WV Modular DRM Version 10 Delta

Changes from Version 9 to 10

Table of Contents

References

DASH - 23001-7 ISO BMFF Common Encryption

DASH - 14496-12 ISO BMFF Amendment

W3C Encrypted Media Extensions (EME)

WV Modular DRM Security Integration Guide for Common Encryption (CENC) : Android Supplement

WV Modular DRM Security Integration Guide for Common Encryption (CENC)

Audience

This document is intended for SOC and OEM device manufacturers to upgrade an integration with Widevine content protection using Common Encryption (CENC) on consumer devices. In particular, if you already have a working OEMCrypto v9 library, and want to upgrade to OEMCrypto v10, then this document is for you. If you are starting from scratch, you should just read WV Modular DRM Security Integration Guide for Common Encryption (CENC).

Overview

There are several new features required for OEMCrypto version 10. The following sections discuss the main new features and give some idea why the new feature is being added. You should refer to WV Modular DRM Security Integration Guide for Common Encryption (CENC) for the full documentation of the API -- this document only discusses changes to the API.

API Version Number

```
uint32_t OEMCrypto_APIVersion();
```

This function should now return 10. If it returns 9, then CDM code will assume that OEMCrypto does not support the new v10 features. Depending on the platform, the library may be run in backwards compatibility mode, or it may be rejected. For example, on Android devices, the CDM code will not load liboemcrypto.so, and will fall back to the level 3 implementation.

Required Maximum Number of Sessions

Since more applications are pre-loading licenses, OEMs are encouraged to allow a larger number of concurrent sessions. For Android, the GTS test suite will require a minimum of 8 sessions -- up from 3. The recommended minimum number of sessions is 50.

Report Current and Maximum Number of Sessions

In addition to requiring more sessions, applications would like more information about the number of sessions to allow resource planning. The following two functions are new:

```
OEMCryptoResult OEMCrypto_GetMaxNumberOfSessions(size_t *count);

OEMCryptoResult OEMCrypto_GetNumberOfOpenSessions(size_t *max);
```

GetNumberOfOpenSessions should report the current number of open sessions. GetMaxNumberOfSessions should report the maximum number of sessions that OEMCrypto can reasonably open. If the maximum number of sessions depends on a dynamically allocated shared resource, the returned value should be a best estimate of the maximum number of sessions.

Report and Enforce Anti-Rollback Capability

Ideally, the Usage Table is saved in a secure file system that has hardware protection preventing the user from reading or writing to it. This can be simulated on top of an insecure file system by encrypting the usage table. However, this does not prevent the user from saving a valid usage table file and copying it back in place later -- rolling back to a previous version, perhaps re-activating entries. If a device has a small amount of secure nonvolatile storage, the generation number can be saved in there. This allows OEMCrypto to detect rollback by comparing this generation number with the one that is saved in the usage table file. If a device does not have any nonvolatile secure memory, it can still support usage tables.

The presence or absence of hardware anti-rollback capability is an important distinction for some content provider operators. For example, a content provider's policy may only allow Ultra High Definition content to be played when anti-rollback hardware is present. Otherwise, the license server will serve license for Standard Definition content only. However, there is no protocol-compatible secure way to prevent anti-rollback query information from being changed in the initial license request. Therefore, the license request will be sent to the server with

insecure information. The client will need to enforce the license server's requirement in the trusted environment when processing the license response in LoadKeys.

The first part of this new API is that the device must report its capability:

bool OEMCrypto_IsAntiRollbackHwPresent(void)

Indicates whether oemcrypto uses anti-rollback hardware to prevent the rollback of the Session Usage Table, such as storing the generation number in a separate secure location.

The second part of the new API is that the device must honor the new *Require_AntiRollback_Hardware* bit in the key control block. If that bit is set, and the device does not support hardware anti-rollback, then it must reject the license.

If the device reports IsAntiRollbackHwPresent is false, then the server will not set the *Require_AntiRollback_Hardware* bit. However, if there is a man-in-the-middle attack, which modifies IsAntiRollbackHwPresent to be true, then the device will detect the attack because the *Require_AntiRollback_Hardware* bit will be set, so the license will be rejected.

