Reverse Engineering PlayReady CDM for EME

Jules TIMMERMAN, Mohamed SABT

SPICY, IRISA

I. Introduction

In 2022, the online video streaming industry was worth around 445 billion USD [6] and is still growing tremendously. Online platforms need to protect their content, usually using cryptography. They encrypt their content (usually videos and musics) using a key they store. They send this secret decryption key to authenticated users that were granted access to said content, usually through a subscription, an online purchase or a rental. Streaming services also need to limit access to the resource once the key was acquired: users should only be able to view the video for the period they were allowed. They should not be able to export it or watch it after their subscription has ended. This is what a Digital Right Management (DRM) system ensures.

Many big companies made their own DRM technologies such as Google's Widevine [4], Microsoft's PlayReady [11] and Apple's FairPlay [1]. But DRM technologies work differently from one another with no common ground. Streaming platforms needed a common interface to be able to use the Web as a way to provide their content. With HTML5 native media playback support, the W3C created in 2012 along Netflix, Google and Microsoft, the Encrypted Media Extension (EME) [21]. This specification created a common JavaScript interface to use DRMs on the web, without the need of plugins. Additionally, browsers do not have to implement an actual DRM themselves: they only have to call existing Content Decryption Modules (CDMs) on the user's device, that is, to redirect the function calls to the actual CDM function. EME was a step toward DRM interoperability on the Web.

There are many concerns and controversies that rose with EME. Many are due to EME not allowing open-source browsers to implement DRM support as they would need to rely on proprietary code, making DRM interoperability harder for newer and/or smaller browsers [2]. It would also mean that the service is the one controlling your computer. This is what "Trusted Computing", sometimes dubbed "Treacherous Computing", is. Other issues are privacy concerns. Indeed, because the License Server (LS) needs information to identify your device, some of these information might be used in a malicious way to track your browser. Additionally, DRMs can be used to fingerprint your browser and identify your device by checking which configurations are supported. This has already been used on Reddit's website [20].

Because DRMs rely on closed source software, reverse engineering becomes the first step in studying it, whether it is for a security or a privacy research. Some DRMs have already been studied like Google's Widevine [18, 19], but some were not closely examined. We plan on investigating PlayReady, in particular the content of the EME messages and how the actual CDM binary works. We also studied potential privacy issues related to PlayReady Test Server. Finally, we examined how licenses were encoded.

II. Background

A. Digital Right Management Technology

A Digital Right Management (DRM) system is a technology that enables companies to provide access to media content, be it music, video, books or software, while protecting their content against piracy and enforcing rules to abide by to the user's device. A DRM system is characterized by two main components: the *License Server* (LS) and the *Content Decryption Module* (CDM). We call this pair a Key System. The CDM is the part responsible for enforcing the rules and decrypting content on the local device. The LS on the other hand, is the entity responsible for providing licenses to the user, so it can consume the content. A license in the context of DRMs is the decryption key (also called Content Key) used to read the content as well as the restrictions and rights associated with said key. There exist different restrictions, allowing many business models. The most common restrictions are time restrictions, granting access to video for only a certain amount of time, requiring that you ask the LS for a new license (or a renewal) if you want to watch the content after it expires.

There are other components that have a role in the DRM ecosystem. Usually, there is the Content Delivery Network (CDN) server that is used by users to get the encrypted videos. There is also the particular application a user is trying to use. This server is the one responsible for user authentication and is the main interface the user directly interacts with.

B. Microsoft PlayReady

PlayReady is a DRM created by Microsoft in 2008 [11]. It is mostly present in Windows 10+ in the browser Edge. PlayReady has three different security levels that can be enforced in the license [17]:

- SL150: this level is the weakest and should only be used for testing as there is no actual protection.
- SL2000: this is the basic level to be used for commercial use. Secrets are protected according to the Compliance and Robustness Rules [12].
- SL3000: this is the highest level to date. It relies on hardware protection to keep data secrets.

