Formal Analysis of Widevine DRM/EME

Stéphanie Delaune, Joseph Lallemand, Gwendal Patat, Florian Roudot, Mohamed Sabt

3 April 2024

DRM Systems

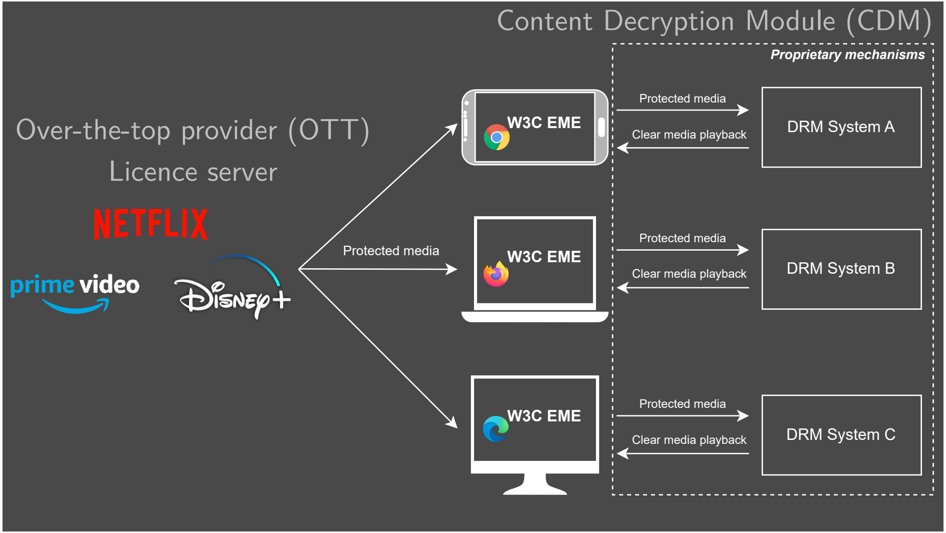
- Digital Rights Management:
  restrict uses of digital content – prevent copy, *etc.*
- Used for music, books, video games, video streaming. . .

Microsoft PlayReady

WIDEVINE

FairPlay

DRM Systems


Content Decryption Module (CDM)
Proprietary mechanisms
Over-the-top provider (OTT)
Licence server
NETFLIX
prime video
Disney+
Protected media
W3C EME
W3C EME
W3C EME
Protected media
Clear media playback
DRM System A
Protected media
Clear media playback
DRM System B
Protected media
Clear media playback
DRM System C

Encrypted Media Extension (EME)

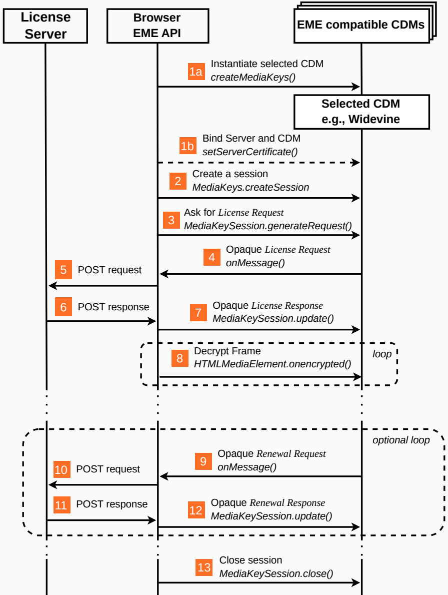
- EME: A standard defined by the World Wide Web Consortium (W3C)
- An API to make DRM use in browsers more uniform
- Integrated into all major browsers
- An “opaque” specification

Contribution

- Our goal: formally study the security of EME instantiated by Widevine
- Reverse engineer the proprietary Widevine protocol
- Define security properties (not present in EME spec)
- Model and prove in Tamarin

EME Specification

- Standard defines EME workflow and messages:
  - Initiate session
  - Initial Licence request/response
  - Licence Renewal request/response
  - Using a licence to decrypt
- Does not specify the content of messages or internal behaviour of the CDM
  → they are proprietary and implementation-specific



License Server

Browser EME API

EME compatible CDMs

1a Instantiate selected CDM *createMediaKeys()*

Selected CDM e.g., Widevine

1b Bind Server and CDM *setServerCertificate()*

2 Create a session *MediaKeys.createSession*

3 Ask for *License Request* *MediaKeySession.generateRequest()*

4 Opaque *License Request* *onMessage()*

5 POST request

6 POST response

7 Opaque *License Response* *MediaKeySession.update()*

8 Decrypt Frame *HTMLMediaElement.onencrypted()*

loop

optional loop

9 Opaque *Renewal Request* *onMessage()*

10 POST request

11 POST response

12 Opaque *Renewal Response* *MediaKeySession.update()*

13 Close session *MediaKeySession.close()*

Reverse engineering of Widevine EME messages – Initial Licence

EME Message	Widevine EME Message Content
Licence Request	$\langle \mathsf{reqID}, \mathsf{nonce}, \{\mathsf{clientID}\}_{\mathsf{privacyK}}, \{\mathsf{privacyK}\}_{\mathsf{serviceCert}}, \mathsf{keyID}, t_1 \rangle = req$ $+$ signature of *req* with deviceK
Licence Response	$\{\mathsf{sessionK}\}_{\mathsf{deviceK}}$, $\langle \mathsf{reqID}, t_1, \Delta t, \mathsf{keyID}, \{\mathsf{contentK}\}_{\mathsf{assetK}}, \{\mathsf{nonce}\}_{\mathsf{contentK}}, \mathsf{policy} \rangle = resp$ $+$ MAC of *resp* with macK_S

