CloudTV Nano SDK Application Note

Contents

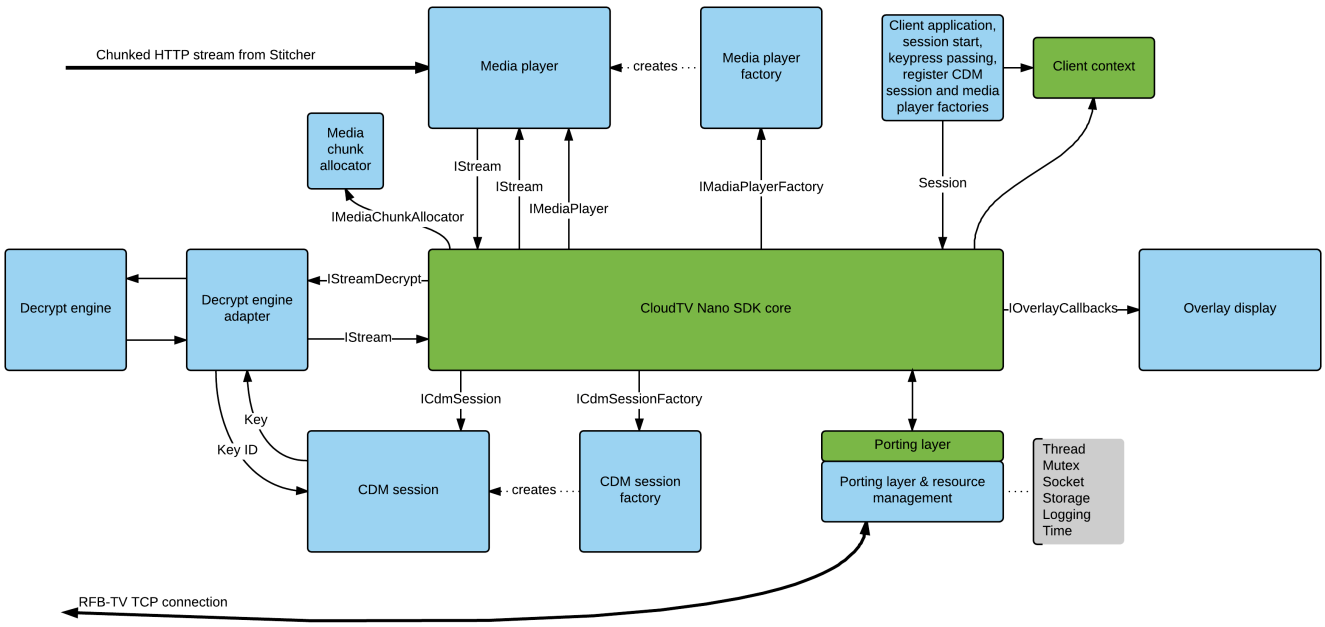
- 1. Introduction
- 2. System overview
- 3. Description of the blocks
- 4. Sequence diagrams
    - 4.1. Session and stream setup
    - 4.2. Protocol extension
- 5. Referenced documents

1. Introduction

This technical application note gives guidance on how the CloudTV Nano C++ SDK *could* be integrated with the STB middleware for a use case that supports encrypted content. This document must be used in combination with the Porting Guide for CloudTV Nano C++ SDK.

2. System overview

The block diagram below depicts the components that are connected to the CloudTV Nano SDK code. The blocks in green are provided by ActiveVideo whereas the blocks in blue are to be created by the integrator. The integrator does not need to provide all blocks if the application doesn't require this. For instance, the decrypt engine and CDM session blocks only need to be supplied in case an encrypted transport stream may be sent, which needs to be decrypted using the CloudTV Nano SDK. Also, CloudTV Nano SDK provides basic HTTP stream and content loaders that don't need to be supplied by the integrator if there is no specific reason to.



The **Media player** loads the chunked HTTP stream from the CloudTV Platform Stitcher, decode it and present the decoded audio and video data. The chunked HTTP potentially contains mixed encrypted and clear transport stream data and uses a proprietary mechanism to signal which parts are encrypted and which parts are in clear (i.e. not encrypted). The **Media player** must implement the `IMediaPlayer` interface. The **Media player** is typically supplied by the integrator but the CloudTV Nano SDK code base also contains an `SimpleMediaPlayer` class that implements the same interface. The **Media player** may or may not use the built-in `HttpLoader` class of CloudTV Nano SDK. The `SimpleMediaPlayer` class also makes use of a stream player that implements the `IStreamPlayer` interface, but contrary to previous versions of the SDK, this interface does not have to be used any more in custom implementations of `IMediaPlayer`.

