USENIX Security '24 Artifact Appendix: Formal Security Analysis of Widevine through the W3C EME Standard

Stéphanie Delaune
Univ Rennes, CNRS, IRISA, France

Joseph Lallemand
Univ Rennes, CNRS, IRISA, France

Gwendal Patat
Fraunhofer SIT | ATHENE, Germany

Florian Roudot
Univ Rennes, CNRS, IRISA, France

Mohamed Sabt
Univ Rennes, CNRS, IRISA, France

A Artifact Appendix

This artifact appendix describes the experiments and case studies conducted in the research paper *Formal Security Analysis of Widevine through the W3C EME Standard.*

A.1 Abstract

Streaming services such as Netflix, Amazon Prime Video, or Disney+ rely on the widespread EME standard to deliver their content to end users on all major web browsers. While providing an abstraction layer to the underlying DRM protocols of each device, the security of this API has never been formally studied. One of the core objectives of this research is to provide a formal security analysis of Widevine, the most deployed DRM instantiating EME.

Relying on TAMARIN, we study two variants of the Widevine DRM system. Our investigation highlights a vulnerability that could allow for unlimited media consumption. Additionally, we present a fix suitable for both mobile and desktop platforms, and formally prove it secure using TAMARIN.

We provide means to reproduce all the proofs and analyses we performed using the TAMARIN prover. We study both the Android version of Widevine (where the renewal request contains a so-called Key Control Block – KCB), and the desktop version (where it does not). For each version, we consider both the actual protocol and the fixed version which includes the mitigation we propose. Altogether, we thus present four protocol models as the artifact for our paper.

A.2 Description & Requirements

A.2.1 Security, Privacy, and Ethical Concerns

Executing the models provided as artifact does not present any security or ethical risks. The vulnerability we found has been responsibly disclosed to Widevine and the W3C, as discussed in the paper.

A.2.2 How to Access

Our artifact is available in a public GitHub repository, with detailed instructions to replicate the experiments discussed in the original paper. https://github.com/Avalonswanderer/eme_widevine_formal_verification/releases/tag/v1.1.

A.2.3 Hardware Dependencies

Running our artifact does not require any specific hardware. We used a fairly standard laptop (2.30GHz Intel Core i7-1068NG7 CPU, 16GB of RAM), and were able to verify all our models in a few minutes.

A.2.4 Software dependencies

Our artifact relies on the following dependencies:

- The TAMARIN Prover[1]. We used version 1.8.0, which is the latest release at the time of writing. TAMARIN itself depends on Haskell-stack[2], GraphViz[3], and Maude[4] (versions 2.7.1 to 3.3.1 are recommended by TAMARIN). Installation instructions for TAMARIN and its dependencies can be found at https://tamarin-prover.com/manual/master/book/002_installation.html. Note that TAMARIN runs natively on Linux or macOS, but not on Windows systems (WSL may be used there).

- Python3[5], which can be installed from most package managers or manually. We used version 3.12, but any relatively recent version should work as well.

[1]https://tamarin-prover.com/
[2]https://github.com/commercialhaskell/stack
[3]https://www.graphviz.org
[4]https://github.com/maude-lang/Maude
[5]https://www.python.org/downloads/

A.2.5 Benchmarks

None.

A.3 Setup

A.3.1 Installation

1. Install TAMARIN (and its dependencies), either with a package manager, by manually downloading the binaries, or by compiling it from sources. Instructions are provided at https://tamarin-prover.com/manual/master/book/002_installation.html.
2. Install Python3.
3. Retrieve the files from our repository.

A.3.2 Basic Test

To check that TAMARIN is properly installed, run

```
$ tamarin-prover test
```

You should see a diagnostic message, ensuring that

- Maude is correctly installed;
- GraphViz is correctly installed;
- unification works properly.

It should conclude with:

```
All tests successful.
The tamarin-prover should work as intended.

           :-) happy proving (-:
```

A.4 Evaluation Workflow

Our repository contains a `README` file and four subfolders: `WithKCB`, `WithoutKCB`, `FixWithKCB`, and `FixWithoutKCB`. These four folders correspond to the four models we developed, to analyze the Android version (`WithKCB`), the desktop version (`WithoutKCB`), and the mitigation we proposed for both versions (`FixWithKCB` and `FixWithoutKCB`).

For the security analysis, we consider 7 security goals that are explained in the paper. In a nutshell, they are:

Goal 1: Confidentiality of the decryption key.

Goal 2: Integrity and authenticity of initial licenses.

Goal 3: Freshness of initial licenses.

Goal 4: Enforcing expiration time of initial licenses.

Goal 5: Integrity and authenticity of renewal licenses.

Goal 6: Freshness of renewal licenses.

Goal 7: Enforcing expiration time of renewal licenses.

In addition, we prove an "executability lemma", as a form of sanity check. This lemma expresses the fact that the normal intended protocol flow can be executed. Its purpose is to ascertain that our models are correct.

We also consider an extra security goal, expressing that the license policies are respected, *i.e.* that the CDM can load a renewal license only if it is authorized by the policy. This goal cannot be stated at the EME level (the API studied in this paper), as policies are not part of EME. Nevertheless, we still include this extra goal, which we call **Goal 8**, for the specific case of Widevine.

