User Guide

DAS on STB Reference application

Version 1.0.1

Status: Reviewed

Filename	DAS_on_STB.docx
Document Nr	~~01ABCJ0000000262~~
Date	April 20, 2020
Author(s)	Patet Nitin
Information Domain	
Client/Project	
Owner	CAS-PU > Embedded Security

CH-1033 Cheseaux, Switzerland.
Tel: +41 21 7320311 Fax: +41 21 7320100
www.nagra.com

Contents

1. Introduction

1.1 Purpose

Purpose of this document is to explain the way DAS on STB reference application is designed and how it functions. This would help integrators understand how the communication, between different entities involved, works.

1.2 Document History

Change logs
1.0.1 / 2020-04-20 / Nitin Patet ▪ Minor corrections.
1.0.0 / 2018-10-17 / Nitin Patet ▪ Updated screenshots of the sample app. (section 5.1)
0.0.2 / 2018-10-16 / Nitin Patet ▪ New Modifications
0.0.1 / 2018-10-15 / Nitin Patet ▪ First issue

1.3 Document Reviewers

Name
Sharma Gaurav

1.4 Used Acronyms

Abbreviation	Definition
CAS	**C**onditional **A**ccess **S**ystem
CCL	**C**onnect **Cl**ient
DVB	**D**igital **v**ideo **b**roadcasting

1.5 References

[1] https://documentation.anycast.nagra.com

[2] DAC API – Reference Guide

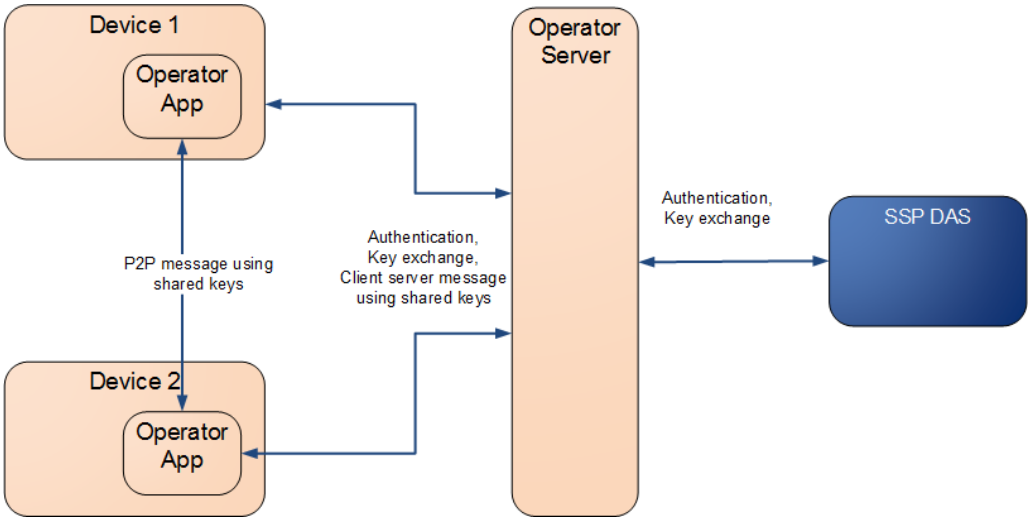
2. Overview

2.1 DAS – Device Authentication Service

SSP, Security Services Platform, provides a service called Device Authentication Service (DAS) and related client SDKs for Open Devices as well as STBs.

This service enables operators to:

- authenticate a device from the operator backend. This authentication is based whenever possible on platform level secure elements like hardware root of trust or TEE shared or not with the DRM platform.
- secure the delivery of symmetric keys provided by the operator and intended to be used by the application running on the device. These keys can then be used by applications to encrypt/decrypt or sign/validate data sent or received to/from either a server in a client-server protocol or other devices in a peer-to-peer protocol.



2.2 DAS on STB

The client-side SDK of the DAS provides a secure authentication service based on the DRM/CA of the device. For DAS on STB, Nagra Connect Client (CCL) provided APIs, called "Link protection session manager", make use of session keys delivered in specific persisted licenses, for enabling DAS use cases. Hence, those session keys can be used by applications to encrypt/decrypt and authenticate/verify any data.