In addition to the CDN and LS, PlayReady offers four optional servers:

- Secure Stop Server [16]: handles the detection of when a user stops playback. It allows for a service to count how many users from the same account are watching a particular content and limit concurrent access.
- Secure Delete Server [15]: ensures the deletion of a particular license that was stored on a client.
- Metering Server [14]: measures user consumption (duration of a playback, episodes watched...)
- Domain Server [13]: manages user domains of the application. A domain is a group of users which are granted access to similar resources by sharing common decryption keys without the need of communicating with the license server.

C. Encrypted Media Extension

1) Components: The Encrypted Media Extension (EME) is a specification created by the W3C to allow the usage of DRM protection without any additional plugins [21]. It introduces several JavaScript functions that can be used by webpages to interact with a CDM installed on the user machine. A simplified graphical explanation of DRM components related to EME can be found in figure 1.

2) Workflow: We can describe a typical EME workflow as follows:

1) `requestMediaKeySystemAccess`: the script specifies a collection of Key Systems and configurations to try.
2) `createMediaKeys`: gives access to the specified CDM

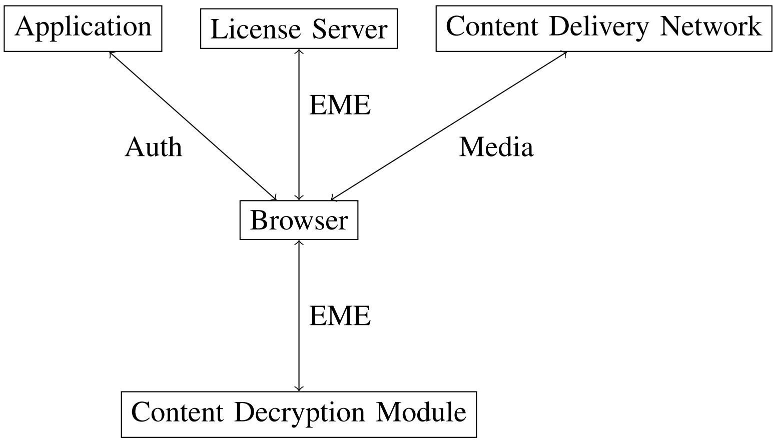

Figure 1. Summary of DRM Components

3) `createSession`: creates a particular instance in the CDM. This marks the end of the initialization phase, and we can now use licenses
4) `generateRequest`: creates the request to be sent to the LS. Initialization data has to be specified. It is usually given by inspecting the encrypted video we are trying to watch (via the `encrypted` event).
5) `message` event: this event is raised whenever the CDM needs to send a message to the LS. The browser then generates the HTTP request with the specified body.
6) `update`: this method has to be called with the response from the LS to the CDM.

We can then loop the last three steps to ask for multiple licenses.

A summary of the workflow can be found in figure 2

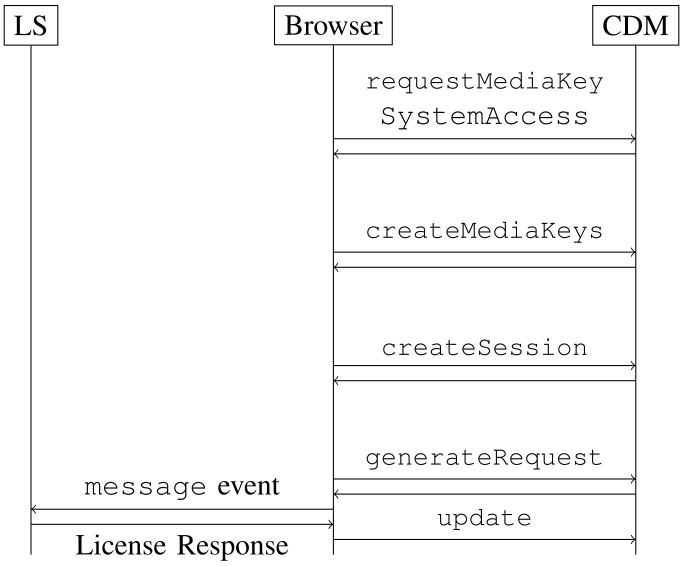

Figure 2. Typical EME workflow

3) Privacy and EME: The EME specification [21] mentions Distinctive Identifiers and Distinctive Permanent Identifiers. It is defined as follows:

- Distinctive Identifier: piece of data or observable behavior shared by only a small subset of individuals. This identifier is supposed to be clearable.
- Distinctive Permanent Identifier: A Distinctive Identifier which cannot be cleared, or is non-trivial to remove. Hardware-based identifiers are an example.

When requesting access to the Key System using `requestMediaKeySystemAccess`, you can specify in the configuration if Distinctive Identifiers can be used. It is up to the CDM to decide whether it will use some. A configuration will not be supported if it requires Distinctive Identifiers but the CDM refuses. These identifiers can be used to collect information about the client. As such, the EME specification [21] recommends limiting or completely avoiding the use of these identifiers. When used outside the client, the identifiers must be encrypted. In general, EME puts forward a per-origin policy, as well as a per-browsing profile policy (like a user session) for identifiers.

In privacy, when investigating whether some data could be used maliciously, we usually investigate two properties:

- *Uniqueness*: this indicates how many users share this value, and if it can be used to identify someone. Ideally (for fingerprinting), the value is unique per user and shared across origin but if only a handful shares it, it can still be used.
- *Stability*: this shows how often the value changes. The data is said to be stable when it rarely changes over time.

III. Contribution

A. EME Messages

In a typical EME workflow as explained in section II-C2, EME messages are sent from the CDM to the LS and vice-versa while passing through the browser in events. We can observe EME functions using a plugin like EME Logger[1]. It allows us to see EME calls, their parameters, their return values and triggered events. In combination with the PlayReady (partial) documentation [10] (the whole documentation is only available to PlayReady licensees), we can begin analyzing the opaque messages that are sent.

For the first tests, we used the Bitmovin Stream Test[2], in combination with the Microsoft PlayReady Test Server[3] as a license server.

1) Initialization: During the first three calls to the EME API, we can mostly see the configurations used by the platform in question, along the key used to identify the PlayReady Key System. Most notably, the session is temporary, that is, non-persistent and stored in-memory instead of on the hard drive. Distinctive identifier are not allowed. Audio and Video capabilities are also specified in the configuration as mentioned in the EME specification [21].

2) License Acquisition: When requesting a license using the `generateRequest` call, we have to supply initialization data, as well as the type of said initialization data (usually "cenc" as in Common Encryption Standard [7]) used by the CDM to determine what licenses it needs. These variables are often supplied by the `encrypted` event that is triggered when playback is initiated on a DRM-protected video. In the case of PlayReady, the initialization data is the header of the file, called a PlayReady Header. This header uses the Simple Object Access Protocol (SOAP) format, which is derived from XML, making the content human-readable. It contains the Key ID (KID) that uniquely identifies the content we are trying to decrypt, as well as other information like the License Server URL or the used encryption method.

Once the CDM has processed the request, it will trigger a `message` event. The content of the message should then be forwarded to the license server by the JavaScript with a POST request. The message also uses the SOAP format. It contains the aforementioned PlayReady Header, as well as other data such as a License Nonce made to counter-act replay attacks, a digital signature and, most importantly, a field called Encrypted Data. It has two subfields : EncryptedKey and CipherValue. This data contains the private client information that is used by the LS to identify the user and send a license using their encrypted ECC Public Key. This key is encrypted using a key called WMRMServer, shared with the License Server. In addition, the license request message is signed using ECDSA. We summarize the different fields and the encryption used in Figure 3 and Figure 4. An example license request is shown in Appendix A.

