AVB (1722/1722.1) security technical proposal

Authors:
Kieran Tyrrell - Sienda Multimedia Ltd
Matt Jackson - Sienda Multimedia Ltd

Date:
14th Jan 2021

Abstract

This document documents the results of discussion with the 1722.1 working group on 13th Jan 2021 about how to use the various security related elements of 1722 and 1722.1 to provide ‘secure’ control of AVB devices.

1722 and 1722.1 provide features for:

- Key management
- Controller Authorisation
- Encrypted Control messages (ECC)
- Signed Control messages (ECC)
- Encrypted Control and Stream PDUs (AES)

No direction or recommendation of how these features should be used is found in 1722 or 1722.1.

Following discussion with the 1722.1 working group, one possible usage of these security features is proposed.

WARNINGS:

- AVB provides many flexible tools and options, and this proposal describes just one. This may not be the best solution for all (or any!) use case!
- No security review has been performed on this proposal. There is no claim that following this proposal will lead to a secure system.

Overview, 1722

1722 provides the following security related features:

AES Encrypted Format (13)

For encrypting streams. Keys may be exchanged using 1722.1 Key Management. Controller must add the same key to the Talker and any Listeners and ENABLE_STREAM_ENCRYPTION

ECC Signed Control Format (16)

The ECC Signed Control Format (ESCF) is used to transport an encapsulated AVTPDU that is cryptographically signed with an Elliptic Curve Cryptography (ECC) algorithm as defined in IEEE Std 1363a™. The signing of an AVTPDU allows a Listener to verify the sender of the AVTPDU and also allows the Listener to verify the integrity of the encapsulated AVTPDU.

ECC Encrypted Control Format (17)

The ECC Encrypted Control Format (EECF) is used to transport an existing control AVTPDU to another end station using elliptic curve public key cryptography.

Overview, 1722.1

1722.1 provides the following security related commands/responses:

Key management (7.6.1)

AUTH_ADD_KEY
AUTH_DELETE_KEY
AUTH_GET_KEY_LIST
AUTH_GET_KEY
AUTH_ADD_KEY_TO_CHAIN
AUTH_DELETE_KEY_FROM_CHAIN
AUTH_GET_KEYCHAIN_LIST

Used to manage, add, remove AES keys (which can be used for stream encryption) and ECC keys (which can be used for signing/encrypting AVDECC control PDUs).

Controller Authorisation (7.6.2)

AUTH_ADD_TOKEN - set or replace the authentication token to be used by the AUTHENTICATE command. Adding an authentication token when one doesn't exist triggers the AVDECC Entity to require that the AVDECC Controller authenticate to be able perform any action that is protected by authentication.

AUTH_DELETE_TOKEN - The AUTH_DELETE_TOKEN command is used to remove the current authentication token and disable authentication.

AUTHENTICATE - The AUTHENTICATE command is used to authenticate an AVDECC Controller with an AVDECC Entity.

DEAUTHENTICATE - The DEAUTHENTICATE command is used to deauthenticate an AVDECC Controller from an AVDECC Entity.

Transport Security Control (7.6.3)

ENABLE_TRANSPORT_SECURITY
DISABLE_TRANSPORT_SECURITY

The transport security commands (ENABLE_TRANSPORT_SECURITY and DISABLE_TRANSPORT_SECURITY) provide the mechanism for enabling and disabling the use of the transport security protocol.

Stream Encryption Control (7.6.4)

ENABLE_STREAM_ENCRYPTION
DISABLE_STREAM_ENCRYPTION

The Stream encryption commands (ENABLE_STREAM_ENCRYPTION and DISABLE_STREAM_ENCRYPTION) provide the mechanism for enabling and disabling the use of encryption on a Stream.

Entity Model Verification (7.6.5)

AUTH_GET_IDENTITY

The AUTH_GET_IDENTITY command is used to get the signature of the AVDECC Entity as signed by the manufacturer to prove that the AVDECC Entity is reporting its entity model correctly.