- Hierarchy of keys:

$$\mathsf{deviceK} \rightarrow \mathsf{sessionK} \rightarrow \mathsf{assetK}, \mathsf{macK}_S, \mathsf{macK}_C \rightarrow \mathsf{contentK}$$
$$\mathsf{assetK}, \mathsf{macK}_S, \mathsf{macK}_C = KDF(tags, req, \mathsf{sessionK})$$

- nonce to ensure freshness
- Timestamps and time-to-live to control licence expiration

Reverse engineering of Widevine EME messages – Licence Renewal

EME Message	Widevine EME Message Content
Renewal Request	$\langle \mathsf{reqID}, \{\mathsf{clientID}\}_{\mathsf{privacyK}'}, \{\mathsf{privacyK}'\}_{\mathsf{serviceCert}}, t_1, t_2, \mathsf{ctr}, \mathsf{nonce}' \rangle$ $+$ MAC with macK_C
Renewal Response	$\langle \mathsf{reqID}, t_1, t_2, \mathsf{ctr} + 1, \mathsf{policy}', \Delta t, \{\mathsf{nonce}'\}_{\mathsf{contentK}} \rangle$ $+$ MAC with macK_S

- Counter ctr to ensure correspondence between request/response
- Two slightly different versions:
  nonce′ in renewal is present on Android, but absent on desktop

Security properties for Widevine

- Specifications are not public, EME gives no security guarantees
  No standard security definitions for such DRM systems 😭

Security properties for Widevine

- Specifications are not public, EME gives no security guarantees
  No standard security definitions for such DRM systems 😭
  ⇒ We propose our own definitions for the security of Widevine 😎

- Attacker scenario:
  trusted CDM and OTT, untrusted network and API user (browser/client)

Security properties for Widevine

- Specifications are not public, EME gives no security guarantees
  No standard security definitions for such DRM systems 😭
  ⇒ We propose our own definitions for the security of Widevine 😎

- Attacker scenario:
  trusted CDM and OTT, untrusted network and API user (browser/client)

- We introduce seven security goals, split into three groups

Security properties for Widevine

- Specifications are not public, EME gives no security guarantees
  No standard security definitions for such DRM systems 😭
  ⇒ We propose our own definitions for the security of Widevine 😎

- Attacker scenario:
  trusted CDM and OTT, untrusted network and API user (browser/client)

- We introduce seven security goals, split into three groups

Security Goal 1: Key Confidentiality

Content decryption keys remain secret.

Security goals for initial licences

Security Goal 2: Integrity

The CDM must load initial licence responses *as they were generated* by the OTT.

Security Goal 3: Freshness

A given licence response can be loaded at most once,
and only by the CDM generating the corresponding request.

Security Goal 4: Expiration

In the initial phase, the CDM can use a decryption key at time t
only if the OTT granted a licence for it expiring at time $t_0 + \Delta t \geq t$.

Security goals for licence renewal

Security Goal 5: Integrity

The CDM must load renewal responses *as they were generated* by the OTT.

Security Goal 6: Freshness

A given renewal response can be loaded at most once, and only by the right CDM, and a CDM loads at most one response per *renewal event*.

Security Goal 7: Expiration

In the renewal phase, the CDM can use a decryption key at time t
only if the OTT granted a (renewed) licence for it expiring at time $t_0 + \Delta t \geq t$.

Formal analysis of Widevine/EME

- We analyse Widevine/EME using the Tamarin prover (both Android and desktop version)
- A fairly complex protocol to model:
  - two roles (CDM & Licence Server), unbounded sessions 👍
  - a stateful API 👍
  - with loops
  - and counters
  - and timers...

Modelling of time in Tamarin

- Widevine messages contain timestamps, and Goals 4 and 7 explicitly mention time
- No built-in support for time-based properties in Tamarin

Modelling of time in Tamarin

- Widevine messages contain timestamps, and Goals 4 and 7 explicitly mention time
- No built-in support for time-based properties in Tamarin
- We propose our own encoding of time
  - Time as an integer %t
  - Each protocol rule receives it In(%t), has event GTime(%t)
  - It can appear in protocol messages and lemmas:

$$\mathsf{GTime}(\%\mathsf{t}_1)@i \ \& \ \mathsf{State}(\%\mathsf{t}_2, \ldots)@i \Rightarrow \%\mathsf{t}_1 << \%\mathsf{t}_2 \ldots$$

  - Attacker chooses the time each rule is executed, a restriction forces consistent choices:

$$\#i < \#j \ \& \ \mathsf{GTime}(\%\mathsf{t}_1)@i \ \& \ \mathsf{GTime}(\%\mathsf{t}_2)@j \Rightarrow \%\mathsf{t}_1 << \%\mathsf{t}_2$$

Conclusion & future work

- We reverse-engineered the Widevine DRM protocol
- We propose definitions for the security of Widevine as an EME instance
- We formally analyse the protocol in Tamarin

Future work

- Privacy properties
- Model for dishonest Licence server
- Other DRM systems

Questions?