CloudTV Nano SDK receives the data supplied by the **Media player** over the IStream interface and will separate the encrypted content (if any)

from the clear content. The encrypted content will be forwarded to the **Decrypt engine adapter**, which will in turn forward it to the **Decrypt engine**. The decrypted transport stream data will be sent back to **CloudTV Nano SDK**, which in turn passes it back to the **Media player** using another IStream interface.

The **Media player factory** is the component that is called by **CloudTV Nano SDK** to create a **Media player** object. A **Media player factory** must be registered for each of the supported stream protocols; there are no default loaders registered for UDP or HTTP.

The **Decrypt engine adapter** gets an encrypted stream from **CloudTV Nano SDK** and forwards it to the **Decrypt engine**. Part of the decryption meta-data, such as actual key identifiers and initialisation vectors (if applicable) are passed by the **CloudTV Nano SDK** as well, over the `IStrea mDecrypt` interface. Preparation of the decryption process, involving the transfer of other metadata such as licenses, is done during the CDM session setup phase.

The **Decrypt engine** takes care of the actual decryption. This is typically executed using a specific piece of hardware but may as well be implemented by software..

The **CDM session object** handles the decryption context of a DRM session. The **CDM session factory** is responsible for creating and destroying these objects and must be registered with the **CloudTV Nano SDK**.

The **Media chunk allocator** is a simple memory allocator interface that manages fixed-sized blocks of media memory. It is used by the **CloudTV Nano SDK** to allocate and free memory that is used to store the media stream when needed. **CloudTV Nano SDK** will take care of all further stream handling, including deep buffering and buffer management. It will make sure that the **Media player** gets a valid (and fully compliant) transport stream that can be played back with low latency.

The **Overlay display** component handles the drawing of overlay images. It implements `IOverlayCallbacks`.

The **Client application** handles the setup of **CloudTV Nano SDK**, the registration of the **Media player factory** (as HTTP media player), the **Decr ypt engine adapter** (as decryption engine) the **CDM session factory** (as proprietary CDM session handler), the **Overlay display** (as overlay engine) and passes the end user key presses to **CloudTV Nano SDK**. It also takes care of any other client-specific setup or resource management. It controls **CloudTV Nano SDK** by means of the Session class. It also sets up the **Client context** object that is needed by **CloudT V Nano SDK** to retrieve box-specific information such as serial numbers and storage locations.

The **Porting layer** connects **CloudTV Nano SDK** to the underlying operating system and platform services. Amongst others, **CloudTV Nano SDK** uses it to open a TCP connection to the CloudTV platform for the RFB-TV protocol.

3. Description of the blocks

The following table lists the blocks and their responsibilities:

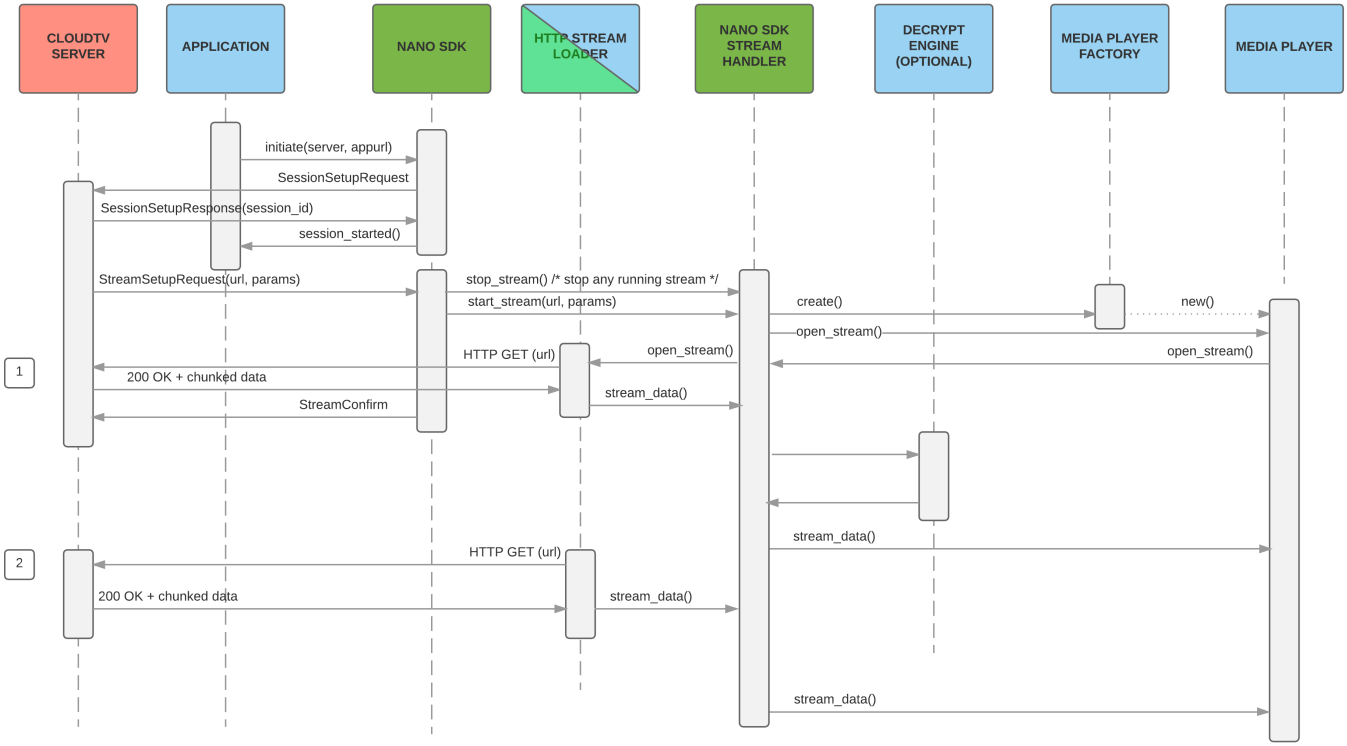
Block	Creator	Responsibilities	Implements	Remarks
CloudTV Nano SDK	ActiveVideo	Session management. Handling the RFB-TV protocol. Calling the **Media player factory** object to create the **Media player**. Controlling the **Media player** and handling the stream it supplies (buffer management and such) and media player events. Splitting the mixed encrypted/clear stream delivered by the **Media player**. Passing the encrypted segments (along with synchronous metadata) to the **De crypt engine adapter**. Calling the **CDM session factory** to create a **CDM session** object and calling this to manage the CDM session. Passing RFB-TV overlay data to the **Overlay display**. Passing the (processed) transport stream back to the **Media player**. Filtering the RFB-TV protocol extension data and passing it to the **Decrypt engine adapter**. Handling key presses issued by the **Client application**.	-	This component can be used as-is. No specific adaptations are needed. This is modeled by the *Session* class.
Decrypt engine	Integrator	Performing decryption of the encrypted transport stream.	-	Optional. Does not directly interface with any ActiveVideo component.
Decrypt engine adapter	Integrator	Control the Decrypt engine. Handle DRM metadata passed to/by **CloudTV Nano SDK** (impementing `ICdm Session`). Pass the encrypted stream obtained from **CloudTV Nano SDK** to the **Decrypt engine**, along with the appropriate metadata (implementing `IStreamDecrypt`). Pass the decrypted stream back to **CloudTV Nano SDK** (using `IStream`).	`ICdmSessionFactory` (pas sed to **CloudTV Nano SDK**). `IStreamDecrypt` (passed to **CloudTV Nano SDK**).	Optional. Handles all DRM specifics.
Media player	Integrator	Load the chunked HTTP data from the CloudTV Platform Stitcher using the URL passed by **CloudTV Nano SDK**. Pass this stream unmodified to **CloudTV Nano SDK** (using `IStream`). Receive the processed stream back from **CloudTV Nano SDK** (using `IStrea m`). Decode the stream and present the decoded stream to the user. Send player events (such as started, stopped and underrun) to the **CloudTV Nano SDK** (using `IMediaPlayer::ICallback`).	-	A similar component also exists in the CloudTV Nano SDK code base. This can be used as an example.

