# Using AUTOSAR high-level specifications for the synthesis of security components in automotive systems

Cinzia Bernardeschi[1], Gabriele Del Vigna[1], Marco Di Natale[2], Gianluca Dini[1], and Dario Varano[1]

[1] Department of Information Engineering, University of Pisa, Largo L. Lazzarino 1, 56122 Pisa, Italy
[2] Scuola Superiore Sant'Anna, Piazza Martiri della Libertà 33, 56127 Pisa, Italy
{cinzia.bernardeschi,gianluca.dini}@ing.unipi.it
{g.delvigna88,dario.varano}@gmail.com
{marco.dinatale}@sssup.it

**Abstract.** The increasing complexity and autonomy of modern automotive systems, together with the safety-sensitive nature of many vehicle information flows require a careful analysis of the security requirements and adequate mechanisms for ensuring integrity and confidentiality of data. This is especially true for (semi-)autonomous vehicle systems, in which user intervention is limited or absent, and information must be trusted. This paper provides a proposal for the representation of high-level security properties in the specification of application components according to the AUTOSAR standard (AUTomotive Open System ARchitecture). An automatic generation of security components from security-annotated AUTOSAR specifications is also proposed. It provides for the automatic selection of the adequate security mechanisms based on a high-level specification, thus avoiding complex and error-prone manual encodings by the designer. These concepts and tools are applied to a paradigmatic example in order to show their simplicity and efficacy.

**Key words:** Security, modelling, AUTOSAR

## 1 Introduction

Robots, unmanned aerial vehicles (UAVs) [1], self-driving cars, and unmanned underwater vehicles (UUVs) [2] are examples of complex networks of autonomous systems that were considered just fiction a few decades ago [3]. They share the common technological denominator of being a networked embedded and control system composed of many sensors, actuators and embedded computers [4]. As with many of these complex networked systems, external intruders can intentionally compromise the proper operation and functionality of these systems.

Modern cars are not exempt from these threats. Currently, automotive systems integrate an increasing number of features aiming at providing active safety and then full autonomy [5], [6]. These functions execute on a distributed architecture of embedded computers, or *Electronic Control Units* (ECUs), with the final objective of controlling the vehicle actuation systems, such as steering, braking, acceleration, lights. ECUs are interconnected by wired networks such as the Controller Area Network (CAN) and Ethernet with the integration of wireless capability, e.g., keyless entry and diagnostic, entertainment systems. This increased connectivity leads to an increasing number of access points and potential cyber security threats. Koscher et al. [7] demonstrate that an attacker who is able to infiltrate any Electronic Control Unit (ECU) can leverage this ability to completely circumvent a broad array of safety-critical systems. Furthermore, Stephen Checkoway et al. [8] demonstrate that remote exploitation is feasible via a broad range of attack vectors including CD players and Bluetooth sub-system. The impact of these cyber security threats is getting more and more relevant as cars are getting more and more autonomous and interconnected. It follows that cyber security is a requirement that has to be addressed since the early stages of the project.

In this paper, we provide a set of *modelling extensions* to address cyber-security requirements at design stage in the AUTOSAR (AUTomotive Open System ARchitecture), an open industry standard for automotive software architectures [9]. AUTOSAR provides a component-based system design and a development approach based on a three-layered architecture: the Application layer, that contains the Application Software Components providing system specific functionality; the Run Time Environment layer, an auto-generated layer providing for the implementation of application functions in concurrent threads and the implementation of communication; and, finally, the Basic software layer, that contains standardised operating system, IO and other services, including libraries and communication services [10].

In AUTOSAR, safety and security services are being standardised with respect to the set of basic services that may be required by the application, such as the basic cryptographic functionalities provided by the Crypto Service Manager (CSM) or the definition of integrity-related message authentication codes (MACs) in messages (SecOC component) [11], [12].

With reference to the AUTOSAR component-based methodology, we show how:

i) to *annotate* a component diagram by means of cyber security concepts during the modelling phase; and,
ii) to automatically synthesize AUTOSAR-compliant *security components* from such an annotated component diagram.

This introduces the following advantages. First, this allows designers to take into account security aspects in the early phases of system design. In-vehicle communications among components over the internal buses can be protected from cyber threats such as eavesdropping, integrity and spoofing. Next, the automated synthesis of security components allows a designer to handle cyber security concepts at a high level without being an expert of security and so avoiding errors and inaccurate selections of methods and algorithms. Finally, the synthesized security components are AUTOSAR-compliant and thus they can readily be plugged into an AUTOSAR application.

With respect to the modelling extensions, we define two sets of elements aimed at the specification of: the *trust level* of a functional component, which gives an indication of the effort required to compromise the component; and, the *security requirements* of a communication between components, which specifies the demand in terms of confidentiality and authenticity on such a communication.

As an example showing the result of the application of our methodology, models and tools, we discuss a set of application components as a typical representative of active safety or autonomous driving subsystems, with communications that are characterized by integrity and/or confidentiality requirements.

Our work can be placed in the research thread about enriching modelling formalisms with security requirements and constraints. Relevant examples are those based on UML such as UMLSec, MDS and MARTE [13], [14], [15], [16], [17]. These proposals tend to address general application domains, i.e., distributed applications or embedded, real-time applications. In contrast, our proposal has been conceived for the automotive domain and to extend the AUTOSAR modelling and development methodology.

The rest of the paper is organised as follows. Section 2 summarizes the AUTOSAR concepts of interest for this work. Section 3 presents the modelling extensions that are required for the specification of application-level security requirements. Section 4 presents our synthesis tool. Section 5 illustrates the application of our methodology, models and tools to an autonomous driving subsystem, a typical representative of active safety which requires integrity and confidentiality of communications. Finally, Section 6 draws final conclusions and illustrates future works.