Concurrent Clear Content Copy

Some video content has a clear leader -- this is several seconds of unencrypted content that can be played while the license is being requested and the license response is being processed. Previously, DecryptCTR had a flag, is_encrypted, that allowed the application to tell OEMCrypto to copy unencrypted data to the secure buffer. OEMCrypto needs to do the copy because the non-secure CPU does not have read or write access to the secure buffer. The problem with the current implementation is that this cannot be synchronous with other session functions. In particular, GenerateDerivedKeys can take hundreds of milliseconds, which can cause the playback to stutter when DecryptCTR is blocked on a mutex. To solve this problem, a new function is being introduced which copies unencrypted content to a secure buffer and does ***not*** block. In particular, it can be called simultaneously with GenerateDerivedKeys, LoadKeys, or other key processing functions for the same session. Also, it can be called before LoadKeys.

```
OEMCryptoResult OEMCrypto_CopyBuffer(const uint8_t *data_addr,
                                     size_t data_length,
                                     const OEMCrypto_DestBufferDesc* out_buffer,
                                     uint8_t subsample_flags);
```

OEMCrypto will verify that the pointers are not null, and will copy the data from data_addr to the buffer specified by out_buffer. The parameter subsample_flags has the same meaning as in DecryptCTR.

Report Key Control Block For Debugging

For debugging license interactions and timeline, application developers would like to view the key control block as it was loaded onto the device. The developer of the OEMCrypto library must be careful that the keys themselves are not accidentally revealed.

```
OEMCryptoResult OEMCrypto_QueryKeyControl(const OEMCrypto_SESSION session,
                                          const uint8_t* key_id,
                                          size_t key_id_length,
                                          uint8_t* key_control_block,
                                          size_t* key_control_block_length);
```

OEMCrypto will perform the usual verification that pointers are not null and that key_control_block_length is 16 bytes. If the buffer is short, the usual OEMCrypto_ERROR_SHORT_BUFFER is returned. It only returns a key control block if LoadKeys was successful, otherwise it returns OEMCrypto_ERROR_NO_CONTENT_KEY. Note: it returns the control block in original, network byte order. If OEMCrypto converts fields to host byte order internally for storage, it should convert them back. Since OEMCrypto might not store the nonce or validation fields, values of 0 may be used instead.

Load Test Keybox or RSA Key

Most of the OEMCrypto test cases require the device have a standard test keybox installed. Previously, installing such a keybox on a production device might overwrite the production keybox, rendering the device unusable when communicating with production license servers. OEMCrypto V10 requires a standardized test keybox to be available in the trusted environment. The test keybox is the same for all devices, so it can be built into the trusted environment executable image, it does not need to be factory provisioned on each device.

By default, the unique factory provisioned production keybox is used. The test keybox will be activated by the new function OEMCrypto_LoadTestKeybox.

```
OEMCryptoResult OEMCrypto_LoadTestKeybox()
```

After this function is called, OEMCrypto should use the test keybox for all operations until OEMCrypto_Terminate is called. The next time OEMCrypto_Initialize is called, the device should revert to using its unique production keybox.

Similarly, on platforms which do not use keyboxes, there is a new function

```
OEMCryptoResult OEMCrypto_LoadTestRSAKey()
```

After this function is called, OEMCrypto will use the test RSA key for all operations and sessions until OEMCrypto_Terminate is called. The next time OEMCrypto_Initialize is called,

the device should revert to its production RSA certificate.

Delete Usage Entry Without Signature

```
OEMCryptoResult OEMCrypto_ForceDeleteUsageEntry(const uint8_t* pst,
                                                size_t pst_length)
```

This function will delete an entry from the Usage Table. This will be used for stale entries without a signed request from the server. As part of this call, OEMCrypto will increment the Usage Table's generation number, and then sign, encrypt, and save the Usage Table. Devices that do not implement a Session Usage Table should return OEMCrypto_ERROR_NOT_IMPLEMENTED.

