Secure Gateway – A concept for an in-vehicle IP network bridging the infotainment and the safety critical domains

Jonas Berg*, Jens Pommer, Chuan Jin, Fredrik Malmin, Johan Kristensson*
Semcon Sweden AB, Gothenburg, Sweden
*Email: {jonas.berg, johan.kristensson}@semcon.com

Abstract

The Secure Gateway is a concept that addresses the need for applications to access the vehicle network and, at the same time, providing a protection for malicious or malfunctioning applications. The objectives are to provide access to the vehicle network from user-installed applications in the infotainment environment, in a secure and controlled way. As the infotainment domain is no longer confined to a single node, an in-vehicle IP (Internet Protocol) based subnet is used. The concept is based on standard technologies and is using a layered architecture, where the networking layer is the lowest level. The messaging layer consists of publish-subscribe messaging using the MQTT (MQ Telemetry Transport) protocol running via the Secure Gateway Core, consisting of a broker with custom add-ons. Communication is encrypted via TLS (Transport Layer Security) certificates, and the broker handles access control to different MQTT message topics. The service layer consists of “resources” and “applications”, which are software abstraction of devices, and users of these resources, respectively. Presence information is handled for resources, which also enables plug-and-play installation of aftermarket additions. In the vehicle network adaptor the IP-network and the safety-critical network meet. There are no open (inbound) IP ports on the vehicle network adaptor.

1 Introduction

A modern vehicle contains many nodes with different types of sensors and actuators. Vehicle electronics is an area that is growing fast due to expanding needs regarding safety aspects, and facilitated by cost reductions on electronics. In addition, we have an infotainment environment that is increasing in functionality and is becoming more and more advanced and dynamic. This combination results in an increasing number of new ideas, and there are wide spread demands to connect these two worlds in a more dynamic way than what is possible today. In essence, there is a need to be able to create applications that can be dynamically installed in the infotainment environment and, when in place, these applications shall be able to access the vehicle’s sensor and actuator data.

A key aspect is that this connection has to be done in a controlled and secure way to avoid that malicious or malfunctioning applications harm the system. If done improperly, such applications might manipulate sensitive vehicle functionality such as brakes or steering and that would be disastrous.

In a large system such as a modern vehicle there are many nodes which perform different tasks independent of each other. Implementing such a large system, which enables for an easy and secure way of communicating between different nodes, is troublesome and hard. A common approach is to divide parts of the system into more manageable objects which are separated from each other. The problem today is to construct a system in such way that it enables the possibility to integrate plug-and-play concepts while still being secure and easy for third party manufacturers to develop, as well as to integrate all nodes into a complete system.

The infotainment domain is no longer confined to a single node. Users would like to control the sound system from the rear seat entertainment units, and it should be possible to easily install aftermarket rear seat entertainment units without any configuration. The user functionality of these units does not differ much from that of the infotainment head unit (IHU). Furthermore, back seat passengers would like to be able to send destination information from a smartphone app to the vehicle navigation system. Also other aftermarket hardware additions need to be handled in a safe and secure way.

For the car manufacturers it is important to be able to decouple infotainment projects from vehicle projects, due to the different paces. Thus well-defined interfaces are required.

Among related research projects are for example the EVITA (E-Safety Vehicle Intrusion Protected Applications) project[1] which primary deals with hardware-protected security relevant component. The projects SeVeCom (SEcure VEhicle COMmunication)[2] and PRESERVE (Preparing Secure Vehicle-to-X Communication System)[3] aim at securing vehicle-to-vehicle and vehicle-to-infrastructure communication. An architecture using an in-vehicle IP

(Internet Protocol) network for communication with consumer electronic devices is described by C. Borcea and coworkers[4].

2 Secure Gateway concept

The Secure Gateway (SG) is a concept that addresses the needs for applications to access the vehicle network and, at the same time, providing protection against malicious or malfunctioning applications.

The objectives of the Secure Gateway are:

- In a secure and controlled way provide access to the vehicle network from applications installed in the infotainment environment.
- Provide an easy way to extend the system with new nodes and applications.
- Provide an open and easy way to access development environments so that different parties can develop new functionality in an efficient way.

Secure Gateway implements an IP based subnet, using other protocols on top. A publish-subscribe messaging pattern connects applications to vehicle related resources via a messaging broker. The main infotainment node is not directly connected to the CAN (Controller Area Network) or Flexray vehicle network.

This paper describes a concept; therefore the implementation is not production ready. Rather, it should be seen as a proof-of-concept.

2.1 Introductory example

To illustrate the idea with the Secure Gateway, consider the following example. In modern vehicles, the climate control module is usually located together with the infotainment system since this is where the knobs and buttons are located. It is also becoming common to integrate more advanced functionality such as adding climate zones in the car and setting the timer for the parking heater. This is functionality that is typically suitable to be controlled using a graphical user interface (GUI).