The algorithms currently supported are the following:

- AES-128 CBC chaining mode with PKCS7 padding for the encryption/decryption
- AES-128 CMAC for authentication/verification

For STB using NOCS compliant chipset, the session keys are protected by the Nagra proprietary hardware key ladder.

3. Entities involved

3.1 DAS Server

Device Authentication Service (DAS) is a service of SSP (Security Services Platform) that offers an authentication mechanism to the backend which can be leveraged to build a trusted exchange of information. More details on this server covered in section [2.1].

The DAS server generates the credentials that contains the session key generated by the API Gateway. These credentials can only be decrypted by the Connect Client on the target STB.

3.2 API Gateway

API Gateway is the cloud system acting as the first point of contact for the application. Only the API Gateway directly interacts with the DAS Server and hence, acts like the first line of defense.

API Gateway is one of the key elements of security in this ecosystem and is responsible for generating session tokens, session keys etc. which are used by the Operator Application to enable different use cases (sign/verify, encrypt/decrypt).

3.3 Application

Operator application that triggers the use cases of Device Authentication, Encryption/Decryption and Sign/Verify, communicating with the API Gateway to fetch the necessary keys.

3.4 DAC – DAS Client

The Device Authentication Client is minimal set of APIs (on top of Connect Client APIs) to the operator application, needed for enabling the following features:

- Device Authentication
- Encryption/Decryption
- Sign/Verify

As the CCL being used underneath requires provisioning, DAC layer also supports communicating with the provisioning server to directly fetch the data.

For more details, refer to DAC API Reference Guide [2].

3.5 CCL – Connect Client

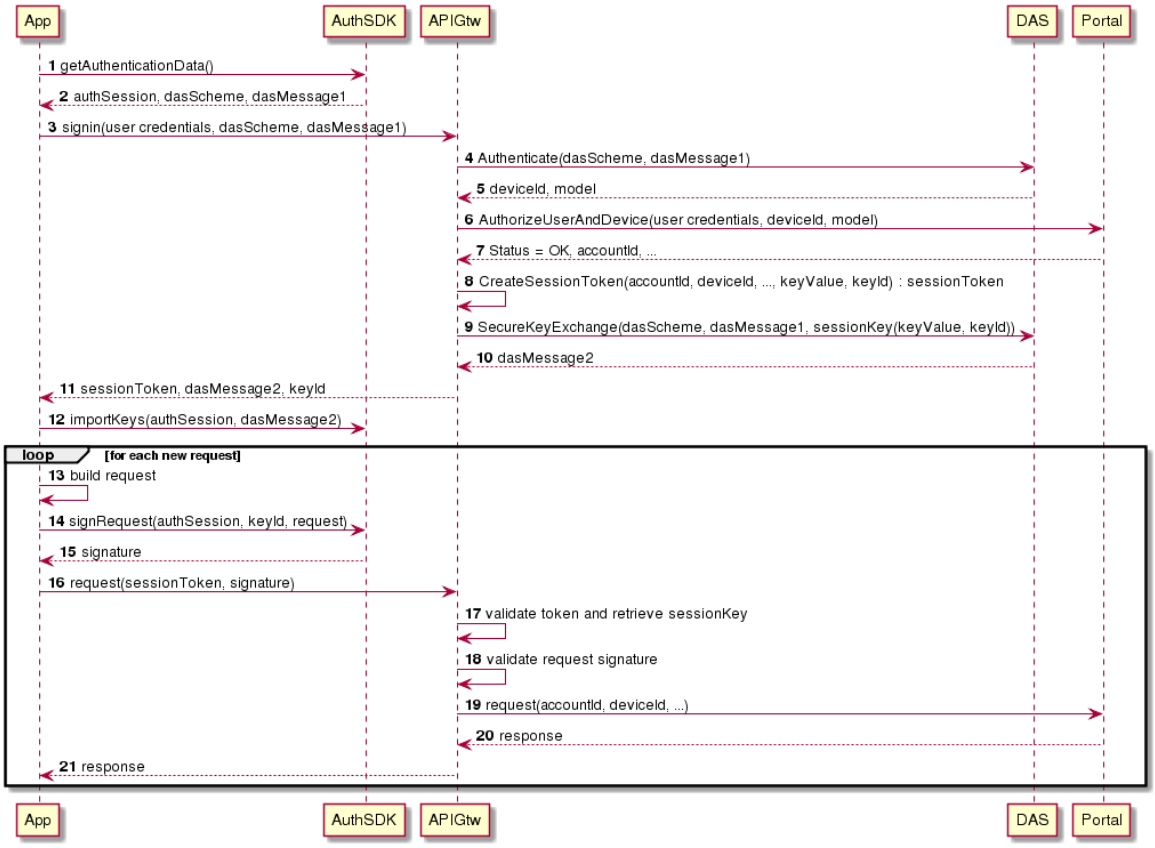
The Connect Client is the common, converged CAS/DRM client of Kudelski Group. It is part of the NAGRA anyCAST Connect solution, driven by the NAGRA anyCAST Security Services Platform (Head End). On the device side,