After the message is sent, we listen for the response of the LS. We can then feed the response directly to the

[1]Slightly modified version with hex dumps: https://github.com/Avalonswanderer/wideXtractor/tree/main/eme_logger_widextractor

[2]https://bitmovin.com/demos/stream-test

[3]https://testweb.playready.microsoft.com/

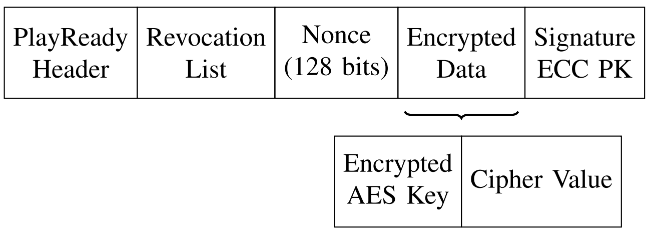

Figure 3. Fields present in a license request message

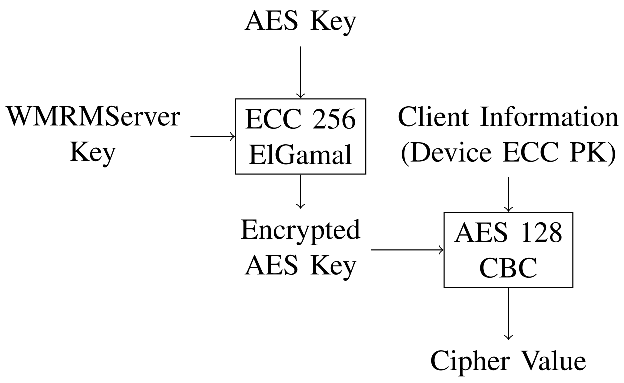

Figure 4. Cryptographic functions used in a license request

CDM with the `update` function. The license is now available and playback can be started.

B. PlayReady Test Server

1) Test Server: Microsoft has created a PlayReady test server that can be used by app developers (even non-licensees) to test how their product works by requesting licenses with custom rights. Some content is available on the platform, but you can also test your own content by specifying content keys and KID. Among all the policies you can ask for in a license, you can specify start and end dates for the license, the minimum security level and whether the license is persistent. The rights and restrictions requested are encoded in the URL query.

Most interestingly, the test server has an option called `clientinfo` which can be used to retrieve the private client information that were sent along the license acquisition request.

2) Client Info: This option is only available on the test server. It is supposed to act as a reflector for the client information that was sent. It is to be noted that the information is encrypted in the license request, making it sensitive information. An example of the response can be found in Appendix B. The main information is the Client time and CDM version, the supported features and the digital certificate chain.

We tried to investigate the different fields and whether they could be used for fingerprinting. As such, we studied two properties of the data: its uniqueness and its stability. More precisely, we focused on two values: *Digest* Value of certificate 0 (the client certificate) and the *Signature Public Key* of the client that was sent after generating the request. We tested on a few different machines using Windows 10 or Windows 11 and Microsoft Edge, as PlayReady was not available in other browsers and operating systems. To conduct the tests, we used a website that would print the client info and the signature public key[4]. This site doubled as a proof of concept of how the test server could be abused to collect information about a user: a malicious website could use JavaScript in the page and call the EME API with the PlayReady Test Server, without any visual effect (like a video) for the user.

We found that the public key as well as the digest value were unique per device and stable (for a few weeks of testing). It is quite interesting that the digest value was changing from client to client when the content of the certificate that was displayed was not different. This would suggest that there were other information that we were not reflected. Furthermore, other fields reflected could be used to track and collect data about a user like the Platform (usually `Windowsx86`) or the Model Number, hinting toward a specific Windows version. In addition, some devices support hardware protection. It can be activated using a Key System allowing security level 3000[5], the hardware security level. It shows a different platform related to the GPU (for example Nvidia or Intel) and another model number, further improving the quality of the fingerprint, while staying unique and stable.