Now imagine that a back seat passenger want to access the above functionality from an application in his tablet computer. When the app has been downloaded and installed in the tablet, he connects the tablet to the car (in this example via WiFi) and the application automatically finds the climate control module. A GUI that contains information of the current status, such as current temperature and fan speed, is presented. Buttons to turn on and off seat heating and changing temperature are also available.

The idea is when you press a button in the GUI in the app running on the tablet, let’s say, to turn on the seat heater, a signal is sent to the climate control module. The climate control module will react on this signal, make the appropriate changes and then send out updated values about its status. The status information is sent to all applications that are interested in the information and not only to the application that sent the signal. This way, the standard climate application running in the infotainment head unit (IHU) is also informed about the change and can update its GUI accordingly.

In order to discuss the mechanisms behind, we need to introduce two components that are central in the Secure Gateway setup – applications and resources.

2.2 Resources

In short, a resource is a software abstraction of devices that are connected to the SG network. A device in this context can be for example a media player, radio, climate control module, temperature sensor or interior lights control. The purpose of the resource concept is to provide an application programming interface (API) to the functionality of these devices. This includes both functionality to read data and status, as well as an interface to control the device using different commands.

In the example above, a resource that can be used to read data such as temperature and fan speed is available. The resource has also the possibility to control some parts of the climate control module such as changing the fan speed. Note that a resource is not required to provide access to all functionality that a device provides. It is possible to have a very simple resource that is connected to a highly advanced device. The climate control module is probably quite advanced, but it might be that the functionality exposed to the SG network is limited to a few functions; for security reasons for example. It is also possible for a device to provide several resources in parallel. For example one very simple resource with limited functionality targeting a special set of applications and a more complex resource that can be used by applications that need more functionality. This split is also an important part in the security concept of the SG. As described later, it's possible to limit the access to different resources to include only selected applications.

As a final note regarding resources, it is important to understand that a resource is not limited to hardware devices; it could also be something that provides data based on calculations or communication with other devices. An example could be a resource that provides speed information based on GPS (Global Positioning System) data.

2.3 Applications

The application, or app, is another important component in the SG network. Applications are typically installed on the IHU and use the APIs of the resources. In modern, dynamic software eco-system, apps are fundamental. The concept of being able to download and install new applications and upgrade them makes it possible to evolve the system over time. New functions can be added and old functions can be updated or even removed if not needed anymore. The use of applications enables new ways to offer a more customized solution for the users, providing the specific functionality and configuration that the user wants or is accustomed to.

The SG concept acknowledges the need for a dynamic application framework which is an important driving force in order to make most use of the SG concept. Without the possibility to install and upgrade applications, the use of the resources and the rest of the SG network are limited. From a SG perspective, the app is simply a user of one or more resources. The guiding principle for the applications in the SG network has been to provide the functionality with an as low as possible threshold.

The fundamental requirements for the applications can be described in the following way. First of all, the application must be able to join the SG network in a simple and controlled way. It should be easy for an application developer to create an application that connects to the system and registers its presence. When connected to the network, the application needs functionality to discover which resources are available. The application also needs functionality to query for the resources' available APIs and usage. Furthermore, it needs a way to establish an encrypted communication with the resources on the SG network and a way to determine which resources that can be trusted. Finally, the application needs mechanisms to communicate with the resources.

The preferred way to be able to access the functionality, mentioned above, would be to implement library functions for different programming languages and that the application developer can include these libraries in a simple way. It is also beneficial if a community is available where different kinds of problems and patterns can be discussed. A key concept is that the APIs used are open and freely available.

3 Guiding principles

In the search for a working solution, a set of principles for the architecture was established to guide the work:

- Base it on standard technologies
- Use a layered architecture approach
- Provide a low threshold to develop and install apps
- Make it easy to extend the system with new nodes
- Build an open environment with possibility to have parallel technologies without compromising security

Throughout the work, these principles have been used when selecting frameworks, and during the discussions how to define APIs and communication, and when setting levels of complexity etc. The principle to build everything based on standard technologies and to use a layered architecture is mostly due to security aspects which are described in more detail below. A low threshold for application developers is considered to be a key aspect if the solution shall have any chance of being widely accepted. It should be noted that the selected messaging protocol (that is described in more detail below) is available for many different platforms and programming languages. The system must also be possible extend easily. Adding a new sensor or actuator should be an easy task. Finally, even though we believe that the Secure Gateway concept described in this document has its benefits, it is important that other solutions can work in parallel. This should also be possible without compromising the security of the Secure Gateway functionality.