Media player factory	Integrator	Create an instance of **Media player** on demand.	`IMediaPlayerFactory` (passed to **CloudTV Nano SDK**).	This is an extremely simple class.
Client application	Integrator	Set up an appropriate **Client context**. Instantiate or connect to an **Overlay display** and **Stream player**. Instantiate an **Media player factory**. Instantiate a **CDM session factory**. Instantiate or connect to a **Decrypt engine adapter**. Instantiate a **Media chunk allocator**. Instantiate **CloudTV Nano SDK** (the `Session` object). Optionally create a state-change callback handler (`ISessionCallbacks` interface) and pass it to the `Session` object on creation. Register **Client context**, **Overlay display**, **Media player factory**, **CDM session factory**, **Decrypt engine adapter** and **Media chunk allocator** with **CloudTV Nano SDK**. Start and stop a session (using the `IControl` interface). Pass end user key presses to **CloudTV Nano SDK** (using the `IInput` interface). Perform any other client-specific actions.	-	This can be based on the example client implementation.
Client context	ActiveVideo	Contains implementation-specific and box-specific data such as serial numbers and storage locations.	-	This is modeled by the *ClientContext* class.
Overlay display	Integrator	Display overlay images (implementing `IOverlayCallbacks`).	`IOverlayCallbacks` (passed to **CloudTV Nano SDK**).	
Media chunk allocator	Integrator	Allocating and freeing of chunks of media memory (implementing `IMediaChunkAllocator`).	`IMediaChunkAllocator` (passed to **CloudTV Nano SDK**).	This can be a simple class and is optional.
Porting layer	ActiveVideo & Integrator	Abstract platform services and operating system functions.	`Condition`, `Thread`, `Mutex`, `Socket`, `DataStore`, `Log`, `Time`.	ActiveVideo provides an implementation for POSIX. The integrator can modify this at will.

4. Sequence diagrams

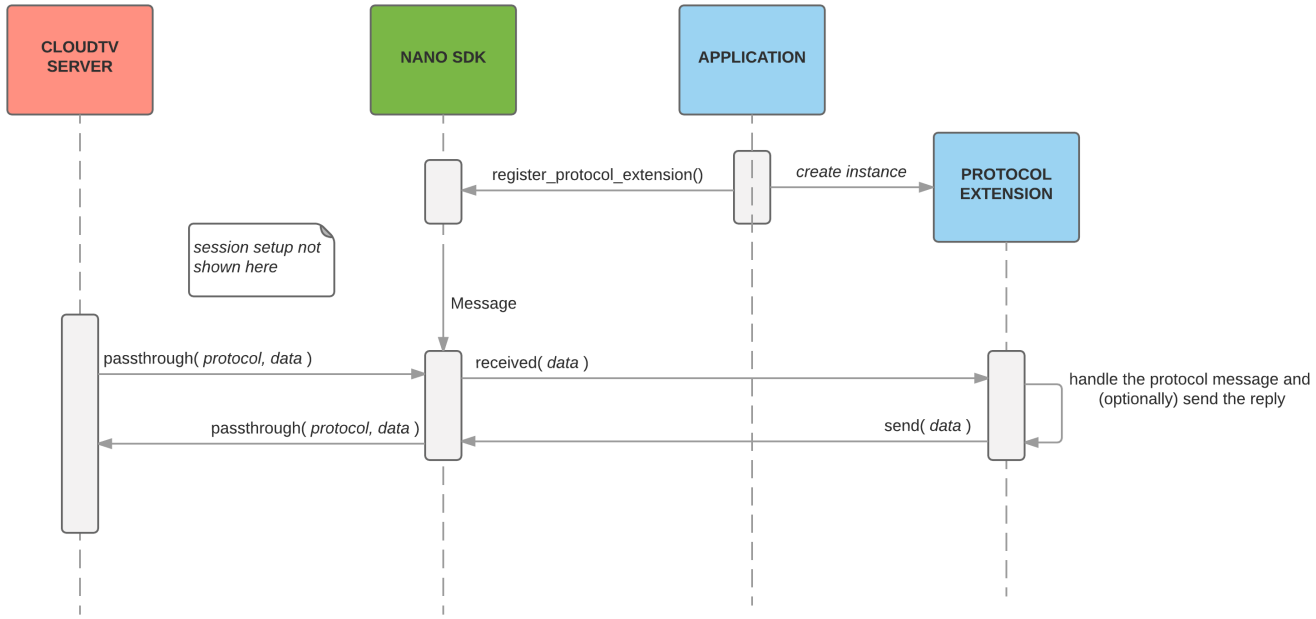
4.1. Session and stream setup

The following diagram depicts the typical sequence of events for setting up a stream. The initialization steps are not shown (they're described in the porting guide). Number 1 shows the sequence for encrypted data and number 2 shows the sequence for clear data. All buffer management is done inside the 'CloudTV Nano SDK' block. The other components must do close to no buffering for optimal user experience.



4.2. Protocol extension

The RFB-TV protocol supports so called 'passthrough' messages that can carry opaque data. CloudTV Nano SDK has abstracted these messages by providing, what we have adopted as, 'protocol extensions'. The CloudTV Platform application and STB client implementation have to agree on the protocol name and the syntax of the data for that protocol. Below is an example of a subclass of 'ProtocolExtensionBase', which implements 'IProtocolExtension', that handles a 'passthrough' message from the CloudTV server.



5. Referenced documents

- Porting Guide for CloudTV Nano C++ SDK