On Microsoft Edge, DRMs are activated by default and no authorization is asked on a per-origin basis. This means that a script could run completely silently. On the other hand, the signature public key and the digest are unique per-origin, meaning a user cannot be tracked from one website to the other with this method as a new key and digest are generated for each origin. Moreover, Microsoft Edge provides an option to clear all data related to Media Foundation, meaning all data linked to the CDM, like persistent licenses, would be purged. It also has the effects of generating a new signature public key and digest. It also changes the values when using hardware protection, thus not giving any additional stability. Although the data can easily be cleared and changed, we had identical values when using a private session or the strict privacy mode from Edge. We did

[4]https://github.com/jules-timmerman/PlayReady-Reverse-Intern/tree/main/ClientInfoPoC

[5]`com.microsoft.playready.recommendation.3000`

have different values when using a different browsing session. It makes PlayReady comply with EME.

C. License Format

Licenses follow the Extensible Media Rights (XMR) system. It is used to encode the policies as well as the encrypted content key. While the specification of this format is only available to PlayReady licensees, we have a good idea of what information are stored in the license thanks to the PlayReady documentation [9, 8]. It explains the used encryption methods as well as a few diagrams with the content of a license.

We used the PlayReady Test Server to sample many licenses that we would then compare to deduce the semantic of the fields. We used a script to automate the sampling phase[6]. It would try all possible combinations of the arguments and store the request. It also does multiple requests of the same license to help with finding random values.

The XMR format seems like it follows a construction akin to Tag-Length-Value. For each field, we have a value on three bytes that identifies the type of data that follows. We then have a length indicator on four bytes representing the combined size of the tag, length and data (including the ending null byte) in bytes. It is followed by the data and terminated by a null byte. A summary of the fields we found are available in Appendix C. This analysis seems to match what had been done previously [3], even though the toolchain developed is not available. There also seems to be a system of blocks, with some tags encapsulating other ones.

There might be a better subdivision: for example, we might be able to split the tag or length in two, giving a separated semantic to each part. The previous analysis [3] seems to have done that separation but not additional meaning was given.

We observed three blocks of data that changed at every request, indicating a form of randomness. The first one was right after the XMR header of the license and represented the license Nonce. We then have a block of 1024 bits. It is the biggest encrypted block, so we believe it is where the Content Key is encrypted using ECC256 ElGamal [8], with some additional information. We know that the Content Key is 128 bit long from the PlayReady documentation [8] and the PlayReady Header, hence the additional data. At the end of the license, we have a 128 bits block. It is the license MAC, called the license signature by PlayReady, generated using AES OMAC1.

[6]https://github.com/jules-timmerman/PlayReady-Reverse-Intern/tree/main/LicenseComparison

XMR Header	Nonce (128 bits)	Policies	Encrypted CK (1024 bits)	ECC Device Key (512 bits)	MAC (128 bits)

Figure 5. Fields present in a license

Another block of interest is the one right after the Content Key. It seems to be a 512-bit key, unique per device and per origin. It is the same as the client info regarding its stability in different browsing contexts, as discussed in section III-B2. In the Security Exploration example [3], it was named *ECC Device Key*. It does not seem to be linked to any cryptographic function we were able to observe so far. A summary can be found in Figure 5

D. PlayReady on Windows

1) Samples: We wanted to investigate the PlayReady binary used by Microsoft Edge. To work on a smaller project than Microsoft Edge, we found two samples created by Microsoft to showcase the usage of PlayReady: one using C++[7] and one using C#[8]. Both applications used the Universal Windows Platform (UWP) to leverage the Windows API. They use Media Foundation, Windows' API for media playback.

2) Reversing the DLL library: The C++ application uses a DLL[9] located in `System32`, Windows default directory for system libraries. This dynamic library shares the same name as the UWP namespace of PlayReady. We tried to reverse it with two different approaches:

- *dynamically*, using the Visual Studio debugger
- *statically*, using Ghidra, a static reversing tool.