Since computer security is a fundamentally hard topic, the chosen method was to reuse ideas from other domains and put them in an automotive context. Since the techniques used on the Internet to communicate efficiently and securely have been proven to work, one of the first decisions made was to rely on standard network technologies as much as possible. This has many benefits, for example the vast documentation resources available. Further, the open source community has provided a lot of working software components that can be used to build secure and reliable solutions without investing a lot of time in scaffolding work.

4 Layered architecture

As mentioned in the introduction, one of the guiding principles for the SG is to use a layered architecture approach. The concept is a network consisting of three layers – a network layer, a messaging layer and a service layer. The layers help to enforce "separation of concerns" and increase security.

The network layer consists of an IP network, with methods for handing out IP addresses to nodes and to advertise these IP addresses to other nodes. The messaging layer consists of publish-subscribe messaging using the Message Queue Telemetry Transport (MQTT) protocol running via the Secure Gateway Core, consisting of a broker and custom add-ons. The top layer in the Secure Gateway concept is the service layer. In this layer, there are two important actors – applications and resources. As described earlier, a resource is a software component that represents a device, for example a media player or a temperature sensor. The resource provides APIs using the messaging layer (MQTT). Applications, installed for example in the IHU, can then use these APIs to read data or control actuators. Below are the different layers discussed in more detail.

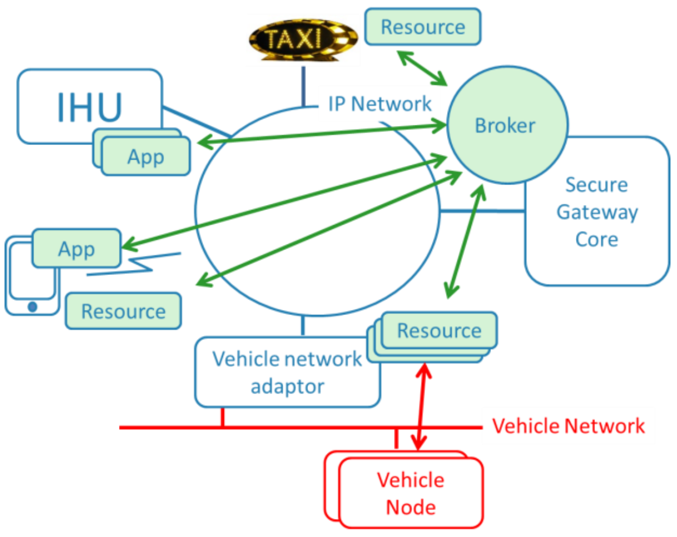

Figure 1 Layers in the Secure Gateway Concept. Blue boxes and connectors represent the network layer, while messaging and service layers are printed in green.

5 Network layer

The base for the Secure Gateway is an IP (Internet Protocol) network running on WiFi or Ethernet, or a combination of both. Standard protocols for communication and configuration has been used; more specifically the network is TCP (Transmission Control Protocol)/IP based with encryption using TLS (Transport Layer Security)/SSL (Secure Sockets Layer). In addition, protocols to handle how to set IP-addresses without the need for a DHCP (Dynamic Host Configuration Protocol) server, and to discover services, are used to simplify the configuration of the network. These techniques and protocols are grouped together to form the lowest layer in the Secure Gateway concept – the network layer.

Using these concepts enables a way to connect a device to the network (for example via an Ethernet cable) that will automatically get an IP-address assigned to itself and also receive the IP-address to the Secure Gateway Core (see below) without any manual configuration.

The Secure Gateway is built on top of TCP/IP stack; all the connection and communication will go through the IP based network. We would like to provide a simple way for each client (e.g. an application on an aftermarket rear seat infotainment unit) to be able to join the Secure Gateway network without any manual configuration by the user, which means adding auto discovery feature to the Secure Gateway network. Therefore, we need to find a good solution to solve the auto discovery problem in the IP based network, and to support plug-and-play functionality. It should be possible for applications and resources to discover the necessary components in the system. Furthermore, we also need to make sure the connection is secure, to prevent the network from malware and attack. Authentication and authorization will be important parts from a security perspective.

There are protocols built on top of TCP/IP stack for enabling for auto discovery to work in a network. Examples are NDP (Neighbor Discovery Protocol), SDP (Service Discovery Protocol), SSDP (Simple Service Discovery Protocol), UPnP (Universal Plug-and-play), DNS-SD (Domain Name System – Service Discovery)[5], mDNS (multicast DNS)[6], Zigbee, DHCP, Jini, SLP (Service Location Protocol), SAP (Session Announcement Protocol), WS-Discovery (Web Services Dynamic Discovery), XMPP Service Discovery (Extensible Messaging and Presence Protocol), CDP (Cisco Discovery Protocol).