## 2 AUTOSAR in a nutshell

The objective of the AUTOSAR consortium and standard (`www.autosar.org`) is to create an open and standardised software architecture for automotive systems allowing for the exchange and integration of software components on a standardised platform. AUTOSAR provides a set of specifications that apply to the software architecture, including the definition of port-based component interfaces and a methodology for the development process.

A fundamental concept in AUTOSAR is the separation between application and infrastructure. With reference to Figure 1, AUTOSAR *Software Components* (SW-Cs) encapsulate application functionalities that run on the AUTOSAR infrastructure. Each software component is represented by a model that consists of a structural representation of the component interfaces, described by ports for data-oriented or client-service interoperability. Each port is typed by a data or operation interface and defines the points of access for the component. In addition, the behaviour description of each component includes the component

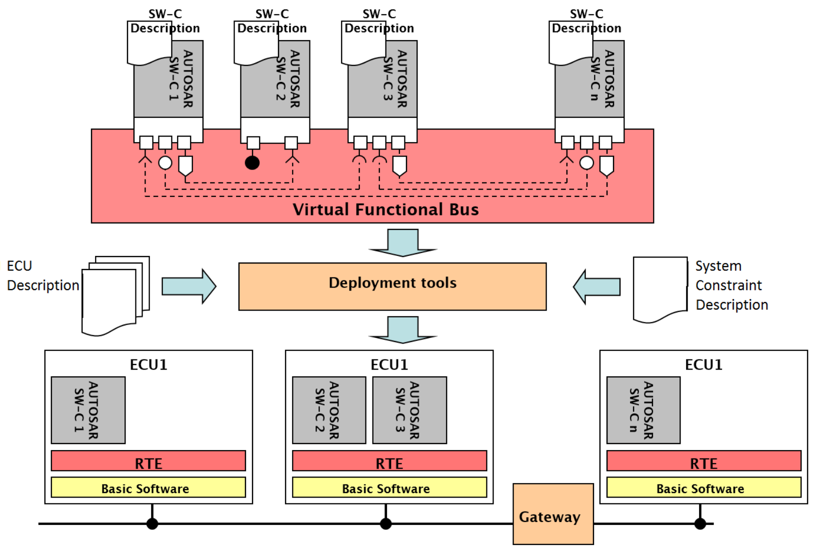


**Fig. 1.** AUTOSAR architecture

functions (or runnables) and the set of events that trigger the execution of those functions. The *Virtual Functional Bus* (VFB) is the set of all the connections that the designer defines between the components' ports to specify the interoperability of components in an automotive system. These connections are defined in AUTOSAR at an abstract level.

From an architectural standpoint, AUTOSAR consists of three layers (see Figure 2): the Application layer, the Runtime Environment (RTE) layer, and the Basic Software (BSW) layer. The Application Layer contains the application implementation in terms of software components. The RTE layer is a middleware that provides for the implementation of components' behaviour (executing the runnables by threads and the related scheduling configuration) and the implementation of the communication among components. Finally, the Basic Software provides basic services and software modules on each ECU, i.e., including the operating system, the communication stack and all drivers. Approximately eighty BSW modules are defined and grouped into service modules, such as System services, Memory Services, Communication Services, and so on. According to the AUTOSAR methodology, software components can access BSW modules only through the RTE. So, thanks to the RTE abstraction layer, software components can be developed independently of the underlying hardware, which means that they have the transferability and reusability property.

In order to provide the actual implementation of software components and their VFB communications over a network of ECUs, AUTOSAR requires the designer to specify the execution platform in terms of network topology and configuration of the ECUs (see *ECU Description* and *System Constraints De-*

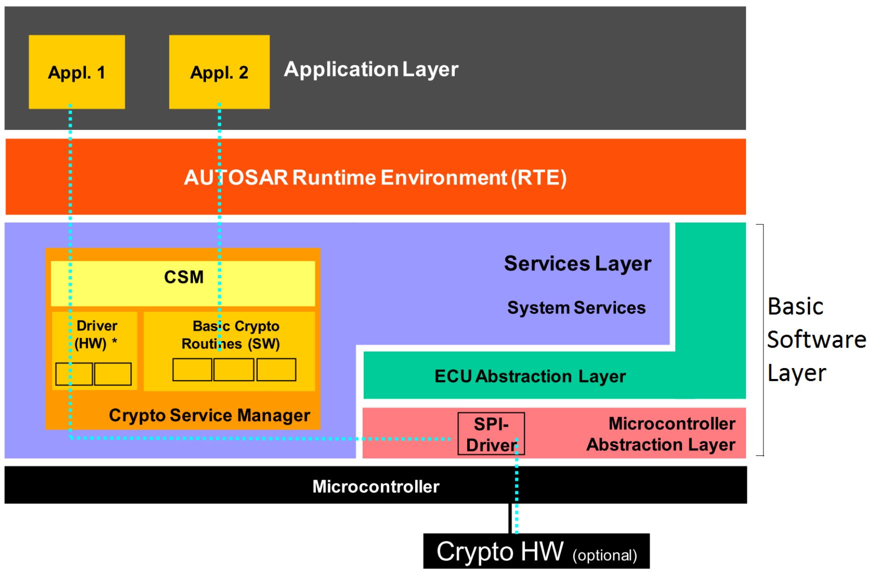


**Fig. 2.** AUTOSAR layers

*scription* in Figure 1) as well as the mapping of software components to ECUs. Each instance of a software component is statically assigned to one ECU. Then, based on the platform description and mapping, a set of synthesis tools generate the appropriate RTE and provide for the configuration of the *Basic Software* (BSW) modules on each ECU.