The binary was stripped, meaning all symbols had been removed, making reversing harder. Our goal when reversing the DRM was to find precisely how the licenses were parsed. By looking at the strings in the binary, we could find some strings that were used in the license response. We tried to set a breakpoint in the binary to functions that used said strings, but it was not hit. It helped us notice a gap between the static disassembly of the functions and how they were loaded in memory: it seemed that the functions were constructed after the DLL was loaded in memory. We dumped the DLL from memory after it was loaded to recover the correct functions.

[7]https://github.com/microsoft/media-foundation/tree/master/samples/MediaEngineEMEUWPSample

[8]https://github.com/microsoft/windows-universal-samples/tree/main/Samples/PlayReady

[9]Called `Windows.MediaProtection.PlayReady.dll`

We were not able to understand the parsing done in the binary in our limited time as it was too time-consuming a task.

IV. Related Works

This work is heavily related to the works that were investigating the inner-workings of Google's DRM, Widevine [19, 18]. Privacy issues and concerns are also raised in [5]. In that paper, they found identifiers that were shared across origins, allowing for effective fingerprinting.

V. Conclusion

In this paper, we tried to investigate if PlayReady complied with EME, like it was done for other DRM technologies. While we did not find any particular identifiers that could be used across origins, the test server can still be abused to query for some components of the user system, like their OS versions or their GPU model. We would need to scale the tests we have done to be able to have a more precise idea on how unique some values can be. In addition to the privacy issues, our understanding of the license format can help to create tools to aid the investigation of PlayReady, like plugins automatically parsing licenses. Lastly, we did find the PlayReady binary and dumped it from memory. While we were not able to do much with it, it is still the first step to reverse it.

Bibliography

References

[1] Apple. *FairPlay*. URL: https://developer.apple.com/streaming/fps/ (visited on 06/12/2023).

[2] EFF. *EFF's Formal Objection to the HTML WG Draft Charter*. URL: https://www.eff.org/pages/drm/w3c-formal-objection-html-wg (visited on 06/13/2023).

[3] Security Exploration. *Microsoft PlayReady*. URL: https://security-explorations.com/microsoft-playready.html (visited on 07/11/2023).

[4] Google. *Widevine*. URL: https://www.widevine.com/ (visited on 05/12/2023).

[5] Mohamed Sabt Gwendal Patat and Pierre-Alain Fouque. "Your DRM Can Watch You Too: Exploring the Privacy Implications of Browsers (mis)Implementations of Widevine EME". In: *In 23rd Privacy Enhancing Technologies Symposium, PETS 2023, Lausanne, Switzerland, July 10-15 2023*. 2023.

[6] Fortune Business Inside. *Video Streaming Market Size, Share and Growth Analysis*. URL: https://www.fortunebusinessinsights.com/video-streaming-market-103057 (visited on 05/12/2023).

[7] ISO/IEC. *ISO/IEC 23001-7:2016(en) Information technology — MPEG systems technologies — Part 7: Common encryption in ISO base media file format files*. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:23001:-7:ed-3:v1:en (visited on 07/10/2023).

[8] Microsoft. *Content Encryption*. URL: https://learn.microsoft.com/en-ca/playready/packaging/content-encryption (visited on 06/13/2023).

[9] Microsoft. *License Acqusition*. URL: https://learn.microsoft.com/en-ca/playready/overview/license-acquisition (visited on 06/13/2023).

[10] Microsoft. *Microsoft PlayReady Documentation*. URL: https://learn.microsoft.com/en-ca/playready/ (visited on 06/14/2023).

[11] Microsoft. *PlayReady*. URL: https://www.microsoft.com/playready/ (visited on 06/12/2023).

[12] Microsoft. *PlayReady Compliance and Robustness Rules*. URL: https://learn.microsoft.com/en-ca/playready/overview/compliance-and-robustness-rules (visited on 06/13/2023).

[13] Microsoft. *PlayReady Domain Server*. URL: https://learn.microsoft.com/en-ca/playready/overview/domain-server (visited on 06/13/2023).

[14] Microsoft. *PlayReady Metering Server*. URL: https://learn.microsoft.com/en-ca/playready/overview/metering-server (visited on 06/13/2023).