We inventoried different auto discovery protocols and tried to determine what the pros and cons are with each system. As a principle of trying to find the easiest and most compatible protocol, we finally ended up using a solution called Zero Configuration Networking (zeroconf). It is a set of technologies consisting of service discovery via the mDNS/DNS-SD protocols, and assignment of network addresses. There is an open-source native Linux implementation for zeroconf available called Avahi. It is compatible with Bonjour, which is the Apple implementation for zeroconf.

To assign IP addresses without the use of a DHCP server, different approaches are used for IPv4 and IPv6. For IPv6 there is automatic configuration of the IP address, partly based on the MAC (media access controller) address of the network interface. When using IPv4 the Address Resolution Protocol (ARP) can be used to set addresses in the range 169.254.1.0 to 169.254.254.255.

6 Messaging layer

6.1 Publish-subscribe messaging pattern

The messaging layer consists of publish-subscribe messaging using the MQTT protocol running via the Secure Gateway Core, consisting of a broker and custom add-ons. In this pattern, there are three actors: publishers, subscribers and a broker. Topics are the denominators for the messages and are used to categorize them in order to find the receivers. In the SG network, a hierarchical topic structure is used `(/topic/subtopic1/subtopic2/…)`.

Messages are sent from a publisher to one or more subscribers, but instead of the publishers knowing which subscribers to send the messages to, all communication is done via a broker. The subscribers register themselves at the broker and list the topics that they are interested in. This information is stored in the broker. All messages from publishers are tagged with a topic and then sent to the broker. Based on this information, the broker can determine which subscribers that shall receive the message and forwards the message accordingly. From the subscribers' perspective, the publishers are hidden; they can only see the broker and has no idea whether the messages are sent from one or many different publishers. From the publishers' perspective, the number of subscribers and other publishers are unknown. To them, there is no information whether their messages are used or not (even though such information of course could be added as an extra layer on top of the pub-sub pattern if needed).

The pattern is illustrated below. In this example, we have one publisher and two subscribers. The subscriber at the top right in the picture informs the broker that it is interested in messages of the topics "`/sensor/temp`" and "`/sensor/fanspeed`". The subscriber at the bottom right is only interested in messages on the topic "`/sensor/temp`". When the publisher wants to inform that a new temperature reading is available, it sends a message on the topic "`/sensor/temp`" with a payload (here "27 C") to the broker. The broker looks up which subscribers that shall receive the message and forwards it to them.

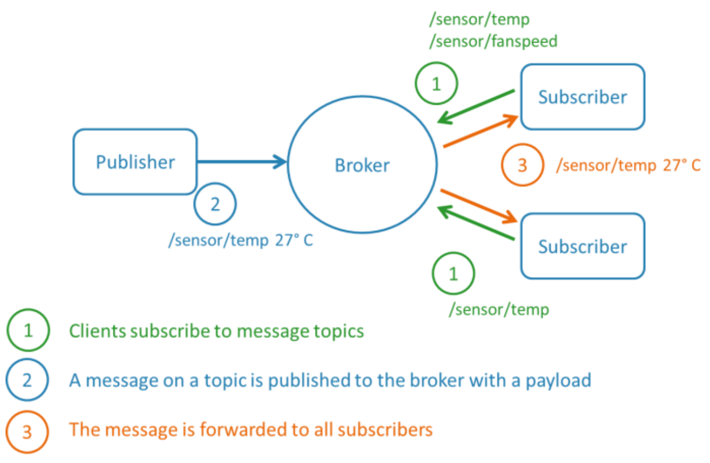
/sensor/temp
/sensor/fanspeed
1
Subscriber
Publisher
Broker
3
/sensor/temp 27° C
2
/sensor/temp 27° C
Subscriber
1
/sensor/temp
1 Clients subscribe to message topics
2 A message on a topic is published to the broker with a payload
3 The message is forwarded to all subscribers

Figure 2 Publish-subscribe pattern in Secure Gateway

The publish-subscribe pattern is a key concept in the SG network and there are a few reasons why it has been selected to be used in the SG network. First, it allows a decoupling between the sender and the receiver of a message. This functionality increases security since the sender and the receiver are not directly aware of their respective existence. From the subscribers' perspective, the sender is not directly known, thus making it harder to extract information that is not intended to be shared.

The decoupling also adds flexibility to the system. Since the system configuration is dynamic – resources and applications can be added and removed at any time – the applications cannot be implemented in a way so that they depend on a specific configuration. This forces the applications to query the broker for available topics without knowing which resources that will provide the actual functionality. On one hand, this adds some complexity for the applications but the benefit that it prevents hard coupling between applications and resources outweighs this. With this approach, it is, for example, possible to replace one resource with another without modifying any apps that uses the functionality. The apps wouldn't even have to know that the replacement had taken place.