A.4.1 Major Claims

Our findings are summarized in Table 1. Most of the proofs required some intermediate lemmas in order to conclude. These proofs have been obtained at first using the interactive mode of TAMARIN. However, for reproducibility purposes, we then automated them, relying on the oracle mechanism available in TAMARIN. The experiments below can therefore be run non-interactively from the command line.

	without fix		with fix	
KCB	with	without	with	without
Goal 1	✓	✓	✓	✓
Goal 2	✓	✓	✓	✓
Goal 3	✓	✓	✓	✓
Goal 4	✓	✓	✓	✓
Goal 5	✗	✗	✓	✓
Goal 6	✓	✓	✓	✓
Goal 7	✗	✗	✓	✓
Goal 8	✗	✗	✓	✓

Table 1: Summary of our results.

Goals marked with ✓ in the table are satisfied, and are proved automatically by TAMARIN in the respective files. Goals against which we found attacks are marked ✗ in the table (see the paper for a description of the attacks). In fact, we used TAMARIN to discover the attack on **Goal 5**, and stored the attack trace in each corresponding TAMARIN file for reproducibility. The goals in a shaded box in the table are broken as well as a consequence of the attack on **Goal 5**. We did not use TAMARIN to derive the same attack again for each of them, and thus the corresponding files do not contain any proof or attack for these goals.

A.4.2 Experiments

All our experiments were performed on a standard laptop, as mentioned earlier. The verification time varies depending on the model, and the goals we are trying to establish. The easiest lemmas are proved in a few seconds, whereas the most difficult ones require around 4 minutes.

Below, we explain the different types of experiments, as well as the results that are expected when running them.

(E1): Proving **Goal 1** in TAMARIN.
Preparation: Go to the folder containing the model for which you want to establish **Goal 1**.
Execution: Run the command given in the `README` file, and recalled below:

```
 tamarin-prover --prove
  --derivcheck-timeout=0
  --heuristic=O
  --oraclename='widevine.oracle'
  widevine.spthy -DSecrecy -DGoal1
```

Results: After execution, you should obtain a summary stating that `ContentKeySecrecy`, *i.e.* **Goal 1**, (and actually all required intermediate lemmas) are verified.

```
==================================================
summary of summaries:
analyzed: widevine.spthy
processing time: 9.16s

OTT2 (all-traces): verified (6 steps)
...
contentKeySecrecy (all-traces): verified (2 steps)
=====================================================
```

(E2): Proving **Goal 2**, **Goal 3**, and **Goal 4** in TAMARIN.
Preparation: Go to the folder containing the model for which you want to establish **Goals 2**, **3**, and **4**.
Execution: Run the command given in the `README` file, and recalled below:

```
tamarin-prover --prove
  --derivcheck-timeout=0
  --heuristic=O
  --oraclename='widevine.oracle'
  widevine.spthy -DSecrecy -DGoalInitialPart
```

Results: After execution, you should obtain a summary stating that all these goals, as well as the required intermediate lemmas, are verified. The correspondence between goals and lemmas is as follows:

- **Goal 2**: `OTTLicenseResponseBeforeLoad`
- **Goal 3**: `LoadRespUnique`
- **Goal 4**: `UseAuthorised`

```
===================================================
summary of summaries:
analyzed: widevine.spthy
processing time: 9.96s

OTT2 (all-traces): verified (6 steps)
...
UseAuthorised (all-traces): verified (4 steps)
=====================================================
```

(E3): Proving **Goal 6** (on all models), as well as **Goals 5**, **7**, and **8** on the fixed versions is similar:

```
tamarin-prover --prove
  --derivcheck-timeout=0
  --heuristic=O
  --oraclename='widevine.oracle'
  widevine.spthy -DSecrecy -DGoal6
tamarin-prover --prove
  --derivcheck-timeout=0 --heuristic=O
  --oraclename='widevine.oracle'
 widevine.spthy -DSecrecy -DGoal578
```

The correspondence between goals and lemmas is as follows:

- **Goal 5**: `OTTRefreshResponseBeforeLoadRefresh`
- **Goal 6**: `LoadRefreshRespUnique`
- **Goal 7**: `UseAuthorised`
- **Goal 8**: `LoadRefreshOnlyIfRenewable`

The runtime is slightly longer for these experiments. TAMARIN should still conclude within a few minutes.

(E4): For **Goal 5**, which does not hold in the models `WithoutKCB` and `WithKCB`, the attack trace has been stored in the corresponding file, and can be replayed[6]
Preparation: Go to the folder containing the model for which you want to establish that **Goal 5** fails.
Execution: Run the command given in the `README` file, and recalled below:

```
tamarin-prover
   --derivcheck-timeout=0
   --heuristic=O
    --oraclename='widevine.oracle'
    widevine.spthy -DSecrecy -DGoal578
```

Results: After execution, you should obtain a summary stating that **Goal 5**, which is to say lemma `OTTRefreshResponseBeforeLoadRefresh`, is falsified, and that an attack trace has been found. All the other lemmas are marked "analysis incomplete" as we did *not* ask TAMARIN to prove them (note that the `--prove` option is not part of the command).

```
==================================================
summary of summaries:
analyzed: widevine.spthy
processing time: 13.28s
...
OTTRefreshResponseBeforeLoadRefresh (all-traces):
    falsified - found trace (77 steps)
...
==================================================
```

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2024/.

[6]The proof scripts are not complete and contain some by sorry instructions. This is normal as there is an attack, and thus this goal can not be proved.