[15] Microsoft. *PlayReady Secure Delete Server*. URL: https://learn.microsoft.com/en-ca/playready/overview/secure-delete-server (visited on 06/13/2023).

[16] Microsoft. *PlayReady Secure Stop Server*. URL: https://learn.microsoft.com/en-ca/playready/overview/secure-stop-server (visited on 06/13/2023).

[17] Microsoft. *Security Level*. URL: https://learn.microsoft.com/en-ca/playready/overview/security-level (visited on 06/13/2023).

[18] Gwendal Patat, Mohamed Sabt, and Pierre-Alain Fouque. "Exploring Widevine for Fun and Profit". In: *43rd IEEE Security and Privacy, SP Workshops 2022, San Francisco, CA, USA, May 22-26*. IEEE, 2022, pp. 277–288. DOI: 10.1109/SPW54247.2022.9833867. URL: https://doi.org/10.1109/SPW54247.2022.9833867.

[19] Gwendal Patat, Mohamed Sabt, and Pierre-Alain Fouque. "WideLeak: How Over-the-Top Platforms

Fail in Android". In: *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2022, Baltimore, MD, USA, June 27-30*. IEEE, 2022, pp. 501–508. DOI: 10.1109/DSN53405.2022.00056. URL: https://doi.org/10.1109/DSN53405.2022.00056.

[20] *Reddit's website uses DRM for fingerprinting*. July 8, 2020. URL: https://iter.ca/post/reddit-whiteops/ (visited on 06/13/2023).

[21] W3C. *Encrypted Media Extension*. URL: https://www.w3.org/TR/encrypted-media/ (visited on 06/12/2023).

Appendix A
License Request

Listing 1. Sample License Request message

```
<PlayReadyKeyMessage type="LicenseAcquisition">
 <LicenseAcquisition Version="1">
  <Challenge encoding="base64encoded">
   <?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>
             <PROTECTINFO>
              <KEYLEN>16</KEYLEN>
              <ALGID>AESCTR</ALGID>
             </PROTECTINFO>
             <KID>4Rplb+TbNES8tGkNFWTEHA==</KID>
             <CHECKSUM>KLj3QzQP/NA=</CHECKSUM>
             <LA_URL>https://profficialsite.keydelivery.mediaservices.windows.
                net/PlayReady/</LA_URL>
             <CUSTOMATTRIBUTES>
              <IIS_DRM_VERSION>8.1.2304.31</IIS_DRM_VERSION>
             </CUSTOMATTRIBUTES>
            </DATA>
           </WRMHEADER>
          </ContentHeader>
          <CLIENTINFO>
           <CLIENTVERSION>10.0.16384.10011</CLIENTVERSION>
          </CLIENTINFO>
          <RevocationLists>
           <RevListInfo>
            <ListID>ioydTlK2p0WXkWklprR5Hw==</ListID>
            <Version>11</Version>
           </RevListInfo>
           [...]
          </RevocationLists>
          <LicenseNonce>MKAsWyCrYniFmdE5MFuVVg==</LicenseNonce>
    <ClientTime>1685106075</ClientTime>
    <EncryptedData
      xmlns="http://www.w3.org/2001/04/xmlenc#" Type="http://www.w3.org
          /2001/04/xmlenc#Element">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#
          aes128-cbc"></EncryptionMethod>
      <KeyInfo
        xmlns="http://www.w3.org/2000/09/xmldsig#">
        <EncryptedKey
          xmlns="http://www.w3.org/2001/04/xmlenc#">
          <EncryptionMethod Algorithm="http://schemas.microsoft.com/DRM
              /2007/03/protocols#ecc256"></EncryptionMethod>
          <KeyInfo
            xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>WMRMServer</KeyName>
          </KeyInfo>
          <CipherData>
            <CipherValue>base64encoded</CipherValue>
          </CipherData>
        </EncryptedKey>
      </KeyInfo>
      <CipherData>
        <CipherValue>base64encoded</CipherValue>
      </CipherData>
    </EncryptedData>
  </LA>
  <Signature
    xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo
      xmlns="http://www.w3.org/2000/09/xmldsig#">
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-
          xml-c14n-20010315"></CanonicalizationMethod>
      <SignatureMethod Algorithm="http://schemas.microsoft.com/DRM
          /2007/03/protocols#ecdsa-sha256"></SignatureMethod>
      <Reference URI="#SignedData">
        <DigestMethod Algorithm="http://schemas.microsoft.com/DRM/2007/03/
            protocols#sha256"></DigestMethod>
        <DigestValue>base64encoded</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>base64encoded</SignatureValue>
    <KeyInfo
      xmlns="http://www.w3.org/2000/09/xmldsig#">
      <KeyValue>
        <ECCKeyValue>
          <PublicKey>base64encoded</PublicKey>
        </ECCKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
 </Challenge>
</challenge>
```

```
          </AcquireLicense>
        </soap:Body>
      </soap:Envelope>
    </Challenge>
    <HttpHeaders>
      <HttpHeader>
        <name>Content-Type</name>
        <value>text/xml; charset=utf-8</value>
      </HttpHeader>
      <HttpHeader>
        <name>SOAPAction</name>
        <value>"http://schemas.microsoft.com/DRM/2007/03/protocols/AcquireLicense
            "</value>
      </HttpHeader>
    </HttpHeaders>
  </LicenseAcquisition>
</PlayReadyKeyMessage>
```

