Product Development

FNV2-SOA Security Architecture

Version 0.3

Version Date: Aug 12, 2019

UNCONTROLLED COPY IF PRINTED

FORD CONFIDENTIAL

1 Table of Contents

2 Glossary

	Description
ACL	Access Control List
ECG	Enhanced Central Gateway
MQTT	Message Queuing Telemetry Transport
RPC	Remote Procedure Call
SOA	Service Oriented Architecture
TLS	Transport Layer Security
GID	Group ID
UID	User ID

3 Summary

The ECG is a critical component of the FNV2 architecture as all the in-vehicle networked nodes are connected to it. To prevent attacks from the outside, the ECG and the connected nodes must implement some security measures. TLS authentication is one of the implemented features. This document provides additional architecture description about security services part of the framework to support authentication and authorization.

4 Requirements

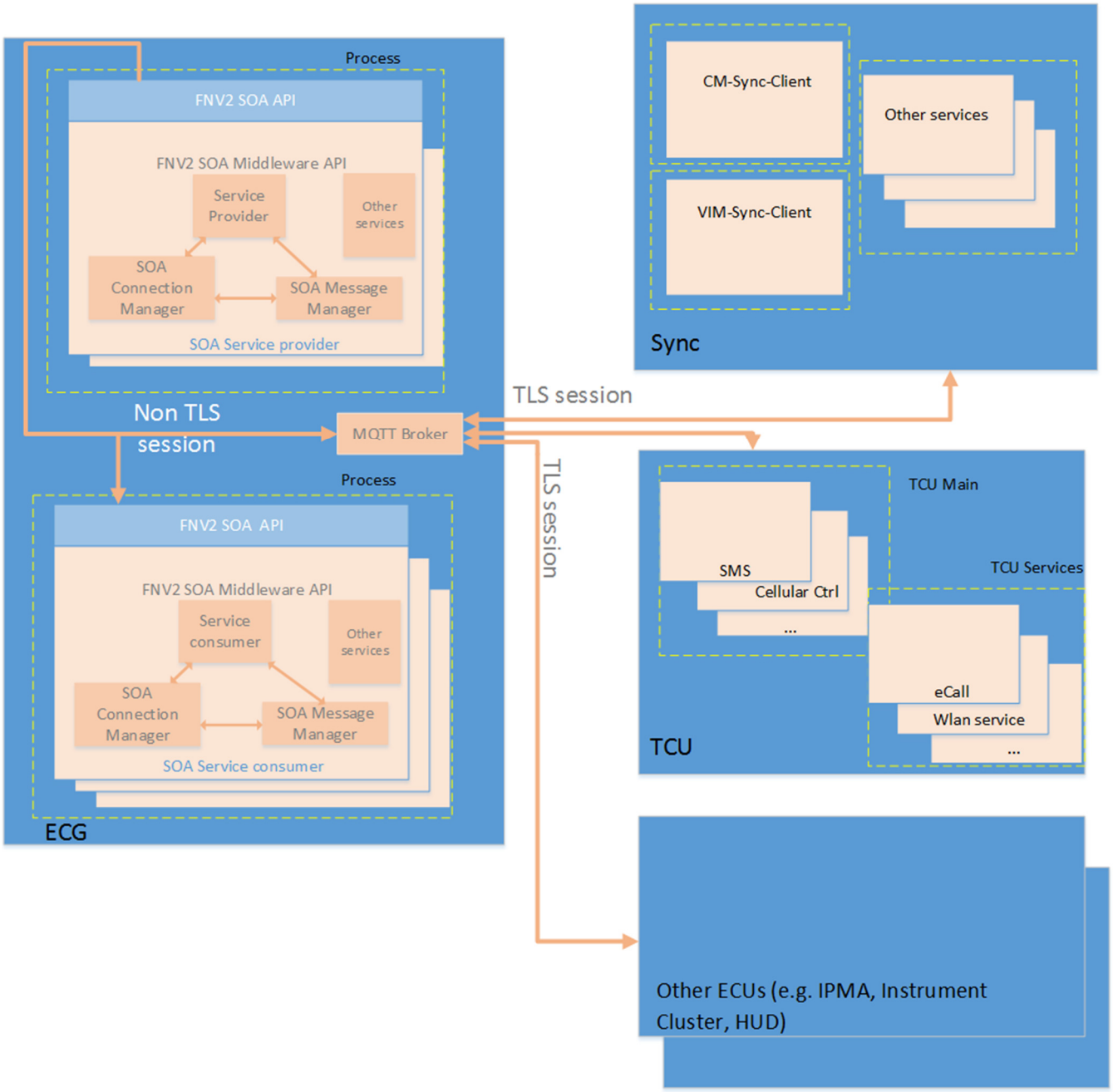
The security requirements are originally specified in the Security HLD document

