initial $REQUEST_{envelope}$ received by CDN (as HTTP cookie):

$$shared\_secret \quad = RANDOM()$$

$$encrypted\_shared\_secret \quad = ECC_{encrypt}(shared\_secret, REQSIGN_{pub})$$

The above could be decrypted by the client:

$$shared\_secret \quad = ECC_{decrypt}(encrypted\_shared\_secret, REQSIGN_{prv})$$

A shared secret could be used to generate a security cookie for fast verification of client access to CDN. Such a cookie could be the function of the following:

$$cdn\_security\_cookie \quad = crypto\_function\ (client\_identity, time, shared\_secret)$$

All consecutive requests to CDN corresponding to PlayReady client browser session could rely on this security cookie (cookie value being verified at CDN server side before serving any content).

The pros

- no need for integration with a content provider infrastructure (beyond the knowledge of a $DELEGSIGN_{key}$ for the verification of requests)
- access to content gets verified (access granted to authorized clients only), decision about access can be conducted without access to clients id database
- access to content is synced with license server (content returned only if within the allowed time period), though no actual syncing needs to be implemented as license state is carried in a delegation token
- excessive / unusual number of requests or CDN download volume can be detected

The cons

- STB compromise and a theft of $DELEGATION_{token}$ and associated $REQSIGN_{key}$ data can still allow for content access (custom generation of $LICENSE_{envelope}$ data), but such an access implicates content compromise any way (download and decryption of content)

REFERENCES

[1] Microsoft Play Ready security research
https://security-explorations.com/mspr_cplus_info.html

[2] Microsoft PlayReady server agreements
https://www.microsoft.com/playready/licensing/server

[3] Security vulnerabilities of Digital Video Broadcast chipsets, HITB talk #2
https://security-explorations.com/materials/se-2011-01-hitb2.pdf

[4] Exploitation Framework for STMicroelectronics DVB chipsets, technical report
https://security-explorations.com/materials/SRP-2018-02-report.pdf

[5] CBC-MAC
https://en.wikipedia.org/wiki/CBC-MAC

[6] HMAC
https://en.wikipedia.org/wiki/HMAC