APPENDIX B
CLIENT INFORMATION

Listing 2. Client Information and Signature Key from a computer with an Nvidia 4070

```
Signature Public Key: x60jY9jVEIWph4Z3+Ow/7pwQRiDJuVZQSjiRJtmVqi+
    c0frm3QzF31lHgTFeBftZH4VzQLaa9TONew+E9w0wDg==

Client Info:

Client Version: 10.0.16384.10011
Client Time: 7/10/2023 5:35:46 PM

Supported Features:

RevocationLists
PlayReady3Features

Device Certificate Info (Cert 0):

Platform: OEM
Type: Device
SecurityLevel: 3000
RobustnessVersion: 16777216
ManufacturerName: Nvidia Corporation
ModelName: AD10X
ModelNumber: AD10X
DigestValue: dMGc6BzMMaVOWDaS03AADOzMrBffsS93dzWpB9g3fCE=
IssuerKey: fiNb/dkIy09gdm23dxS6nFHSrbvkDmLM0ZOCLZTndoekbPR+
    H5a2jvAGh36bqQkfbBg5x/2PdqPw/rTr5kuIUQ==

Certificate Chain:

→ Cert: 1
ManufacturerName: Nvidia Corporation
ModelName: AD10X
ModelNumber: AD10X
DigestValue: 15Ri+OkuHnaW321J23bp4/vSrxVVdb84zprsEGTsgAw=
Platform: OEM

→ Cert: 2
ManufacturerName: Nvidia Corporation
DigestValue: pFOP5LRTEEOsd/knDWcNnWU0jwoLzFRd9Z0byKwXmQA=

→ Cert: 3
ManufacturerName: Microsoft
ModelName: PlayReady SL3000 Device Port + Link CA
ModelNumber: 1.0.0.1
DigestValue: bk7YOJRioSgnzjpZgLasowaL96LFIBHDx6B0z+JoDPE=
```

APPENDIX C
LICENSE POLICIES

Tag	*Name*	*Description*
0x010012	Begin Date / End Date	Epoch time of when the license starts / expires
0x010030	First Expiration	License will expire after that amount of second and playback will not be able to start again
0x010034	Security Level	Minimum Security Level required by the license to allow playback
0x01000a	KID and CK	KID then the encrypted data with the Content Key, separated by 0x000100030080 (indicating 80 bytes long data)
0x00002a	ECC Device Key	Exact usage unknown (unique per client and origin)
0x01000b	Signature	License AES OMAC1 Signature