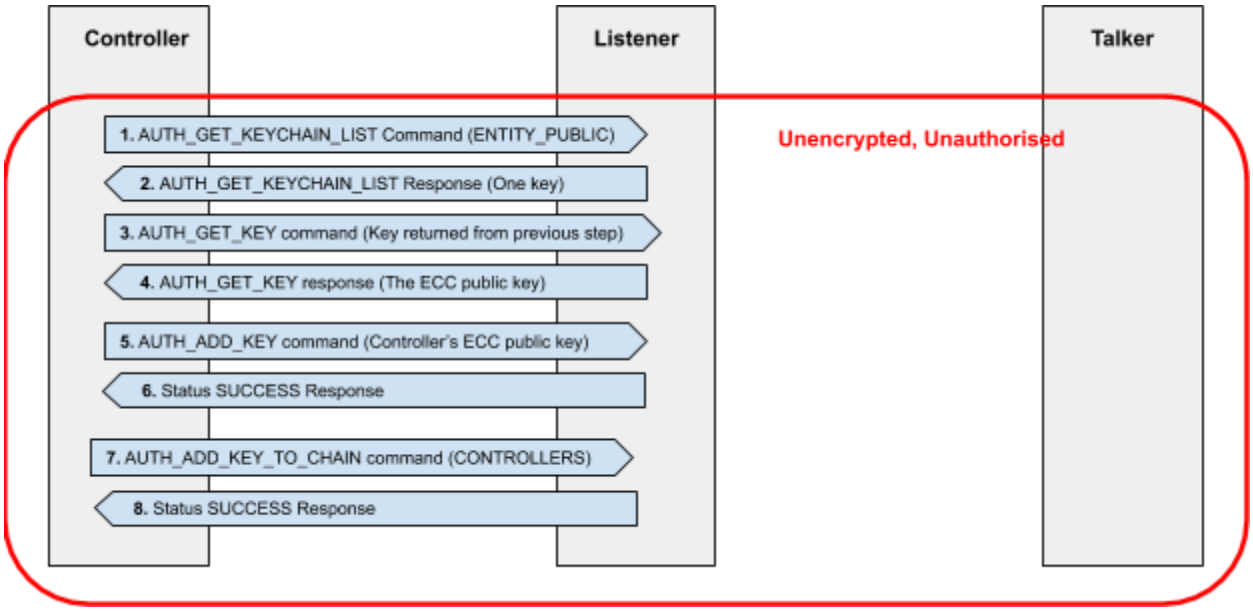
Proposal

The objective is to:

- Exchange ECC public keys so that a secure communication channel can be established between the Controller and Entity(s)
- Use the secure ECC encrypted channel to share an AES key
- Use the secure ECC encrypted channel to enable transport security
- Use the secure AES encrypted channel to authenticate the controller
- (optional) Use the secure AES encrypted channel to enable stream encryption