Description
1- MQTT Authentication process must be protected from an attacker listening on the wire. (TLS session)
2- Access to MQTT Topics must have Access Control applied per user
3- A service must be able to authenticate which client sent a given request. This cannot use client provided data in the payload.
4- RPC operations over the SOA system must not allow a third-party to spoof a reply to a request.
5- The location of a client on a Sync/TCU/ECG should not matter from an authentication point of view.

5 High Level Design

5.1 Transport layer security

Securing the communication is only required for network nodes outside of the ECG. A two way TLS session is necessary to authenticate ECG and SYNC, TCU or any other modules on Ethernet participating in the SOA communication with ECG. Once the certificates are received and validated, the RSA private keys can be exchanged.



5.1.1 Certificate/Key Management

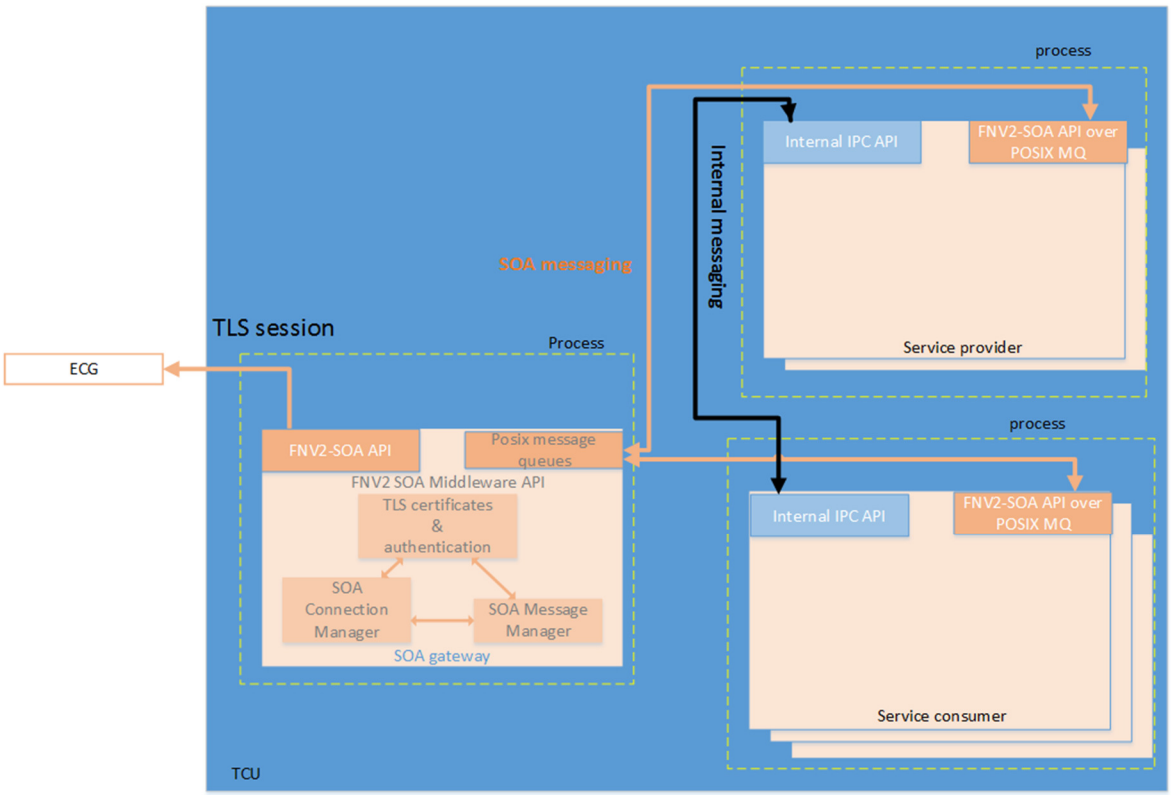
For additional security and improved performance, the certificates and keys on Sync/TCU or other ECUs are not managed by each of the services willing to access the ECG. A dedicated component, the SOA gateway running in a different process with elevated privileges is in charge of the certificate/key management. This component will establish a single TLS session on behalf of all the clients.