The broker can also add extra filter to control the access level of different message channel. Some clients might only be allowed to read data, why they will behave as subscribers only. Other clients could have both read and write permission, so that they are both subscribers and publishers. We can even more specify the channel/topic to be allowed to read or write or both by different clients. This is done by a built-in feature called access control list (ACL), in which we declare the client and the topics the client can access together with type of permissions (read/write).

6.2 Protocol selection

There are many communication protocols that could handle data transmission between different nodes, and that are more-or-less supporting the publish-subscribe pattern. A few of them are: AMQP (Advanced Message Queuing Protocol)[7], MQTT (MQ Telemetry Transport)[8], XMPP, STOMP (Streaming Text Oriented Messaging Protocol), DNP3 (Distributed Network Protocol), DDS (Data Distribution Service), OpenMAMA (Open Middleware Agnostic Messaging API) and COAP (Constrained Application Protocol). Important characteristics are features, security, reliability and availability of software libraries for many programming languages. The main candidates were MQTT and AMQP.

MQTT is a protocol published by IBM in 1999, and is intended for Internet-of-things (IoT) and machine-to-machine (M2M) communication. Both MQTT and AMQP support authentication and encryption based on SSL/TLS.

We ended up choosing MQTT, since it is an simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. MQTT is very easy to get started with, and the design principles are matching our needs in this project. AMQP is also a very good and powerful protocol which can be used in many scenarios, but is a little more heavyweight.

This messaging protocol together with the broker and a set of broker extensions forms the messaging layer that resides on top of the network layer.

6.3 Broker additions

Two extensions to the broker have been elaborated to some extent in the project.

The first extension – the Gatekeeper – is related to the security functionality of the Secure Gateway concept. To be able to have different authentication levels in a dynamically way, a software module called the "Gatekeeper" has been added to the system. The Gatekeeper will control which apps that have access to which resources and how the resources can be used (read only or read and write, for example). The gatekeeper use the access control lists functionality provided by MQTT to determine access rights.

The second extension – the Service Manager – is a component that provides functionality to the system to keep track of the current available resources. Since the MQTT protocol does not include functionality for presence, such a component is required in order to provide the applications with their needed information. The service manager functionality is implemented using MQTT's "last-will" concept together with a specific topic structure. The component contains a "last-will-executor" that send out information when resources are disconnected. This is described in some detail below.

7 Connection to the vehicle network (CAN/Flexray)

Communication with vehicles networks, such as CAN or Flexray is an important part of the concept and is handled using vehicle network adaptors. In the vehicle network adaptor the IP-network and the safety critical network interface and can communicate with each other, using the messaging and service layers in the Secure Gateway.

The idea is to implement resources that provide the functionality of the actual nodes on the vehicle network to the applications on the Secure Gateway. It is important to realize that it's not a one-to-one mapping between the functions on the vehicle nodes and the functionality provided by these resources. Instead, it is encouraged to group the functionality into APIs that make sense to the applications. It is a good opportunity to make use of the idea that there can be several resources that provide similar functionality but with different access rights as mentioned earlier in the resources section. These kinds of resources typically reside on a vehicle network gateway.

For security reasons the vehicle network adaptor is connecting to the broker only. The vehicle network adaptor has no open (inbound) port on the IP network.

8 Message topic structure

The next issue after selecting MQTT protocol is challenge to define the interface between different nodes in the system.

We propose a well-defined topic structure that supports the different needs in the system. Since MQTT is a public/subscribe protocol which send/receive data on topics or channels, having a predefined common topic structure is necessary for all clients.

A data message could contain for example a temperature reading. The data content is sent in a message on a corresponding topic, which could be for example `data/climateservice/temperature` and with a payload of 27.5. The availability of this signal is indicated by a separate message, having the topic: `dataavailable/climateservice/temperature` and the value `True`. The availability message is published as retained message, which means a newly connected client will have the latest availability information about each of the interesting signals.

Note that both applications and resources are both publishers and subscribers. Typically applications publish commands, while resources publishes data.

The general topic structure for MQTT in our system is defined as follows:

```
typeOfMsg/resourceName/signalName
```

- **typeOfMsg** - Describes what type of message it is. We currently implement four types; command, data, commandavailable and dataavailable.
- **resourceName** - The name of the resource that is providing the data, or to which resource the command is aimed. Each resource is its own MQTT client.
- **signalName** - The name of the signal that is provided, or the actual command to perform.

For example: `dataavailable/climateservice/temperature`

In this example, the typeOfMsg is `dataavailable`, the resourceName is `climateservice` and the signalName is `temperature`. This simple example illustrates a signal containing the temperature is currently available from the climate service if the payload value is `True`, and unavailable otherwise.

The typeOfMsg can be one of four different types.