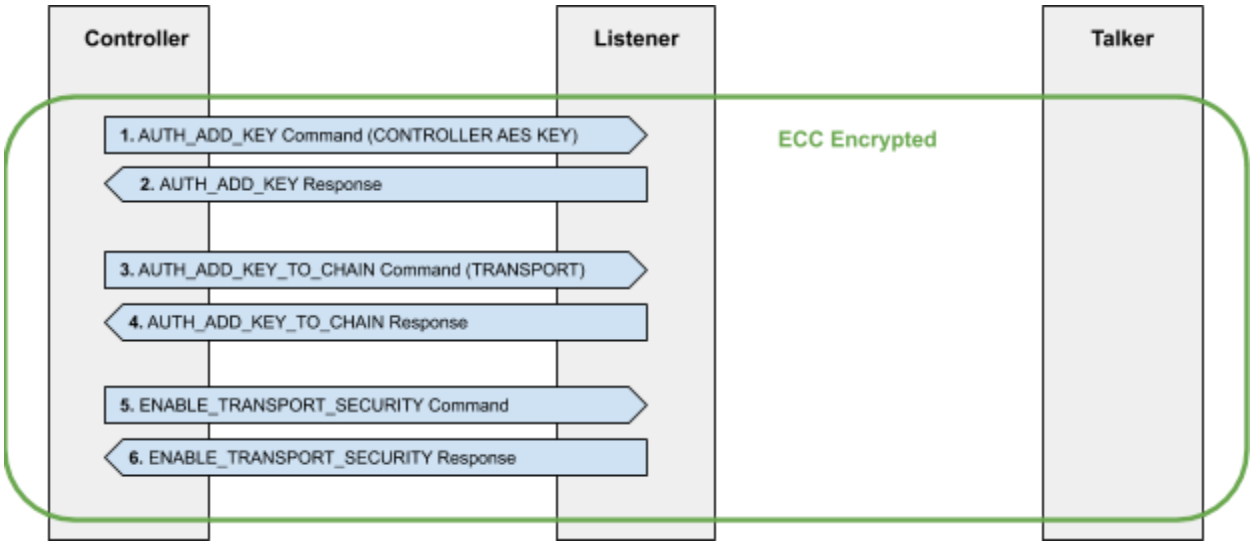
Exchange ECC public keys

Controller and Entity(s) exchange public keys so that a secure encrypted channel can be created with ECC Encrypted Control Format:



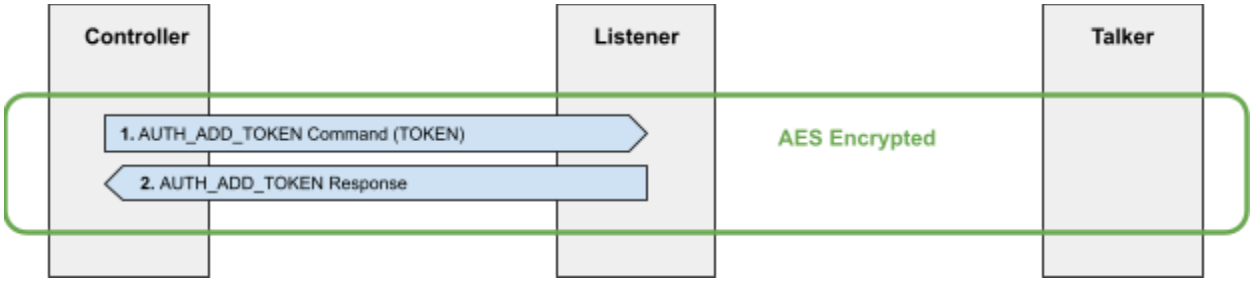
Use the secure ECC encrypted channel to share an AES key

ECC Encrypted Control Format PDUs may now be used to share an AES key, add it to the transport security keychain, and enable transport security:



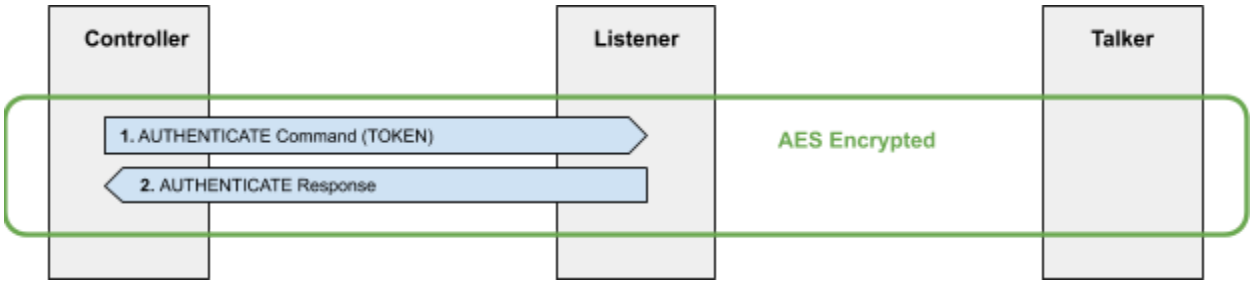
Controller Authentication

Now that there is encrypted communication between the Controller(s) and Entity(s), Controller(s) may enable ‘protection by authentication’ by adding an auth-token:



As soon as an auth token has been added, any protected command requires an authenticated controller.