5.1.2 Gateway architecture

As the gateway is going to be implemented on a variety of modules, the proposed architecture is flexible enough to accommodate POSIX compliant OSes as well as different internal communication mechanisms within the module.

The SOA API on the ECG support MQTT as the transport layer. On the other ECUs it could be various mechanisms as QNX message queues or POSIX or non POSIX compliant IPCes. So it means that the SOA API needs to support other transport architectures than just MQTT. To minimize the impact on the SOA API, POSIX message queues has been selected as the IPC between ECU clients and the Gateway.

The clients on the ECU can use whatever IPC is appropriate for internal messaging. For communicating with the gateway and ECG, the SOA API is going to be used. The POSIX message queue or MQTT transport layers do not change the public API exposed to any client (ECG, Sync, TCU and others).



5.2 Client identification and authorization

In addition to providing an interface to the ECG and managing the TLS session, the gateway will be used to identify the request from the ECUs to the ECG.

5.2.1 Identification

Any consumer requesting a service needs to be authenticated and authorized. For this purpose, a unique identifier has to be generated. This identifier cannot rely on the clients as they cannot be trusted. So this role is covered by the gateway. Each message received by the gateway and forwarded to the ECG is tagged with an identifier.

5.2.1.1 On Sync/TCU and other fast ECUs

On Linux, the gateway can rely on OS services as the Unix Domain Socket and the getsockopt() API to get the credentials (user id) of the client. These values can then be used to generate the identifier. As this identifier also needs to be unique across all the nodes, the proposed format is the following:

clientIdentifier = "<NodeID>_<UserID>"

Where:

The **nodeID** is a string representing the name of the node which is **ECG**, **SYNC** or **TCU** or any other unique name to characterize the node/ECU on Ethernet. On the fast ECUs(Sync, TCU, etc.) this will come from the TLS certificate (CommonName).

The **UserID** is an additional information necessary if the Node ID is not sufficient. This comes from the process userID if available and provided by the OS running on the ECU. For better readability, the userID can be mapped to a unique string.

For a first implementation, we have the flexibility to just use the NodeID if this is sufficient from a security perspective. Particularly on Sync where all the processes run as root, the UserID does not allow to differentiate them. The nodeID can then be the only identifier provided by the gateway.

5.2.1.2 On ECG

On the ECG, even if the applications are trusted, we still need a mechanism to prevent a non authorized consumer to access a service. This is true for both native system services part of the ECG firmware but also for downloaded apps and services .

All the services running on ECG will have a user ID stored in the /etc/passwd file. The QNX *System launch and monitor* has the capability to launch processes with a given UID. The SOA API client can then use getuid() to retrieve the userID of the running process.

For downloaded apps, they will be associated with a manifest file including the client identifier (at least the NodeID and UserID).

5.2.2 Authorization

Once each message can be associated with a client identifier, a mechanism is required to check if a given client is allowed to get access to a particular service.