Figure 3 shows an example of sender-receiver application which comprises two interconnected software components, `SWC1` and `SWC1`. The input data port `in1` of `SWC1` is connected to output data port `out2` of `SWC2`. The two data ports have an interface type `interface1`. In addition, the input data port `in2` of `SWC2` is connected to the output data port `out1` of `SWC1`. The two data ports have interface type `interface2`. The two software components have also two `CSMport` client-server ports that we introduce in the next section. The application has been developed using Rational Rhapsody, a modelling tool by IBM. Rhapsody builds AUTOSAR concepts on top of UML and allows us to define extensions for them by leveraging the typical extension mechanisms of the UML language, namely, profiles and stereotypes.

## 2.1 Security as a service

Within the BSW layer, AUTOSAR makes security mechanisms available to the developers in three different modules: a) the *Secure On-board Communication* (SecOC) module [18], which routes IPDUs (Interaction layer Protocol Data Units) with security requirements; b) the *Crypto Abstraction Library* (CAL) [19], which implements a library of cryptographic functions; and, finally, c) the *Crypto*

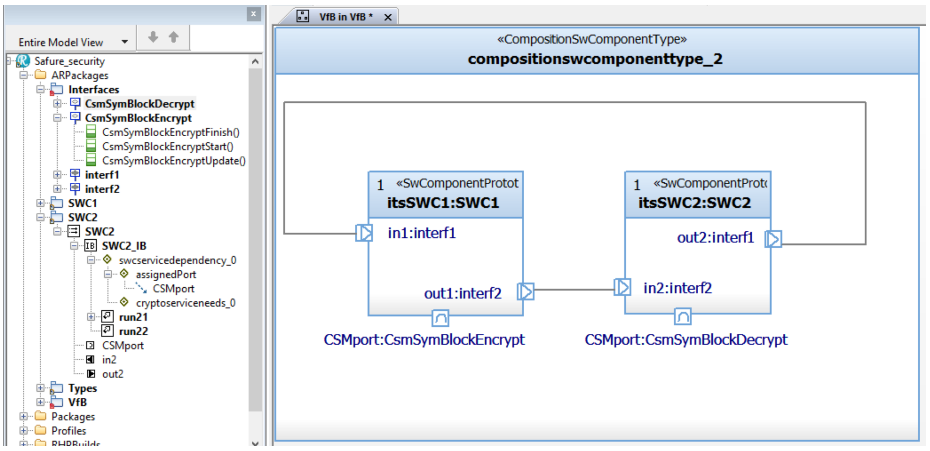


**Fig. 3.** Sender-Receiver application in Rhapsody

*Service Manager* (CSM) [20], which provides software components with cryptographic functionalities implemented in software or hardware. The cryptographic functionalities of the CSM include hash calculation, generation and verification of message authentication code, random number generation, encryption and decryption using symmetric and asymmetric algorithms, generation and verification of digital signature, and, finally, key management.

In AUTOSAR, *services* can be seen as an hybrid concept between BSW modules and software components. Software components that require AUTOSAR services use standardised service interfaces. The dependency of a software component from an AUTOSAR service is modelled by adding ports (hereinafter referred to as "service ports") to the software component. The interface for these ports needs to be one of the standardised service interfaces defined in the AUTOSAR documentation. A port interface has a single attribute called `isService`, that is set to true if the interface is actually provided by AUTOSAR services instead of another application component. Furthermore, the internal behaviour of the software component shall contain a `SwcServiceDependency`, which is used to add more information about the required service (a more detailed explanation of this element can be found in Section 4).

The Crypto Service Manager CSM provides an abstraction layer, with a standardised interface of cryptographic functionalities to higher software layers. The services offered by the CSM can be used locally only: it is not possible to access them from a different ECU.

In Figure 3, the software components `SWC1` and `SWC2` include two service ports of type `CsmSymBlockEncrypt` and `CsmSymBlockDecrypt`, respectively, to use the encryption/decryption service of the CSM respectively.

As shown on panel in the left hand-side of the figure, the functions available at the interface are, respectively, functions `SymBlockEncryptStart()`, `SymBlockEncryptUpdate()`

and `SymBlockEncryptFinish()` for `CsmSymBlockEncrypt`, and functions `SymBlockDecryptStart()`, `SymBlockDecryptUpdate()` `SymBlockDecryptFinish()` for `CsmSymBlockDecrypt`. Furthermore, the description of each software component includes the `SwServiceDependency` on `CryptoServiceNeeds` (left hand side of the figure).

This is the typical use of CSM functions by application components according to AUTOSAR. However, there are two drawbacks to this approach. Component developers are responsible for the selection and use of the right cryptographic functions for guaranteeing an adequate level of integrity and confidentiality to the data they exchange. In addition, the AUTOSAR model does not show on which component communication(s) the encrypted information is supposed to be used. This information is hidden inside the component implementation and disappears completely from the model.

## 3 Extending AUTOSAR for security

In terms of security, AUTOSAR models focus on the mechanisms that should be implemented as part of the BSW layers and are therefore to be considered as architecture patterns. However, AUTOSAR mostly disregards the application level, that is, for instance, how the designer of an application with security issues should specify that its communications need to be suitably protected.

We define a possible extension to AUTOSAR consisting of two types of modelling elements, defined in Rhapsody by means of stereotypes:

- the trust specification of a functional component and
- the security requirement specification of a communication between components.

### 3.1 Trust specification of functional components

A functional component, either a software component or a port, may be associated to a *Trust Specification* which specifies to what extent the element can be trusted to provide the expected function, or service, with respect to attacks targeted to compromise the functionality of the element (a metamodel view of the extensions is shown in Figure 4).