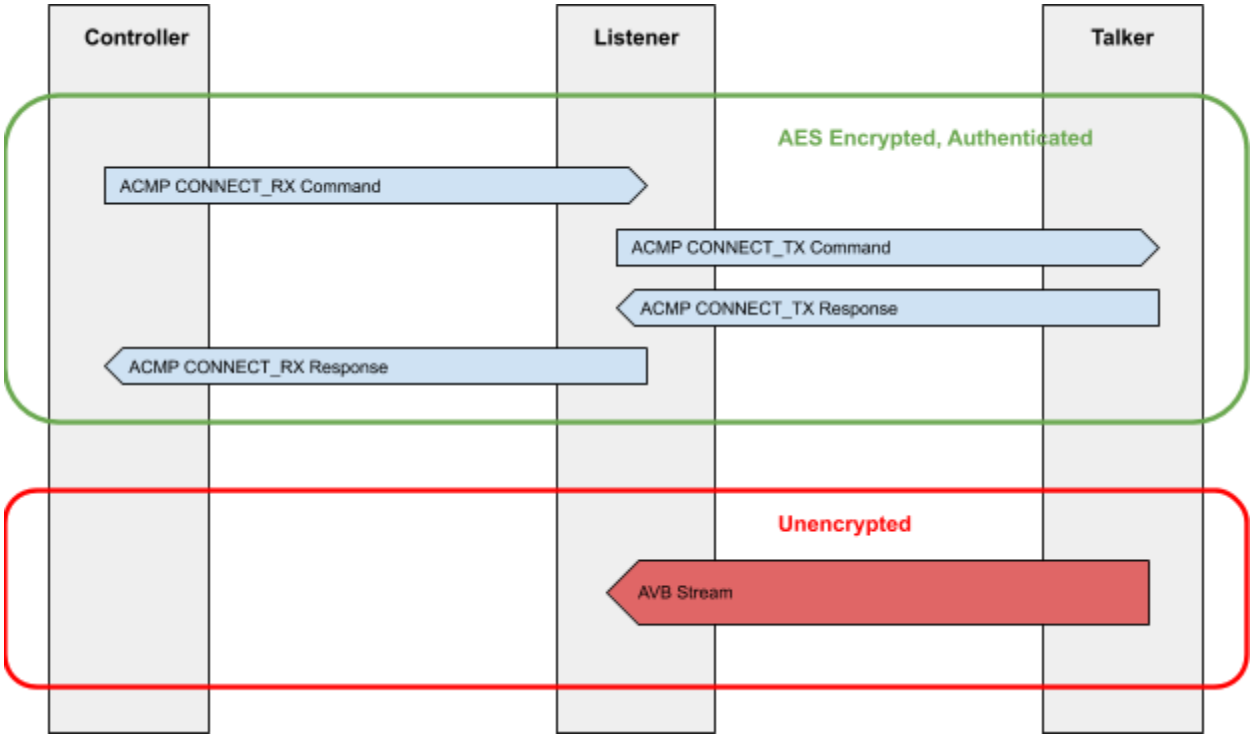
A Controller may now authenticate with the AUTHENTICATE command:



On receipt of the AUTHENTICATE command, an Entity should correlate the AES key used by the AES session with the Controller ID of the authenticated Controller. This is because both the AES session and the Controller ID are required to confirm the identity of the Controller. This prevents a Controller from a different AES session spoofing the Controller ID of an authenticated Controller.

The complete above described procedure would then be repeated for each other Entity on the network to be controlled in the same way.