The Type HDCP_Capability is Now an Enumeration

OEMCrypto_HDCP_Capability is now an enumeration explicitly defining the values. The meaning of the values has not changed. However, this slightly changes the signature of OEMCrypto_GetHDCPCpability.

```
typedef enum OEMCrypto_HDCP_Capability {
  HDCP_NONE  = 0,  // No HDCP supported, no secure data path.
  HDCP_V1    = 1,  // HDCP version 1.0
  HDCP_V2    = 2,  // HDCP version 2.0
  HDCP_V2_1  = 3,  // HDCP version 2.1
  HDCP_V2_2  = 4,  // HDCP version 2.2
  HDCP_NO_DIGITAL_OUTPUT = 0xff  // No digital output.
} OEMCrypto_HDCP_Capability;

OEMCryptoResult OEMCrypto_GetHDCPCapability(HDCP_Capability *current,
                                            HDCP_Capability *maximum);
```

Usage Table Updates

Some APIs introduced in OEMCrypto v9 are ambiguous about whether the usage table is saved when they are called. It seems reasonable that should since they should since otherwise those methods would always be called in conjunction with UpdateUsageTable.

In particular, the usage table should be updated and saved to the file system as part of each of these functions:

- LoadKeys - if the usage table was modified, it should be saved.
- UpdateUsageTable
- DeactivateUsageEntry
- ReportUsage

- DeleteUsageEntry
- ForceDeleteUsageEntry
- DeleteUsageTable - either the file should be deleted from persistent storage, or a blank table should be written to persistent storage.

Unit Tests Modifications

The suite of unit tests were previously delivered with many tests disabled. Developers were expected to install a test keybox or allow a test keybox to be installed by the tests. Then they had to explicitly run the disabled tests. With version 10, all devices must have a test keybox or test certificate available in addition to the factory provisioned keybox or certificate. So now relevant unit tests will all be run and not disabled.

In addition to tests for new features, the following existing features will be explicitly tested:

1. When OEMCrypto_OpenSession fails due to too many sessions, it should return OEMCrypto_ERROR_TOO_MANY_SESSIONS, rather than some other error code.
2. Required maximum number of sessions. For many platforms, such as Android, devices should support at least 8 sessions.

Also, because some features are optional on some platforms, the main function in oemcrypto_test.cpp has been modified so that it skips unit tests for features that are not supported. For example, the generic crypto tests will be skipped if OEMCrypto_Generic_Encrypt returns OEMCrypto_ERROR_NOT_IMPLEMENTED.

On some platforms, such as Android, there is a strict list of features that must be supported in order to be certified. Please see the supplement to this document for your platform if there are any doubts.

The unit tests in oemcrypto_test.cpp are designed so that these features are not tested if they are not implemented. In general, if a feature is not implemented, then the OEMCrypto library should return OEMCrypto_ERROR_NOT_IMPLEMENTED for those functions.

- Keybox functionality. If OEMCrypto_GetKeyData is not implemented, then the device will not use a keybox to generate license requests, or to request an RSA certificate. These devices will need to have an RSA certificate installed separately.
- Certificate functionality. If OEMCrypto_GenerateRSASignature is not implemented, then it will not use an RSA certificate to generate license requests. Many content providers prefer to use RSA certificates to generate license requests because it allows them to use a stand-alone server instead of relaying requests to a Widevine server. All devices must either have a keybox or support RSA certificates. Most platforms will support both.
- Load Certificate functionality. If a device does have a keybox, but does not implement OEMCrypto_RewrapDeviceRSAKey, it will not be able to request an RSA certificate

from the Widevine provisioning server. This essentially makes it unable to use RSA certificates.

- Generic Crypto. If Generic_Encrypt is not implemented, then the generic cryptographic API is not tested. Some applications use modular DRM functionality and root of trust to send secure data, such as business data or account data, from the application to the server. These functions are not used to play DRM protected video or audio.
- Usage Tables. Usage tables are a way to store usage information and offline licenses. If a device does not support usage tables, it will have limited capabilities to store offline licenses and will not be able to process secure stops.

Key Control Block Changes

The functionality described above requires the following changes to the key control block. The key control block should still be verified in the LoadKeys function. To allow for backwards compatibility, OEMCrypto should accept a key control block for previous versions of the API.

The four verification bytes in the key control block are valid if they are “kctl”, “kc09” or “kc10”. An OEMCrypto that implements the new v10 functions described in this document should accept any of these three strings.

Bit 28 of the Control Bits will now be the *Require_AntiRollback_Hardware* bit.

bit 31..29	***bit 28***	***bit 27..15***	***bit 14..0***
Previously defined	Require_AntiRollback_ Hardware *0 = not require* *1 = require*	Reserved *set to 0*	Previously defined