A trust specification consists of:

- a *trust specification identifier* (`trustSpecID`), which identifies the specification, and
- a *trust level* (`trustLevel`) which provides an indication of the extent to which the element can be trusted. The `trustLevel` is an attribute of type `trustLevelType` that corresponds to an integer in the range 1 to 5, being 1 the highest trust level and 5 the basic one.

It follows that a trust specification is formally defined as:

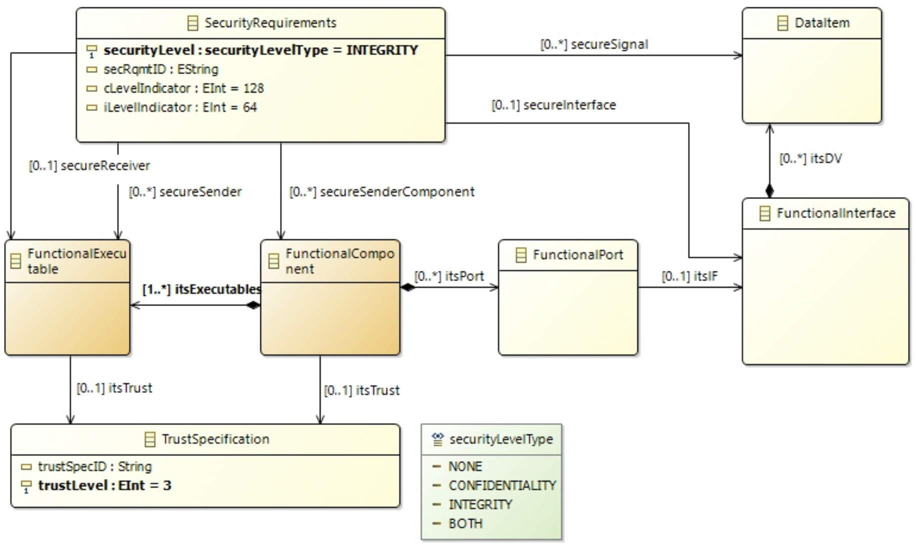


**Fig. 4.** The metamodel with the proposed AUTOSAR extensions

```
TrustSpecification = ⟨ trustSpecID: String,
                       trustLevel: trustLevelType ⟩
```

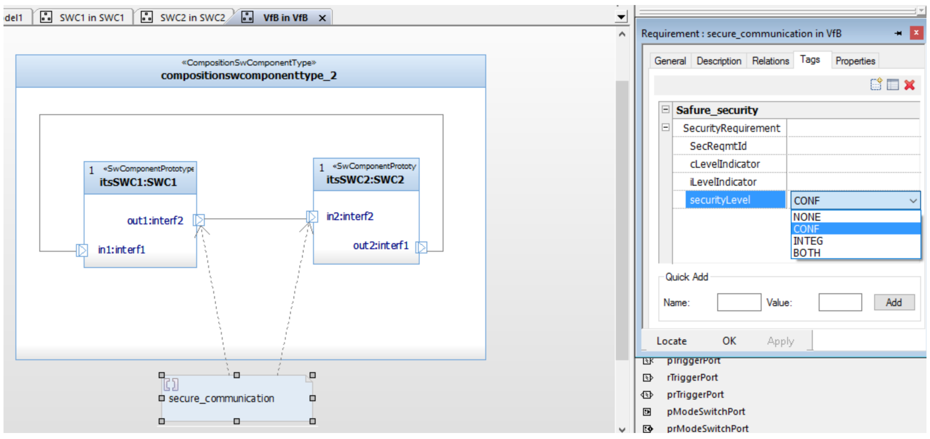
The trust specification is a measure of the effort required to create and carry out attacks to the element. An high trust level corresponds to a low probability of successful attack. The notion of trust specification is similar to *attack potential* in [21].

The trust level attribute is intended to be used in the mapping phase, upon defining the allocation of software components to ECUs. Components with high trust level should be assigned to high secure ECUs. As the mapping of software components to ECUs is outside the scope of this paper we are not going to refer to trust specification any further.

## 3.2 Security requirement specification of communications

With reference to Figure 5, a security specification on a communication consists of four attributes:

- a *security requirement identifier* (`secRqmtID`), which uniquely identifies the requirement;
- a *security level* (`securityLevel`), which specifies the desired secure communication options. It is of type `securityLevelType`, an enumerated that contains four self-explicative values that codify: no security, confidentiality, integrity, and, both confidentiality and integrity;
- a *confidentiality level indicator* (`cLevelIndicator`) and
- an *integrity level indicator* (`iLevelIndicator`).



securityLevel: CONF

**Fig. 5.** Secure communication.

A security requirement is formally defined as:

```
SecurityRequirement = ⟨ secRqmtID: String,
                        securityLevel: securityLevelType,
                        cLevelIndicator: EInt,
                        iLevelIndicator: EInt ⟩
```

where

```
securityLevelType = { CONF,INTEGR, BOTH, NONE }
cLevelIndicator = { 128, ....}
iLevelIndicator = { 64, .... }
```

The attributes *confidentiality level indicator* and *integrity level indicator* provide a quantitative indication of confidentiality and integrity, respectively, of communication. A confidentiality level indicator `cLevelIndicator` equal to 128 means that the computation complexity necessary to break the communication confidentiality should not smaller than $\mathcal{O}(2^{128})$. This requirement can be fulfilled by using the AES-128 cipher, for example. Analogously, an integrity level indicator `iLevelIndicator` equal to 64 means that the computation complexity necessary to break the communication integrity (i.e., to find a collision) should not be smaller than $\mathcal{O}(2^{64})$. According to AUTOSAR Secure On-Board Communication, this requirement can be fulfilled by using a 64-bit Message Authentication Code (MAC), or larger.