- **command** – Messages for controlling resources, sent by applications (or other resources). Resources listening to the command will react according to it.
- **data** – Messages that contain some data e.g. status information from a resource. The data could for example be a reading of vehicle speed.
- **commandavailable** - Message that contains information about the availability of a certain command signal. The payload of signal is either True or False.
- **dataavailable** – Message that contains information about the availability of a certain data signal. The payload of signal is either True or False.

There is a special topic `resourceavailable/servicename/presence` we defined for representing a resource being online or offline. When the resource is offline, the Service Manager inside Secure Gateway will do the cleanup and notify that the data and commands for this resource no longer is available. This is done via the `dataavailable` and `commandavailable` topics, so only the Service Manager listens to the `resourceavailable` messages.

Also scalability is handled via the `dataavailable` and `commandavailable` topics, as these are sent per data message type and per command type. A luxury vehicle could have a climateservice with more options than the one in an economy vehicle. An application using the climateservice resource can adapt to the availability of data and commands available in the particular car, for example by adapting the graphical user interface accordingly.

9 Usage example

To sum up the definition of the topic structure: a resource can provide data and receive commands and react on them, or even send commands to other resources to control them. Resources are mainly information providers and action executors.

An application can send commands to control resources, and subscribe to data from resources. Applications are mainly data consumers and command issuers.

Below will the topic usage be exemplified.

9.1 From application point of view

An app developer knows already when designing the app which signals that are required, and which signals that are optional.

If the application is interested in the temperature:

- Connect to the broker.
- Subscribe to `dataavailable/climateservice/temperature`. If a message `True` is received on this topic, the signal is available. If the message is `False`, then the signal is not available.
- If the signal is available, subscribe to the topic `data/climateservice/temperature`.
- Arrange the GUI layout depending on which signals are available.
- A callback will be activated upon the reception of data on the subscribed topics. A callback will also be activated upon changes in the availability of previously requested signals, and can be used for GUI layout changes.

Thus in order to use this, the app developer needs to know two constants:

- Full topic for the data of the signal, for example: `data/climateservice/temperature.`
- Full topic for the availability of the signal, for example: `dataavailable/climateservice/temperature`.

If the app is using more than one signal, the procedure above is repeated for each signal. It is not necessary for the app developer to use (subscribe to) the dataavailable messages. It is merely provided as a support to app developers, in order to make presence easier to detect. The hierarchy of the topics will later be used for access control in the broker. Right now, the full topics can be regarded as constants. Note that these topics may differ between car manufacturers, but hopefully can be standardized for at least each car manufacturer.

9.2 From resource point of view

A resource makes data available to the IP network. It might also receive commands.

A temperature sensor might behave like this:

- Publish available signals, for example: `dataavailable/climateservice/temperature True` etc.
- Publish the data when available, for example: `data/climateservice/temperature 25.3` etc.
- Set the "last will" in the broker to: `resourceavailable/climateservice/presence False.`

"Last will" is a message that is published automatically by the broker if the client disconnects abnormally.

9.3 From (extended) broker point of view

Due to the implementation details of MQTT, each MQTT client can only supply one last will. This means that the broker can only transmit one topic per client, when they disconnect. We therefore need some feature that convert "resource not available" to a bunch of messages "signal not available". In our terminology this feature is the "service manager", and is implemented as a separate MQTT client.

When the connection to the climateservice is lost, the broker is publishing the last will of the climate service:
`resourceavailable/climateservice/presence False`

That message is received by the Service Manager, that emits `dataavailable/climateservice/temperature False` and similar messages for each data and command topic related to the climateservice.

9.4 From end user point of view

Let's take the rear seat entertainment remote control application as an example. The procedure is as below:

- Open the application from the rear seat entertainment unit.
- The application is using network discovery, and looks for the IP address of Secure Gateway Core.
- The application tries to connect to the Secure Gateway Core.
- The application and Secure Gateway Core verify each other's certificate.
- The connection established if the verification succeeds, and all further communication will be encrypted.

As seen above, the user only need to open the application, and then all the steps following will be done automatically. This procedure is exactly what we want to achieve, i.e. to auto discover the service and connect to it securely without any user configuration.

10 Aftermarket additions

The Secure Gateway makes it easy to add aftermarket installations to the infotainment domain. We have already discussed addition of rear seat entertainment units. Secure Gateway can also help establish a connection from a smartphone app to the vehicle, so that a rear seat passenger can control the music system or send destination information to the vehicle navigation system.

Also hardware additions can be handled by the Secure Gateway. We have (rather naively) implemented an Ethernet enabled taxi sign that is controlled by a taximeter app in the IHU via the Secure Gateway. This eliminates the need for additional displays in the vehicle. This concept is especially interesting for bodybuilder extensions to trucks and other commercial vehicles.