The algorithms currently supported are the following:

- AES-128 CBC chaining mode with PKCS7 padding for the encryption/decryption
- AES-128 CMAC for authentication/verification

4. Using the DAS to secure access tokens

DAS can be used by operators in combination with their user management service to first authenticate client application connecting and then strongly bind any kind of access tokens to these authorized application instances. The following sequence diagrams details a proposal of flow.



The strong binding of the sessionToken to the application instance is realized by interfacing the sessionKey **keyId** (step 14) in each request using the **sessionToken** (step 16). Previously, the sessionKey **keyValue** was securely delivered to the authorized application instance using the DAS in step 9. In details:

1. The application requests the Authentication SDK for authentication data.

2. The Authentication SDK returns the authentication data (dasMessage1) associated to a session object so that keys can be imported later (step 12) protected for this session. The crypto scheme is also important to provide to the server side, it can either be hardcoded in the application or exported from the Authentication SDK depending of the type of SDK.

3. The application performs a signin request to the server API Gateway by providing user credentials, the authentication data and the crypto scheme.

4. The API Gateway calls the DAS to authenticate the device by providing the scheme and the authentication data.

5. The DAS returns the deviceId and optionally the model. This deviceId might be a DRM deviceId or not depending of the scheme used.

6. The API Gateway requests the operator Portal to authorize the user on that instance of the device of this model.
7. The portal validates the user credentials on this device and then returns a status and the accountId.
8. The API Gateway now creates a session token for this accountId on this deviceId running this application. The API Gateway will later enforce that each time the token is used in a request, the keyValue associated with the token is used to sign the request data. To enable that functionality, the API Gtw needs to maintain the link between the keyId and keyValue and the session token.
9. The API Gateway requests the DAS service to protect the keyId and keyValue for the device by providing the dasMessage1 coming from the Authentication SDK and both field in the sessionKey input structure of the secureKeyExchange operation.
10. The DAS protects the keyId and keyValue for the device and returns a dasMessage2.
11. The API Gateway can now returns the sessionToken, the dasMessage2 and the keyId to the application.
12. The application imports the dasMessage2 into the Authentication SDK.
13. Now, for each request that uses the sessionToken and needs to be authenticated, the application first starts to create the request with all its argument and payload.
14. Then the application requests the Authentication SDK to sign the request using the keyId delivered in step 11.
15. The Authentication SDK securely computes the signature of the requests by using the keyValue referred by the keyId.
16. The application now sends the request to the API Gateway with its signature.
17. The API Gateway first validates the sessionToken and retrieves the keyValue and all other parameters associated to the sessionToken.
18. The API Gateway then validates the request signature thanks the keyValue.
19. The API Gateway can finally calls the portal to process the authenticated request by provided parameters retrieved from the sessionToken.
20. The portal returns the answer to the API Gateway.
21. The API Gateway returns the answer to the application.