The definitions in the metamodel of Figure 4 allow for a security requirement to be applicable to any of the following.

– Selected data items exchanged between two ports by two components.
– All the data items exchanged between a specified pair of sender and receiver runnables.
– All the data produced by a component.

– All the data belonging to a given interface specification.

## 4 Automatic generation of security components

Given an AUTOSAR specification with secured communications, a script, written in Python, has been developed to automatically add security software components to the system according to the security properties defined for the communication in the AUTOSAR model.

The security requirements are implemented by using the services provided by the CSM. In order to use these services, components must have suitable elements (client ports, interfaces, and so on), and perform the following actions. On the sender side, the component: 1) identifies the data to protect as input; 2) invokes the CSM service to secure the data; 3) sends the secured data on the specified communication channel/RTE call. On the receiver side, the component invokes the CSM service to extract the secured data.

The script allows the user to choose between two possibilities. The first is to extend the already existing sender or receiver components of the secured communication. Alternatively, new components can be added, acting as filters.

Technically, this procedure has been implemented in the model, by extending the AUTOSAR description field of the ports involved in the communication (tag `desc`) by two additional tags: the tag `SecurityNeeds`, which specifies the security level (none, confidentiality, integrity or both), and the tag `NewComponent`, which indicates if a new (filter) software component must be added.

The standardised format for exchanging data between different AUTOSAR compliant tools is AUTOSAR XML (ARXML). The input parameters of the script are the name of the input ARXML file and, optionally, a name for the output file. If no name is specified for the output file, the script use a default name for it.

Suppose that we are interested in confidentiality. Therefore, a message should be encrypted by the sender and decrypted by the receiver. For this purpose, the CSM offers the Symmetrical Block Encryption (and Decryption) service, which guarantees confidentiality of the received data, under the condition that the key used for computation is not compromised by an external entity.

The steps in order to specify that a software component wants to use the Encryption/Decryption service, are the following:

– The software component that wants to use the Encryption/Decryption service needs to have a client port.
– The interface of such a port has to be named (all the names of all the interfaces provided by the CSM are defined by the AUTOSAR standard):
  • `CsmSymBlockEncrypt` if the software component is a sender, or
  • `CsmSymBlockDecrypt` if, otherwise, it is a receiver.
– `isService`, a flag of the port interface, specifies whether communication occurs between a software component and an AUTOSAR service (true) or not (false).

– `serviceKind`, an attribute of the port interface, provides further details about the nature of the applied service. In our case it must be set to `criptoServiceManager`.
– The internal behaviour element of the software component must contain a `SwcServiceDependency`, which makes it possible to associate ports defined for a software component to a given AUTOSAR service.
– `SwcServiceDependency` must contain both the required AUTOSAR service (`cryptoServiceNeeds`) and one or more `RoleBasedPortAssignment`, which is a container for references to the service client ports of the software component (defined at the first point).

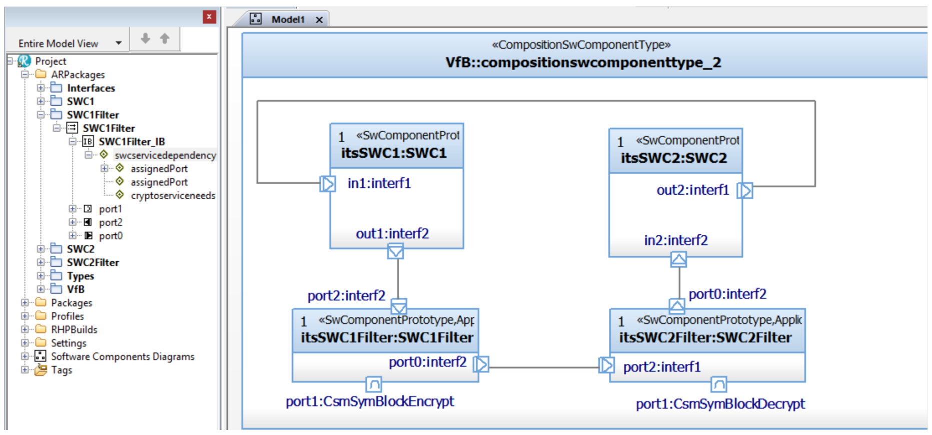


**Fig. 6.** Model with security components.

If we specify `SecurityNeeds=CONF` and `NewComponent=TRUE` for both the sender and the receiver of the secured communication in Figure 5, then executing the script on the ARXML representing the system in Figure 3 results in the system shown in Figure 6. Notice that the modified ARXML file is AUTOSAR-compliant and so can be imported in Rhapsody. Thus all the elements added by the script are visible in the graphical representation of the system. As it turns out, the script generated two components (`SWC1Filter` and `SWC2Filter`). On the sender side, the filter component `SWC1Filter` takes the data produced by the sender component `SWC1`, encrypts and sends them out to the filter component `SWC2Filter`. On the receiver side, this component receives the encrypted data, decrypts and sends them to the receiver component `SWC2Filter`.

## 5 A simple example

There are many instances of AUTOSAR communications that are sensitive to security issues and require support for guaranteeing the confidentiality and integrity of the exchanged information.

Many active safety and autonomous driving applications make use of information coming from sensory input devices, such as lidars, radars and cameras in order to sense the surrounding environment and detect the roadmarks and objects (vehicles, pedestrians) on and around the street.

To improve the precision in the detection of the vehicle position and to assist in the navigation, these data are typically integrated with information coming from the GPS system. GPS information can also be considered as not only characterized by integrity, but also confidentiality requirements.