11 Security considerations

The security efforts have focused on two main aspects. The first and foremost work has been to protect the vehicle network from the infotainment domain. The second aspect has been to protect the resources from malicious use from applications. A guiding principle in the security for the SG is that it shall not be possible to extract information from one vehicle (such as keys, certificates, etc.) and by using this information, compromising the security of another vehicle. Though, there is no explicit protection for tampering with your own vehicle.

11.1 Layered Architecture Principle

There are several motivators for the layered architecture principle. First of all, a layered solution is an important method to achieve low coupling between different parts of the system and having modules with high cohesion. Being a good software principle, it usually adds to the maintainability of the source code which lowers the risk of severe bugs. And since bugs are often exploited to break into systems, this principle adds to security.

From a security perspective the layered architecture opens up for each layer to have its own security mechanisms that work independently of the others. With that approach, each layer can be modified in order by making modifications on one layer. It is also possible to replace or even add a complete layer in order to support the communications between subscribers and publishers.

In addition, to access devices on the vehicle network, the communication is routed via resources on specific vehicle network adaptors. Since there is no direct translation between the protocols used on the vehicle network and the Secure Gateway's IP-network, all communication between applications in the Secure Gateway network must use resources on the vehicle network gateways to access the vehicle network. By using this technique, security mechanisms can be added in several locations – on the WiFi network, on the TCP/IP-network, protection by means of the messaging protocol, additional controls in the broker and its extensions and finally in the resources. By combining these mechanisms, the security risks can be mitigated.

11.2 Standard Security Solutions

Another aspect of the security issues is to make sure that the selected solutions actually perform as expected. One of the decisions made in the project was to minimize the investment of in-house implementation. The idea is instead to use off-the-shelf standard components and connect these in the best way possible. Since the communication in the Secure

Gateway is based on TCP/IP a simple security measure is to make sure that the communication is encrypted using TLS/SSL. The available open source solutions for this have proven to be secure enough to secure communication on the Internet. Hence, the technology was considered a good choice for the communication in the Secure Gateway. By using TLS/SSL all the messages between the broker, the resources and the applications are encrypted.

11.3 Authentication using certificates

By using Transport Layer Security (TLS) protocol together with MQTT it is also possible to increase the security when it comes to authentication. Instead of using a username password approach, certificates are used in the messaging layer to authenticate both resources and applications. The applications and the resources can, using the same mechanisms, also verify the authenticity of the broker.

It is common to use the user name and password to do the authentication, but in our case we treat each client as a standalone entity, i.e. one person is able to have one or more applications (clients), and different applications should have different permissions, e.g. application A is only able to read data from the vehicle and application B with higher permissions can both read and write data back to vehicle. Also different people can download the same application with the same permissions. It is up to the client identity rather than the user identity to determine the permissions. So our security is client-based rather that user-based.

Therefore, we choose to use certificates instead of user name and password, and inside each certificate, there is a built-in ID for this particular client for access control purpose. So the certificate will encrypt the connection between client and Secure Gateway, and will require that both the Secure Gateway Core and client side should provide a valid certificate, so that they can communicate in a secure way after verification. Since we would like to make sure that only valid clients with valid certificates can connect to the Secure Gateway and start to read data or even write data back to control the car, otherwise there might be a risk that a dangerous application without certificate probably will damage the vehicle network, which will bring potential risk to the driver and passengers.

We use the regular certificate based TLS support as used in web servers/browsers. For our case in Secure Gateway, not only the server side, but also each client needs to provide a valid certificate to be able to proceed. Since the client application might need to e.g. write to CAN, we need to be careful to make sure only authenticated clients with the correct permission are able to write to CAN.

We have an access control list in Secure Gateway for the authorization purpose after each client is being authenticated. The Secure Gateway will refer to the access control list to decide whether or not a request (read/write) from a client is allowed.

Regarding the security aspect of the solution, we use our own self-signed CA certificate thus far, and distribute clients' certificates manually. When it comes to the public key infrastructure (PKI), it is hard to provide a complete solution within the scope of this project. Such solution might involve certificate trust chain, certificate distribution and revocation etc. This part can be further developed in future related projects.

On the Wifi network, standard encryption protocols such as WPA2 are used. It would be possible to secure the communication using certificates also here, but that is not something that has been addressed.

11.4 Resource Access Levels

In order to analyze the risk compared to the benefit of exposing functionality or data from the vehicle network to the infotainment domain, a concept with resource access levels was discussed during the project. Four levels of resources were identified:

- **Open** – Resources that are available to all applications within the Secure Gateway network. An example could be a resource that provides vehicle speed or temperature.
- **Privileged** – Resources that is available to applications with special credentials. This is typically for third party developers that have been selected by the OEM.
- **Secure** – Resources that are sensitive or contains data that must not be manipulated or spread. This level is typically only for the OEM or trusted partners. This could, for example, be access to some of the safety functions or personal information.
- **Locked** – The final level is used for functionality or data that should not be exposed at all to the infotainment domain. An example is brakes; it might be the case that the risk of exposing braking functionality to applications in the infotainment domain outweighs the possible benefits. Should that be the case, there should not be any resources that expose this functionality.