A typical use case is related to the ECG Connection Manager (ECG-CM). Only specific apps (e.g. Sync-CM client) on Sync are authorized to open a WiFi connection on TCU.

This verification can take place in different locations like the Gateway, the ECG broker or the service itself. A centralized approached is preferred and the broker ACL can take into account this requirement. From an implementation point of view, as the ACL is topic-based, the Sync, TCU and ECG client identifier should be part of the topic structure. Also, for each request from any consumers accessing a service, the type of service requested has to be part of the topic structure for the ACL to grant the appropriate permissions. The proposed addition to the topic structure is as follow:

...<ServiceType/<clientIdentifier>/...

Where:

The **ServiceType** is already part of the topic structure. This is the last part of the string including the component name and service (e.g. "VIM/DASHBOARD" or VIM/DASHBOARD/ENGINESPEED). This is typically what the ECG connection manager would be needing to make the distinction between setting up a WiFi or Cellular access.

The **clientIdentifier** is what has been defined above based on **"<NodeID>_<UserID>"**

This authentication sub-topic should not be exposed to the developer as the purpose is for authorization management only. This will be added by the API or gateway and potentially taken away by the broker custom ACL implementation (through the use of the broker plugin API).

The security requirements are applicable to the request-response messages. For that case, the sub topic is only applicable to the publish message. Blocking the publish prevents the service to receive the request and consequently to send a response.

The ACL built in the Mosquitto broker is a basic implementation as the ACL configuration is a simple text file which could be tricky to update.
The Mosquitto_auth plugin supports a large number of databases which is more suitable for ACL management. The detailed analysis is available here: ACL details. The plugin fetches the ACL information from a database and does not rely anymore on the broker ACL configuration file. In the current plugin implementation, there is a restriction which has to be resolved as the client has to be disconnected and reconnected for the plugin to check the updated client ACL. The way the ACL database is updated is outside of the scope of this document. It could be done by the service manager getting the meta data of the app manifest from the package manager (ALM Package Manager HLD).

5.2.2.1 Typical configuration of the ACL for TCU

Client identifier	Permission	Service Type	Authorized endpoint	Pub/Sub
TCU_MAIN	Allow	VIM/ROLLINGAVERAGESERVER	VIM/ROLLINGAVERAGESERVER/TCU_MAIN	Publish/Subscribe
TCU_SERVICES	Allow	VIM/BODYCONTROLSERVER	VIM/BODYCONTROLSERVER/TCU_SERVICES	Publish/Subscribe

The client identifier is the Unix userID provided by the gateway when the clients connect to it. The UnixID of tcu_main and tcu_services processes are mapped to TCU_MAIN and TCU_SERVICES strings including the NodeID. The service Type is what is used in the endpoint (e.g. /SERVICES/REQUEST/ECG/VIM/ROLLINGAVERAGESERVER)