Position information coming from the GPS, together with object and road position information coming from sensors are typically forwarded to several navigation and active safety functions, including, for example, Path planning, Lane keeping and Lane Departure warning. These functions, in turn, produce commands for the actuation systems (steering, throttle and brakes) for which integrity must be preserved. The low-level control systems for throttle, brakes and steering proceed to arbitrate among these incoming requests and determine the final actuation commands that go to the low-level control software.

The corresponding AUTOSAR model with sensors and actuators as `SensorActuatorsSwComponentType` and `ApplicationSwComponentType` elements is shown in Figure 7.

Confidentiality and integrity requirements are added to the communications between the `GPS` and the `PathPlanning` components, and between the latter and all the actuators. Other communications are only characterized by integrity requirements. The security tag has been added to the `desc` tag of the corresponding ports before exporting the ARXML file.

After the system is modelled using Rhapsody, we export the ARXML file. The file is processed by our generation script, which provides for the synthesis of the security mechanisms and the generation of a new ARXML file. Figure 8 shows the part of the model related to the GPS software component and its communication with other components obtained as a result of this step.

The figure shows the original elements and the newly generated ones. The generated filters components have client ports to call the CSM functions for integrity (MAC) and confidentiality (symmetric block encryption).

The automatic generation of the security components allows the developers to work at a high level of abstraction in a completely transparent way, without requiring knowledge of the details of the CSM and the cryptographic routines in it. In addition to filters, our script also generates the RTE calls for reading and writing over sender/receiver ports. The RTE calls for the `PathPlanningFilter` software component are shown below.

```
#if (!defined RTE_PathPlanningFilter_H)
#define RTE_PathPlanningFilter_H
#include <Std_types.h>
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */
;
Std_ReturnType Rte_Read_port0_data(int data);
;
Std_ReturnType Rte_Write_port9_data(int data);
;
Std_ReturnType Rte_Read_port10_data(int data);
;
Std_ReturnType Rte_Read_port11_data(int data);
;
Std_ReturnType Rte_Write_port8_data(int data);
;
Std_ReturnType Rte_Write_port12_data(int data);
;
Std_ReturnType Rte_Write_port7_data(int data);
;
Std_ReturnType Rte_Write_port8_data(int data);
;
Std_ReturnType Rte_Call_port3_CsmMacGenerateFinish();
Std_ReturnType Rte_Call_port3_CsmMacGenerateStart();
Std_ReturnType Rte_Call_port3_CsmMacGenerateUpdate();
Std_ReturnType Rte_Call_port4_CsmMacVerifyFinish();
Std_ReturnType Rte_Call_port4_CsmMacVerifyStart();
Std_ReturnType Rte_Call_port4_CsmMacVerifyUpdate();
Std_ReturnType Rte_Call_port5_SymBlockEncryptFinish();
Std_ReturnType Rte_Call_port5_SymBlockEncryptStart();
Std_ReturnType Rte_Call_port5_SymBlockEncryptUpdate();
Std_ReturnType Rte_Call_port6_SymBlockDecryptFinish();
Std_ReturnType Rte_Call_port6_SymBlockDecryptStart();
Std_ReturnType Rte_Call_port6_SymBlockDecryptUpdate();
#ifdef __cplusplus
} /* extern "C" */
#endif /* __cplusplus */
#endif
```

The names of the RTE functions have a standard format. For each port, the call for the RTE is composed by four parts. The first part is the return type, which is `Std_ReturnType`. The second one is defined by the port's interface. In case of a SenderReceiver interface, it is `Rte_Write` for sender ports and `Rte_Read` for receiver ones. In case of a ClientServer interface it is `Rte_Call`. The third part is the name of the port. Finally, the fourth one is defined again by the interface. In case of SenderReceiver interface, it is the name of the data related to the interface, whereas in the case of ClientServer interface it is the name of the operation related to the interface.