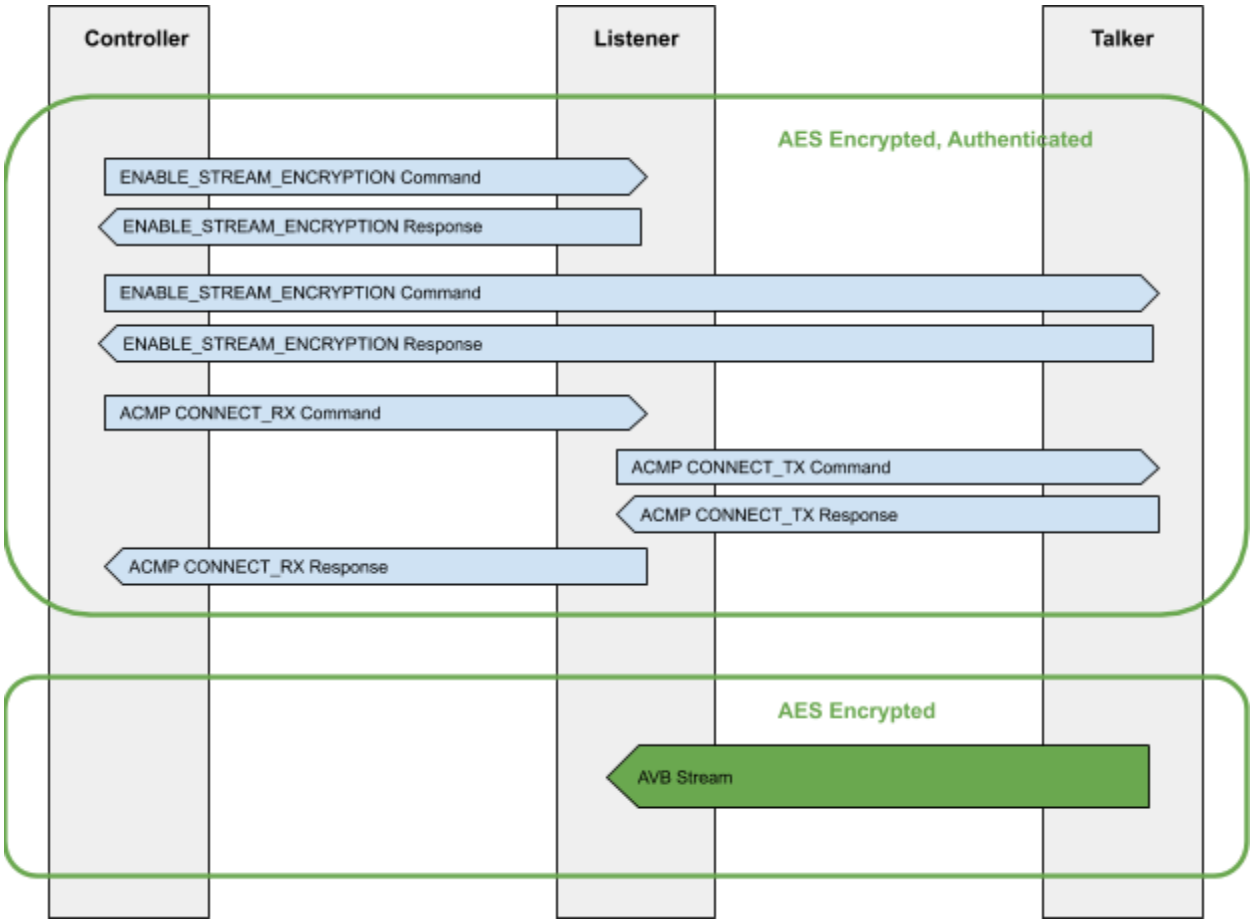
At this stage, stream connections (which are protected by authentication) would only be possible from authenticated controllers:



It should be noted that although all AVDECC messages are now encrypted and only authorised Controllers may perform protected commands such as connecting streams, the actual streams themselves are NOT encrypted.