5. Reference code for DAS on STB

Multiple entities involved in DAS, as explained in Section [3], interact with each other in a specific way to enable following use cases

1. Device Authentication
2. Encryption/Decryption
3. Sign/Verify

5.1 Reference application: *clidacapp*

The reference application for DAS on STB is an application with command line interface.

5.1.1 DAC – Device Authentication Client

DAC is a simplified layer on top of specific APIs of CCL absolutely needed for enabling DAS use cases.

5.1.2 Building the app

Instructions:

make clean

make

5.1.3 Running the app

```
nagra@nagra-virtual-machine:/mnt/hgfs/axavier_dac_profile/projects/clidacapp$ ./dacstbapp

Description:
        DAC STB application.
        The options bellow are globle options applying to all dacstbapp commands,
        Use 'dacstbapp <command> -h' to display the help message of a specific command.

Usage: dacstbapp command arguments[options]

Commands:                               Arguments and options:
        encrypt         | enc           -b <inputbuffer> -k <keyidentifier> -i <ivkey> -p <persistlicense_flag> -l <loglevel>
        decrypt         | dec           -b <inputbuffer> -k <keyidentifier> -i <ivkey> -p <persistlicense_flag> -l <loglevel>
        sign            | sgn           -b <inputbuffer> -k <keyidentifier> -p <persistlicense_flag> -l <loglevel>
        verify          | vfy           -b <inputbuffer> -s <signedbuffer> -k <keyidentifier> -p <persistlicense_flag> -l <loglevel>
        installlicense  | ins           -p <persistlicense_flag> -l <loglevel>
Options:
        --help          | -h            Display the Application help message
```

Description of commands:

1. Encrypt

```
nagra@nagra-virtual-machine:/mnt/hgfs/axavier_dac_profile/projects/clidacapp$ ./dacstbapp encrypt

Description:
        The command dacstbapp encrypt (dacstbapp enc) is used to encrypt the given input buffer

Usage: dacstbapp encrypt [options] -b <inputbuffer> -k <keyidentifier> -i <ivkey> -p <persistlicense_flag> -l <loglevel>

Options:
        --inputbuffer           | b <Text>      : Give Input buffer (Base64)
        --keyidentifier         | k <Text>      : Give Key Identifier (Base64)
        --ivkey                 | i <Text>      : Give Initialization Vector key (Base64)
        --persistlicense        | p <Num>       : Enable/Disable the persisted license storage. In default its enabled
        --loglevel              | l <Num>       : Debug level: 0 - disable, 1 - default, 2 - entry exit with error,
                                                  3 - information with error, 4 - debug, information and error
```

2. Decrypt

```
nagra@nagra-virtual-machine:/mnt/hgfs/axavier_dac_profile/projects/clidacapp$ ./dacstbapp decrypt

Description:
        The command dacstbapp decrypt (dacstbapp dec) is used to decrypt the given input buffer

Usage: dacstbapp decrypt [options] -b <inputbuffer> -k <keyidentifier> -i <ivkey> -p <persistlicense_flag> -l <loglevel>

Options:
        --inputbuffer           | b <Text>      : Give Input buffer (Base64)
        --keyidentifier         | k <Text>      : Give Key Identifier (Base64)
        --ivkey                 | i <Text>      : Give Initialization Vector key (Base64)
        --persistlicense        | p <Num>       : Enable/Disable the persisted license storage. In default its enabled
        --loglevel              | l <Num>       : Debug level: 0 - disable, 1 - default, 2 - entry exit with error,
                                                  3 - information with error, 4 - debug, information and error
```

3. Sign

```
nagra@nagra-virtual-machine:/mnt/hgfs/axavier_dac_profile/projects/clidacapp$ ./dacstbapp sign

Description:
        The command dacstbapp sign (dacstbapp sgn) is used to sign the given input buffer

Usage: dacstbapp sign [options] -b <inputbuffer> -k <keyidentifier> -p <persistlicense_flag> -l <loglevel>

Options:
        --inputbuffer           | b <Text>      : Give Input buffer (Base64)
        --keyidentifier         | k <Text>      : Give Key Identifier (Base64)
        --persistlicense        | p <Num>       : Enable/Disable the persisted license storage. In default its enabled
        --loglevel              | l <Num>       : Debug level: 0 - disable, 1 - default, 2 - entry exit with error,
                                                  3 - information with error, 4 - debug, information and error
```