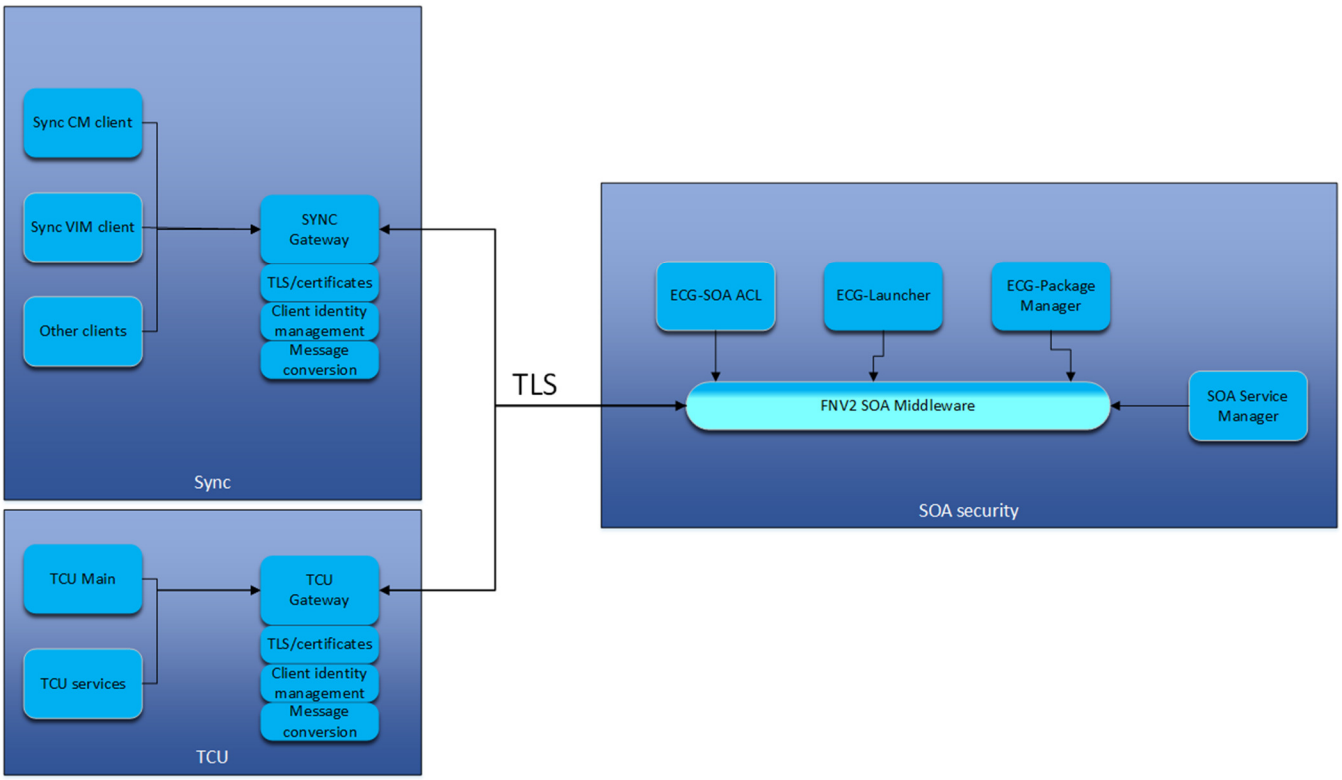
The gateway identifier in the table below is the information the gateway from the endpoint and the client identifier

Client	Publishes to	Gateway Identifier	Actual endpoint provided by the Gateway to ECG	Authorization
TCU_MAIN	/SERVICES/REQUEST/ECG/VIM/ROLLINGAVERAGESERVER	VIM/ROLLINGAVERAGESERVER/TCU_MAIN	/SERVICES/REQUEST/ECG/VIM/ROLLINGAVERAGESERVER/TCU_MAIN	Allowed
TCU_MAIN	/SERVICES/REQUEST/ECG/VIM/BODYCONTROLLERSERVER	VIM/BODYCONTROLLERSERVER/TCU_MAIN/	/SERVICES/REQUEST/ECG/VIM/BODYCONTROLLERSERVER/TCU_MAIN	Denied
TCU_SERVICES	/SERVICES/REQUEST/ECG/VIM/BODYCONTROLLERSERVER	VIM/BODYCONTROLLERSERVER/TCU_SERVICES	/SERVICES/REQUEST/ECG/VIM/BODYCONTROLLERSERVER/TCU_SERVICES	Allowed

The custom ACL implementation done in the auth_plugin of the ECG will parse the endpoint and compare the gateway identifier with what is defined in the ACL database.

5.3 Interfaces

The gateway multiplexes the Sync and TCU access requests from all the client to a single connection going to the ECG. It includes the certificate and private key to setup the TLS session with the ECG. In addition, it also includes the capability to set the permission level of the client requests.



6 Performance (if applicable)

The memory is limited on TCU and Gateway implementation should not exceed few MB.