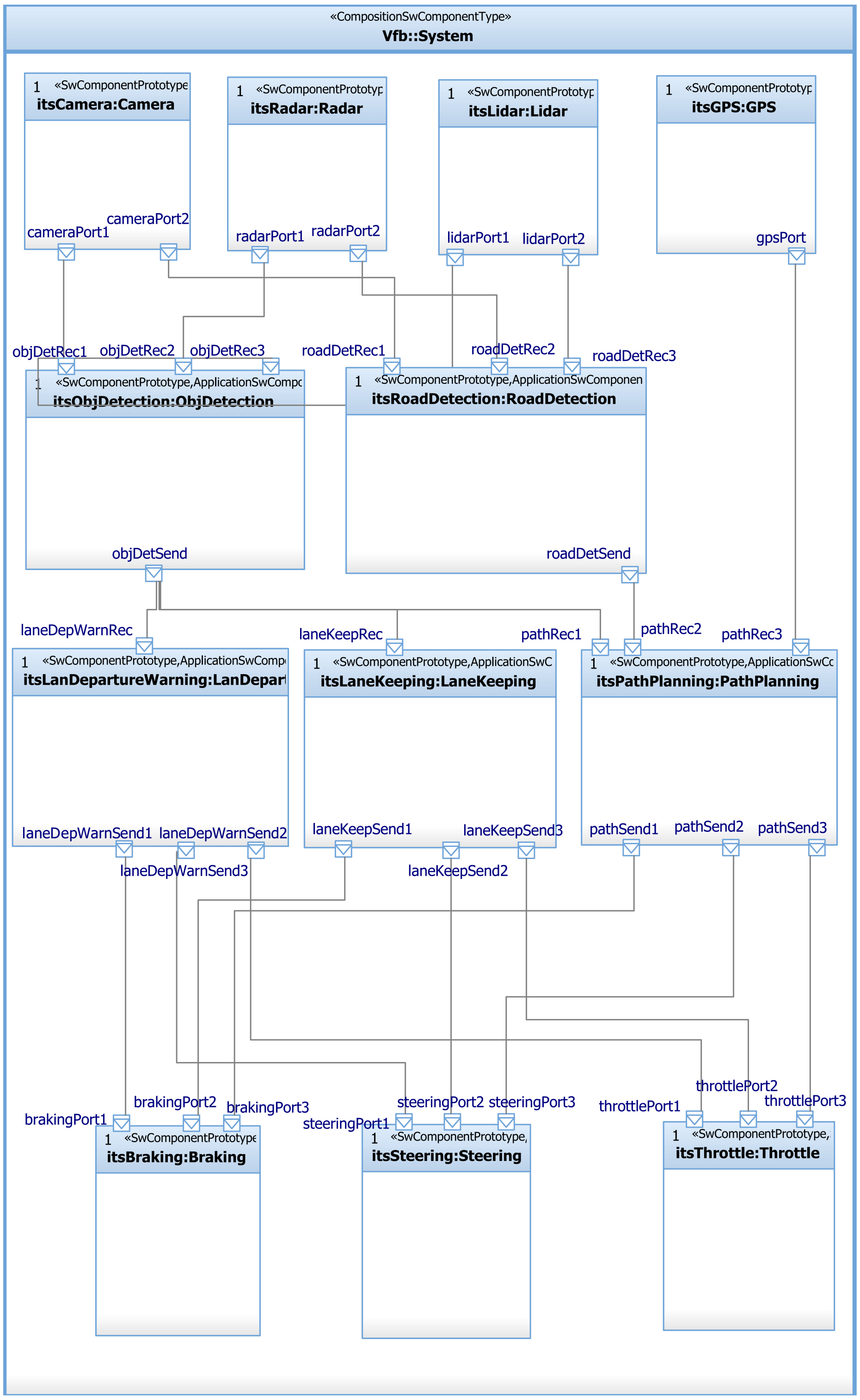


**Fig. 7.** AUTOSAR specification of the sample model.

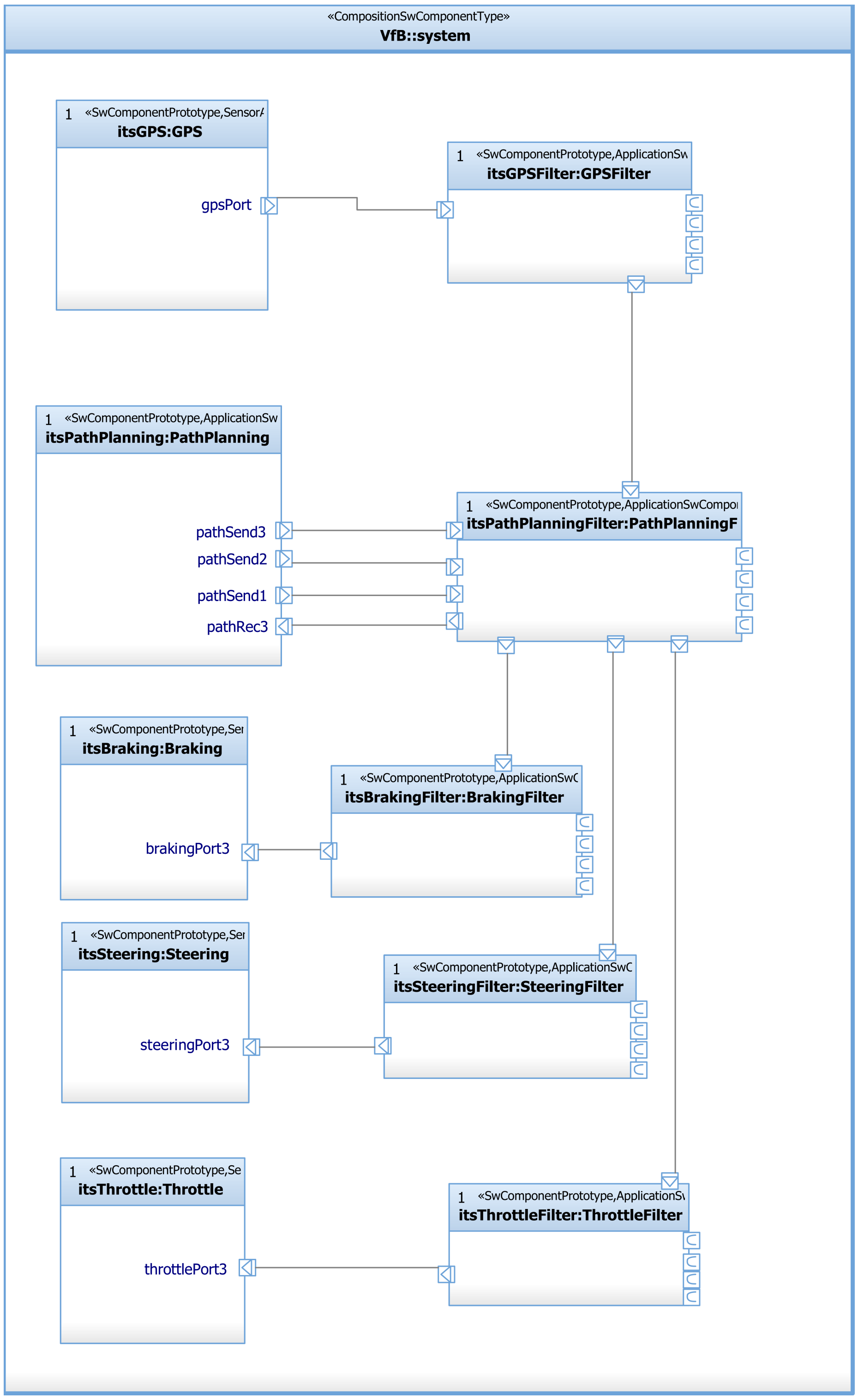

«CompositionSwComponentType»
VfB::system
1 «SwComponentPrototype,SensorA
itsGPS:GPS
gpsPort
1 «SwComponentPrototype,ApplicationSw
itsGPSFilter:GPSFilter
1 «SwComponentPrototype,ApplicationSw
itsPathPlanning:PathPlanning
pathSend3
pathSend2
pathSend1
pathRec3
1 «SwComponentPrototype,ApplicationSwCompo
itsPathPlanningFilter:PathPlanningF
1 «SwComponentPrototype,Se
itsBraking:Braking
brakingPort3
1 «SwComponentPrototype,ApplicationSwC
itsBrakingFilter:BrakingFilter
1 «SwComponentPrototype,Se
itsSteering:Steering
steeringPort3
1 «SwComponentPrototype,ApplicationSwC
itsSteeringFilter:SteeringFilter
1 «SwComponentPrototype,Se
itsThrottle:Throttle
throttlePort3
1 «SwComponentPrototype,ApplicationS
itsThrottleFilter:ThrottleFilter

**Fig. 8.** Part of the sample model with security components.

## 6 Conclusions

This paper advocates the modelling of security issues in the early phases of system design. Extensions to AUTOSAR for expressing the trust level of components and the security requirement of communications has been introduced. A tool has been implemented that automatically generates filters that implement the security issue on communications by using the CSM services. The tool is intended to be used to ease the work of system designers and to avoid oversight caused by the complexity of the AUTOSAR standard.

This paper is focused on security requirements specification, leaving verification of achievement of security goals to further work.

More in general, modeling should be not limited to just the communication aspect. Rather, modeling should address security as much as possible. For instance, well-known security engineering best-practices make it possible to harden the software components. These include, for example, using safe string libraries, diligent input validation, and checking function "contracts" at module boundaries. Modeling should allow the designer to require the employment of these practices. For instance, a class diagram, possibly extended by means of proper stereotypes, may mandate the use of a `SecureString` class instead of a customary `String` library.

In a similar fashion, modeling should address other aspects of the system. According to the "defence in depth" principle, just hardening the software components is in general not sufficient. Another design countermeasure consists in reducing the attack surface. Consider the Bluetooth vulnerability documented by Checkoway et al. [8], for example. Using a safe `strcpy` library function would certainly harden the Bluetooth implementation component. However, further security improvements could derive from requiring that, in contrast to current procedures, the Bluetooth component will respond to pairing requests only after user interaction.