4. Verify

```
nagra@nagra-virtual-machine:/mnt/hgfs/axavier_dac_profile/projects/clidacapp$ ./dacstbapp verify

Description:
        The command dacstbapp verify (dacstbapp vfy) is used to verify the given input buffer

Usage: dacstbapp verify [options] -b <inputbuffer> -s <signedbuffer> -k <keyidentifier> -p <persistlicense_flag> -l <loglevel>

Options:
        --inputbuffer           | b <Text>      : Give Input buffer (Base64)
        --keyidentifier         | k <Text>      : Give Key Identifier (Base64)
        --signedbuffer          | s <Text>      : Give Signed buffer (Base64)
        --persistlicense        | p <Num>       : Enable/Disable the persisted license storage. In default its enabled
        --loglevel              | l <Num>       : Debug level: 0 - disable, 1 - default, 2 - entry exit with error,
                                                  3 - information with error, 4 - debug, information and error
```

5. Installlicense

```
nagra@nagra-virtual-machine:/mnt/hgfs/axavier_dac_profile/projects/clidacapp$ ./dacstbapp installlicense

Description:
        The command dacstbapp installlicense (dacstbapp ins) is used for license installation processing

Usage: dacstbapp installlicense [options] -p <persistlicense_flag> -l <loglevel>

Options:
        --persistlicense        | p <Num>       : Enable/Disable the persisted license storage. In default its enabled
        --loglevel              | l <Num>       : Debug level: 0 - disable, 1 - default, 2 - entry exit with error,
                                                  3 - information with error, 4 - debug, information and error
```

5.1.4 Code organization

```
components
    app_log                                          Component used for logging
        src
            com
                cpp_log
                    nv_app_log.cpp
                nv_app_log.c
                nv_app_log.h
    dac                                              DAC component source code. This
        src                                          component directly interacts with CCL.
            com
                nv_dac.c
                nv_dac_provisioning.c
                nv_dac_provisioning.h
                nv_exchange.c
                nv_exchange.h
    dacapi                                           DAC header file.
        src
            com
                nv_dac.h
libs                                                 Directory containing Connect Client
    prmc                                             libraries.
        linux
            gcc-linux
                cma.single.NonNocsSTBNoKop.debug.gcc-linux.a
                cma.single.NonNocsSTBNoKop.release.gcc-linux.a
                cta.single.NonNocsSTBKop.debug.gcc-linux.a
                cta.single.NonNocsSTBKop.release.gcc-linux.a
                KOP_libs
                    libcx_checksum.a
                    libcx_endofcode.a
                    libcx_startofcode.a
                libtal.gcc-linux.a
                Readme.txt
                talsamplelib.tal_sample.debug.gcc-linux.a
projects
    clidacapp
        DACConfig.txt
        dacstbapp
        include
            clidacapp.h
            getHomeDirectory.h

        Makefile
        src
            clidacapp.c                              File containing the entry point (main
            getHomeDirectory.c                       function) to start program execution.

    common
        include
            applicationController.h
            cJSON.h
            dasGatewayProxy.h
            dasMessageUtils.h
            JSonTestHelper.h
        src
            applicationController.c
            cJSON.c
            dasGatewayProxy.c
            dasMessageUtils.c
            JSonTestHelper.c
```

5.1.5 App architecture

The reference app is designed in such a way that the communication between different entities is shown clearly, making it convenient to understand the sequence of calls and data transfer between these entities.

The *applicationController* shows how the application can make use of different functionalities supported by DAC (encrypt, decrypt, sign, verify & installlicense). It also calls the APIs of proxy gateway (*DasGatewayProxy_Authenticate* & *DasGatewayProxy_SecureKeyExchange*).

The dasGatewayProxy imitates how the gateway would behave when calling the DAS server APIs (*Authenticate* & *SecureKeyExchange*).