Stream Encryption

To enable stream encryption the ENABLE_STREAM_ENCRYPTION command should be performed before starting a stream:



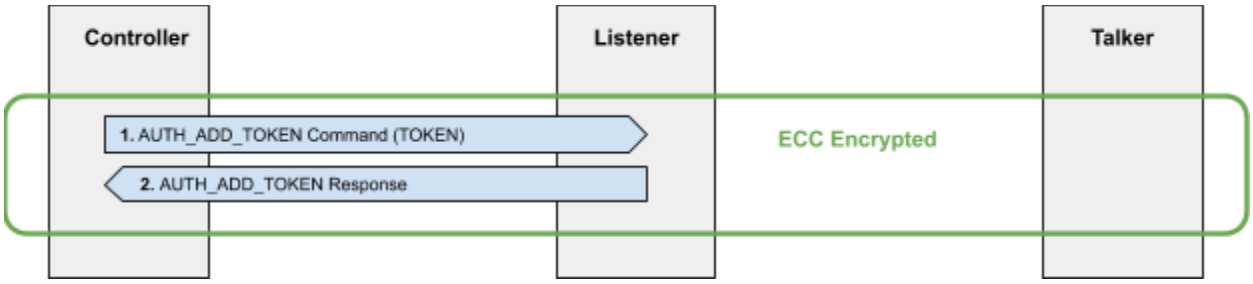
Notes:

- Stream encryption only needs to be enabled once per stream, not before each connection
- The ENABLE_STREAM_ENCRYPTION command enables encryption on a single stream, and so some streams may be encrypted and other not
- Both the Talker and the Listener must use the same AES key!
- The AES key used for stream encryption need not (and perhaps should not) be the same AES key as used for the control session encryption. If a separate AES key is to be used for stream encryption then this key can be exchanged using either the ECC protected key exchange, or within the existing AES session

Alternative Controller Authentication and Control

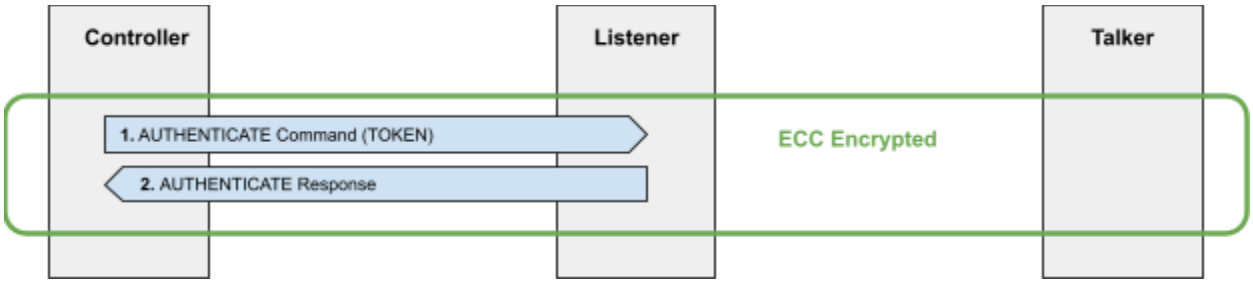
The use of a network wide AES ‘session’ is primarily for convenience and for consideration of low power devices on the network. AES is computationally much simpler than ECC, and AES hardware is often available on low power MCU class devices. An alternative to using an AES session is to use the ECC Signed or Encrypted PDUs for all AVDECC communication, and to base the Controller Authorisation on ECC Signatures/sessions instead:

Now that there is encrypted communication between the Controller(s) and Entity(s), Controller(s) may enable ‘protection by authentication’ by adding an auth-token:



As soon as an auth token has been added, any protected command requires an authenticated controller.

A Controller may now authenticate with the AUTHENTICATE command:



Possible disadvantages of this alternative are:

- Higher CPU load on Controllers and Entities
- Entities must handle a different session with each Controller
- Controllers must handle a different session with each Entity
- ACMP communication between Listener and Talker is more difficult as there is no shared session. The Listener may have to establish its own ECC session with the Talker, and Talkers may have to maintain many ECC sessions (one for each Listener).

Possible advantages to this alternative:

- ECC signing can be used by Entities to verify the source of commands, which may provide more security than relying on trust within an AES session. (One controller within an AES session could spoof the Controller ID of another controller within the same session.)