### Acknowledgement

This work has been developed under the framework of the European project SAFURE (Safety And Security By Design For Interconnected Mixed-Critical Cyber-Physical Systems) under grant agreement No 644080.

## References

1. Simone Martini, Davide Di Baccio, Francisco Alarcón Romero, Antidio Viguria Jiménez, Lucia Pallottino, Gianluca Dini, and Aníbal Ollero. Distributed motion misbehavior detection in teams of heterogeneous aerial robots. *Robotics and Autonomous Systems*, 74:30–39, 2015.
2. Andrea Caiti, Vincenzo Calabro, Gianluca Dini, Angelica Lo Duca, and Andrea Munafo. Secure cooperation of autonomous mobile sensors using an underwater acoustic network. *Sensors*, 12(2):1967–1989, 2012.

3. Alexander M Wyglinski, Xinming Huang, Taskin Padir, Lifeng Lai, Thomas R Eisenbarth, and Krishna Venkatasubramanian. Security of autonomous systems employing embedded computing and sensors. *Micro, IEEE*, 33(1):80–86, 2013.
4. A. Sangiovanni-Vincentelli and Marco Di Natale. Embedded system design for automotive applications. 10:42–51, October 2007.
5. Erico Guizzo. How Google's self-driving car works. *IEEE Spectrum Online, October*, 18, 2011.
6. Arman Barari. GM Promises Autonomus Vehicles by End of Decade. `http://www.motorward.com/2011/10/gm-promisesautonomous-vehicles-by-end-of-decade`, October 17 2011.
7. Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE, 2010.
8. Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.
9. AUTOSAR, (`http://www.autosar.org/`).
10. Marco Di Natale and A. Sangiovanni-Vincentelli. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. 98(4):603–620, April 2010.
11. AUTOSAR. *Specification of Safety Extensions: AUTOSAR Release 4.2.1.*
12. AUTOSAR. *Specification of Security Extensions: AUTOSAR Release 4.2.1.*
13. Jan Jürjens. Umlsec: Extending uml for secure systems development. In *UML 2002—The Unified Modeling Language*, pages 412–425. Springer, 2002.
14. Jan Jürjens. Towards development of secure systems using umlsec. In *Fundamental approaches to software engineering*, pages 187–200. Springer, 2001.
15. David Basin, Jürgen Doser, and Torsten Lodderstedt. Model driven security for process-oriented systems. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 100–109. ACM, 2003.
16. Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin. On the need for extending marte with security concepts. In *International Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2011)*, 2011.
17. UML MARTE – The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems. `http://www.omgmarte.org/`.
18. AUTOSAR. *AUTOSAR Specification of Module Secure Onboard Communication: AUTOSAR Release 4.2.2.*
19. AUTOSAR. *AUTOSAR Specification of Crypto Abstraction Library: AUTOSAR Release 4.2.2.*
20. AUTOSAR. *AUTOSAR Specification of Crypto Service Manager: AUTOSAR Release 4.2.2.*
21. EVITA. Deliverable D2.3: Security requirements for automotive on-board networks based on dark-side scenarios. EU FP7 Project no. 224275, "E-safety vehicle intrusion protected applications," (`www.evita-project.org`), 2009.