There is no specific support for the different access levels in the current version of the Secure Gateway concept. Currently the access levels shall be seen as a tool to support the discussion related to risk versus benefit.

11.5 Gatekeeper

The broker access control list is combined with a Secure Gateway component called Gatekeeper, which can dynamically change the permissions for the clients. For example, if your kid is playing with the remote control wildly from rear seat, you can revoke the write permission of the remote control by using the IHU, which has a more powerful permission. It is however possible to keep the read permission for the remote control, so that it still can receive data but not to be able to send command to control the vehicle. And you can always assign the write permission back to the remote control client if you wish.

It is also possible for the gatekeeper to take actions if the remote control app on the phone should try to access the data anyway. Such an action could be to disconnect the phone altogether. No such functionality has been elaborated in the Secure Gateway work, though.

12 Prototype implementation

A Secure Gateway architecture prototype have been implemented, and connected to a production car. Data was read from the CAN bus, and CAN commands were sent by using diagnostic messages.

The broker was running on an embedded Linux board, of the type BeagleBone. For service discovery on the IP network, the Avahi software was used. As MQTT broker was Mosquitto used, and the Python programming language was used for most of the prototyping work.

We used the Paho MQTT client for the Python programming language. Also the vehicle network adaptor was running on the same embedded Linux board, which had CAN ports enabled by a CAN expansion board manufactured by Towertech.

We plan to open-source our prototype implementation of the Secure Gateway concept architecture during May 2015.

13 Discussion and future work

To define a good interface via the topics of MQTT is a difficult task; we finally ended up with an abstract layer, which is the well-defined topic structure for all the resources and applications to use. Future work can include better defined and more generic topics.

The current version of the Secure Gateway is by no means a production ready system. The demonstrations during the course of the project have shown that it has great potential and that it opens up for a more dynamic infotainment system. Though, much work is still needed to find appropriate software components to use in a live system. One obstacle is to conclude where to fit the Secure Gateway into the vehicle system architecture. There are a few alternatives, but quite a lot of work is still needed to make this integration.

More specifically, the following are examples on further work that is needed in the area:

- More work in the Public Key Infrastructure (PKI) area is needed. Parts of the security in the Secure Gateway concept relies on certificates and the different methods how to distribute and protect the certificates today are not sufficient. There is a risk that certificates stored in devices are extracted and used in other devices.
- Mechanisms to supervise the Secure Gateway have not been evaluated.
- Certificate revocation must be handled.
- The gatekeeper component in the system is a very rudimentary extension to the broker. The concept can be elaborated further and, beneficially, done together with the work on the PKI. This work should also try to find the mechanisms to be used to grant permissions to the applications. One question is if it should be done automatically or if it should involve the driver/owner of the vehicle.
- Key management and distribution are intrinsically difficult, and must be solved.
- Software and certificate updates are important to keep a high security level, but are not addressed here.
- Since all communication is routed via the broker, the broker is a single point of failure. More investigation is needed to find solutions to possible denial of service attacks. For example by adding redundancies.
- The resource access levels area needs to be explored further. One problem is that it is hard to prevent that certificates in one application is extracted and used in another application. This makes it hard to differentiate between entities operating on the "privileged" level. Certificates on the "Secure" level can be protected by other means (such as hardware protections) and the problem is not applicable for the open level.

14 Acknowledgements

This work is partially funded by the Swedish government agency VINNOVA’s FFI program, in the project PLINTA. The authors would like to thank the collaborating parties in the project (including an major vehicle OEM).

15 References

[1] EVITA project. Retrieved from http://www.evita-project.org/

[2] SEVECOM project. Retrieved from http://www.transport-research.info/web/projects/project_details.cfm?id=46017

[3] PRESERVE project. Retrieved from https://www.preserve-project.eu/

[4] Automotive Proxy-Based Security Architecture for CE Device Integration, C. Borcea et al, "Mobile Wireless Middleware, Operating Systems, and Applications", ISBN 978-3-642-36659-8, p.62-76, 2013.

[5] DNS-SD protocol. Retrieved from http://www.ietf.org/rfc/rfc6763.txt

[6] mDNS protocol. Retrieved from http://www.multicastdns.org/

[7] AMQP protocol. Retrieved from https://www.amqp.org/

[8] MQTT protocol. Retrieved from http://mqtt.org/